INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Media Ingest Systems - The Road To SaaS

ALEXANDRA ISABEL ARAÚJO BRANCO
outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Media Ingest Systems
The Road To SaaS

Alexandra Isabel Araújo Branco

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

Orientador: Dr. Nuno Bettencourt
Supervisor: João Estevinho

Porto, 15 de outubro de 2022

Para ti, Pai

Que sempre acreditaste em mim e me tornaste na pessoa que hoje sou.

Resumo

Durante os últimos anos, os avanços da computação na *cloud* e das tecnologias de conteinerização vieram facilitar a distribuição do *software*. Assim surge o conceito de *Software as a Service* (SaaS), que se traduz na disponibilização do *software* através da internet. Porém, a adoção desta prática permanece complexa, uma vez que envolve alterações no quotidiano das organizações e no processo de desenvolvimento e entrega do *software*. A presente dissertação pretende demonstrar a aplicabilidade deste conceito através do provisionamento de uma infraestrutura capaz de suportar a implantação do *software* MAM4PRO. Foram definidas quatro hipóteses que visam responder à questão de investigação colocada, e são avaliadas após a implementação da solução.

Após a identificação e contextualização do problema foi estudado o estado atual da arte e das tecnologias que permitem concretizar os objetivos. Inicialmente, realizou-se a análise SWOT e o *Value Proposition Canvas* para validar a oportunidade. Posteriormente, o método *Analytic Hierarchy Process* (AHP) demonstrou que a tecnologia Kubernetes é a mais adequada para orquestração dos *containers* do MAM4PRO. Para provisionar a infraestrutura escolheu-se aplicar técnicas de *Infrastructure as Code* (IaC) com recurso à tecnologia Terraform. Adicionalmente, para garantir o valor da solução utilizou-se a ferramenta *House of Quality* (HOQ) inserida na primeira fase do método *Quality Function Deployment* (QFD). Esta etapa conduziu à identificação dos requisitos funcionais e não funcionais da solução, que por sua vez permitiram a definição dos casos de uso do sistema.

Após a análise da solução desenhou-se a sua arquitetura considerando as tecnologias escolhidas, os requisitos e os casos de uso definidos.

Por último, procedeu-se à execução de testes funcionais desenhados com base nos casos de uso do sistema, para analisar o comportamento da solução implementada e verificar as hipóteses de investigação definidas. A primeira e a segunda hipóteses, que dizem respeito à automatização do provisionamento da infraestrutura e implantação do MAM4PRO, não se comprovaram. Por outro lado, a terceira hipótese verifica-se, uma vez que o tempo de implantação do MAM4PRO foi inferior ao valor definido. Em relação à quarta hipótese, que dizia respeito à possibilidade de alterar algumas configurações do MAM4PRO depois de implantado, também se verificou. Os resultados obtidos revelaram-se promissores, uma vez que a automatização e a eficiência conseguida com esta solução simplifica o trabalho manual normalmente exigido nestes contextos.

Em suma, os objetivos desta dissertação foram alcançados permitindo a distribuição do MAM4PRO como um serviço.

Palavras-chave: *Containers*, *Infrastructure as Code*, Kubernetes, *Software as a Service*, Terraform.

Abstract

Over the last years, the increasing cloud computing and containerization technologies came to improve the distribution of software. Thus, creating the concept of Software as a Service (SaaS), which regards the distribution of software through the internet. Still, applying this practice remains complex given the necessary changes in the company's daily tasks and software development and delivery. In that concern, the present dissertation aims to demonstrate the applicability of this concept through the provisioning of an infrastructure able to support the deployment of the software MAM4PRO. Four hypotheses aiming to answer the research question were defined and are evaluated after the solution implementation.

Following the problem identifying and contextualizing, the current state of the art and the technologies available to reach the objectives were explored. Initially, SWOT analysis and a Value Proposition Canvas were realized to validate the opportunity. Then, the Analytic Hierarchy Process (AHP) revealed that Kubernetes is the most appropriate technology for the orchestration of the MAM4PRO's containers. To provision the infrastructure, techniques of Infrastructure as Code (IaC) using the Terraform technology were selected. Additionally, to confirm the value of the solution the House of Quality (HOQ) tool which integrates the first step of the Quality Function Deployment (QFD) method was applied. This led to the identification of the function and non-functional requirements of the solution, which in turn allowed the definition of the use cases.

Following the analyses of the solution, its architecture was designed considering the chosen technologies, requirements, and the defined use cases.

Finally, functional tests designed considering the use cases of the system were performed to analyze the behavior of the implemented solution and verify the research hypotheses. The first and second hypotheses, which concern the automatization of the infrastructure provisioning and MAM4PRO deployment, were not confirmed. Contrariwise, the third hypothesis was corroborated since the deployment time of the MAM4PRO software was lower than the initially defined value. Regarding the fourth hypothesis, which concerns the possibility of changing some configurations of the MAM4PRO after deployment, it was also verified. The obtained results revealed promising since the automatization and efficiency obtained with this solution simplify the manual tasks usually required in these contexts.

Overall, the objectives of this dissertation were achieved by allowing the distribution of the MAM4PRO as a service.

Keywords: Containers, Infrastructure as Code, Kubernetes, Software as a Service, Terraform.

Agradecimentos

Agradeço ao meu orientador Professor Nuno Bettencourt, por todo o acompanhamento, disponibilidade e orientação ao longo destes meses e ao Instituto Superior de Engenharia do Porto, pelo curso que me proporcionaram sempre pautado pela qualidade e exigência.

Um muito obrigado à MOG Technologies, por me ter garantido todas as condições para a frequência do curso e desenvolvimento desta dissertação.
Um agradecimento muito especial ao João Estevinho e ao André Costa e Silva por me terem dado esta oportunidade e me terem sempre apoiado e encaminhado na melhor direção possível desde o primeiro minuto. Tenho a certeza de que sem eles nada disto seria possível.

