Continuous Integration e *Continuous Delivery* em plataformas de *e-health*

Mestrado em Engenharia Informática – Computação Móvel

Rafael Tomás Correia

Leiria, setembro de 2023

Continuous Integration e *Continuous Delivery* em plataformas de *e-health*

Mestrado em Engenharia Informática – Computação Móvel

Rafael Tomás Correia

Trabalho de Projeto realizado sob a orientação dos Professores Ricardo Martinho, Rui Rijo e Carlos Ferreira.

Leiria, setembro de 2023

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o Autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, Curso de Mestrado em Engenharia Informática – Computação Móvel, no ano letivo 2022/2023, da Escola Superior de Tecnologia e Gestão do Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos.

Resumo

O presente projeto aborda o desenvolvimento e implementação de um *pipeline* de *continuous integration* e *continuous delivery* (CI/CD) voltado para a área da saúde, com ênfase numa plataforma dedicada ao apoio de cuidadores informais que prestam assistência em cuidados paliativos, denominada Help2Care-Pal. Este relatório descreve detalhadamente o processo de configuração e execução do referido *pipeline*, bem como a sua monitorização, num contexto sensível à qualidade e eficiência dos serviços disponibilizados.

A plataforma Help2Care-Pal compreende uma aplicação móvel para os cuidadores (*frontoffice*) e uma aplicação *web* para os profissionais de saúde (*backoffice*). Esta solução proporciona uma interação fundamental entre os cuidadores informais e os profissionais de saúde, através da disponibilização de conteúdos multimédia que capacitam e orientam os cuidadores. Esses conteúdos incluem imagens, vídeos, áudio, textos e *links* para recursos externos, todos geridos e disponibilizados de forma personalizada pelos profissionais de saúde. A aplicação móvel possibilita o acesso direto a esses recursos, desempenhando um papel importante no apoio aos cuidadores informais na prestação de cuidados paliativos.

O projeto abrange as diversas fases do ciclo de desenvolvimento de software e operações (DevOps), incluindo planificação, desenvolvimento, *build*, teste, *deployment*, operação e monitorização. O projeto foca-se nas fases de teste e *deployment* com o desenvolvimento e implementação do *pipeline* CI/CD que assegura a qualidade contínua da plataforma e a sua manutenção, garantindo uma entrega segura e eficiente.

O *pipeline* CI/CD foi implementado usando Jenkins e Cypress, e está integrado com o ambiente de produção. O *pipeline* foi bem-sucedido em melhorar a qualidade e confiabilidade da plataforma digital, e também ajudou a reduzir o tempo necessário para entregar novos recursos e correções de *bugs*.

Este projeto representa uma contribuição significativa para a área da saúde, destacando a importância da CI/CD como abordagens essenciais para garantir a eficácia e a qualidade das plataformas de suporte nesta área. Além disso, demonstra a aplicação prática destes conceitos num cenário real, ilustrando os benefícios tangíveis que podem ser alcançados por meio da implementação cuidadosa de práticas de DevOps.

Palavras-chave: *continuous delivery*, *continuous integration*, DevOps, *e-health*, *pipeline*

Abstract

The present project addresses the development and implementation of a continuous integration and continuous delivery (CI/CD) pipeline focused on the healthcare sector, with an emphasis on a platform dedicated to supporting informal caregivers providing palliative care, known as Help2Care-Pal. This report provides a detailed description of the configuration and execution process of the pipeline, as well as its monitoring, within a context that is sensitive to the quality and efficiency of the services provided.

The Help2Care-Pal platform consists of a mobile application for caregivers (front office) and a web application for healthcare professionals (back office). This solution facilitates essential interaction between informal caregivers and healthcare professionals by providing multimedia content that empowers and guides caregivers. These contents include images, videos, audio, texts, and links to external resources, all managed and customized by healthcare professionals. The mobile application enables direct access to these resources, playing a significant role in supporting informal caregivers in providing palliative care.

The project encompasses various phases of the DevOps cycle, including planning, development, building, testing, deployment, operation, and monitoring. The project primarily focuses on the testing and deployment phases with the development and implementation of the CI/CD pipeline, ensuring continuous quality of the platform and its maintenance, thus guaranteeing a secure and efficient delivery.

The CI/CD pipeline was implemented using Jenkins and Cypress and is integrated with the production environment. The pipeline successfully improved the quality and reliability of the digital platform, while also helping to reduce the time required to deliver new features and bug fixes.

This project represents a significant contribution to the healthcare field, highlighting the importance of CI/CD as essential approaches to ensuring the effectiveness and quality of support platforms in the healthcare sector. Furthermore, it demonstrates the practical application of these concepts in a real-world scenario, illustrating the tangible benefits that can be achieved through the careful implementation of DevOps practices.

Keywords: continuous delivery, continuous integration, DevOps, e-health, pipeline

Índice

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de siglas e acrónimos

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CP	Cuidados Paliativos
CSS	Cascading Style Sheets
CRUD	Create Read Update Delete
DSL	Domain-Specific Language
ESTG	Escola Superior de Tecnologia e Gestão
GCE	Google Compute Engine
GCP	Google Cloud Platform
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NPM	Node Package Manager
SDLC	Ciclo de Vida do Desenvolvimento de Software
SSH	Secure Shell
TI	Tecnologias de Informação
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine (Máquina Virtual)
WSL	Subsistema Windows para Linux

1. Introdução

A *continuous integration* (CI) e *continuous delivery* (CD) são práticas de desenvolvimento de software que automatizam a criação, o teste e o *deployment*[1] de aplicações de software. A CI garante que a aplicação está sempre atualizada e que as alterações são feitas de forma controlada e consistente [1]. A CD automatiza a instalação da aplicação no ambiente de produção, garantindo que as novas funcionalidades e correções de erros são lançadas de forma rápida e fiável [2].

Com o previsível crescimento da indústria da saúde digital, tanto a Statista [3] como a Grand View Research [4] prevêem o crescimento do mercado global de *e-health*, a Statista prevê que a receita aumente 11,16% até 2027, enquanto a Grand View Research prevê uma taxa de crescimento anual de 17% entre 2023 e 2030 no mercado global. Como resultado desse crescimento, espera-se um aumento na procura por software, ou seja, é necessário introduzir no mercado de forma mais rápida e com qualidade novas aplicações assim como atualizar as existentes. Nesse contexto, a CI e a CD desempenharão um papel cada vez mais importante para as aplicações de saúde, contribuindo para melhorar a qualidade, a segurança e a eficiência do software [5], [6].

O software médico é crítico em termos de segurança e quaisquer erros no código podem ter consequências graves para os doentes. A utilização de um *pipeline* de CI/CD permite automatizar processos que ajudam a assegurar que o software é desenvolvido, testado e *deployed* de forma mais rápida e confiável. É assim possível automatizar diferentes tarefas, como compilação de código, correr testes ou fazer *deploy* nos ambientes de teste e de produção.

Há sempre desafios para implementar um *pipeline* CI/CD na indústria da saúde, não só devido às normas existentes para software médico [7] (e por isso poderem existir tarefas ou ações difíceis de automatizar), mas também por diferentes desafios encontrados pelas organizações [8]. Alguns destes desafios são a falta de colaboração e comunicação entre as equipa de desenvolvimento e a equipa de operações, ou a falta de empregados qualificados,

[1] Termo utilizado para descrever o processo de colocação de um artefacto de software num ambiente de testes ou de produção, após ter sido construído e atualizado num repositório.

uma vez que requer competências e conhecimentos de desenvolvimento e operação, além de uma mudança de mentalidade [8].

1.1.Contexto

Este relatório de projeto foi desenvolvido no âmbito do 2.º ano do Mestrado em Engenharia Informática – Computação móvel, no ano letivo 2022/2023, lecionado na Escola Superior de Tecnologia e Gestão, do Politécnico de Leiria. Serve como documentação do trabalho desenvolvido no decorrer do projeto e como análise dos resultados obtidos. O projeto desenvolvido implicou a criação de um *pipeline* CI/CD utilizando como caso de estudo uma plataforma digital de *e-health* denominada de Help2Care-Pal.

O desenvolvimento da plataforma Help2Care-Pal é a continuação do projeto Help2Care [9], desenvolvido pelo Politécnico de Leiria com a finalidade de capacitar os cuidadores informais, nas tarefas que estes têm de efetuar no dia-a-dia para cuidar dos doentes. A plataforma Help2Care-Pal parte da continuação do trabalho efetuado com a plataforma Help2Care, mas foca-se nos grupos de cuidadores com doentes em cuidados paliativos. Esta nova plataforma usufruiu também do financiamento por parte da Fundação "la Caixa"[2].

1.2. Objetivos

Este projeto tem como objetivo aplicar as metodologias CI e CD em aplicações de saúde com a implementação de um *pipeline* CI/CD. Este projeto surgiu com a necessidade de recuperação da plataforma Help2Care, que se encontrava sem atualização há um considerado intervalo de tempo, para ser usada como base para o desenvolvimento da plataforma Help2Care-Pal. Sendo o objetivo final do desenvolvimento da plataforma Help2Care-Pal a sua colocação em produção. O desenvolvimento de um *pipeline* CI/CD tem como objetivo a criação de um artefacto que permita manter a qualidade e fiabilidade da plataforma, permitindo também agilizar as correspondentes atualizações. Conseguindo implementar de forma rápida e com qualidade a colocação de novas atualizações em produção, permite também obter, mais rapidamente, o *feedback* dos utilizadores.

[2] https://fundacaolacaixa.pt/

1.3. Estrutura

Este relatório começa, no capítulo Conceitos e Estado da Arte, por dar a conhecer os diferentes conceitos base do projeto assim como para a leitura deste relatório, seguido de um pequeno contexto da utilização de CI/CD na área de saúde. O capítulo é fechado com um pequeno estado de arte e considerações do mesmo. O capítulo seguinte, Fases do projeto e metodologias, introduz as diferentes fases pelo qual o projeto passou, assim como as metodologias utilizadas ao longo do desenvolvimento do projeto. Seguindo as fases do projeto apresentadas o próximo capítulo, Preparação do ambiente de desenvolvimento, apresenta as ações realizadas para preparar o ambiente de desenvolvimento, seguido do capítulo Arquitetura de um *pipeline* CI/CD e Ações onde é feita a introdução ao *pipeline* CI/CD desenvolvido neste projeto. No capítulo Implementação de um *pipeline* de CI/CD é apresentada a configuração do *pipeline*, e a configuração para utilização da ferramenta de testes Cypress, seguido pela explicação referente à implementação dos testes. É também apresentado e explicado o *dashboard* da ferramenta Jenkins e das diferentes ações realizadas no mesmo, assim como os resultados obtidos. No capítulo Caso de Estudo é apresentado o projeto Help2Care-Pal, os passos necessários para a sua utilização como caso de estudo, e a implicação da utilização do *pipeline* no mesmo. São também explicadas as ações realizadas para colocar o caso de estudo em produção e assim ter *feedback* por parte dos utilizadores. Este relatório é fechado com um capítulo de conclusões e trabalho futuro.

2. Conceitos e Estado da Arte

Este capítulo tem como principal objetivo apresentar os conceitos-chave abordados neste projeto, assim como descrever o problema em questão e, apresentar abordagens que podem ser utilizadas para ajudar a resolver esse problema.

2.1. Conceitos

Este subcapítulo introduz os diferentes conceitos necessários para uma boa compreensão do projeto descrito neste relatório. É realizada uma introdução às diferentes metodologias existentes de desenvolvimento de software e as suas diferenças, desde as metodologias tradicionais às metodologias ágeis, assim como também à metodologia DevOps e à utilização de CI e CD e a implementação e utilização de um *pipeline* CI/CD.

2.1.1. Modelos do Ciclo de Vida do Desenvolvimento de Software (SDLC)

Ciclo de Vida do Desenvolvimento de Software (SDLC do inglês *Software Development Lifecycle*) é um processo de construção e manutenção de software. Este processo abrange todo o processo de desenvolvimento de software, desde o planeamento inicial do projeto e obtenção de requisitos até ao desenvolvimento do projeto, teste, *deployment* e manutenção. Este processo compreende várias metodologias, que podem ser divididas em dois grupos, metodologias tradicionais e metodologias ágeis [10].

Neste subcapítulo apresenta-se os dois diferentes grupos, as metodologias tradicionais e as metodologias ágeis, focando as fases de teste e *deployment* onde a utilização de CI/CD mais diferenças introduz no SDLC.

A. Metodologias Tradicionais

As metodologias tradicionais são baseadas numa sequência de etapas, como definição de requisitos, construção de soluções, testes e *deployment*, e requerem também a definição e documentação de requisitos no início do projeto. As metodologias tradicionais são dependentes de um conjunto de processos pré-determinados e documentação contínua realizada ao longo do desenvolvimento [10].

Após a finalização total da etapa de desenvolvimento a equipa de desenvolvimento envia todo o software para a equipa de testes para iniciar a próxima fase. Desta forma existe pouca

comunicação entre ambas as equipas existindo uma separação entre elas. Este fenómeno é geralmente conhecido como "existência de silos". Esta separação entre as equipas pode levar a problemas, uma vez que a equipa de testes pode não ter informação suficiente para testar eficientemente o software. Esta separação aumenta também a repetição de processos, uma vez que ao serem encontrados problemas, a equipa de desenvolvimento tem de os corrigir e a equipa de testes voltar a repetir todo o processo de testes [11], [12].

A fase de *deployment* representa um passo critico na transição do software do desenvolvimento para a produção. Durante esta fase são realizadas várias atividades, incluindo preparação da documentação, treino dos utilizadores finais, se necessário, migração de dados (quando existe essa necessidade) configuração do ambiente de produção, planeamento e execução do *deployment*, verificação e validação do software no ambiente de produção para confirmar o bom funcionamento do mesmo. Estas ações são realizadas na sua maioria de forma manual, uma vez que a intervenção humana é necessária para garantir uma transição controlada do software do ambiente de desenvolvimento para o ambiente de produção [12]–[14].

B. Metodologias Ágeis

As metodologias ágeis baseiam-se em métodos de desenvolvimento de software adaptativos, onde não existe um planeamento detalhado e as tarefas futuras estão apenas relacionadas com características do software que devem ser desenvolvidas. Desta forma o software é testado frequentemente, minimizando assim o risco de falhas no futuro. A interação com os clientes é também um dos pontos fortes das metodologias ágeis. A comunicação entre as diferentes equipas e a documentação mínima são também características das metodologias ágeis [15].

Nestas metodologias, ao contrário das metodologias tradicionais em que é implementado um grande processo, o ciclo de vida do desenvolvimento é dividido em partes mais pequenas, sendo estas chamadas de incrementos ou iterações, onde é abordada cada uma das fases convencionais do desenvolvimento. Seguindo o Manifesto para o Desenvolvimento Ágil de Software, esta metodologia prioriza indivíduos e interações, software funcional, colaboração com o cliente e resposta à mudança.

Nas metodologias ágeis, a fase de testes não é uma fase separada das restantes, mas ocorre simultaneamente com a fase de desenvolvimento e envolve a participação de todos os

membros da equipa do projeto [11]. Nas metodologias ágeis os testes seguem uma estrutura simples, não sendo necessária grande documentação, uma vez que se foca na comunicação e colaboração entre as equipas, permitindo também maior conhecimento sobre o software por parte da equipa de testes. Esta fase também promove a comunicação com o cliente, aumentando também a fiabilidade dos testes [11].

C. Diferenças entre metodologias tradicionais e ágeis

A Tabela 2.1 apresenta uma comparação entre as metodologias tradicionais e as metodologias ágeis, com base nos artigos [15]–[18], que permite de uma forma sintetizada perceber as diferenças entre metodologias tradicionais e ágeis.

Tabela 2.1 - Diferenças entre as metodologias tradicionais e ágeis

Características	Metodologias Tradicionais	Metodologias Ágeis
Fundamentos	Sistemas inteiramente definíveis, previsíveis e são construídos através de um plano minucioso e abrangente.	Software adaptável de alta qualidade é criado por pequenos grupos que utilizam procedimentos de *design* contínuo, testes e aperfeiçoamento baseados em *feedbacks* rápidos e alterações.
Abordagem do planeamento	Planeamento e documentação extensivos antecipadamente.	Planeamento adaptativo e abordagem iterativa.
Gestão de riscos	Mitigação dos riscos através de planeamento detalhado e análise de riscos.	Gestão de riscos através de contínua adaptação e monitorização.
Colaboração em equipa	Estrutura hierárquica e isolada, como silos.	Equipas multifuncionais e auto-organizadas com alta colaboração.
Alterações	Resistência a alterações ou requerem pedidos formais.	Aceita mudanças e recebe-as de braços abertos ao longo do processo de desenvolvimento.