Ao Daniel Moreira, por me chamar sempre à razão e não me deixar fugir do meu objetivo. Por me fazer crer até ao último minuto que era capaz e me ter prometido um brinde de *champagne* no final da defesa desta dissertação.
À Sara Fidalgo, obrigada por todos os almoços, pausas para café e jantares. Por todas as conversas sobre tudo e sobre nada que, sem ela saber, me ajudaram a encontrar a força e energia para continuar. Agora percebo o quão importantes foram.
Ao Luís Cerqueira, por me fazer sempre ter os pés assentes na Terra e me fazer acreditar em mim. Por me mostrar que no final do dia o mais importante é fazermos sempre aquilo em que acreditamos.
À Maria Pinto, por me encorajar sempre e me fazer acreditar que todos os sonhos são possíveis com garra e determinação.

À minha família por ter sido o pilar mais importante ao longo destes anos. À minha irmã Catarina Branco, pelos conselhos preciosos, pela disponibilidade e encorajamento nos momentos mais importantes desta difícil jornada. Obrigada por leres imensas versões desta dissertação e me ajudares a melhorar sempre. És a melhor irmã que eu poderia ter.
À minha mãe Alice Branco, por todos os ensinamentos e por me incentivar sempre a ser mais e melhor.
Ao meu pai Porfírio Branco, que estará certamente mais orgulhoso de mim do que eu própria. Ainda que a tua presença não se concretize fisicamente, sei que estás sempre presente para me dar a força necessária para continuar. Espero um dia conseguir ser, pelo menos metade, do ser humano que tu foste.

Por fim, quero agradecer a todos os meus amigos, ao Slimmy, ao Tomé, à Bárbara, à Anabela, à Daniela, à Rafaela - com vocês do meu lado foi mais fácil.
Um obrigado a todas as pessoas com quem me cruzei nos últimos anos e que de alguma forma me incentivaram e motivaram a seguir em frente nesta caminhada.

A todos vós, muito obrigada.

Alexandra Branco

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Código

Lista de Acrónimos

AHP	*Analytic Hierarchy Process*
API	*Application Programming Interface*
AR	*Action Research*
ASP	*Application Service Provider*
AV	Análise de Valor
AWS	*Amazon Web Services*
CD	*Continuous Deployment*
CDE	*Continuous Delivery*
CI	*Continuous Integration*
CLI	*Command Line Interface*
CNCF	*Cloud Native Computing Foundation*
CPU	*Central Processing Unit*
CRM	*Customer Relationship Management*
DC/OS	*Datacenter Operating System*
DMZ	*De-Militarized Zone*
DNS	*Domain Name System*
DSR	*Design Science Research*
EC2	*Elastic Compute Cloud*
ERP	*Enterprise Resource Planning*
FFE	*Fuzzy Front End*
HCL	*HashiCorp Configuration Language*
HDMI	*High Definition Multimedia Interface*
HOQ	*House of Quality*
HP	*Hewlett-Packard*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
IA	Índice Aleatório
IaC	*Infrastructure as Code*
IC	Índice de Consistência
IP	*Internet Protocol*
ISEP	Instituto Superior de Engenharia do Porto
LXC	*Linux Containers*
MEI	Mestrado em Engenharia Informática

MFC	*Microsoft Foundation Classes*
MTBF	*Mean Time Between Failures*
MTTR	*Mean Time To Repair*
NCD	*New Concept Development*
NPD	*New Product Development*
OCI	*Open Container Initiative*
OCP	*Open Container Project*
QFD	*Quality Function Deployment*
RAM	*Random Access Memory*
RC	Razão de Consistência
REST	*Representational State Transfer*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
SaaS	*Software as a Service*
SDI	*Serial Digital Interface*
SSH	*Secure SHell*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TI	Tecnologias da Informação
TMDEI	Tese de Mestrado do Departamento de Engenharia Informática
UDP	*User Datagram Protocol*
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
vCPUs	*Virtual Central Processing Unit*
VLAN	*Virtual Local Area Network*
VM	Máquina Virtual
VMs	Máquinas Virtuais
YAML	*YAML Ain't Markup Language*

Capítulo 1

Introdução

Esta dissertação foi realizada no âmbito da Unidade Curricular *Tese de Mestrado do Departamento de Engenharia Informática* (TMDEI) do *Mestrado em Engenharia Informática* (MEI) do *Instituto Superior de Engenharia do Porto* (ISEP).

Neste capítulo, inicialmente apresenta-se o contexto em que a dissertação se insere, o problema que se pretende solucionar, quais os objetivos a alcançar e a metodologia de investigação adotada. Posteriormente são apresentadas as questões e hipóteses de investigação que se pretendem verificar. Por último, é apresentado o plano de trabalhos e a estrutura desta dissertação.

1.1. Contexto

Nos últimos anos assistimos a uma transformação digital em vários negócios e setores de atividade, como é o caso da saúde e da educação. No entanto, a transformação digital não é específica de uma indústria [1], esta pode ser aplicada em qualquer organização que procure integrar tecnologia nas várias áreas do negócio, resultando em mudanças fundamentais na forma como operam e agregam valor para o cliente[1]. Nos ambientes de produção e pós-produção audiovisual esta transformação foi visível ao longo dos anos, e atualmente a televisão e o cinema já funcionam com conteúdos totalmente digitais [2]. O surgimento de dispositivos tecnológicos cada vez mais capazes contribuíram para esta transformação, e nos dias de hoje, computadores pessoais, *smartphones* ou *tablets* funcionam também como dispositivos de exibição e captura [3].

A MOG Technologies é uma empresa portuguesa sediada na Maia e fundada em 2007, cuja principal atividade é o desenvolvimento de novas plataformas tecnológicas para colocar no mercado produtos e soluções inovadoras que automatizem os processos de trabalho dos operadores de conteúdos multimédia profissional, nomeadamente estúdios de vídeo, produtoras, estações de televisão, entre outras. Atualmente comercializa o produto MAM4PRO, um sistema de *ingest* de nível empresarial que oferece uma ampla variedade de operações com *media*.

O *ingest* faz parte do quotidiano das empresas que trabalham com grandes volumes de *media* e é o termo técnico utilizado na indústria do *broadcast* para definir o processo de preparação dos conteúdos multimédia para edição [3]. Em particular, os sistemas de *ingest* auxiliam os operadores nas suas tarefas diárias através do processamento automático de fluxos, que podem incluir a captura de vários sinais de vídeo, a extração de metadados, transcodificação[2], gestão e distribuição do conteúdo.

[1] `https://enterprisersproject.com/what-is-digital-transformation`

[2] do inglês *transcode*, consiste na alteração da codificação atual.

1.2. Problema

Quando uma ordem de encomenda do MAM4PRO dá entrada na MOG Technologies, as equipas de *Product Assembly* e *Quality Assurance* são responsáveis pela preparação do *hardware* e instalação do *software*, que depois de preparados e validados seguem para as instalações do cliente.

Atualmente, o processo de instalação do MAM4PRO consiste na execução de um instalador (figura 1.1A), que dada a complexidade do sistema e a diversidade de clientes, permite através de caixas de seleção e ficheiros de configuração ativar ou desativar determinadas funcionalidades (figura 1.1B). As desvantagens associadas a este tipo de abordagem são evidentes, além de ser um processo manual e, por isso sujeito a falhas, é também demorado e altamente dependente de *hardware*. No entanto, estas desvantagens podem ser colmatadas melhorando a forma como o *software* é distribuído.

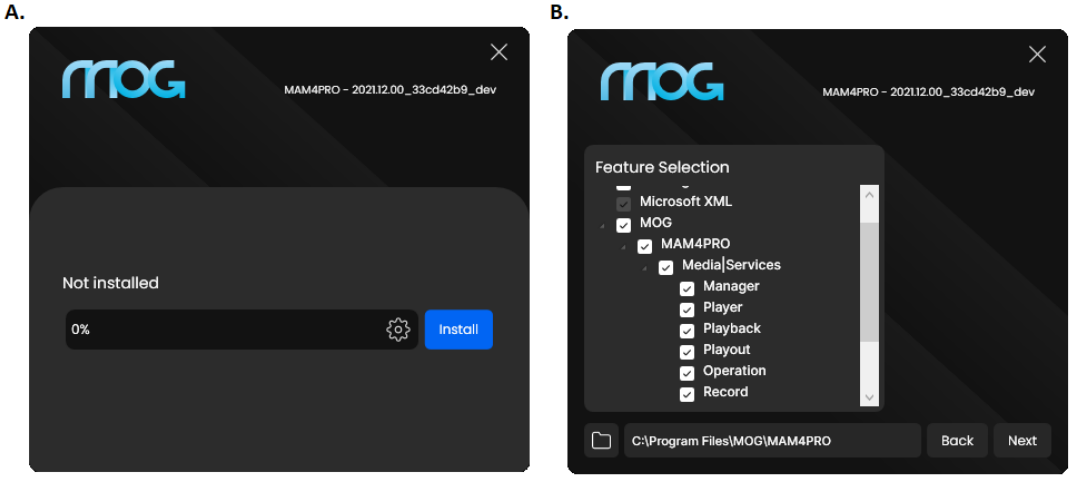


Figura 1.1. Processo de instalação do MAM4PRO. (A) Início da instalação. (B) Caixas de seleção que permitem selecionar as funcionalidades/serviços a instalar.

A evolução tecnológica da última década, juntamente com as melhorias da velocidade de conexão à internet, a capacidade de processamento de dados na *cloud*, o surgimento de arquiteturas de micro serviços e os utilizadores cada vez mais exigentes, permitiram evoluir a forma como o *software* é distribuído, surgindo o conceito de *Software as a Service* (SaaS) [4]. Embora seja uma forma de distribuição do *software* que está a ser amplamente adotada e em crescente desenvolvimento na indústria, a MOG Technologies possui casos de uso em que ainda existe uma alta dependência de *hardware* e por isso torna-se difícil uma total disponibilização do *software* como um serviço.

1.3. Objetivos

O objetivo desta dissertação é o provisionamento de uma infraestrutura (*i.e.*, *cloud* privada alojada nas instalações da MOG Technologies) capaz de suportar o *deployment* de *software*, nomeadamente do MAM4PRO. Assim, o que se pretende alcançar é um cenário híbrido (figura 1.2), isto é, manter o processo atual de distribuição do MAM4PRO mas também começar a disponibilizá-lo como um serviço na *cloud* (*clouds* públicas e privadas). Internamente a solução a implementar foi denominada *MOG One*.

Figura 1.2. Cenário híbrido de distribuição do *software* que se pretende alcançar.

Devem ser aplicados os conceitos de SaaS e *Infrastructure as Code* (IaC), bem como utilizadas ferramentas e tecnologias capazes de suportar esta realidade. Destacam-se ainda outros objetivos, nomeadamente:

- Garantir a portabilidade e a escalabilidade dos serviços para que rapidamente a infraestrutura possa ser utilizada para o desempenho de outras funções (*e.g.,* criação de ambientes de testes específicos, execução de testes automáticos);
- Integrar o desenvolvimento numa *pipeline* de *Continuous Integration* (CI)/*Continuous Delivery* (CDE) para que o processo de criação e destruição da infraestrutura seja automatizado;
- Desenvolvimento de um *dashboard* capaz de estabelecer comunicação com a *Application Programming Interface* (API) da infraestrutura para permitir o escalonamento e a elasticidade dos serviços.

1.4. Metodologia

Esta dissertação documenta a análise e implementação de uma solução capaz de automatizar o provisionamento de uma infraestrutura que suporte o *deployment* de *software*, objetivando a melhoria do processo de distribuição do MAM4PRO. Dadas estas características e os requisitos definidos (*cf.,* capítulo 3) optou-se por adotar a *Design Science Research* (DSR) como metodologia de investigação, uma vez que esta se foca na resolução de um problema através da criação de um artefacto tecnológico e se apresenta como uma metodologia científica para a conceção e desenvolvimento de produtos e serviços no domínio de *Tecnologias da Informação* (TI)[3]. Foram aplicadas todas as diretrizes desta metodologia pela seguinte ordem:

- A relevância para o problema (*cf.,* secção 1.2);
- *Design* como um processo de pesquisa (*cf.,* capítulo 2);
- Rigor de pesquisa (*cf.,* capítulo 3);
- *Design* como um artefacto (*cf.,* capítulo 5);
- A avaliação do *design* (*cf.,* capítulo 6);
- Contribuições (*cf.,* secção 7.2);
- Comunicação de pesquisa (*cf.,* secção 7.2).