Frequência de entregas	Longos ciclos de desenvolvimento com menos entregas (*releases*)	Iterações curtas e frequentes com entregas regulares.
Documentação	Realce em documentação extensiva.	Mínima documentação, focando-se no desenvolvimento de software de alta qualidade.
Envolvimento do cliente	Envolvimento limitado do cliente durante o processo de desenvolvimento.	Envolvimento ativo do cliente e *feedback* contínuo.
Qualidade do software	Supervisão e planeamento intensos e extensos. Testes extensos, tardios e de grande escala.	Colocação em produção, entregas, *feedback*, melhorias, validação, integração e testes contínuos.

2.1.2. DevOps

A metodologia DevOps é uma abordagem de desenvolvimento de software que destaca a comunicação e colaboração entre as equipas de desenvolvimento, testes e operações. Esta metodologia tem como objetivo reduzir o ciclo de vida do desenvolvimento de software e melhorar a monitorização de toda a infraestrutura, automatizando e integrando os processos entre as diferentes equipas. Esta metodologia é uma mudança da abordagem tradicional de desenvolvimento de software, onde as equipas de desenvolvimento, teste e operações trabalhavam de forma independente, como se de silos se tratassem. Esta abordagem tradicional leva muitas vezes a atrasos e problemas, uma vez que as equipas não têm uma compreensão partilhada do software ou dos processos envolvidos no desenvolvimento ou lançamento do software [19].

A metodologia DevOps quebra assim estes silos reunindo as diferentes equipas e incentivando-as a trabalhar juntas. Isto é conseguido através de diferentes práticas, como comunicação, colaboração, automação e desenvolvimento iterativo, significando assim uma mudança cultural [20].

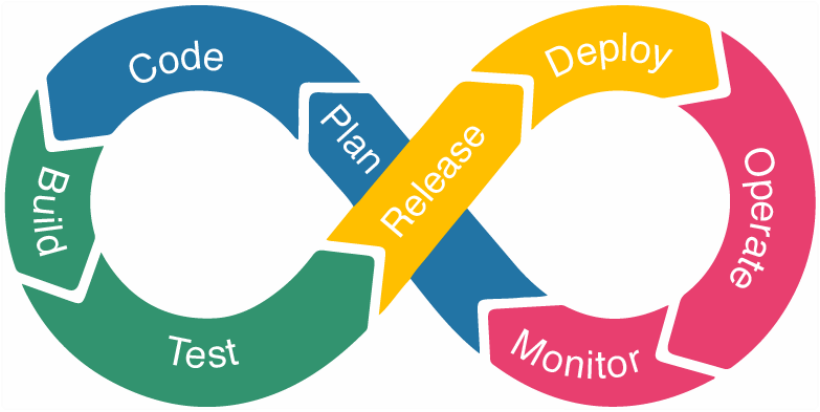

Figura 2.1 - Ciclo de vida DevOps [21]

O ciclo de vida DevOps, representado na Figura 2.1 é um processo de desenvolvimento continuo de software, que utiliza as melhores práticas DevOps para planear, construir, integrar, testar, *deploy*, monitorizar, operar e fornecer *feedback* contínuo ao longo do ciclo de vida do software. Assim, este ciclo de vida é dividido em diferentes fases [21]:

- Planear (*Plan*) envolve a identificação dos diferentes requisitos do software, *design* da arquitetura do sistema e a criação de um plano de desenvolvimento e *deployment;*
- Desenvolver (*Code*) consiste no desenvolvimento do código do software, seguindo os requisitos definidos na fase anterior;
- Compilação (*Build*) envolve a compilação do código desenvolvido num artefacto funcional e testável;
- Testar (*Test*) engloba todos os testes realizados ao artefacto produzido na fase anterior para garantir o cumprimento dos requisitos definidos e que o mesmo esteja livre de erros;
- Lançar (*Release*) consiste na criação de uma versão do software pronta para ser *deployed*;
- *Deploy* envolve a disponibilização do software aos seus utilizadores;
- Operar (*Operate*) envolve a monitorização do software já em produção e a correção de problemas encontrados;
- Monitorizar (*Monitor*) envolve a recolha de dados sobre o desempenho e utilização do software.

As diferentes fases do ciclo de vida DevOps são também muitas vezes divididas nos sete Cs, *continuous development*, *continuous integration*, *continuous testing*, *continuous*

deployment, *continuous feedback*, *continuous monitoring* e *continuous operations*. Esta associação entre as fases e os Cs é possível observar-se na Figura 2.2.

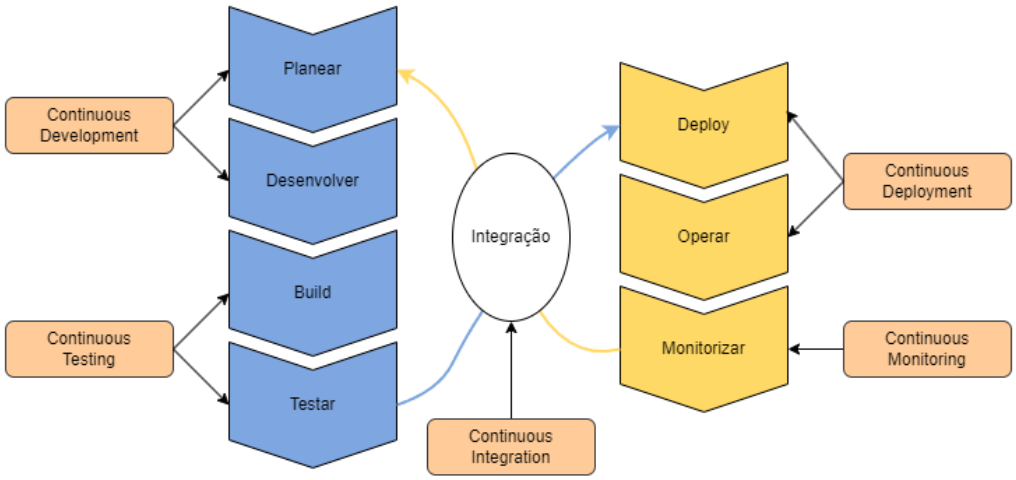

Figura 2.2 - Fases do ciclo de vida DevOps e 7 C's (fonte: edureka!)

A metodologia DevOps tem vindo a evoluir e muitas ferramentas têm sido desenvolvidas para ajudar a suportá-la. Na Tabela 2.2 é possível observar diferentes exemplos de ferramentas divididas por três fases, construção (*build*), *deployment* e operações (*operations*) [20].

Tabela 2.2 - Ferramentas de automação para DevOps [14]

Ferramenta	Fase DevOps	Tipo Ferramenta	Formato de Configuração	Linguagem	Licença
Ant	Construção	Construção	XML	Java	Apache
Maven	Construção	Construção	XML	Java	Apache
Rake	Construção	Construção	Ruby	Ruby	MIT
Gradle	Construção	Construção	Groovy	Java e Groovy	Apache
Jenkins	Construção	Integração Contínua	UI[3]	Java	MIT

[3] Interface gráfica para o utilizador

TeamCity	Construção	Integração Contínua	UI	Java	Comercial
Bamboo	Construção	Integração Contínua	UI	Java	Comercial
Puppet	*Deployment*	Gestão de Configurações	DSL[4] similar a JSON[5]	Ruby	Apache
Chef	*Deployment*	Gestão de Configurações	Ruby	Ruby	Apache
Ansible	*Deployment*	Gestão de Configurações	YAML	Python	GPL[6]
Loggly	Operações	Gestão de Logs	-	Baseado na Nuvem	Comercial
Graylog	Operações	Gestão de Logs	-	Java	*Open Source*
Nagios	Operações	Monitorização	-	C	*Open Source* e GPL
New Relic	Operações	Monitorização	-	-	Comercial
Cacti	Operações	Monitorização	-	PHP	GPL

A Tabela 2.2 compara diferentes ferramentas existentes no mercado atual. Estas ferramentas estão divididas pelas três fases: DevOps, construção, *deployment* e operações. É também possível comparar qual o formato de configuração usado por cada uma delas, a linguagem ou o tipo de licença. Esta é uma comparação importante uma vez que se deve escolher as

[4] Domain-Specific Language
[5] JavaScript Object Notation
[6] Licença publica GNU (https://www.gnu.org/licenses/gpl-3.0.html)

ferramentas de acordo com o projeto em que se vai aplicar a metodologia DevOps para assim obter o melhor rendimento possível de cada uma das ferramentas utilizadas.

2.1.3. *Continuous Integration* e *Continuous Delivery*

Martin Fowler [22] refere que o termo *continuous integration* é introduzido por Kent Beck como parte das práticas definidas para a metodologia *extreme programming* [23]. Fowler define CI como uma prática de desenvolvimento de software que envolve a integração frequente do trabalho realizado pelos membros de uma equipa, normalmente de forma diária, resultando assim em múltiplas integrações por dia. Cada uma das integrações é submetida a uma verificação automatizada, através de testes, a fim de detetar erros de desenvolvimento. Esta prática tem demonstrado significativa redução de problemas de integração e permite assim que a equipa desenvolva software coeso e fiável de forma mais ágil.

Mais tarde, Jez Humble e David Farley [2], estendem a ideia de CI para uma abordagem de CD como um conceito de *pipeline*. A ideia de CD proveio do primeiro princípio do manifesto ágil [24], "A nossa maior prioridade é, desde as primeiras etapas do projeto, satisfazer o cliente através da entrega rápida e contínua de software com valor".

Atualmente, a CI tem-se tornado uma prática fundamental no desenvolvimento de software, oferecendo inúmeros benefícios, como redução de riscos e melhor visibilidade do processo de desenvolvimento. Ao integrar o trabalho de forma frequente e automatizar os processos de compilação e teste, as equipas podem assim detetar rapidamente erros de integração, garantindo um produto de software coeso e confiável [6]. Além disso, a CI serve de base para a CD, assim a CD estende os benefícios da CI ao permitir que sejam realizadas entregas frequentes e automatizadas em ambientes de produção. Com a CD as equipas de desenvolvimento conseguem fornecer rapidamente novos recursos e atualizações aos utilizadores, obter *feedback* mais rápido, assim como promover a colaboração entre clientes e desenvolvedores.

É essencial reconhecer que tanto CI como CD requerem uma base sólida em ambas as práticas, assim como também uma bateria de testes bem projetada e abrangente. Testes automatizados confiáveis e monitoramento abrangente são fundamentais para garantir que as implantações não introduzam regressões ou afetem negativamente o desempenho do sistema. A avaliação regular e a melhoria do *pipeline* de implantação são necessárias para manter um equilíbrio entre velocidade e estabilidade.

A Figura 2.3 permite observar as diferenças existentes entre CI e CD, como CI permite que exista CD, uma vez que esta é dependente da existência de CI. É também possível observar-se as diferentes etapas de CI e CD.

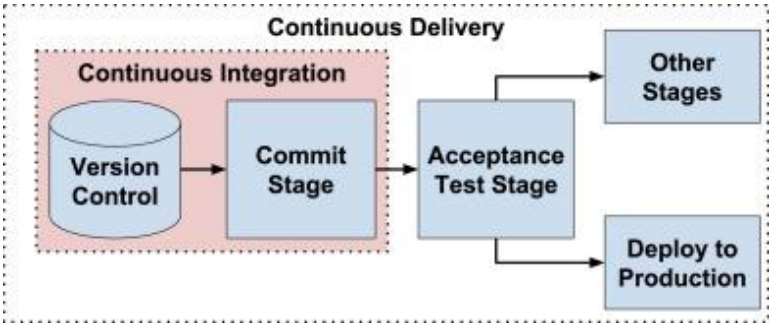

Figura 2.3- A diferença conceptual entre CI e CD (fonte:[25]**)**

2.1.4. Pipeline de CI/CD

Para uma correta aplicação de CI e CD é necessário implementar um *pipeline* de automação para simplificar todas as ações que têm de ser realizadas e ,com o premir de um botão, efetuar essas ações. Para implementar esse *pipeline* existem diversas ferramentas disponíveis no mercado. Alguns exemplos de ferramentas existentes são: Jenkins, CircleCi, TeamCity, Bamboo, GitLab, Buddy, Azure DevOps e GitHub Actions [26]–[28].

Na Tabela 2.3 é possível comparar algumas das diferentes ferramentas existentes para a aplicação de CI e CD. Todos estes exemplos têm em comum suportar diferentes plataformas assim como suportar diferentes linguagens de programação. As ferramentas que apresentam planos pagos, têm diferentes preços das licenças consoante o tamanho da equipa, uso e recursos adicionais.

Tabela 2.3 - Comparação de ferramentas para construção do pipeline de CI/CD (fonte: [26]–[28]**)**

Ferramenta	Principais recursos	Configuração	Preço da Licença
Jenkins	Open-Source, extensível, amplo ecossistema de plugins	Utilização de Jenkinsfiles ou interface gráfica	Gratuito
CircleCI	Baseado em nuvem, rápidas complicações	Arquivos YAML	Gratuito para uso limitado, disponíveis planos pagos

Travis CI	Baseado em nuvem, Forte comunidade	Arquivos YAML	Gratuito para projetos de código aberto, disponíveis planos pagos
GitLab CI/CD	Integrado ao GitlLab, revisão de código embutido	Arquivos YAML	Gratuito com a edição comunitária do GitLab, disponíveis planos pagos
Azure DevOps (Pipelines)	Integração com o ecossistema da Microsoft	Arquivos YAML	Gratuito para equipas pequenas, disponíveis planos pagos
Bamboo	Integração com o ecossistema da Atlassian	Bamboo Specs	Planos pagos com base no número de agentes de compilação
GitHub Actions	Integração com o GitHub, automação fácil	Arquivos YAML	Gratuito com uso limitado, disponíveis planos pagos
TeamCity	Recursos avançados, fácil integração	Interface baseada na web	Gratuito para equipas pequenas, disponíveis planos pagos
Buddy	Configuração simplificada, extensas integrações	Interface baseada na web	Gratuito para uso limitado, disponíveis planos pagos

2.2.*Continuous Integration* e *Continuous Delivery* na Saúde

A CI/CD pode ser especialmente benéfica para o setor de saúde, pois pode ajudar a melhorar a qualidade e a segurança do software de saúde. O setor de saúde é um dos setores mais regulamentados do mundo, e o software de saúde deve atender a uma série de requisitos rigorosos de conformidade. A CI/CD pode ajudar a garantir que o software de saúde seja

desenvolvido e *deployed* de forma consistente e confiável, o que pode ajudar a reduzir o risco de violações de segurança e outros problemas [5].

Tanto em [6] como em [5] discutem-se os benefícios da CI/CD para o setor de saúde. Os artigos argumentam que a CI/CD pode ajudar a melhorar a qualidade, a segurança e a eficiência do software de saúde. Os artigos também discutem os desafios da implementação da CI/CD na saúde, como a necessidade de formação e recursos e a necessidade de superar a resistência da indústria.

Alguns exemplos específicos de como a CI/CD está a ser usada no setor de saúde são:

- O Mayo Clinic está a usar CI/CD para desenvolver e fazer *deploy* de software de saúde de forma mais rápida e eficiente;
- O Memorial Sloan Kettering Cancer Center está a usar CI/CD para melhorar a qualidade e a segurança do seu software de saúde;
- O Kaiser Permanent está a usar CI/CD para reduzir os custos de desenvolvimento e manutenção de software de saúde.

A CI/CD é um conceito poderoso que pode ajudar as organizações de saúde a melhorar a qualidade, segurança, produtividade, visibilidade e os custos do desenvolvimento de software. No entanto, há desafios que precisam ser superados para implementar a CI/CD na saúde, como a necessidade de formação e recursos específicos para a área da saúde, e a necessidade de superar a resistência da indústria à mudança.

A metodologia CI/CD permite às equipas de desenvolvimento implementar em código regras necessárias para cumprir as leis e normas de segurança e privacidade impostas pelas agências reguladoras e assim automatizar a confirmação do cumprimento destas regras e normas. [33].

2.3.Métodos de Pesquisa

Com a finalidade de obter conhecimento sobre as áreas abordadas neste projeto, assim como também sobre projetos semelhantes já realizados, foi realizada uma pesquisa do estado de arte e conceitos relacionados com o projeto.

A pesquisa foi conduzida utilizando diferentes motores de busca de literatura académica, incluindo as plataformas, Google Scholar, Springer, IEEE Xplore e PubMed. Estas plataformas abrangem uma ampla gama de publicações académicas, como livros, revistas

especializadas, conferências e literatura médica A realização da pesquisa com base em palavras-chave e termos (*continuous integration*, *continous delivery*, saúde, DevOps e *pipeline* CI/CD) teve como objetivo identificar recursos que permitissem a obtenção de conhecimento teórico e de abordagens que complementassem o conhecimento existente sobre o tema do projeto e sobre como proceder na implementação do *pipeline* CI/CD. A pesquisa realizada permitiu encontrar publicações sobre a utilização de CI/CD e DevOps na saúde, nos 7 artigos seguintes [5]–[7], [29]–[32]. Foram também encontradas 3 publicações referentes a aplicações em cuidados paliativos [33]–[35], assim como referências de suporte aos métodos e ferramentas utilizadas durante o desenvolvimento do projeto.

2.4. Trabalhos Relacionados

Este subcapítulo apresenta diferentes trabalhados relacionados com a utilização de CI/CD em *e-health*, encontrados após a pesquisa realizada.

A. *Continuous Security though Integration Testing in an Electronic Health Records Systems* [32]

Neste artigo, os autores implementam um software de *continuous security* para um sistema de registos de saúde. Para conseguirem atingir este objetivo fazem uso das metodologias CI e CD. Os autores encontram duas razões para que, normalmente, este tipo de desenvolvimento não seja utilizado: métricas e práticas de desenvolvimento de software. Para o desenvolvimento do software fizeram uso de Pytest[7] BDD e *Behavior Driven Development* [36], que permitiu melhorar a colaboração entre os programadores e os planeadores dos produtos. Os autores fizeram uso do *Common Vulnerability Scoring System* (CVSS) [37] para obter a pontuação dos testes unitários realizados e assim medir cada gravidade identificada como também o risco associado.

B. *Challenges in Adopting Continuous Delivery and DevOps in a Globally Distributed Product Team: A Case Study of a Healthcare Organization* [29]

Os autores apresentam os desafios que uma organização de saúde passou para adotar CD e DevOps [38] num projeto com uma equipa localizada em três países diferentes: India,

[7] https://docs.pytest.org/en/7.2.x/

Alemanha e Estados Unidos da América, existindo diferentes tarefas em cada um dos países como é possível observar-se na Figura 2.4.

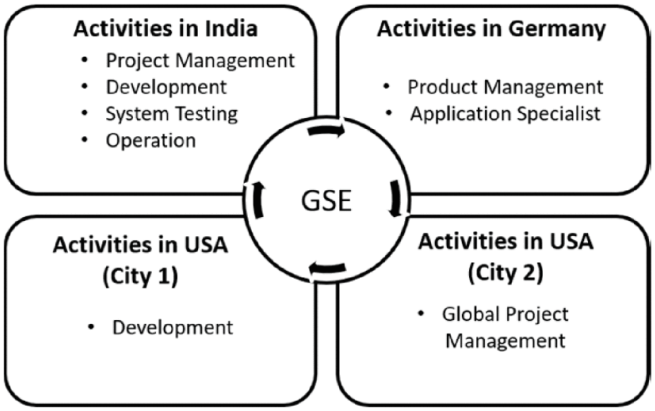

Figura 2.4 - Localização das equipas do projeto e respetivas atividades (fonte: [27])

Foram identificados diferentes desafios ao longo do estudo realizado, tais como, desafios de comunicação e colaboração entre as equipas, desafios de gestão de mudanças e configurações, desafios de segurança e conformidade e desafios de escalabilidade. Sendo identificados posteriormente diferentes práticas para ajudar a organização a superar os desafios encontrados, incluindo, o investimento em ferramentas e infraestruturas que suportam CD e DevOps, a criação de uma cultura de colaboração e comunicação entre as equipas, desenvolver um processo de gestão de mudanças e configuração eficaz e eficiente, adotar uma abordagem de segurança e conformidade proativa e projetar sistemas escaláveis desde o início do desenvolvimento. É concluído no artigo que a adoção da CD e do DevOps pode ser um desafio para organizações globalmente distribuídas, mas também pode trazer uma série de benefícios. Ao identificar e superar os desafios, as organizações podem melhorar a velocidade, confiabilidade e qualidade das suas entregas de software.

C. *Implementation of DevOps in healthcare systems* [30]

O autor explora, tal como os autores do artigo anterior, a implementação de DevOps em sistemas de saúde através de uma revisão sistemática da literatura. Nesta, o autor explica o porquê de utilizar-se DevOps na indústria da saúde, como implementar, quais as melhores práticas, assim como os seus benefícios e o aumento da interação e satisfação dos pacientes. O autor reforça que a utilização de típicas metodologias SDLC irá aumentar os gastos do projeto como reduzir a aproximação do projeto com os seus utilizadores, dando assim a utilização de DevOps como uma solução para estes problemas. Para a boa implementação de DevOps, é reforçada a utilização de CI e CD nos diferentes passos a seguir para a

implementação de DevOps. Ainda refere como nos Estados Unidos da América os sistemas de saúde estão a beneficiar da utilização de DevOps, uma vez que cada vez mais doentes estão a recorrer a telemedicina.

D. *DevOps in regulated* software *development: Case medical devices* [7]

Os autores referem como desde 2017 a utilização de DevOps se tem tornado popular na indústria de desenvolvimento de software, mas que em ambientes regulamentados como por exemplo na saúde nem sempre é algo simples uma vez que existem leis que devem ser seguidas. Uma vez que o foco da DevOps está na constante entrega de um software com qualidade e contínua melhoria, é necessário que tarefas repetitivas sejam automatizadas. Assim são criados testes automatizados que asseguram a qualidade do software, mas sendo esta uma área regulamentada este processo não é tão fácil, uma vez que a validação do cumprimento das leis é feita de forma manual.

Os autores examinaram dois dispositivos médicos e duas normas IEC/ISO para software de saúde, sendo estas normas a IEC 62304[8] e a IEC 82304-1[9]. Os autores concluíram que neste caso de estudo estas normas requerem especial atenção para uma integração contínua (CI), mas impede um *deployment* contínuo após a primeira disponibilização aos utilizadores. Os autores referem também que é necessário o desenvolvimento de novas ferramentas e métodos para permitir a utilização de DevOps em desenvolvimento de software regulamentado.

2.5. Conclusões

Como se pode verificar, existe uma enorme quantidade de aplicações digitais de saúde [35] e esse número tende a crescer [39]. Com isso, a necessidade de apresentar aplicações de qualidade e entregues de forma eficiente tem aumentado também. Como em outros setores, a área da saúde necessita de se manter atualizada, entregando aplicações mais rapidamente sem comprometer a qualidade.

No entanto, ao realizar a pesquisa, verifica-se que a adoção das práticas CI e CD em aplicações de saúde não é algo largamente documentado, comparando com a sua utilização noutras áreas. Isto pode estar ligado à necessidade de cumprir diferentes leis, sejam elas

[8] https://www.iso.org/standard/38421.html
[9] https://www.iso.org/standard/59543.html

internacionais ou nacionais, mas que podem apresentar barreiras à implementação destas metodologias. Além disso, há relatos de alguma resistência por parte da indústria como documentado nos artigos [6] e [5].

Embora existam tentativas da sua utilização ou a adaptação para a utilização em certas fases do desenvolvimento das aplicações de saúde, muitas vezes isso requer validação manual rigorosa devido à natureza sensível e crítica destas aplicações.

A utilização de CI/CD e, mais amplamente, a adoção de DevOps na indústria da saúde estão, de facto, em crescimento, como constatado em [30]. No entanto, qualquer utilização deve ser cuidadosamente ponderada, tendo em conta o objetivo da aplicação, o seu público-alvo e as normas existentes [40]. Pode ser necessário desenvolver novas ferramentas ou métodos [7] específicos para responder às necessidades do setor. Assim a adoção das práticas de CI/CD no contexto da saúde deve ser avaliada individualmente em cada projeto, pois este não é um processo padrão devido à complexidade destes projetos.

3. Fases do projeto e metodologias

Este capítulo apresenta as diferentes fases pelo qual o projeto passou ao longo do seu desenvolvimento. Nestas estão englobadas as fases da construção e implementação do *pipeline* CI/CD, assim como as fases do projeto Help2Care-Pal, uma vez que o projeto vai ser desenvolvido fazendo uso do *pipeline* CI/CD. É também descrita a metodologia Kanban, adotada ao longo do desenvolvimento do software.

3.1. Fases do Projeto

O projeto foi dividido em três grandes fases, compostas por diferentes tarefas. As fases definidas foram:

1. preparação do ambiente de desenvolvimento;
2. criação e implementação do *pipeline* CI/CD;
3. desenvolvimento do caso de estudo.

A primeira fase, a preparação do ambiente de desenvolvimento, englobou as tarefas:

1. atualização das versões das *frameworks* e bibliotecas do projeto Help2Care para este sofrer as alterações necessárias para o caso de estudo, Help2Care-Pal;
2. a criação do ambiente de desenvolvimento.

Após a primeira fase estar completa foi possível iniciar a segunda fase: a criação e implementação do *pipeline* CI/CD. Esta foi composta pelas seguintes tarefas:

1. criação das máquinas virtuais para os ambientes de teste e produção numa plataforma de computação em nuvem;
2. configuração das diferentes máquinas virtuais;
3. configuração da ligação *pipeline*-repositório.

A última fase - desenvolvimento do caso de estudo, é também a fase mais longa do projeto, sendo esta dividida nas tarefas:

1. adaptação do código do projeto Help2Care para início do desenvolvimento do projeto Help2Care-Pal;
2. desenvolvimento das funcionalidades requisitadas;
3. desenvolvimento dos testes à plataforma *backoffice;*

4. *deployment* do projeto;
5. manutenção e implementação de alterações com base no *feedback* dos utilizadores.

A última fase do projeto permitiu a utilização do *pipeline* CI/CD e assim perceber a sua implicação no desenvolvimento do caso de estudo e na sua manutenção.

3.2. Kanban

O Kanban é uma metodologia de gestão visual que tem como objetivo melhorar a eficiência, a colaboração e a visibilidade do fluxo de trabalho. Esta metodologia foi utilizada em junção com o Scrum e recorrendo à plataforma Trello para ajudar na gestão dos objetivos de cada sprint.

A plataforma Trello, utiliza o conceito de Kanban digital, sendo as tarefas representadas por cartões movidos entre colunas, representando diferentes estágios do processo. O quadro deste projeto estava dividido em seis etapas, *design*, *backlog*, *to-do*, *doing*, *testing*, *done*, como é possível observar na Figura 3.1. Os novos cartões eram criados na etapa *backlog*, sendo este movidos pelas diferentes etapas consoante o seu estado de desenvolvimento.

Cada uma das seis etapas está definida da seguinte forma:

- ***Design:*** Existindo uma componente de aplicação web, esta etapa apresenta a evolução dos componentes do *design* da interface do utilizador;
- ***Backlog:*** Lista de tarefas por ordem de prioridade. Como referido anteriormente é nesta etapa que são criados os cartões (tarefas);
- ***To-do:*** Lista das tarefas pendentes para a próxima *sprint*;
- ***Doing:*** Lista das tarefas em desenvolvimento*;*
- ***Testing:*** Lista das tarefas terminadas, mas que ainda têm de ser testadas ou o design revisto;
- ***Done:*** Etapa final onde se encontram todos os cartões que representam as tarefas terminadas, revistas e aceites por todos os membros da equipa de desenvolvimento.

Para ajudar na organização do quadro do Trello, Figura 3.1, e na organização de todos os membros da equipa de desenvolvimento foram criadas etiquetas que associam cada cartão a uma parte do projeto. Assim como também cada cartão era associado ao/s membro/s da equipa responsável pelo seu desenvolvimento.

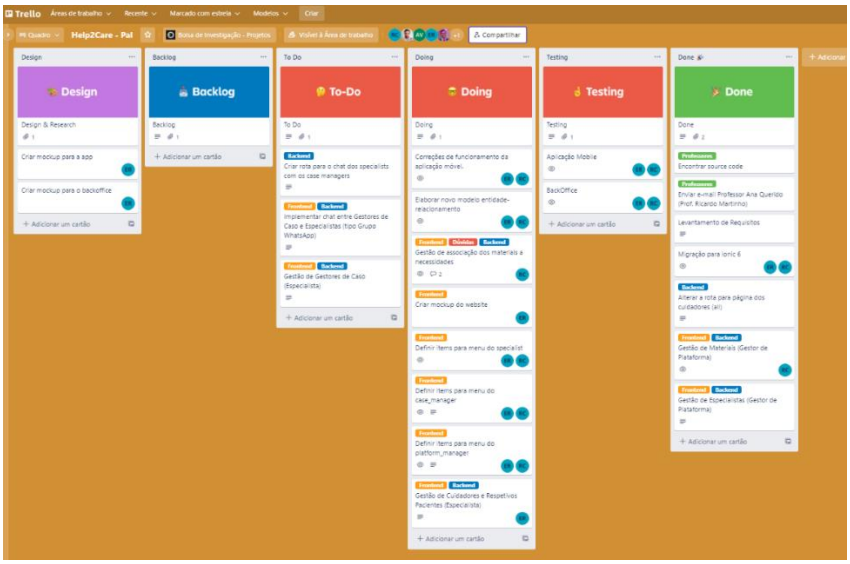

Figura 3.1 - Exemplo do quadro Trello utilizado na gestão do projeto

4. Preparação do ambiente de desenvolvimento

Como primeira fase do projeto, esta fase consistiu na atualização das versões das *frameworks* e bibliotecas do projeto Help2Care para versões mais recentes e estáveis, e assim permitir a sua execução e permitir iniciar as alterações necessárias para implementação dos requisitos para o projeto Help2Care-Pal. Nesta fase foi também configurado e preparado o ambiente de desenvolvimento local.

4.1. Atualização do projeto Help2Care

O projeto Help2Care, desenvolvido em 2019, encontrava-se desatualizado por falta de atualizações tanto no projeto em si como nas versões das *frameworks* e bibliotecas que o sustentavam. Como resultado, não era possível executar o projeto no seu estado original.

Para resolver o problema, na aplicação web (*backoffice*) procedeu-se a uma extensa atualização das versões de todas as dependências do projeto. O projeto estava inicialmente na versão 5.7 do Laravel, enquanto a versão mais recente disponível no início do desenvolvimento do projeto era a 8.7. Portanto, seguiu-se rigorosamente a documentação fornecida pelo Laravel para realizar a migração entre essas diferentes versões. Para cada atualização do Laravel, também foi necessário atualizar as versões das bibliotecas e dependências que requisitassem essa atualização. Em alguns casos, foi também necessário realizar alterações no código-fonte, pois o Laravel introduziu mudanças que não eram retro compatíveis.

Outra atualização crucial ocorreu na versão do PHP, que passou da versão 5.6 para a versão 7.3. O PHP é a linguagem principal utilizada no desenvolvimento do código do projeto. Essa atualização foi essencial, pois a versão 7.3 do PHP tornou-se o requisito mínimo obrigatório para a versão do Laravel utilizada.

Todo este processo de atualização também se estendeu às versões das bibliotecas e dependências JavaScript, que desempenham um papel importante no desenvolvimento do código-fonte para as diversas páginas *web* do projeto.

Este processo de atualização de versões foi também realizado na aplicação móvel (*frontoffice*) seguindo a mesma dinâmica utilizada para a aplicação web (versão a versão).

Neste caso, foi necessário atualizar a *framework* Ionic, que permite o desenvolvimento de aplicações móveis que podem ser *deployed* tanto em Android como iOS.

4.2. Configuração do ambiente de desenvolvimento

A partir deste momento é focada apenas a aplicação *web*, uma vez que foi esta a aplicação utilizada como caso de estudo para a implementação do pipeline CI/CD. Como a aplicação *web* do projeto Help2Care já fazia uso do Laravel, manteve-se a mesma *framework* para o desenvolvimento do Help2Care-Pal. Com a atualização das versões foi possível fazer-se uso de novas ferramentas disponibilizadas pelo Laravel, como o Laravel Sail.

A ferramenta Laravel Sail permite simplificar a configuração e manutenção do ambiente de desenvolvimento fazendo uso de contentores Docker. Os contentores Docker consistem num pacote de software que inclui tudo o que é necessário para executar uma aplicação: código, dependências e configurações. Estes contentores permitem executar uma aplicação em qualquer máquina desde que tenha o Docker instalado.

O Docker é uma plataforma, e um conjunto de ferramentas, que permitem criar, executar e gerir aplicações em contentores. Estes contentores são leves, independentes e executáveis, tendo incluído tudo o que é necessário para executar um software ou parte dele, desde código, bibliotecas ou ferramentas de sistema. O Docker permite assim que o software dentro de um contentor seja executado em diferentes sistemas operativos mantendo assim o ambiente de execução dentro do contentor. Tendo como exemplo da sua arquitetura a Figura 4.1.