[3] https://medium.com/@3D_Ideation/simplifying-design-science-research-action-research-and-design-research-bf564959402b

1.5. Questões de Investigação

Considerando os objetivos propostos, definiu-se uma questão de investigação na qual assenta o desenvolvimento desta dissertação: *"De que forma se conseguirá automatizar a criação de uma infraestrutura capaz de suportar o deployment do MAM4PRO?"*

1.6. Hipóteses de Investigação

Com base na questão de investigação colocada surgem as seguintes hipóteses de investigação, que devem ser corroboradas com o desenvolvimento desta dissertação:

- **H. 1:** Todas as tarefas de provisionamento da infraestrutura devem ser automatizadas;
- **H. 2:** Todas as tarefas de instalação do MAM4PRO devem ser automatizadas;
- **H. 3:** Deve ser possível implantar o MAM4PRO em até um minuto;
- **H. 4:** Depois de implantado o MAM4PRO deve ser possível alterar algumas configurações (*i.e.,* versão do *software*, número de serviços de um *deployment* e o *Uniform Resource Locator* (URL) de acesso ao MAM4PRO).

1.7. Plano de Trabalhos

De acordo com as metas estabelecidas na Unidade Curricular de TMDEI o desenvolvimento deste projeto dividiu-se em duas fases: a primeira que corresponde a toda a compreensão do problema e do contexto em que a dissertação se insere, à revisão da literatura e ao *design* da solução, e a segunda fase que corresponde à implementação da solução e avaliação dos resultados obtidos (figura 1.3).

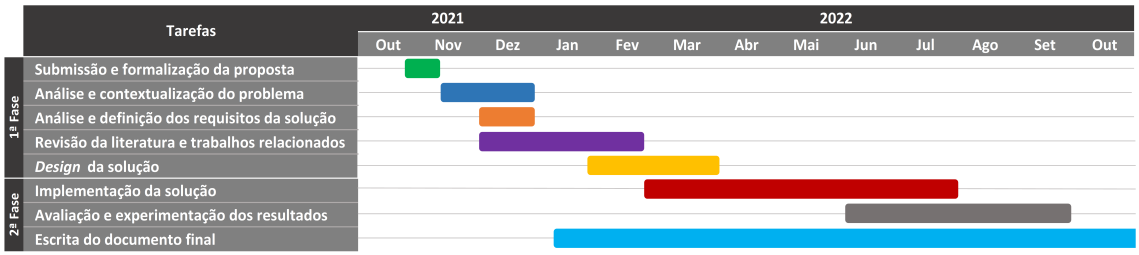


Figura 1.3. Plano de trabalhos estabelecido juntamente com o supervisor da empresa e acompanhado com reuniões semanais.

1.8. Estrutura da Tese

No primeiro capítulo, intitulado **Introdução**, é descrito o âmbito e o contexto desta dissertação, qual o problema que se pretende solucionar, os objetivos que se desejam alcançar e a metodologia de investigação adotada. São apresentadas as questões e hipóteses de investigação e o plano de trabalhos.

No segundo capítulo, intitulado **Estado da Arte**, é apresentada uma revisão da literatura com foco nos conceitos teóricos inerentes a esta dissertação e as principais tecnologias associadas.

No terceiro capítulo, intitulado **Análise**, é apresentado o valor deste projeto, do ponto de vista do negócio e do utilizador, definidos os requisitos funcionais e não funcionais da solução e apresentados os casos de uso do sistema.

No quarto capítulo, intitulado **Desenho da Solução**, é apresentado o *design* da solução e justificadas as decisões tomadas.

No quinto capítulo, intitulado **Implementação da Solução**, são apresentados todos os detalhes de implementação e tecnologias utilizadas, relacionando-os com os casos de uso definidos no terceiro capítulo.

No sexto capítulo, intitulado **Experimentação e Avaliação**, são avaliados os resultados obtidos, enumerando os testes funcionais realizados bem como o comportamento da solução perante os mesmos.

No capítulo final, intitulado **Conclusões**, são apresentadas as conclusões relacionadas com o projeto na sua globalidade, incluindo os objetivos que foram concretizados e o trabalho futuro a ser desenvolvido.

Capítulo 2

Estado da Arte

A evolução tecnológica dos últimos anos conduziu a uma procura crescente por *softwares*, devendo estes, estarem sempre disponíveis e acessíveis aos seus utilizadores. Sommerville [5] destaca que para a sociedade moderna funcionar adequadamente é necessário existir sistemas de *software* complexos, pois estes são essenciais para serviços como energia, comunicação e transportes.

O mercado cada vez mais competitivo tem colocado as organizações em constante desenvolvimento de novas funcionalidades, disponíveis para diferentes plataformas e dispositivos. Isto não só aumenta a exigência e complexidade do processo de desenvolvimento de *software* como também aumenta as exigências relacionadas com a infraestrutura, dificultando a aplicação de novas versões de forma eficaz em produção [6]. Além disso, também as diferentes competências e experiências das equipas de desenvolvimento (Dev[1]) e de operações (Ops[2]) contribuem para uma evolução do *software* mais atribulada [7]. Enquanto as equipas de desenvolvimento procuram a evolução do *software* voltada para o lançamento de novas funcionalidades, trazendo alguma instabilidade, as equipas de operações procuram alcançar a estabilidade evitando mudanças que possam trazer risco (figura 2.1), originando assim conflitos entre elas [8].