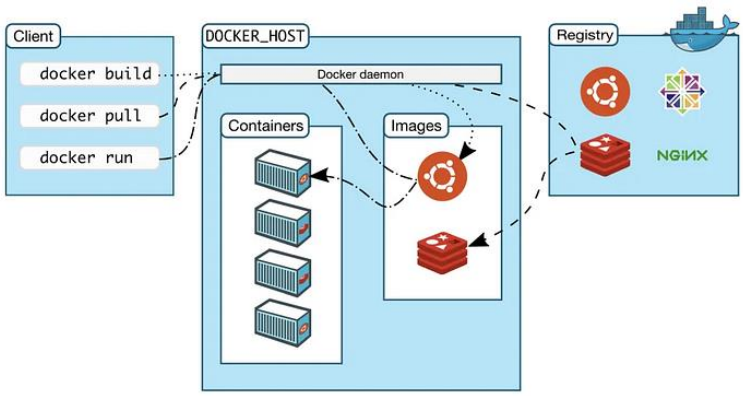

Figura 4.1 - Arquitetura Docker (fonte: Docker)

Desta forma foi possível manter-se sempre o ambiente de desenvolvimento entre as diferentes máquinas dos diferentes membros da equipa envolvida no desenvolvimento do

projeto Help2Care-Pal. Evitou-se assim problemas de compatibilidade ou de diferentes sistemas operativos nas máquinas evitando assim perdas de tempo na resolução destes problemas.

Utilizando o Laravel Sail foram criados contentores de suporte à base de dados, MySQL, e de suporte à aplicação, um servidor *web* Nginx com a linguagem PHP configurada. Foram utilizadas as configurações pré-definidas pelo Laravel Sail tanto para a base de dados como para o servidor *web*, estas disponíveis para alteração no ficheiro `docker-compose.yml`, criado na pasta do projeto. Nesta configuração estão também definidas as diferentes portas para os diferentes contentores, por exemplo, se for definida a porta 3306 para o contentor da base de dados, ao se utilizador o endereço IP local, o acesso à base de dados é feito através do endereço 127.0.0.1:3306.

5. Arquitetura de um *pipeline* CI/CD e Ações

Neste capítulo é apresentada a arquitetura do *pipeline* CI/CD criado no âmbito do projeto desenvolvido e as diferentes ações realizadas ao longo do seu processo de desenvolvimento, fazendo uso do *pipeline* CI/CD.

5.1. Arquitetura

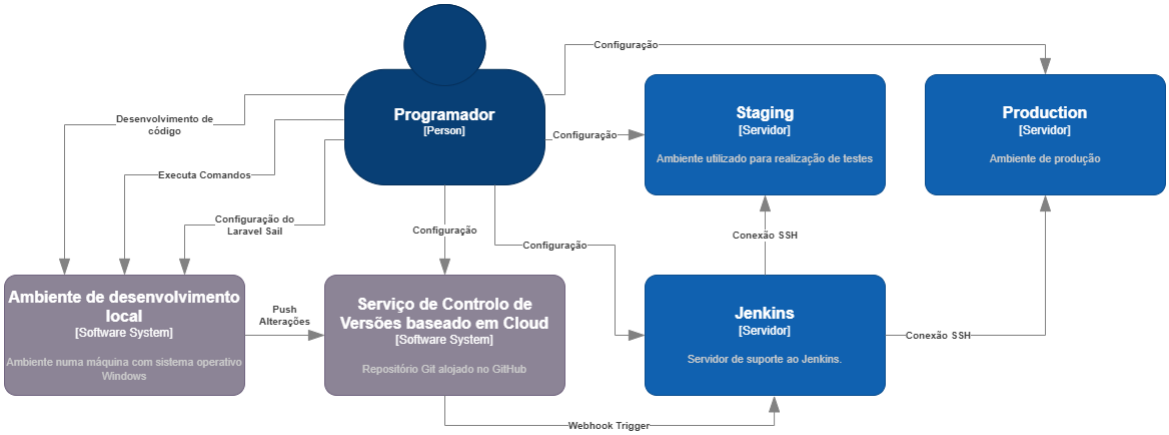

Figura 5.1 – Diagrama do contexto do pipeline CI/CD do caso de estudo Help2Care-Pal

O projeto visa a criação de um *pipeline* CI/CD e a sua utilização para desenvolvimento e manutenção de aplicações. Para atingir esse objetivo a arquitetura do projeto está dividida em duas partes. A primeira parte é denominada por ambiente de desenvolvimento local e a sua conexão com o serviço de controlo de versões. A segunda parte da arquitetura é composta pelos restantes sistemas presentes no diagrama da Figura 5.1: conexão com o servidor de controlo de versões, Jenkins, Staging e Production. Esta parte do *pipeline* CI/CD é denominada por ambiente de teste e *deployment*.

Seguindo o diagrama da Figura 5.1, tanto o editor de código como o ambiente de desenvolvimento local são considerados sistemas externos uma vez que estes não estão diretamente ligados ao funcionamento do *pipeline* CI/CD. Este, apenas é acionado quando é realizado um *push* para o repositório presente no servidor de controlo de versões, neste caso um repositório Git alojado no GitHub.

1. Ambiente de desenvolvimento local

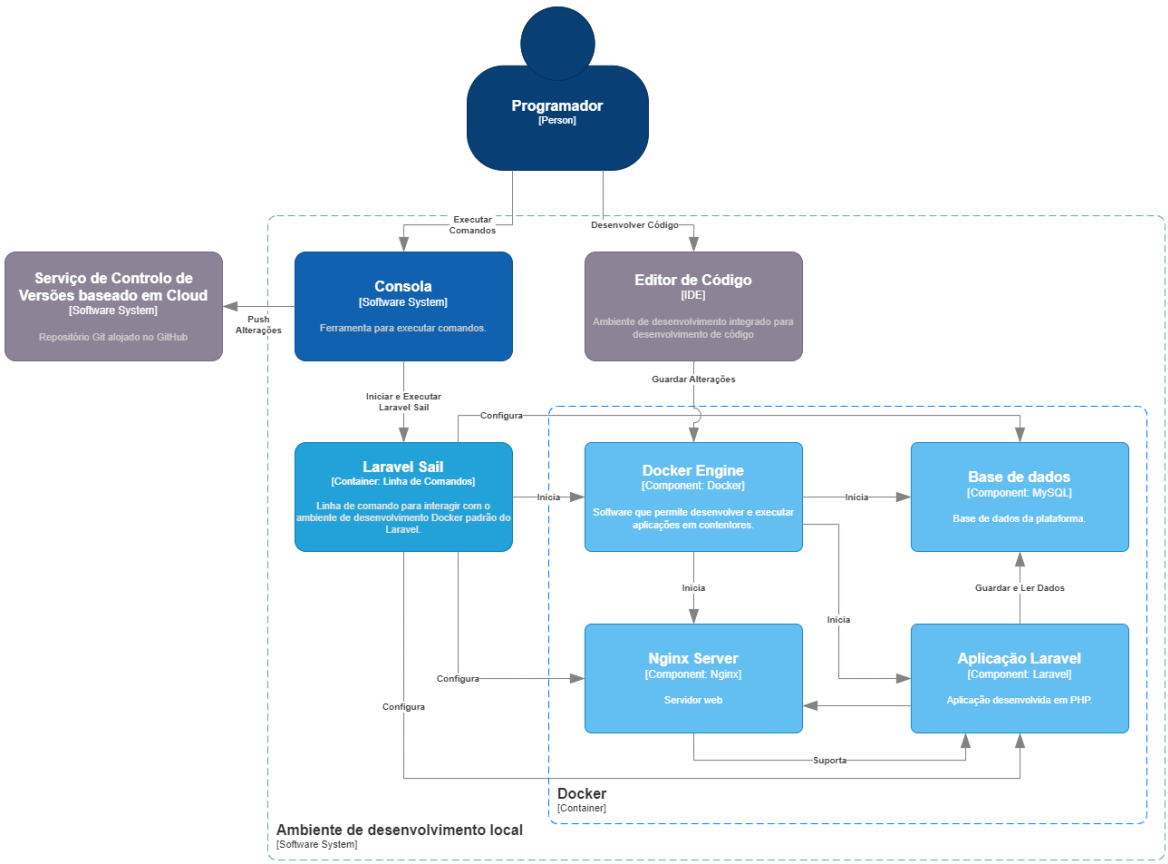

Figura 5.2 - Diagrama referente ao ambiente de desenvolvimento local

O ambiente de desenvolvimento local, conforme ilustrado na Figura 5.2, engloba a consola, editor de código, Laravel Sail e o Docker. Para isso, é essencial que os arquivos do projeto estejam armazenados numa pasta, obtidos previamente a partir do repositório do GitHub.

No caso do projeto Help2Care-Pal, fazendo uso do Laravel Sail e do Docker, foram criados diferentes contentores, um para suportar a base de dados em MySQL, um para suportar a aplicação Laravel e um para o servidor *web* Nginx com a linguagem PHP configurada. A configuração do Docker permite que estes contentores comuniquem entre si, sendo esta conexão configurada automaticamente pelo Laravel Sail, mas que pode ser alterado, no ficheiro `docker-compose.yml`, presente na diretoria do projeto.

No caso de máquinas com o sistema operativo Windows instalado é necessário também ter instalado e configurado o subsistema Linux do Windows (WSL) para utilizar os comandos para iniciar e executar o Laravel Sail.

É possível aceder aos contentores criados no Docker através do endereço de IP local (127.0.0.1) diferenciando apenas as portas definidas para cada contentor. No caso do contentor MySQL, este está associado à porta 3306 e o contentor Nginx está associado à porta 80, sendo estes acessíveis respetivamente através dos endereços, 127.0.0.1:3306 e 127.0.0.1:80. A Figura 5.3 ilustra a arquitetura dos contentores criados através da utilização do Laravel Sail e da sua configuração.

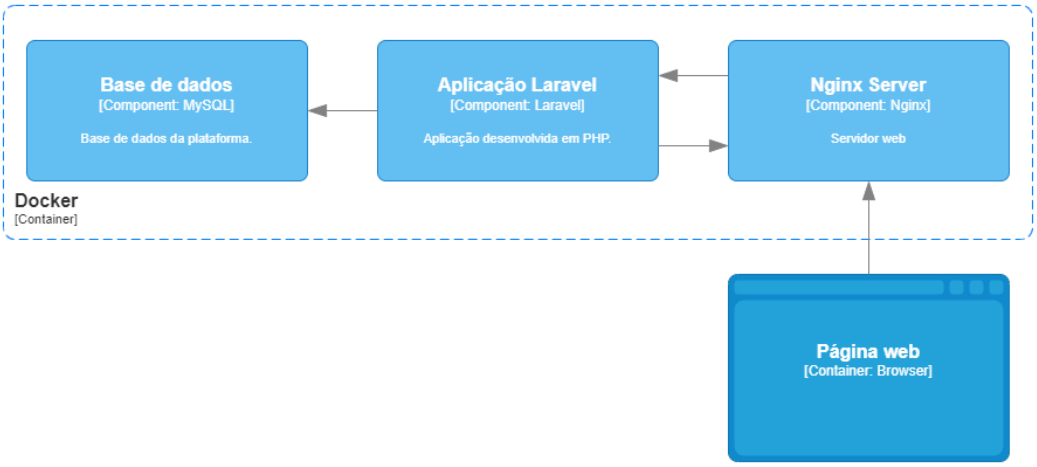

Figura 5.3 - Arquitetura dos contentores no Docker

2. Teste e *deployment*

Para suportar a segunda parte da arquitetura do *pipeline* CI/CD fez-se uso da Google Cloud Platform (GCP). Esta é uma plataforma de computação em nuvem oferecida pela Google, que fornece uma ampla gama de serviços e recursos para suportar o desenvolvimento e *deployment* de projetos de Tecnologias de Informação (TI). No caso deste projeto fez-se uso do serviço Google Compute Engine (GCE) que de forma resumida permite criar e gerir máquinas virtuais (VMs).

Seguindo o diagrama presente na Figura 5.1 para configurar os sistemas Jenkins, Staging e Production utilizou-se as ferramentas disponibilizadas pelo Google Compute Engine (GCE), configurando-se três VMs. Numa das VMs utilizadas foi configurado o Jenkins e uma ferramenta de automação de código aberto amplamente utilizada no desenvolvimento de software. Esta ferramenta oferece recursos para a criação, integração e entrega contínua de projetos, permitindo que as equipas de desenvolvimento automatizem tarefas repetitivas e acelerem o ciclo de desenvolvimento. No caso deste projeto, o Jenkins é configurado para monitorizar repositórios de código-fonte, como o GitHub que foi utilizado neste projeto.

As outras duas VMs foram configuradas para suportar o *deployment* do projeto, uma para realização de testes (Staging) e outra para disponibilização aos utilizadores (Production). A VM onde o Jenkins se encontra instalado está ligada com as outras duas através de chaves SSH. Esta ligação entre as máquinas virtuais permite ao pipeline CI/CD transferir os ficheiros, obtidos do repositório, para a máquina virtual de testes e para a máquina virtual usada pelos utilizadores. Ambas as VMs onde é feito o *deploy* da plataforma Help2Care-Pal, como o ambiente de desenvolvimento local estão configurados utilizando a ferramenta Laravel Sail.

A Figura 5.4 tem representado em maior detalhe a segunda parte da arquitetura do *pipeline*. Neste caso é apenas detalhada a arquitetura da Production, uma vez que a Staging e a Production têm a mesma arquitetura e configuração, assim como, mencionado anteriormente, igual ao ambiente de desenvolvimento local (Figura 5.2).

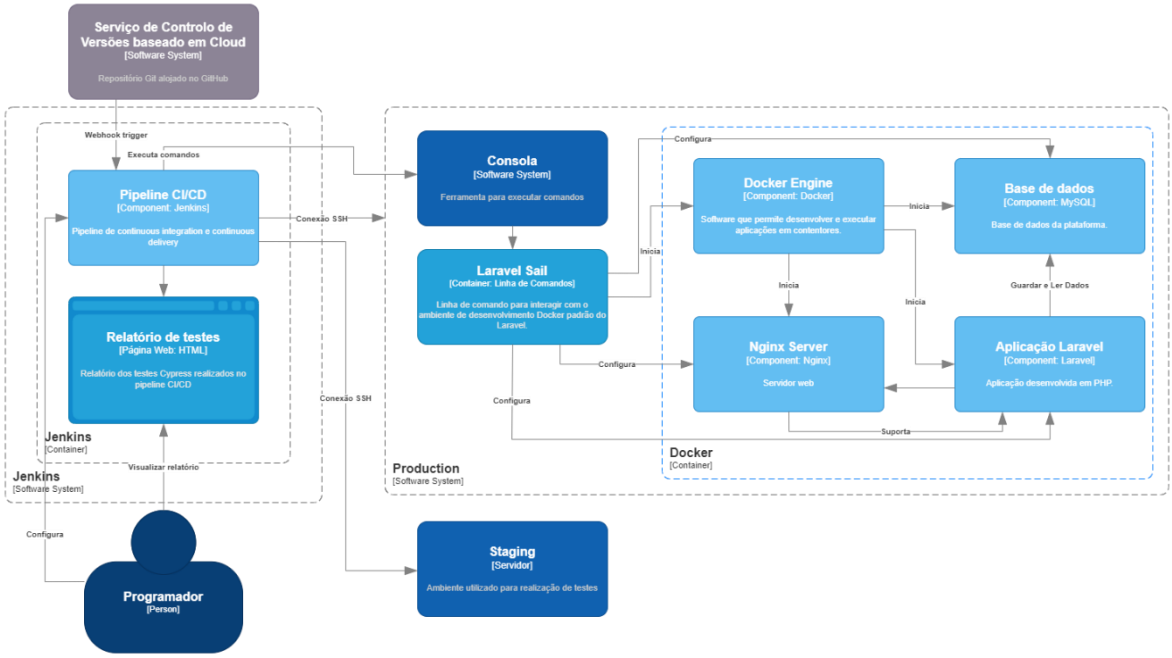

Figura 5.4 - Diagrama referente ao ambiente de teste e *deployment*

O *pipeline* CI/CD tem definida também uma fase de testes automatizados. No âmbito do projeto recorreu-se à *framework* Cypress. O Cypress é uma *framework* de teste de *frontend* de código aberto, projetado para facilitar a escrita e execução de testes automatizados para aplicações *web*. Uma das características do Cypress é a sua arquitetura *all-in-one*, que combina os aspetos de teste e execução num único ambiente.

5.2. Ações

A Figura 5.5 permite observar as ações realizadas, durante o processo de desenvolvimento do projeto Help2Care-Pal fazendo-se uso do pipeline CI/CD, de uma forma resumida e esquematizada.

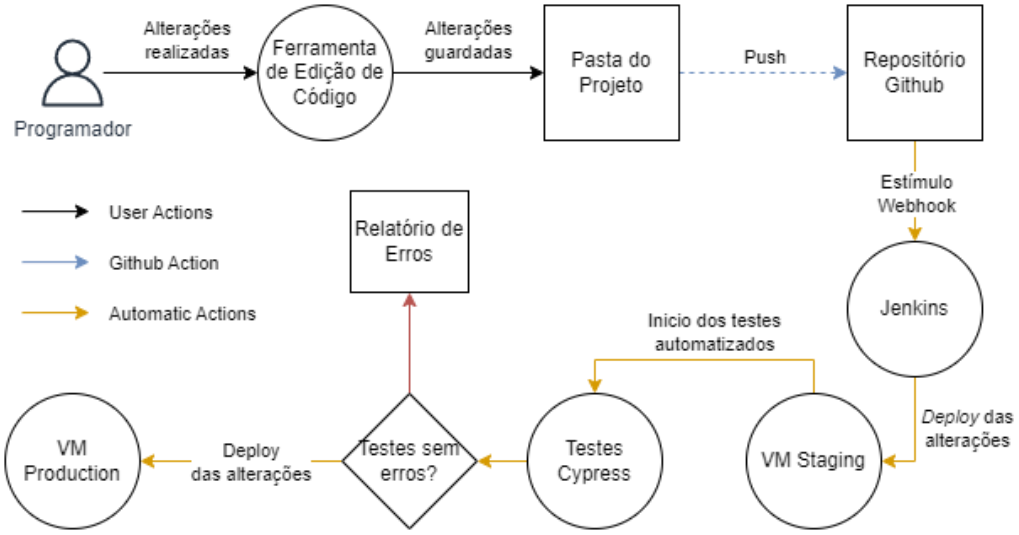

Figura 5.5 - Ações realizadas durante o processo de desenvolvimento

Antes de iniciar o processo do *pipeline* e para o ativar, o programador precisa executar algumas ações. O desenvolvimento de um projeto é realizado da mesma forma, independentemente da utilização ou não de um *pipeline* CI/CD. Portanto, as primeiras etapas desse processo envolvem a modificação no código-fonte através de uma ferramenta de programação ou edição de código.

Após a realização de alterações no código, a próxima ação é, novamente, um passo comum no desenvolvimento de projetos que fazem uso de repositórios de controlo de versões centralizado. Isso implica efetuar um *commit* das alterações e, em seguida, enviá-las (*push*) para o repositório. É neste momento que o *pipeline* CI/CD é acionado.

Este acionar do *pipeline* é possível graças à configuração de um *webhook* (mecanismo que permite a comunicação automática entre dois sistemas ou aplicações). Neste caso o *webhook* funciona como um estímulo (conhecido como *trigger*) e permite que o *pipeline* tenha o conhecimento que foram realizadas alterações no repositório.

Com a notificação enviada pelo *webhook*, o *pipeline* inicia o seu processo, descarrega os ficheiros do mesmo repositório centralizado, cria ou atualiza os ficheiros na VM Staging, executa os testes automatizados, usando a VM Staging como ambiente de testes.

Caso todos os testes realizados sejam bem-sucedidos, as novas atualizações são enviadas para a VM onde o projeto está em produção, realizando também os comandos necessários para o contínuo funcionamento do software sem ser necessária a intervenção de um administrador do sistema.

Caso existam erros durante os testes, é criado um relatório onde é possível ao programador perceber o que falhou. Assim é possível realizar as alterações necessárias para o bom funcionamento da atualização e iniciar o processo do *pipeline* de novo.

6. Implementação de um *pipeline* de CI/CD

Neste capítulo é descrita a implementação do *pipeline* de *continous integration* e *continuous delivery* (*pipeline* CI/CD), onde são apresentadas as configurações do *pipeline* e as configurações e implementação dos testes em Cypress realizados no mesmo. Seguidamente são apresentadas as *builds*[10] do *pipeline* e o que é feito caso existam falhas. Por fim são apresentados também os resultados e relatórios obtidos e as ações necessárias para manutenção do *pipeline*.

6.1. Configuração do *pipeline* CI/CD

Para a implementação do *pipeline* CI/CD fez-se uso da Google Cloud Platform (GCP) para hospedar três máquinas virtuais (VMs). Uma das máquinas irá hospedar o Jenkins, e cada uma das restas irá ser configurada para ambas terem o mesmo ambiente e hospedar o software. Uma destas duas máquinas (Staging) será utilizada para realizar os testes às novas funcionalidades enquanto a outra máquina (Production) irá hospedar o software que será utilizado para que os utilizadores finais do projeto possam aceder à aplicação através da página de *web*.

A máquina virtual para hospedar o Jenkins foi criada através de uma máquina virtual *ready-to-deploy* existente no *Marketplace* (Mercado) da GCP, sendo apenas necessário configurar o Jenkins após a VM estar em funcionamento.

A configuração do Jenkins implicou a instalação dos seguintes *plugins*: Git, GitHub, Pipeline, Pipeline: Stage View, NodeJS, e Publish over SSH. Outros *plugins* podem ser instalados conforme as necessidades e objetivos do *pipeline* em questão.

O próximo passo foi a instalação do NodeJS nas "Ferramentas de Configuração Global" na seção "Gestão Jenkins", esta instalação é necessária para que seja possível realizar os testes com o Cypress.

Em seguida, as VMs Staging e Production foram configuradas. Esta configuração é a mesma para ambas as máquinas, pois ambos os ambientes devem ser iguais para permitir que os testes sejam realizados num ambiente igual ou semelhante aquele em que o software está em funcionamento quando o mesmo se encontrar em produção e disponível aos seus utilizadores

[10] Nome dado a cada iteração do *pipeline*

finais. Além disso, as VMs devem ser configuradas de acordo com a *golden image* existente no ambiente de desenvolvimento, daí ser utilizado o Docker para a criação de contentores que permitem esta similaridade entre os diferentes ambientes e a sua portabilidade pelas diferentes máquinas utilizadas pelos intervenientes no desenvolvimento do software.

Depois de ambas as máquinas estarem configuradas e ativas, o Jenkins foi ligado a estas através de chaves *Secure Shell* (SSH), o que permitiu executar comandos e transferir arquivos automaticamente para ambas as máquinas virtuais. Para que esta ligação fosse possível, foi necessário criar um par de chaves pública/privada para a VM do Jenkins. A chave pública foi então inserida nas chaves autorizadas em cada uma das outras VMs.

Em seguida, foi necessário voltar à configuração de cada uma das VMs (Staging e Production) e copiar a chave privada de cada uma. Esta chave foi colocada na "Configuração de Sistema" do Jenkins, dentro do separador "Publicação por SSH". Um servidor SSH foi então criado com as informações de cada máquina virtual, estando assim concluída a ligação entre a VM do Jenkins e as duas VMs para hospedar o software.

A automatização do *pipeline* foi a última fase da configuração do Jenkins. Nesta fase foi necessário fazer a conexão com o repositório do projeto que é gerido pelo *pipeline* assim como definir os passos que devem ser seguidos após ser acionado. Primeiro foi necessário criar um *webhook* no repositório onde está guardado o projeto. De seguida, no Jenkins adicionou-se um novo item do tipo *pipeline* definido como "*GitHub project*" e onde foi colocado o URL para o repositório.

O passo seguinte permitiu que o *pipeline* se inicie automaticamente quando recebe um pedido através do *webhook* com a ativação dos *triggers*, selecionando a opção "*GitHub hook trigger for GITScm polling*". Este *trigger* é acionado sempre que é realizado um *"push"* para o repositório centralizado, ou seja, quando um dos programadores fez alterações no código do projeto no seu ambiente local de desenvolvimento e enviou essas alterações para o repositório. Existindo esta alteração do código, é assim necessário atualizar também os mesmos ficheiros, mas nas VMs e consequentemente testar as alterações realizadas.

Sem o *pipeline* e o seu automatismo todo este processo teria de ser realizado manualmente, através da linha de comandos de cada uma das máquinas virtuais. A Figura 6.1 permite de forma simples e sintética perceber o fluxo desde o programador até ao início do funcionamento do *pipeline* CI/CD, com os diferentes "atores" no processo.

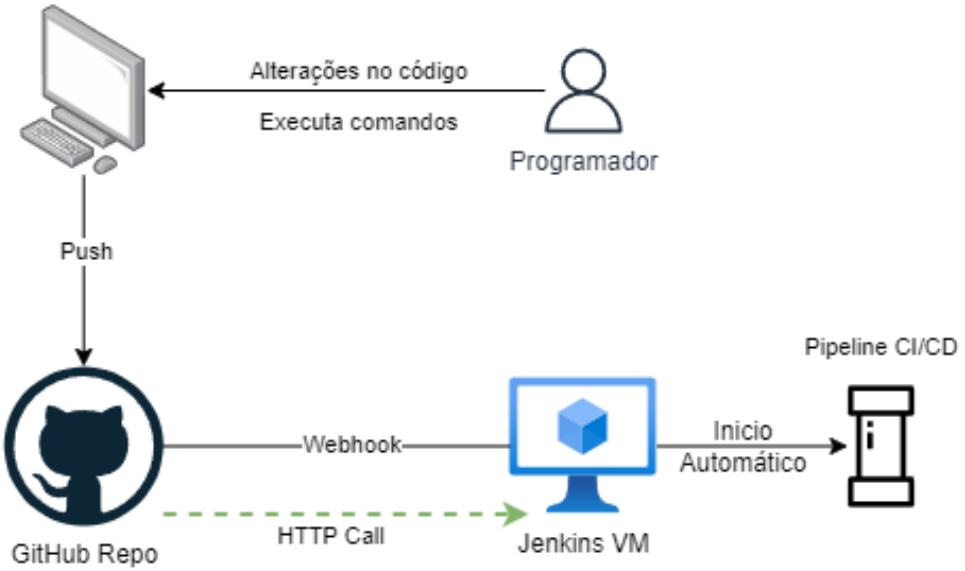

Figura 6.1 - Fluxo desde o programador até iniciar o *pipeline*

Por fim, é necessário indicar ao Jenkins quais os passos que este deve seguir no pipeline construído. Pode ser feito de duas formas, através da secção "Pipeline Syntax" onde são definidos os passos a seguir, como se pode observar o exemplo da Listagem 1. Como alternativa, esses passos podem ser definidos num ficheiro com o mesmo formato, por padrão chamado "Jenkinsfile" e presente na raiz do projeto.

```
pipeline{
  agent any
  stages {
    stage('Build/Deploy app to staging') {
      steps {
        echo 'Building/Deploying app to staging'
      }
    }
    stage('Run automated tests') {
      steps {
        echo 'Running automated tests'
      }
    }
    stage('Perform manual testing') {
      steps {
        echo 'Performing manual testing '
      }
    }
    stage('Release to production') {
      steps {
        echo 'Releasing to production'
      }
    }
  }
}
```

Listagem 1 - Exemplo da formatação do código do Pipeline Syntax

6.2. Configuração do Cypress

Para o *pipeline* conseguir realizar os testes ao software é necessário criar uma etapa (*stage*) onde estão especificados os comandos necessários para configurar o Cypress e iniciar o seu processo. Estes comandos são considerados os passos (*steps*) da etapa em questão. Para a sua utilização é necessário ter instalado o NPM[11], que permite gerir os pacotes e dependências necessárias para o funcionamento do Cypress.

Como verificado na Listagem 1, os diferentes passos a seguir numa etapa são colocados dentro da função `step`. No caso do *pipeline* CI/CD implementado foram definidos 9 passos distintos para a realização e automatização dos testes, estando definidos na Listagem 2.

[11] Gestor de pacotes para o Node.JS

```
stage('Run automated tests'){
  steps {
    echo 'Running automated tests'
    sh 'npm prune'
    sh 'npm cache clean --force'
    sh 'npm i'
    sh  'npm  install  --save-dev  mochawesome  mochawesome-merge
                mochawesome-report-generator'
    sh 'rm -f mochawesome.json'
    sh 'npx cypress run --config baseUrl="url" --record --key=”key”
                --reporter mochawesome --headless'
    sh   'npx   mochawesome-merge   cypress/reports/mochawesome-
                reports/*.json > cypress/reports/output.json'
    sh  'npx  marge  cypress/reports/output.json  --reportDir  ./  --
                inline'
    sh 'rm -r cypress/reports/mochawesome-reports/*.json'
  }
}
```

Listagem 2 - Código exemplo com comandos de configuração do Cypress

O primeiro passo, `npm prune`, é realizado para remover pacotes de bibliotecas ou dependências que não estejam a ser utilizadas, ou seja não estão definidas no ficheiro `package.json`. Desta forma é possível libertar espaço que esteja a ser utilizado desnecessariamente, melhorar o desempenho e evitar conflitos entre os diferentes pacotes utilizados.

```
npm cache clean --force
```

Listagem 3 - Linha de código para limpar a cache

O comando seguinte, Listagem 3, permite limpar a cache usada para guardar os pacotes descarregados anteriormente, e permite como o comando anterior libertar espaço no disco, resolver erros de instalação caso existam e melhor também o desempenho do NPM. Estes dois primeiros passos são opcionais, mas recomendados prevenindo assim possíveis erros de conflito entre as diferentes iterações do *pipeline*.

```
npm install --save-dev mochawesome mochawesome-merge mochawesome-
                report-generator
```

Listagem 4 - Linha de código para instalação de dependências especificas

Seguidamente, procede-se à instalação dos pacotes referenciados no ficheiro `package.json` com o comando, `npm install`, sendo este seguido do comando, Listagem 4,para instalar as dependências Mochawesome, Mochawesome Merge e Mochawesome

Report Generator. A *flag*[12] `--save-dev` indica que estas dependências são apenas instaladas em contexto de desenvolvimento e não em contexto de produção. Estas dependências permitem a criação de relatórios dos testes de fácil leitura, ou seja, não é um passo obrigatório, mas apenas feito para proporcionar uma melhor experiência a quem lê os relatórios e possibilitar uma rápida leitura do mesmo.

O quinto passo, `rm -f {ficheiro}`, permite remover um ficheiro, no caso da implementação efetuada no projeto é removido o ficheiro `mochawesome.json` para evitar conflitos com a criação do mesmo ficheiro no fim de cada iteração do pipeline. Uma vez que este é o ficheiro em que está presente o relatório dos testes, portanto é imperativo que o mesmo seja corretamente criado.

```
npx cypress run {flags}
```

Listagem 5 - Linha de código para iniciar o Cypress

O sexto passo, consiste na iniciação do Cypress e o início dos testes, Listagem 5, com este comando podem ser inseridas diferentes *flags* para configurar o seu funcionamento, neste caso são utilizadas as *flags*:

1. `config`: Especifica o ficheiro de configuração, e neste caso permiti definir o URL onde são realizados os testes, `--config baseUrl="url"`,
2. `record`: Indica ao Cypress que os resultados dos testes devem ser guardados, neste caso no ficheiro `cypress/results.json`,
3. `key`: Especifica a chave API para o Mochawesome, que é usada para gerar o relatório,
4. `reporter`: Define qual a biblioteca a ser utilizada para a construção dos relatórios, neste caso o Mochawesome, `--reporter mochawesome`,
5. `headless`: Indica ao Cypress que os testes devem ser realizados sem abrir o *browser*, e
6. `browser`: Indica qual o browser que deve ser utilizado para correr os testes.

Seguidamente, Listagem 6, no sétimo e penúltimo passo, é realizada a junção de todos os relatórios em apenas um relatório, mantendo a extensão de ficheiro, sendo possível realizar esta ação com o comando. Neste caso é necessário indicar todos os ficheiros que se

[12] *Flag* – opção do comando que altera o comportamento do mesmo

pretendem fundir inserindo a sua localização onde estes são guardados na máquina, `cypress/reports/mochawesome-reports/*.json`, ou seja, todos os ficheiros JSON presentes na pasta `mochawesome-reports` que se encontra dentro da pasta `reports`. Esta junção será guardada no ficheiro indicado à direita do `>`, ou seja, no ficheiro `output.json` na pasta `reports`.

```
npx mochawesome-merge {ficheiros} > {ficheiro}
```

Listagem 6 - Linha de código para realizar a junção de relatórios

O último comando, Listagem 7, irá transformar o ficheiro JSON gerado no comando anterior em um ficheiro `html`, isto para melhor a visualização dos relatórios gerados a partir dos resultados dos testes. A *flag* `--reportDir` é utilizada para especificar a paste destino do ficheiro html, caso não seja especificada será criado na pasta atual. A *flag* `--inline` indica se os recursos são escritos diretamente no ficheiro ou inseridos como ligações externas.

```
npx marge cypress/reports/output.json --reportDir ./ --inline
```

Listagem 7 - Linha de comando para transformar um ficheiro JSON em HTML

6.3.Implementação dos testes automatizados

Como mencionado anteriormente, foi utilizada e configurada a ferramenta Cypress como ferramenta para a realização dos testes automatizados à plataforma Help2Care-Pal. Como parte do *pipeline* CI/CD, os diferentes ficheiros com o código fonte dos testes colocaram-se no mesmo repositório que o código fonte da plataforma digital do caso de estudo. Neste caso foi criada uma diretoria onde foram colocados todos os ficheiros necessários para o bom funcionamento dos testes implementados. Esta pasta deve ser indicada no ficheiro Jenkinsfile como uma das pastas a excluir quando transferidos os ficheiros para as máquinas virtuais dos ambientes de *staging* e de produção, uma vez que os mesmos não vão ser utilizados nestes ambientes. Na Figura 6.2 mostra-se a estrutura de pastas geral de um projeto de testes utilizando a ferramenta Cypress.