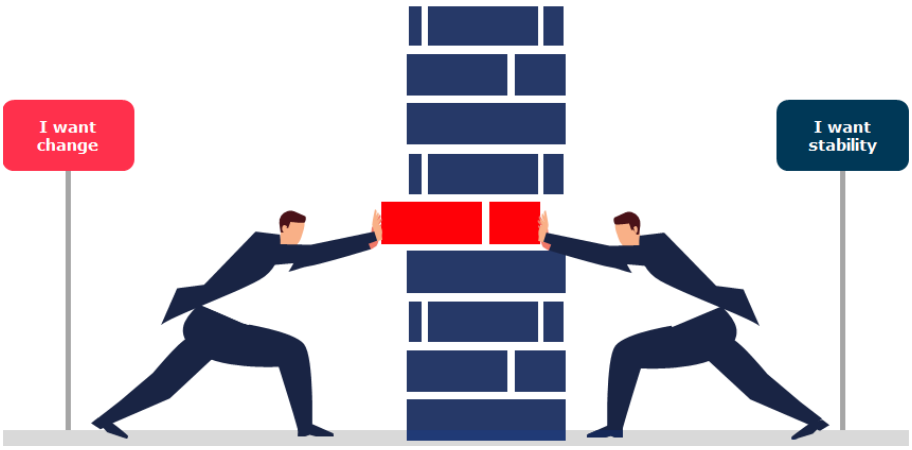


Figura 2.1. Ilustração típica associada ao DevOps[3], conhecida como *Wall of Confusion* [9].

O DevOps consiste na mistura de desenvolvimento e operações, e procura resolver os conflitos entre as equipas através da união das pessoas, processos e tecnologia para entregar

[1] do inglês, *Development.*
[2] do inglês, *Operations.*
[3] `http://technopedia.info/devops/devops-wall-of-confusion-explained/`

valor continuamente aos clientes[4]. Com a adoção de uma cultura DevOps, o desenvolvimento do *software* baseia-se no processo de integração e entrega contínua, oferecendo visibilidade e *feedback* constante a todos os envolvidos [10]. Assim, as equipas ganham a capacidade de responder melhor às necessidades dos clientes, de disponibilizar novas funcionalidades com maior velocidade, de aumentar a confiança nas aplicações que criam e de atingir os objetivos do negócio mais depressa [11].

A Amazon definiu DevOps como uma combinação de filosofias culturais, práticas e ferramentas que visam aumentar a capacidade de uma organização em entregar *software*, contribuindo assim para um melhor atendimento aos clientes e maior competitividade[5].

O processo clássico de desenvolvimento de *software* [5], normalmente conhecido por Modelo em Cascata[6], é caracterizado pelo encadeamento entre as várias fases conhecidas como:

- **Definição dos requisitos**: levantamento dos requisitos, objetivos e características do sistema junto dos seus utilizadores;
- ***Design* do *software***: definição da arquitetura do sistema e tudo aquilo que será necessário para o seu funcionamento;
- **Implementação e testes unitários**: o sistema é desenvolvido e testado recorrendo a testes unitários;
- **Integração e testes**: integração e testes ao sistema como um todo;
- **Operação e manutenção**: instalação do sistema e correção de todos os erros que possam ocorrer e que não foram descobertos nas etapas anteriores.

Apesar da cultura DevOps ser diferente do modelo clássico de desenvolvimento do *software* no que respeita ao processo e dinâmica entre as equipas de desenvolvimento e operações, algumas das atividades são semelhantes. Destacam-se quatro fases principais[4]:

- **Planear**: idealizar, definir e descrever as funcionalidades e capacidades dos *softwares*;
- **Desenvolver**: inclui todos os aspetos de programação (*i.e.,* escrita, teste, revisão e integração do código) bem como, a compilação do código;
- **Distribuir**: processo de implantar os *softwares* em ambientes de produção de forma consistente e fiável. Esta fase também inclui implementar e configurar a infraestrutura base;
- **Operar**: esta fase envolve a manutenção, a monitorização e a resolução de problemas das aplicações em ambientes de produção.

2.1. *Continuous Integration*, *Delivery* e *Deployment*

No desenvolvimento moderno de *software*, o objetivo é que várias pessoas contribuam simultaneamente para a mesma aplicação, integrando os desenvolvimentos num único código-fonte. Este processo pode gerar alguns conflitos, uma vez que um desenvolvedor pode estar a trabalhar no mesmo recurso que outras pessoas da organização. Para combater este problema surgiu o conceito de CI[7] (figura 2.2), que procura garantir que mudanças

[4] `https://azure.microsoft.com/en-us/overview/what-is-devops/#devops-overview`
[5] `https://aws.amazon.com/pt/devops/what-is-devops/`
[6] do inglês, *Waterfall Model*.
[7] `https://aws.amazon.com/pt/devops/continuous-integration/`

no código de uma aplicação são desenvolvidas, testadas e consolidadas regularmente num repositório partilhado[8]. A CI permite que as organizações tenham ciclos de lançamento do *software* mais curtos e frequentes, melhorando a qualidade e aumentando a produtividade das equipas [12].

Aliado à CI surge o conceito de CDE (figura 2.2) que procura automatizar o lançamento do código-fonte desenvolvido na etapa de CI. O objetivo da CDE é garantir que o código de uma aplicação está sempre pronto para implantação num ambiente de produção [12]. Para garantir a eficácia deste conceito é importante que a CI esteja bem definida e devidamente integrada na *pipeline* de desenvolvimento.

Como complemento à CDE existe ainda outro conceito, o *Continuous Deployment* (CD) (figura 2.2), que corresponde ao lançamento automático das mudanças feitas por um desenvolvedor, permitindo que os clientes tenham acesso rápido a novas versões do *software* [13].

Figura 2.2. As três principais fases do processo de desenvolvimento de *software*: CI, CDE e CD[9].

2.1.1. Máquinas Virtuais

No passado, as organizações executavam as aplicações em servidores físicos, não existindo forma de limitar os recursos utilizados por cada aplicação. Se várias aplicações estivessem em execução no mesmo servidor físico podia acontecer de uma aplicação ocupar maior parte dos recursos disponíveis, afetando o desempenho das restantes aplicações[10]. Além disso, como as aplicações executavam todas no mesmo ambiente sem qualquer tipo de isolamento, podia ocorrer a atualização involuntária de dependências utilizadas por outras aplicações [14]. Uma possível solução para estes problemas seria colocar aplicações diferentes em execução em servidores físicos diferentes, porém, além de aumentar consideravelmente os recursos necessários, é também mais caro e difícil de manter para as organizações.