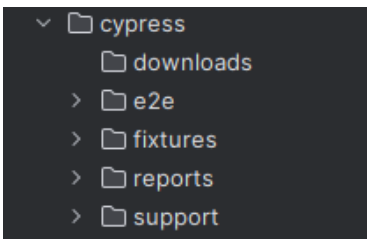

Figura 6.2 - Esquema de pastas do Cypress

Para a realização dos testes foram criados utilizadores de teste, cada um deles com um papel diferente na plataforma (administrador, especialista, gestor de caso), para assim ser possível testar todas as funcionalidades. Para se abranger o maior número de funcionalidades possíveis com o mesmo utilizador, os primeiros testes desenvolvidos fizeram uso do utilizador com a função de administrador, uma vez que este tem acesso a todas as funcionalidades existentes na plataforma. Assim é possível reutilizar os testes criados alterando apenas o utilizador que está autenticado na plataforma. Cada tipo de utilizador tem as suas funções, e com isso acesso restrito dentro da plataforma, excetuando como referido o administrador.

A plataforma do caso de estudo é também constituída por páginas informativas sobre o projeto assim como uma página inicial que podem ser acedidas por qualquer tipo de utilizador sem ser necessário fazer autenticação. Uma vez que o caso de estudo foi implementado a partir do projeto já existente, mantendo a maioria das funcionalidades e sendo desenvolvidas novas, implementou-se numa primeira fase testes às funcionalidades existentes, para assim haver a certeza que continuavam a funcionar de forma correta e para se realizarem as alterações a partir de uma versão funcional. Assim foi possível desenvolver-se tanto a plataforma como os respetivos testes ao mesmo tempo e dessa forma ter sempre uma versão funcional.

A. Página inicial e páginas informativas

Com o objetivo de tornar os testes mais abrangentes e adaptáveis para a possibilidade de inserção ou remoção futura de páginas, bem como para a navegação por meio de um menu localizado na parte superior das páginas, foi criado um ficheiro JSON. Este arquivo contém informações essenciais sobre cada página, que são utilizadas nos testes, como o exemplo representado na Listagem 8.

```
[
  {
    "name": "Contactos",
    "suffix": "/contactos"
  }
]
```

Listagem 8 - Exemplo da informação de cada página

Para permitir também a reutilização de código foi criado o ficheiro `WelcomeUtils.js` onde estão desenvolvidas as diferentes ações de teste efetuadas nestas páginas e que são comuns em todas as páginas.

```
Class WelcomeUtils {
  clickToolbarButton(name){
    cy.get('[data-cy="toolbar"]')
          .contains(name)
          .click();
  }
  checkUrl(suffix){
    cy.url().should('include',suffix);
  }
  notEmpty(){
    cy.get('[data-cy="content"]').should('not.be.empty');
  }
}
```

Listagem 9 – Código do ficheiro WelcomeUtils.js

Estas funções permitem (seguindo a Listagem 9 de cima para baixo): 1) testar se existe um botão na barra superior da página como o texto pretendido (`name`) e clicar para passar à página pretendida, e 2) verificar se o URL da página atual (`suffix`) é o pretendido e uma função para verificar se a página não se encontra vazia. A junção destes dois ficheiros permite criar assim um teste dinâmico em que caso seja necessário alterar algo, não é preciso fazê-lo em diversos pontos do código, mas sim apenas em um local. Para utilizar os dados do ficheiro JSON e as funções do ficheiro `WelcomeUtils.js` estes têm de ser importados no ficheiro de testes pretendido. Neste caso a Listagem 10 apresenta o código dos testes tanto da página inicial como das páginas informativas. Uma vez que são utilizadas funções de um ficheiro à parte, assim como os dados do ficheiro JSON, é necessário fazer a importação de ambos os ficheiros para ser possível utilizar. As importações são realizadas de forma diferente, os dados do ficheiro JSON são importados através da criação de uma variável, como apresentado na segunda linha da Listagem 10.

```
import WelcomeUtils from "../auxiliars/WelcomeUtils";

const pages = require('../../fixtures/welcome_pages.json')

describe('Access Front Pages', function () {

  const welcomeUtils = new WelcomeUtils();

  pages.forEach((page) => {
    it('successful_access_'+page.name, function () {
      cy.visit('/');
      welcomeUtils.clickToolbarButton(page.name);
      welcomeUtils.checkUrl(page.suffix);
      welcomeUtils.notEmpty();
    })
  })
})
```

Listagem 10 - Código do ficheiro de testes às páginas informativas

Desta forma são percorridos os diferentes objetos presentes no ficheiro JSON, ou seja, a informação de cada página, e os testes necessários iniciando o teste de cada página sempre na página inicial do *website*, clicando no botão correspondente no topo da página e após aceder a essa página, verificando se está no URL correspondente e se a página não está vazia.

B. Organização dos testes para utilizadores autenticados

Para a reutilização de código foram criados dois ficheiros onde estão desenvolvidas as funções com o código para os testes que são necessários em diferentes partes do *website*, sendo estes o ficheiro `Utils.js` e o ficheiro `FillFormsUtils.js`.

O ficheiro `Utils.js` é destinado a suportar todas os testes comuns a todas as páginas como, carregar num botão, configurar o texto de um título de uma página, clicar num botão de um menu, procurar um elemento de uma tabela, entre outras ações.

O ficheiro `FillFormsUtils.js` foi destinado a testar ações que ocorrem apenas em páginas com formulários, como os diferentes campos existentes num formulário, mas que podem existir em formulários diferentes, como por exemplo: inserir texto, inserir ficheiros ou escolher opções em *dropdowns*[13]. Desta forma foi possível eliminar código repetido e

[13] https://semantic-ui.com/modules/dropdown.html

também facilitar a leitura dos diferentes testes e do que estes testavam em cada uma das situações.

Estes ficheiros, assim como outros criado de forma a evitar a repetição de código e melhorar a organização de todos os testes, é possível observar-se na Figura 6.3

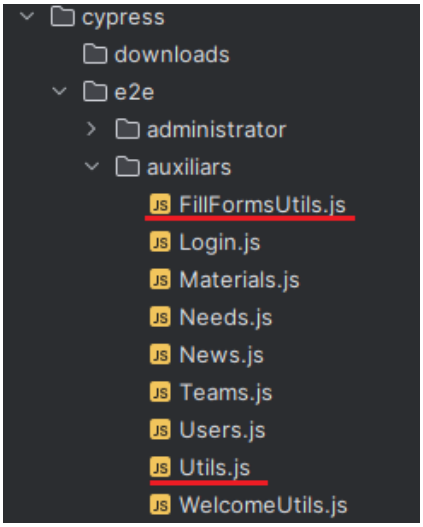

Figura 6.3 - Organização dos ficheiros auxiliares

Para se aceder e obter informações dos elementos HTML das páginas *web* o Cypress faz uso das classes definidas no atributo `class` de um elemento HTML ou do `id`, também possível de ser definido como `class`. Uma vez que estes atributos podem ser alterados em futuras atualizações do código-fonte e seguindo as estratégias de boas práticas[14] do Cypress foi adicionado, nos elementos do código da plataforma Help2Care-Pal, um atributo, `data-cy`, para que, caso existam alterações nos atributos `class` ou `id`, ser sempre possível ao Cypress encontrar os elementos nas diferentes páginas sem ser necessário alterar o código dos testes. Estes diferentes atributos do elemento HTML estão representados na Listagem 11.

```
<h5 class="card-title" style="margin:auto!important;" data-
                cy="title">Materiais de Capacitação</h5>
```

Listagem 11 - Exemplo da utilização do atributo data-cy

Foram também criadas diferentes pastas para cada tipo de utilizador de forma a organizar os testes consoante o seu papel na plataforma. Estes ficheiros de testes foram também agrupados em pastas consoante a área da plataforma a ser testada. Por exemplo, todos os testes relacionados com notícias em que o utilizador autenticado é o administrador estão na

[14] https://docs.cypress.io/guides/references/best-practices

pasta `noticias` dentro da pasta `administrador`. Desta forma os testes são fáceis de encontrar quando existem problemas ou erros e seja necessário fazer alguma alteração. Seguindo esta organização também os ficheiros com testes têm nomes indicativos da ou das funcionalidades que são testadas.

C. Organização do código no ficheiro de testes

Um ficheiro de testes é normalmente constituído por importações de outros ficheiros e/ou pacotes necessários para o bom funcionamento dos testes, seguido de uma função onde são escritos os testes a realizar. Esta função, `describe`, é constituída por dois argumentos: 1) o nome do conjunto de testes; e 2) uma função de retorno (*callback function*) onde estão implementados os diferentes testes.

```
describe('My First Test', () => {
  it('Does not do much!', () => {
    expect(true).to.equal(true)
  })
})
```

Listagem 12 - Código exemplo de testes em Cypress

A função de retorno da função `describe` pode conter diferentes tipos de funções. Para definir os diferentes testes ou casos de teste (*test case*) é utilizada a função `it`, como é possível observar no código exemplo da Listagem 12. A função `it` segue a mesma estrutura da função anterior, ou seja, é constituída por dois argumentos: 1) o nome do caso, e 2) a função de retorno. É aqui que os diferentes passos do teste são definidos, correspondendo assim às funcionalidades a serem testadas.

São também usadas duas funções que permitem realizar passos antes do início dos casos de teste, ou antes do início de cada um dos casos (funções `before` e `beforeEach` respetivamente), exemplo na Listagem 13.

Com a função `before` é possível preparar o ambiente para os casos de teste que se vão realizar, como por exemplo criar um utilizador necessário para os testes definidos. A função `beforeEach` permite realizar ações que se repetem em todos os casos de teste existentes, desde que essas sejam realizadas antes das ações que diferem entre os testes. Assim é possível evitar a repetição de código e permite melhorar a sua leitura e o seu entendimento. Nestes casos os passos repetidos podem ser a realização da autenticação, ou a necessidade

de aceder a uma página específica, ou seja, todos esses passos comuns em todos os casos são colocados na função `beforeEach` e o Cypress realiza-os antes de cada um dos casos.

```
before(() => {
  login.visit();
  login.loginUser('ADMIN_USERNAME', 'ADMIN_PASSWORD')
  cy.url().should('contains', 'users');
  utils.clickButton('create-new')
  let data = usersFunctions.generateData();
  userName = data.name;
  data.role = 'platform_manager'
  usersFunctions.createHealthCareProUser(data);
})
beforeEach(() => {
  login.visit();
  login.loginUser('ADMIN_USERNAME', 'ADMIN_PASSWORD')
  cy.url().should('contains', 'users');
})
```

Listagem 13 - Exemplo do uso das funções before e beforeEach

D. Organização dos casos de teste

Cada conjunto de testes seguiu também uma organização de forma a percorrer todos os cenários de cada uma das funcionalidades da plataforma testada. Assim, é sempre realizado um caso de teste que percorre o caminho ideal (normalmente também denominado de *happy path*) para o correto funcionamento da funcionalidade. Após o caso de teste para o caminho ideal, são criados os casos em que o caminho não chega ao objetivo desejado, mas informa o utilizador desse problema, ou seja, os caminhos secundários como mensagem de erro pela autenticação falhar, um campo errado num formulário ou uma tabela sem dados para apresentar.

Focando nas funcionalidades da plataforma do nosso caso de estudo, existem situações em que há vários caminhos ideais, uma vez que a mesma funcionalidade permite chegar a um objetivo por diferentes caminhos. Por exemplo, uma das funcionalidades da plataforma é a criação de um cuidador (tipo de utilizador), podendo neste caso criar apenas o cuidador, criar o cuidador com paciente associado, mas sem necessidades associadas ao paciente, e/ou criar o cuidador com paciente associado com necessidades associadas. Todos estes diferentes caminhos chegam ao objetivo de criar um cuidador, e todos estes casos foram testados.

6.4. Jenkins Dashboard

O Jenkins disponibiliza um *dashboard* onde se pode aceder às informações de cada *pipeline* criado, e onde é possível ter informações rápidas sobre os diferentes *pipelines* como o estado da última *build*, há quanto tempo ocorreu com sucesso, com falha e a duração, assim como a possibilidade de iniciar uma *build* manualmente. Acedendo a um *pipeline*, são disponibilizadas diversas opções para configurar, manter e analisar o funcionamento do *pipeline*.

Na página de *Stage View* do *pipeline* é possível observarem-se as últimas *builds* e qual o seu estado, assim como o estado de cada uma das diferentes etapas do *pipeline* em cada uma das *builds (*verde – sucesso, amarelo – incompleto/pendente e vermelho – falha), como se pode observar na Figura 6.4 e na Figura 6.5.

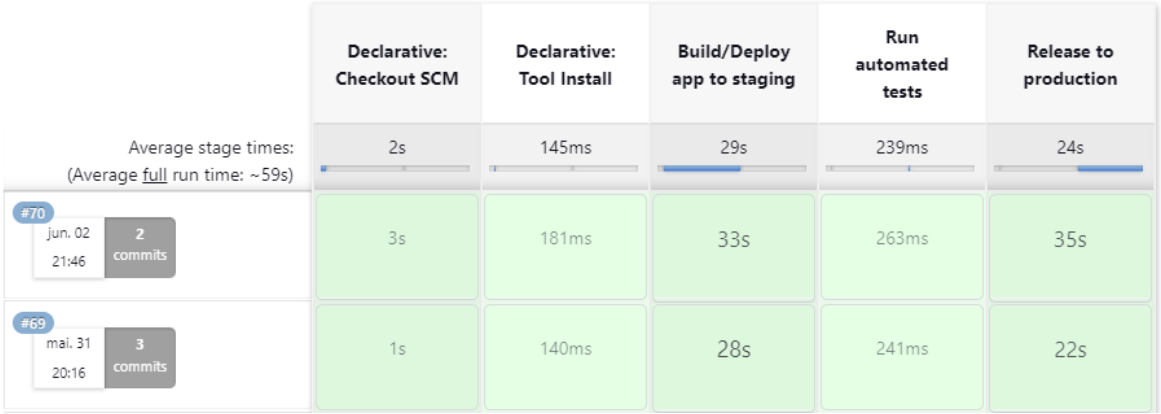

Figura 6.4 - Excerto da apresentação das *builds* na vista de Stage View do *dashboard* do *pipeline*

Outra das opções disponibilizada é um resumo das *builds* em que é dada a informação dos diferentes *commits* que foram testados em cada uma das *builds* e assim é possível confirmar que alterações puderão ou não estar a causar problemas. Acedendo às informações de uma *build* a partir do histórico (demonstrado na Figura 6.5) é possível aceder a mais opções e obter detalhes específicos da *build* selecionada.

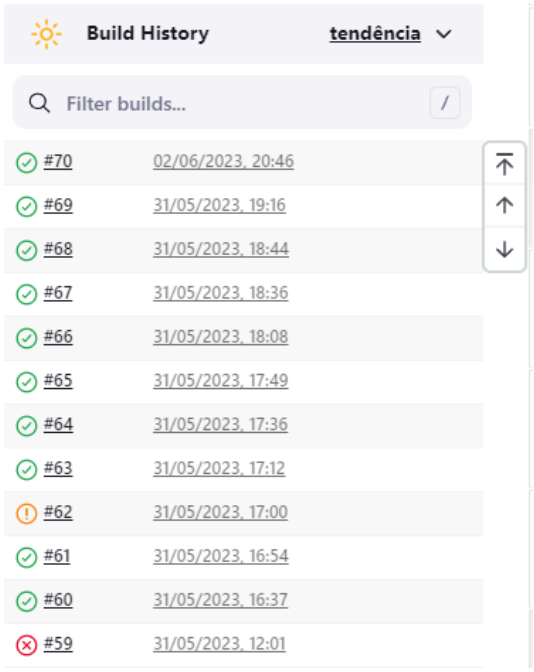

Figura 6.5 - Excerto do histórico de *builds*

Na página de detalhes de uma *build* é possível aceder à consola do *pipeline* e verificar a forma como este procedeu ao longo das diferentes etapas e que comandos foram utilizados, e também aceder aos resultados dos testes. Sendo configurada a criação de relatórios estes são também acedidos a partir desta página.

6.5. *Build* e Falhas

Para um bom funcionamento do *pipeline* foram seguidas algumas regras entre os membros da equipa responsáveis pelo desenvolvimento da plataforma do caso de estudo, não só para evitar *builds* desnecessárias do *pipeline* como também para uma boa gestão de recursos utilizados.

O *pipeline*, ao estar ligado ao repositório centralizado (Git) tem também definido que ramo do repositório deve esperar as notificações de alterações (neste caso, o ramo ‘main’). Assim, foi decidido que todas as alterações seriam realizadas em novos ramos (estratégia também denominada de *feature branching*), sendo posteriormente feito um *pull request* e assim feita a junção de código entre o ramo ‘main’ e o ramo de desenvolvimento, sendo neste momento que o *pipeline* seria ativado.

Feature branching é uma estratégia de desenvolvimento de software utilizada em sistemas de controlo de versões. Esta prática permite gerir o desenvolvimento de novas funcionalidades de um projeto através da criação de ramos (*branches*) separados para cada

nova funcionalidade ou tarefa realizada. Esta estratégia permite realizar desenvolvimento paralelo, aumentado a velocidade de desenvolvimento e a entrega de novas funcionalidades, reduzindo também o risco de conflitos entre os diferentes códigos desenvolvidos pelos diferentes membros da equipa. No caso do *pipeline* reduz o número de *builds*, que poderiam ser por vezes desnecessárias.

Outra regra seguida pela equipa foi a de, sempre que possível, deixar o *pipeline* no final do dia a verde, ou seja, ao final do dia a última *build* ser sempre uma *build* em que todas as etapas foram concluídas com sucesso. Caso não fosse possível, tentar sempre resolver os problemas encontrados pelo *pipeline* dentro de um curto intervalo de tempo.

6.6. Resultados e Relatórios de Testes

Os resultados e relatório dos testes são uma das partes essenciais do *pipeline* CI/CD e estes podem ser obtidos de várias formas. O Cypress disponibiliza, no seu próprio *site*, através da Cypress Dashboard, a possibilidade de criar um projeto, ao qual se pode ligar um projeto Cypress através de uma chave e do seu ID, e assim obter relatórios dos testes realizados, bem como diferentes estatísticas. Esta conexão é realizada através da inserção do ID no ficheiro de configuração do Cypress (`cypress.config.js`) como `projectId` e como observado na Listagem 2. Quando executado o comando `npx cypress run` é necessário adicionar as *flags* `--record` e `--key`, colocando-se nesta última a chave disponibilizada no Cypress Dashboard.

Na Figura 6.6 é possível observar-se os diferentes *commits* e os respetivos resultados dos testes em cada um deles. Assim de uma forma resumida tem-se a informação da quantidade de testes realizados com sucesso e os que falharam. É também depois possível aceder aos detalhes de cada uma das *builds* realizadas.

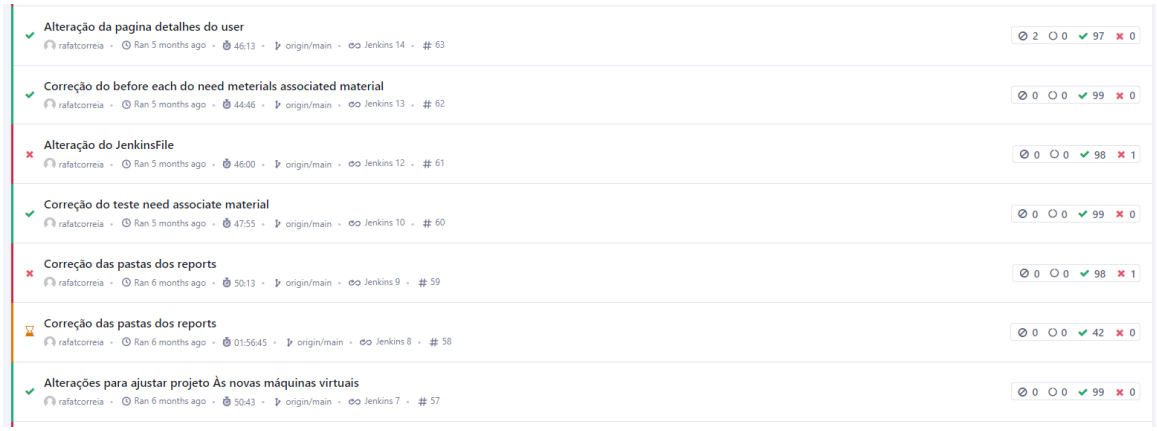

Figura 6.6 - Excerto da Cypress Dashboard

Na *dashboard* do Jenkins é possível consultar o resultado dos testes na consola da respetiva *build*, onde se pode observar como os testes foram executados, se estes foram ou não bem-sucedidos e também no final um resumo de todos os testes realizados. Como se pode observar na Figura 6.7, são apresentados os testes realizados, a sua duração, a quantidade de casos, o número de testes bem-sucedidos e mal sucedidos. Esta informação pode ser observada em tempo real enquanto o *pipeline* está a executar, uma vez que esta está a ser escrita para a consola do Jenkins durante a sua execução.

	Spec		Tests	Passing	Failing	Pending	Skipped
✗	administrator/create_platform_manager.cy.js	00:11	1	-	1	-	-
✗	administrator/needs/need_details_func.cy.js	00:08	3	-	1	-	2
✗	administrator/needs/need_materials_associated_material_details.cy.js	00:19	5	-	1	-	4
✗	administrator/needs/need_materials_list_func.cy.js	00:48	3	-	3	-	-
✓	administrator/needs/needs_create.cy.js	921ms	2	-	-	-	2
✗	administrator/needs/needs_list_func.cy.js	00:08	2	-	1	-	1

Figura 6.7 - Excerto do resumo dos resultados dos testes na consola da *build*

6.7. Manutenção

De forma a manter o contínuo funcionamento do *pipeline* CI/CD e a contínua atualização da aplicação a ser monitorizada deve ter-se em conta alguns aspetos importantes. A falha de

algum destes aspetos pode comprometer os objetivos descritos acima. Os aspetos importantes a ter em conta são:

1. contínuo funcionamento das máquinas virtuais que suportam tanto o Jenkins como a ambiente utilizado para testes;
2. contínuo desenvolvimento dos testes em paralelo com o desenvolvimento de novas funcionalidades;
3. atualização dos testes em caso de atualizações no código da aplicação que alterem os processos existentes;
4. utilização das boas práticas de Cypress e de *feature branching;*
5. contínua monitorização do *dashboard* do Jenkins;
6. contínua monitorização dos resultados e relatórios dos testes realizados após atualização à aplicação.

7. Caso de Estudo

O projeto Help2Care-Pal é composto por uma plataforma digital inovadora de *e-health* que pretende apoiar os cuidadores informais no seu próprio cuidado e no do seu familiar, em situações de cuidados paliativos (CP). O projeto é uma parceria entre o Politécnico de Leiria e a equipa Beja+, financiado pela Fundação "la Caixa", e tem como objetivos gerais desenvolver uma intervenção de saúde digital colaborativa e participativa de apoio ao cuidador informal e gestor de caso em CP, com potencial de transferência de conhecimento para outros contextos e com recurso a novas intervenções/materiais de apoio integradas numa plataforma digital. Esta permite:

- Capacitar o cuidador informal na gestão de situações complexas;
- Monitorizar a condição do doente e cuidador informal pelos gestores de caso em CP;
- Aplicar estratégias de autocuidado ao cuidador informal;
- Rastrear a sobrecarga e o risco de luto complicado do cuidador informal.

O projeto parte de uma reestruturação do projeto Help2Care (projeto financiado pela Fundação para a Ciência e Tecnologia), que já se destinava a apoiar a capacitação dos cuidadores informais. O projeto Help2Care-Pal é uma especialização para cuidados paliativos. A plataforma original Help2Care é constituída por uma aplicação *web* e uma aplicação móvel. A aplicação *web* consiste num *backoffice* para os profissionais de saúde acederem e fazerem a gestão dos utilizadores e conteúdos da plataforma. A aplicação foi criada seguindo as diretrizes da *framework* Laravel e seguindo um modelo Model-View-Controller (MVC).

A aplicação móvel foi criada utilizando a *framework* Ionic e Angular, e é destinada aos cuidadores informais. A aplicação móvel conecta com a aplicação *web* através de um *Application Programming Interface* (API) que permite fornecer dados à aplicação móvel e a comunicação entre os CI e os gestores de caso.

O projeto Help2Care-Pal é uma ferramenta inovadora que tem o potencial de ajudar a melhorar a qualidade de vida dos cuidadores informais e dos doentes em cuidados paliativos.

7.1. Recuperação do projeto Help2Care

A plataforma digital do projeto original Help2Care teve a sua última alteração no ano de 2019, tendo sido necessária uma atualização extensa em ambas as aplicações para as versões mais recentes das *frameworks* e bibliotecas utilizadas.

O projeto Help2Care foi desenvolvido até ao ano de 2019 utilizando as versões existentes na altura, sendo no caso do *backoffice* a utilização da *framework* Laravel na sua versão 5.4, implicando a utilização da versão 5.6 do PHP. Com isto era necessário realizar uma atualização tanto da versão do Laravel, como das respetivas dependências existentes no projeto, mantendo todas a funcionalidades do projeto Help2Care.

Esta atualização foi realizada versão a versão até à mais estável existente atualmente (versão 8). De forma a manter a atualização o mais estável possível ao longo de todo o processo, para subir da versão atual para a seguinte foi necessário fazer a atualização das dependências a cada passo realizado. Como é possível visualizar na Figura 7.1, foram necessários sete incrementos nas versões do Laravel para ser possível atingir a versão mais estável para as dependências existentes no projeto, fazendo-se sempre acompanhar também a atualização da versão do PHP.

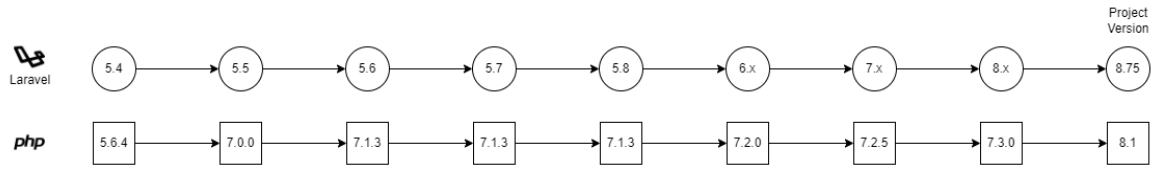

Figura 7.1 - Esquema dos passos para atualização do projeto original Help2Care

As dificuldades encontradas durante a atualização da *framework* Laravel, assim como para todas as dependências existentes, foram uma das razões para o desenvolvimento do *pipeline* CI/CD. Caso já existisse este *pipeline*, com maior facilidade se teria percebido se todas as funcionalidades do projeto estavam em pleno funcionamento ao longo de todos os incrementos das versões.

7.2. Levantamento dos requisitos de software para o projeto Help2Care-Pal

Para efetuar a transição da plataforma Help2Care para a especialização da plataforma Help2Care-Pal, realizou-se um levantamento de requisitos de software entre os professores

responsáveis pelo projeto e a equipa de profissionais no terreno que serão os utilizadores finais da plataforma.

Após várias e contínuas reuniões, os requisitos obtidos foram divididos pelos papéis previstos na nova plataforma. Os requisitos funcionais gerais obtidos foram os seguintes:

- Gestor de Plataforma:
    - Realizar a gestão de especialistas (Create, Retrieve, Update e Delete - CRUD);
    - Realizar a gestão de materiais de capacitação (CRUD);
    - Realizar a gestão de necessidades (CRUD);
    - Realizar a gestão das associações entre necessidades e materiais de capacitação;
    - Realizar a gestão de equipas (CRUD);
    - Realizar a associação dos profissionais de saúde às equipas.
- Especialista:
    - Realizar a gestão de gestores de caso (CRUD);
    - Realizar a gestão de cuidadores informais (CRUD);
    - Realizar a gestão de doentes (CRUD);
    - Realizar a gestão da associação entre cuidadores informais e doentes;
    - Realizar a gestão da associação entre gestores de caso e o binómio cuidador informal/doente;
    - Comunicação via *chat* com a Equipa.
- Gestor de Caso:
    - Registar novas necessidades;
    - Associar/Dissociar materiais aos cuidadores informais;
    - Comunicação via *chat* com a Equipa.
- Criação de Equipas, constituídas por especialistas e gestores de caso para criação de *chat*.

Com estes requisitos foi necessário realizar alterações nas funcionalidades existentes para definir que utilizadores tinham acesso aos dados disponibilizados e acesso às funcionalidades respetivas para cada um dos papéis previstos. Como se pode verificar na Figura 7.2, houve necessidade de alteração dos papéis dos utilizadores da plataforma original Help2Care para a plataforma Help2Care-Pal.

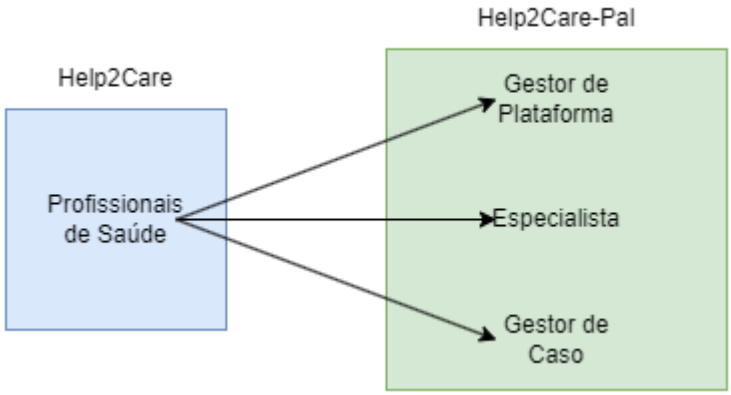

Figura 7.2 - Divisão do papel "Profissional de Saúde" da plataforma original Help2Care para os papéis na nova plataforma Help2Care – Pal

7.3.Ambiente de desenvolvimento

Com a atualização da versão do Laravel para a mais recente, foi possível fazer-se uso da ferramenta Laravel Sail, e assim criar o ambiente de desenvolvimento local através desta ferramenta. O Laravel Sail necessita da instalação prévia o Docker, e no caso de sistemas operativos Windows também da configuração do WSL onde serão executados os comandos para iniciar o Laravel Sail.

Uma das características do Laravel Sail é a disponibilização de serviços pré-configurados, como um servidor *web* (Nginx ou Caddy), bases de dados (MySQL, PostgreSQL ou SQLite), serviço de cache (Redis) ou serviço de e-mail (Mailhog). Estes diferentes serviços estão prontos para uso assim que o Sail for iniciado e também podem ser configurados de acordo com as necessidades do projeto. No caso da plataforma Help2Care-Pal, fez-se uso de um servidor *web* Nginx e uma base de dados MySQL. Também é possível adicionar outros serviços que não estão pré-configurados no Laravel Sail, alterando o ficheiro de configuração do Docker, `Docker-compose.yml`. Para a plataforma foi adicionado um serviço de *websockets* para processamento em tempo real das funcionalidades da aplicação. A Figura 7.3 apresenta um exemplo dos diferentes contentores criados pelo Laravel Sail.

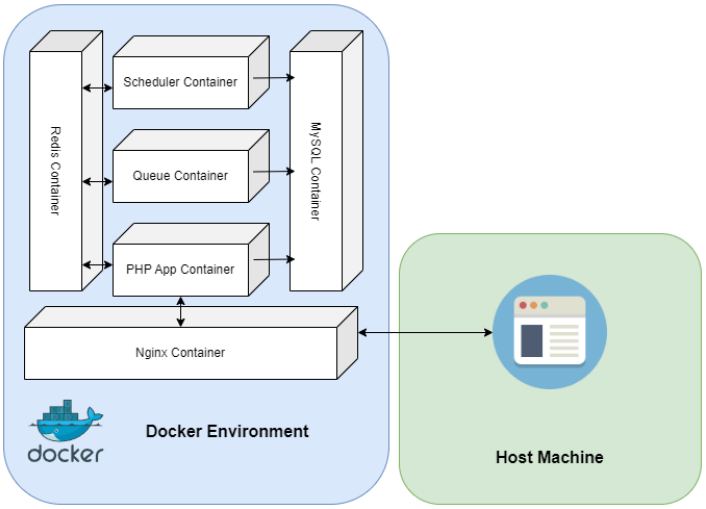

Figura 7.3 - Esquema exemplo do ambiente criado pelo Laravel Sail (fonte: Laravel Sail e Docker)

Depois de configurar e inicializar o Laravel Sail, é possível aceder-se à aplicação através de um *browser* ou uma API usando um URL local do tipo http://localhost, ou através de um domínio personalizado, caso este seja necessário e configurado.

7.4. Desenvolvimento e *Deploy*

Com o ambiente de desenvolvimento preparado e o *pipeline* CI/CD funcional iniciou-se o desenvolvimento das funcionalidades para cumprir os requisitos definidos para a plataforma Help2Care-Pal. Dos 17 requisitos definidos inicialmente, apenas os requisitos para a criação das equipas, associação de profissionais de saúde às equipas e os requisitos ligados ao *chat* das equipas, foram funcionalidades criadas de origem. Os restantes requisitos foram cumpridos através da alteração das funcionalidades existentes da plataforma Help2Care.