Para solucionar este problema, a virtualização de aplicações com recurso a *Máquinas Virtuais* (VMs) apresenta-se como uma boa opção. Uma *Máquina Virtual* (VM) é um ambiente virtual que funciona como um sistema computacional com o seu próprio *Central Processing Unit* (CPU), memória, interface de rede e armazenamento (figura 2.3A). Este sistema virtual é criado a partir do *hardware* e de um *software* chamado *hypervisor*, responsável por criar a VM e alocar os recursos computacionais necessários. A máquina física onde o *hypervisor* está instalado e a VM é criada denomina-se *host*. As VMs que usam os recursos da máquina *host* são chamadas de *guest*[11].

A virtualização de aplicações com recurso a VMs permite que várias VMs estejam hospedadas num único servidor físico, garantindo que as aplicações são executadas de forma isolada, oferecendo maior segurança [14]. Assim, as VMs apresentam-se como uma boa opção para

[8] https://www.redhat.com/en/topics/devops/what-is-ci-cd
[9] https://medium.com/tilicholabs/what-is-ci-cd-c7c047b80e6b
[10] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
[11] https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine

testar novas aplicações ou configurar ambientes de produção. Este método de virtualização permite um melhor aproveitamento dos recursos e oferece maior escalabilidade, dado que as VMs podem ser adicionadas ou atualizadas mais facilmente, reduzindo assim os custos de *hardware*. As VMs garantem mais opções de recuperação de desastres, pois permitem implementar mecanismos de tolerância a falhas e redundância.

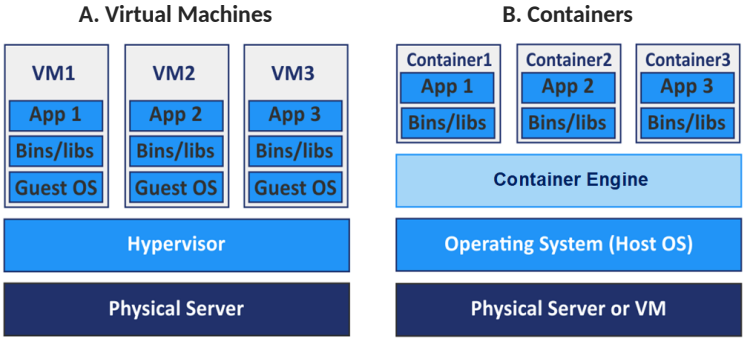


Figura 2.3. Comparação entre (A) máquinas virtuais e (B) *containers*[12].

2.1.2. *Containers*

Os *containers* são uma forma de virtualização ao nível do sistema operativo e revolucionaram a forma como se desenvolve, distribui e executa o *software*. São o resultado do encapsulamento de uma aplicação juntamente com as suas dependências. Assim, os desenvolvedores podem criar *software* localmente sabendo que será sempre executado de forma idêntica, independentemente do ambiente de execução. As equipas de operações podem assim concentrar-se na rede, recursos, tempo de atividade e despender menos tempo a configurar ambientes e respetivas dependências [15].

À primeira vista os *containers* (figura 2.3B) podem parecer ser apenas uma forma leve de VMs, uma vez que também garantem o isolamento do sistema operativo. No entanto existem casos de uso bastantes difíceis ou impossíveis de implementar com VMs que se tornam possíveis com *containers*. Os principais benefícios dos *containers* comparativamente às VMs (figura 2.3) são:

- Criação e implantação de aplicações de forma mais fácil e ágil;
- Maior portabilidade, eliminando problemas causados por mudanças subtis nos ambientes de execução;
- Os *containers* partilham os recursos do *host* tornando-os mais rápidos e eficientes;
- Podem ser parados e iniciados rapidamente;
- São executados em diferentes sistemas operativos e compatíveis com as principais *clouds* públicas garantindo assim portabilidade na sua distribuição;
- Permitem a exibição de informações e métricas de nível aplicacional;
- A natureza leve dos *containers* permite que seja possível executar um número elevado de *containers* no mesmo *host*;
- Com *containers* as aplicações são divididas em partes menores e independentes, ao invés de um único monolítico executado numa máquina de propósito único.

[12] `https://www.nakivo.com/blog/docker-vs-kubernetes/`

2.1.2.1. Docker

Os sistemas Unix possuem o comando `chroot` que fornece uma forma simples de isolamento no sistema de ficheiros. Em 1998, o FreeBSD apresentou o utilitário Jail, que estendeu o comando `chroot` também aos processos do sistema [15]. Anos mais tarde, em 2001, o Solaris Zones apresentou uma tecnologia de conteinerização completa, mas que estava limitada ao Solaris OS. No mesmo ano, a SWsoft (atualmente Parallels) lançou a tecnologia Virtuozzo para Linux, chegando inclusive a disponibilizar o código-fonte. Foi então que a Google iniciou o desenvolvimento de `CGroups`, um recurso do *kernel* Linux que limita e isola o uso de recursos de uma coleção de processos. Em 2008, surgiu o projeto *Linux Containers* (LXC), que reúne `CGroups`, `namespaces` e a tecnologia `chroot` para fornecer uma solução completa de conteinerização [15].

Em 2013, surgiu o Docker, uma tecnologia baseada em LXC, mas que permitiu que a abordagem de conteinerização começasse a ser amplamente utilizada pela indústria [15]. Quando surgiu, o Docker era um monolítico, isto é, todos os serviços eram altamente acoplados e tudo era executado num único processo, mas com o passar dos anos surgiu a necessidade de separar a tecnologia em várias partes[13]. Assim que a empresa decidiu tornar o código-fonte do Docker *open-source*[14] uma grande comunidade decidiu começar a contribuir para a correção de *bugs* e acréscimo de melhorias. A rápida ascensão do Docker levou a que este se tornasse um padrão na abordagem de *containers*, e fez surgir a *Open Container Initiative* (OCI) [15].

A OCI é um projeto colaborativo da *Linux Foundation* para estabelecer padrões comuns para o desenvolvimento e implementação de *containers*. Foi apresentada pela primeira vez como *Open Container Project* (OCP) na DockerCon, em junho de 2015. Atualmente, conta com o apoio de várias empresas proeminentes no mercado tecnológico, no entanto, o projeto permanecerá independente de qualquer organização comercial específica[15].

Em 2016, após lançamento da OCI, na nota de lançamento da versão `1.11`[16] do Docker, a empresa informa que a sua instalação passava a ser composta por quatro binários: `docker`, `docker-containerd`, `docker-containerd-shim` e `docker-runc`. Ter uma ferramenta separada em várias partes específicas faz com que várias pessoas se especializem em cada uma delas, com o objetivo de melhorar a tecnologia. Na mesma nota, a empresa apresenta o novo modelo de funcionamento da tecnologia, composto por:

- ***Docker Engine:*** é responsável por receber os comandos do utilizador através da *Command Line Interface* (CLI) e enviar essas instruções ao *container runtime*;
- ***Container Runtime:*** interface compatível com as especificações OCI capaz de executar e extrair qualquer imagem de *container* que seja também compatível com a OCI;
- ***RunC:*** lida com toda a gestão e criação dos *containers* e atualmente é desenvolvido pela OCI.

Com a criação da OCI e com este modelo de funcionamento da tecnologia Docker começam a surgir vários *container runtimes*, como é o caso do containerd[17] e do CRI-O[18].

[13] `https://blog.lsantos.dev/entendendo-runtimes-de-containers/`
[14] um *software open-source* pode ser visualizado, modificado e distribuído por qualquer pessoa.
[15] `https://www.techtarget.com/searchitoperations/definition/Open-Container-Initiative`
[16] `https://docs.docker.com/engine/release-notes/prior-releases/#1110-2016-04-13`
[17] `https://containerd.io/`
[18] `https://cri-o.io/`

I. Imagens Docker

Uma imagem Docker é um pacote binário que encapsula todos os ficheiros e dependências necessárias para executar um determinado programa dentro de um *container*[19]. As imagens podem ser criadas de duas formas, através de um *snapshot*[20] de um *container* em execução ou através de um ficheiro de *build* chamado `Dockerfile` [16]. Uma imagem pode ainda ter por base uma outra imagem já existente[21].

II. Dockerfile

Um `Dockerfile` é um ficheiro de texto que contém um conjunto de etapas que permitem automatizar a criação de uma imagem de *container* Docker. Pode ser visto como uma *«receita»* de como construir a imagem [17]. No exemplo a seguir (código 2.1) é mostrada a criação de uma imagem através de um `Dockerfile`.

```
1 FROM node:10
2
3 WORKDIR /usr/src/app
4 COPY package*.json ./
5 RUN npm install
6 COPY .  .
7 CMD [ "npm","start" ]
```

Código 2.1. Dockerfile responsável pela criação de uma imagem de *container*.

Neste caso, através da instrução `FROM` define-se que a imagem a construir é baseada numa versão disponível no Docker Hub (`node:10`). De seguida é especificado o diretório (`/usr/src/app`) onde serão executados os comandos, copiados os ficheiros para a imagem (`COPY`) e instaladas as dependências do `npm` necessárias para a aplicação (`RUN`). Por último, com a instrução `CMD` define-se qual o comando (`npm start`) que será executado assim que o *container* iniciar.

Depois de construído o ficheiro `Dockerfile` que define a imagem do *container* esta pode ser compilada através do comando `docker build` (código 2.2).

```
1 $ docker build -t tmdei/app .
```

Código 2.2. Compilação da imagem `tmdei/app` com base num Dockerfile previamente criado.

Um *container* é uma instanciação de uma imagem, por isso depois de compilada a imagem é preciso inicializar o *container* conforme demonstra o código 2.3.

```
1 $ docker run -d -p 8000:8000 tmdei/app
```

Código 2.3. Execução de um *container* baseado na imagem `tmdei/app` previamente criada.

III. Repositório de Imagens: *Registry*

As imagens depois de criadas precisam de ser armazenadas num local partilhado para que possam ser distribuídas pela comunidade, a este local chamamos *registry*. O Docker disponibiliza atualmente um *registry* público, o Docker Hub. Também podem ser criados *registries* privados permitindo que as organizações tenham os seus próprios locais de armazenamento.

[19] `https://kubernetes.io/docs/concepts/containers/images/`
[20] registo de um estado do sistema, aplicação ou ficheiro num determinado ponto no tempo.
[21] `https://docs.docker.com/get-started/overview/`

2.2. Arquiteturas de Micro Serviços

A conteinerização de aplicações impulsionou a transformação arquitetural de alguns sistemas que até então eram desenhados segundo uma arquitetura monolítica (figura 2.4A) onde todos os processos eram altamente acoplados e compostos por um único serviço[22]. As arquiteturas monolíticas são mais simples de implementar e desenhar, e por isso mais baratas numa primeira fase de desenvolvimento, porém apresentam imensas desvantagens que podem aumentar o seu custo, nomeadamente:

- A escalabilidade e a flexibilidade estão comprometidas, uma vez que sempre que é necessário criar uma nova instância do sistema todo ele precisa de ser replicado [18];
- Existe uma alta dependência de componentes de código, fazendo com que a inclusão ou manutenção de componentes do sistema possa causar inconsistências ou comportamentos inesperados [19];
- A falha de um único componente compromete todo o sistema dada a dependência entre os seus componentes [20];
- Pouca flexibilidade e liberdade na escolha tecnológica [20], o que significa que se uma nova funcionalidade exigir uma tecnologia diferente todo o sistema precisa de ser alterado ou a arquitetura do sistema precisa de ser revista.