As alterações às funcionalidades existentes foram necessárias uma vez que houve divisão do papel profissional de saúde em três papeis diferentes e com isso as responsabilidades de cada um desses papeis.

Em paralelo ao desenvolvimento do código-fonte da plataforma foram também desenvolvidos os testes correspondentes a cada uma das funcionalidades da plataforma. Foram realizados em média 4 testes por cada página referente às funcionalidades de cada requisito. Um requisito envolvendo o CRUD, tem pelo menos 5 páginas diferentes para cada uma das funcionalidades: *create*, *retrieve*, *update e remove*. Foram assim desenvolvidos em média 20 testes diferentes por requisito. O número de testes dependia sempre da funcionalidade a ser testada e do número de caminhos que esta tenha.

Seguindo a prática de *feature branching*, foram criadas diferentes *branches* para cada um dos requisitos definidos. Cada uma destas *branches* corresponde a pelo menos uma *build* realizada pelo *pipeline*. Cada *merge* de cada *branch* com o *branch* main, ativou o *pipeline* CI/CD e assim realizados os testes automatizados à nova funcionalidade e o seu *deploy* na máquina virtual Production e assim disponível para os utilizadores. Sempre que os testes encontravam alguma falha, fazia-se a resolução do problema na respetiva *branch*, uma vez que estas não foram eliminadas quando realizado o *merge*, e realizado novamente o *merge* para a *branch* main.

Com todas as funcionalidades de cada requisito desenvolvidas e a plataforma preparada para utilização foi realizado um primeiro teste com utilizadores, onde foi possível obter o *feedback* desses utilizadores através da utilização da plataforma como também foram realizados testes de usabilidade – observação direta e PSSUQ[15]-, realizados no âmbito de outro projeto de mestrado que utilizou o a plataforma Help2Care-Pal como caso de estudo. Foram assim registados problemas encontrados pelos utilizadores como também alterações a serem realizadas para melhorar o funcionamento da plataforma de acordo com as necessidades dos seus utilizadores. Após resolvidos os problemas encontrados nesta primeira interação de utilizadores, os testes foram repetidos com os mesmos utilizadores sendo realizadas novas melhorias na plataforma.

Todas as melhorias realizadas nas duas interações com os utilizadores de teste, foram registadas no *pipeline* através de duas diferentes *builds*, correspondente às duas *branches* criadas para realizar as alterações necessárias.

O próximo passo foi a disponibilização da plataforma aos utilizadores finais, neste caso os profissionais de saúde da equipa Beja+ e os cuidadores informais selecionados pela mesma. Procedeu-se a uma apresentação das funcionalidades da plataforma à equipa e disponibilizou-se o acesso à mesma.

Ao longo da utilização da plataforma pela equipa foram necessárias fazer pequenas alterações e ajustes na plataforma de acordo com as necessidades da equipa no terreno. Com a utilização do *pipeline* CI/CD, foi possível manter a plataforma em produção e ser atualizada à medida que as alterações foram sendo terminadas e testadas.

[15] Post-Study System Usability Questionnaire – Consiste num questionário padronizado de 16 questões que avalia a satisfação do utilizador.

8. Conclusões e Trabalho Futuro

Este projeto teve como objetivo principal a implementação de um *pipeline* CI/CD para aplicações de saúde de forma a automatizar a validação de uma plataforma digital Help2Care-Pal, através de testes, e também a sua colocação em produção. Pretendia-se, portanto, a contínua integração de novas funcionalidades ou atualizações às já existentes na plataforma original Help2Care, e agilizar também a colocação dessas alterações em produção de forma rápida e fiável, mantendo entregas contínuas.

A implementação do *pipeline* permitiu à equipa responsável pelo desenvolvimento do caso de estudo, plataforma Help2Care-Pal, implementar as novas funcionalidades como também alterar as já existentes de acordo com os requisitos definidos e, em paralelo, desenvolver os respetivos testes. Assim, sempre que uma das funcionalidades fosse terminada e realizado o *merge* com o *branch* principal, toda a plataforma era testada. Desta forma era possível avançar no desenvolvimento se os testes fossem bem-sucedidos ou corrigir imediatamente problemas encontrados.

O *pipeline* permitiu assim colocar a plataforma Help2Care-Pal em produção de forma mais rápida, obter *feedback* dos utilizadores mais frequentemente e mais rápido, e integrar as novas funcionalidades ou alterações necessárias à medida que foram sendo desenvolvidas. A obtenção de *feedback* através de testes de usabilidade, observação direta e PSSUQ, realizados no âmbito de outro projeto de mestrado que utilizou o mesmo caso de estudo, implicou também a realização de alterações na plataforma. A colocação destas modificações em produção foi agilizada com o *pipeline* CI/CD.

A utilização do *pipeline* CI/CD permitiu também manter a qualidade e segurança da plataforma, graças aos testes automatizados realizados a cada iteração da plataforma. Permitindo assim que a plataforma Help2Care-Pal se encontre em produção de forma funcional e a ser utilizada pela equipa Beja+.

A implementação de um *pipeline* CI/CD permite expandir de forma ágil, segura e consistente as plataformas que façam uso desta ferramenta. Esta ferramenta permite também prever trabalho futuro que possa ser realizado, garantindo uma melhor integração e entrega aos utilizadores finais de novas funcionalidades.

Como trabalho futuro, existem novos requisitos para serem implementados na plataforma Help2Care-Pal, após *feedback* fornecido tanto pela equipa Beja+, como pelos gestores da plataforma. Esses requisitos consistem em:

1. Resolver erros encontrados no *chat* da equipa;
2. Alteração da visualização de notificações referentes aos pedidos de ajuda colocados pelos cuidadores informais;
3. Coerências na apresentam dos dados pelas diferentes listagens existentes na plataforma;
4. Criação de ordenações para todas as listagens;
5. Alterações visuais para melhorar a usabilidade da plataforma.

A configuração de um *dashboard*, tipo Grafana, de monitorização do ambiente de produção é um dos possíveis trabalhos futuros para o projeto, fornecendo aos gestores da plataforma mais um elemento de controlo de qualidade e gestão de recursos da mesma. A este *dashboard* podem ser adicionadas as diferentes métricas disponibilizadas pelos diferentes artefactos que constituem o projeto:

1. *Pipeline* CI/CD, obtendo as diferentes *builds* realizadas, o estado atual do *pipeline* (estado da última *build*) e relatórios de testes;
2. Máquina virtual Production, estado da máquina virtual e respetivos recursos a ser utilizados;
3. Métricas retiradas na plataforma Help2Care-Pal, como por exemplo, utilização da plataforma ou número de ações realizadas.

Assim, e como todo o trabalho descrito neste relatório se baseia no contínuo melhoramento e contínuo desenvolvimento, a manutenção é um dos trabalhos futuros não só para esta plataforma como também para todo o software existente.

Referências Bibliográficas

[1] S. A. I. B. S. Arachchi e I. Perera, «Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management», em *2018 Moratuwa Engineering Research Conference (MERCon)*, 2018, pp. 156–161. doi: 10.1109/MERCon.2018.8421965.

[2] J. Humble e D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[3] Statista Market Insights, «eHealth - Worldwide». Acedido: 11 de Setembro de 2023. [Em linha]. Disponível em: https://www.statista.com/outlook/dmo/digital-health/ehealth/worldwide?currency=EUR

[4] Grand View Research, «eHealth Market Size, Share & Trends Analysis Report By Product (Big Data For Health, EHR, mHealth, Health Information Systems), By Services, By End Use, By Region, And Segment Forecasts 2023 - 2030». Acedido: 20 de setembro de 2023. [Em linha]. Disponível em: https://www.grandviewresearch.com/industry-analysis/e-health-market

[5] S. C. Mathews, M. J. McShea, C. L. Hanley, A. Ravitz, A. B. Labrique, e A. B. Cohen, «Digital health: a path to validation», *npj Digital Medicine*, vol. 2, n. 1. Nature Publishing Group, 1 de dezembro de 2019. doi: 10.1038/s41746-019-0111-3.

[6] Matt Ferrari, «Continuous Integration/Continuous Delivery (CI/CD) Means Continuous Improvement». Acedido: 8 de dezembro de 2022. [Em linha]. Disponível em: https://www.cleardata.com/ci-cd-healthcare/

[7] T. Laukkarinen, K. Kuusinen, e T. Mikkonen, «DevOps in regulated software development: Case medical devices», em *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Results Track, ICSE-NIER 2017*, Institute of Electrical and Electronics Engineers Inc., Jun. 2017, pp. 15–18. doi: 10.1109/ICSE-NIER.2017.20.

[8] M. S. Khan, A. W. Khan, F. Khan, M. A. Khan, e T. K. Whangbo, «Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic

Review», *IEEE Access*, vol. 10, pp. 14339–14349, 2022, doi: 10.1109/ACCESS.2022.3145970.

[9] J. E. Ferreira Caroço, «HELP2CARE-APP: HELPING CAREGIVERS DELIVERING BETTER CARE».

[10] Y. Beng Leau, W. Khong Loo, W. Yip Tham, e S. Fun Tan, «Software Development Life Cycle AGILE vs Traditional Approaches».

[11] A. Sinha e P. Das, «Agile Methodology Vs. Traditional Waterfall SDLC: A case study on Quality Assurance process in Software Industry», em *2021 5th International Conference on Electronics, Materials Engineering and Nano-Technology, IEMENTech 2021*, Institute of Electrical and Electronics Engineers Inc., 2021. doi: 10.1109/IEMENTech53263.2021.9614779.

[12] Y. Bassil, «A Simulation Model for the Waterfall Software Development Life Cycle», 2012. [Em linha]. Disponível em: http://iet-journals.org/archive/2012/may_vol_2_no_5/255895133318216.pdf

[13] F. I. Maulana, V. Susanto, P. Shilo, J. Gunawan, G. Pangestu, e D. R. B. Raharja, «Design and Development of Website Dr.Changkitchen Diet Catering Using SDLC Waterfall Model», em *ACM International Conference Proceeding Series*, Association for Computing Machinery, Set. 2021, pp. 75–79. doi: 10.1145/3479645.3479652.

[14] M. Kramer, «BEST PRACTICES IN SYSTEMS DEVELOPMENT LIFECYCLE: AN ANALYSES BASED ON THE WATERFALL MODEL», *Review of Business & Finance Studies*, vol. 9, n. 1, pp. 77–84, 2018, [Em linha]. Disponível em: https://ssrn.com/abstract=3131958www.theIBFR.com

[15] M. Stoica, M. Mircea, e B. Ghilic-Micu, «Software Development: Agile vs. Traditional», *Informatica Economica*, vol. 17, pp. 64–76, ago. 2013, doi: 10.12948/issn14531305/17.4.2013.06.

[16] M. or M. Javanmard e M. Alian, «Comparison between Agile and Traditional software development methodologies», Ago. 2015.

[17] L. Khong, L. Yu Beng, T. Yip, e T. Soofun, «Software Development Life Cycle AGILE vs Traditional Approaches», Ago. 2012.

[18] S. Nerur, R. Mahapatra, e G. Mangalaraj, «Challenges of Migrating to Agile Methodologies», *Commun. ACM*, vol. 48, n. 5, pp. 72–78, Mai. 2005, doi: 10.1145/1060710.1060712.

[19] Sourojit Das, «DevOps Lifecycle : Different Phases in DevOps». Acedido: 30 de agosto de 2023. [Em linha]. Disponível em: https://www.browserstack.com/guide/devops-lifecycle

[20] C. Ebert, G. Gallardo, J. Hernantes, e N. Serrano, «DevOps», *IEEE Softw*, vol. 33, n. 3, pp. 94–100, mai. 2016, doi: 10.1109/MS.2016.68.

[21] R. T. Yarlagadda, «DevOps and Its Practices», 2021. [Em linha]. Disponível em: www.ijcrt.org

[22] M. Fowler e M. Foemmel, «Continuous integration». Acedido: 12 de junho de 2023. [Em linha]. Disponível em: http://martinfowler.com/articles/continuousIntegration.html

[23] K. Beck e M. Fowler, *Planning extreme programming*. Addison-Wesley Professional, 2001.

[24] K. Beck *et al.*, «The agile manifesto». The Agile Alliance, www. agilemanifesto. org, 2001.

[25] E. Laukkanen, J. Itkonen, e C. Lassenius, «Problems, causes and solutions when adopting continuous delivery—A systematic literature review», *Information and Software Technology*, vol. 82. Elsevier B.V., pp. 55–79, 1 de fevereiro de 2017. doi: 10.1016/j.infsof.2016.10.001.

[26] Simplilearn, «10 Best CI/CD Tools Used by Programmers Today and Why You Should Learn Them in 2023». Acedido: 19 de dezembro de 2022. [Em linha]. Disponível em: https://www.simplilearn.com/best-ci-cd-tools-article

[27] C. C. Kitakabee, «Top 14 CI CD Tools for your DevOps project». Acedido: 19 de dezembro de 2022. [Em linha]. Disponível em: https://www.browserstack.com/guide/top-ci-cd-tools

[28] Katalon, «Best 14 CI/CD Tools You Must Know | Updated for 2022». Acedido: 19 de dezembro de 2022. [Em linha]. Disponível em: https://katalon.com/resources-center/blog/ci-cd-tools

[29] R. K. Gupta, M. Venkatachalapathy, e F. K. Jeberla, «Challenges in Adopting Continuous Delivery and DevOps in a Globally Distributed Product Team: A Case Study of a Healthcare Organization», em *Proceedings - 2019 ACM/IEEE 14th International Conference on Global Software Engineering, ICGSE 2019*, Institute of Electrical and Electronics Engineers Inc., Mai. 2019, pp. 30–34. doi: 10.1109/ICGSE.2019.00020.

[30] R. T. Yarlagadda, «Implementation of DevOps in healthcare systems», USA, Jun. 2017. Acedido: 30 de junho de 2023. [Em linha]. Disponível em: www.jetir.org

[31] Greg Sienkiewicz, «How to Implement DevOps in Healthcare». Acedido: 8 de dezembro de 2022. [Em linha]. Disponível em: https://www.macadamian.com/learn/how-to-implement-devops-in-healthcare/

[32] S. Purkayastha, S. Goyal, T. Phillips, H. Wu, B. Haakenson, e X. Zou, «Continuous Security through Integration Testing in an Electronic Health Records System», em *Proceedings - 2020 International Conference on Software Security and Assurance, ICSSA 2020*, Institute of Electrical and Electronics Engineers Inc., 2020, pp. 26–31. doi: 10.1109/ICSSA51305.2020.00012.

[33] G. Kreps, «Communication and Palliative Care: E-Health Interventions and Pain Management», em *Handbook of Pain and Palliative Care*, Springer International Publishing, 2018, pp. 71–81. doi: 10.1007/978-3-319-95369-4_5.

[34] J. Mills, J. Fox, R. Damarell, J. Tieman, e P. Yates, «Palliative care providers' use of digital health and perspectives on technological innovation: a national study», *BMC Palliat Care*, vol. 20, n. 1, Dez. 2021, doi: 10.1186/s12904-021-00822-2.

[35] A. M. Finucane, H. O'donnell, J. Lugton, C. Swenson, e C. Pagliari, «Digital Health Interventions in Palliative Care: A Systematic Meta-Review and Evidence Synthesis», doi: 10.1101/2020.09.16.20195834.

[36] L. P. Binamungu, S. M. Embury, e N. Konstantinou, «Maintaining behaviour driven development specifications: Challenges and opportunities», em *25th IEEE*

International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Abr. 2018, pp. 175–184. doi: 10.1109/SANER.2018.8330207.

[37] L. Allodi, S. Banescu, H. Femmer, e K. Beckers, «Identifying relevant information cues for vulnerability assessment using CVSS», em *CODASPY 2018 - Proceedings of the 8th ACM Conference on Data and Application Security and Privacy*, Association for Computing Machinery, Inc, Mar. 2018, pp. 119–126. doi: 10.1145/3176258.3176340.

[38] C. Ebert, G. Gallardo, J. Hernantes, e N. Serrano, «DevOps», *IEEE Softw*, vol. 33, n. 3, pp. 94–100, mai. 2016, doi: 10.1109/MS.2016.68.

[39] Statista, «Digital Health - Worldwide». Acedido: 22 de dezembro de 2022. [Em linha]. Disponível em: https://www.statista.com/outlook/dmo/digital-health/worldwide?currency=EUR#revenue

[40] N. Purtova, E. Kosta, e B. J. Koops, «Laws and regulations for digital health», em *Requirements Engineering for Digital Health*, Springer International Publishing, 2015, pp. 47–74. doi: 10.1007/978-3-319-09798-5_3.