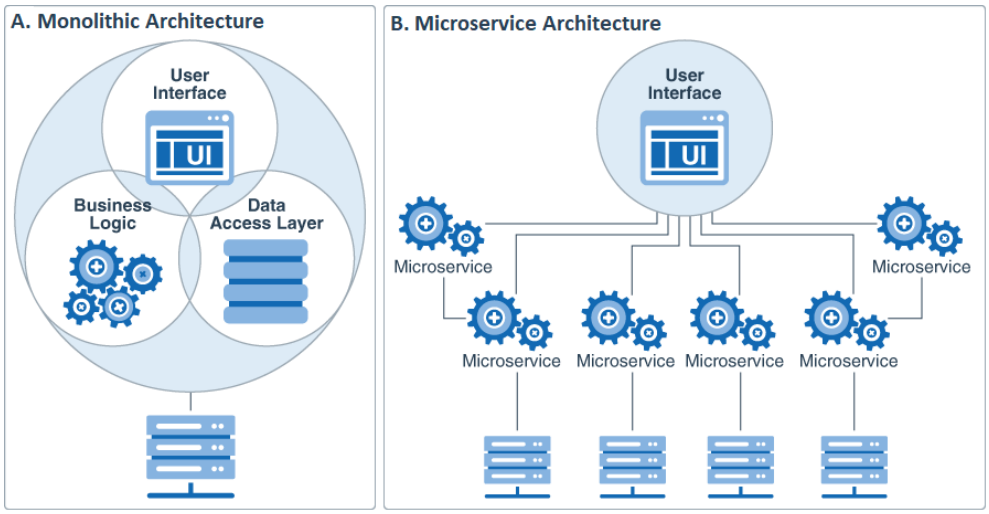


Figura 2.4. Comparação entre (A) arquiteturas monolíticas e (B) arquiteturas de micro serviços[22].

Por oposição às arquiteturas monolíticas temos as arquiteturas baseadas em micro serviços (figura 2.4B) que têm ganho cada vez mais destaque, e embora sejam mais caras e complexas de implementar numa primeira fase, estas apresentam bastantes vantagens quando comparadas com arquiteturas monolíticas[23]. Este tipo de arquitetura é composta por vários micro serviços, onde cada um possui o seu próprio processo e é capaz de comunicar com os restantes através de APIs bem definidas. Cada serviço é desenvolvido em torno de um conjunto de regras de negócio específicas e é implementado de forma independente[22].

Chen *et al.* [19] afirmam que as arquiteturas de micro serviços facilitam os processos de manutenção, reutilização, escalabilidade, disponibilidade e implantação automática.

[22] https://azure.microsoft.com/en-us/solutions/microservice-applications/#overview
[23] https://docs.oracle.com/pt-br/solutions/learn-architect-microservice/index.html

Desta forma, alguns dos desafios e dificuldades das arquiteturas monolíticas são quebrados, nomeadamente:

- A escalabilidade e a flexibilidade do sistema tornam-se muito mais simples uma vez que os serviços podem ser replicados de forma independente, ao invés da replicação de todo o sistema [18];
- A manutenção e a evolução dos serviços é mais fácil e estável, uma vez que os desenvolvedores vão trabalhar com códigos que são executados de forma independente;
- A falha de um serviço não implica a falha de todo o sistema [20];
- A independência entre os serviços permite que cada um possa ser desenvolvido com recurso a uma tecnologia diferente, tornando a arquitetura mais flexível [20].

Estas características reduzem os custos de desenvolvimento e manutenção, sendo por isso um sistema mais sustentável ao longo do tempo. Apesar das vantagens que as arquiteturas baseadas em micro serviços apresentam, é de notar os diversos desafios para as organizações que optem por adaptar os seus sistemas[24]. Para além das alterações que as aplicações terão de sofrer, também o modo como as equipas operam deve ser ajustado.

2.3. *Software as a Service*

O surgimento de arquiteturas de micro serviços e os utilizadores cada vez mais exigentes, juntamente com as melhorias da velocidade de conexão à internet e a capacidade de processamento de dados na *cloud* permitiram evoluir a forma como o *software* é distribuído, surgindo o conceito de SaaS. A diferença é simples, ao invés de o *software* ser comprado e instalado nos computadores dos utilizadores, agora encontra-se disponível na internet [21].

A ideia por trás do SaaS não é fundamentalmente nova, uma abordagem semelhante chamada *Application Service Provider* (ASP) já estava a ser adotada em 1990 [22]. O SaaS pode ser visto como uma extensão do ASP, no entanto, o ASP é um modelo que possibilita o acesso a aplicações e *software* através de uma infraestrutura de rede e o SaaS é um modelo de licenciamento e entrega de *software* através da internet.

A disponibilização do *software* como um serviço pode ser visto como uma prioridade, em particular nas áreas de *software* de *Customer Relationship Management* (CRM) e *Enterprise Resource Planning* (ERP) [22]. O SaaS apresenta várias vantagens, nomeadamente:

- Acesso ao *software* de forma instantânea sem a necessidade de instalações;
- Redução dos custos com *hardware*, uma vez que o processamento é disponibilizado pela *cloud* a ser utilizada;
- Custos apenas relacionados com os recursos e tráfego utilizados;
- Atualizações oportunas e constantes;
- Compatibilidade entre dispositivos;
- Flexibilidade e escalabilidade.

Apesar das novas aplicações e sistemas serem projetados desde o início para disponibilização como um serviço, existem organizações que procuram transformar e adaptar as arquiteturas dos seus sistemas atuais. A transformação arquitetural de um sistema é algo trabalhoso

[24] `https://www.redhat.com/en/topics/microservices/what-are-microservices`

e demorado e que precisa de ser pensado tendo em conta as desvantagens que o SaaS apresenta:

- Necessidade de investimento na formação dos colaboradores e em equipas de segurança;
- Falta de controlo no *hardware*, uma vez que as tecnologias que permitem implementar um modelo de negócio baseado em SaaS oferecem alguma abstração relativamente aos servidores onde os serviços estão em execução;
- Gastos periódicos relacionados com operação, suporte e manutenção da solução.

As novas tecnologias de desenvolvimento de aplicações *web* que garantem que as páginas não são renderizadas a cada acesso[25] e as arquiteturas de *software* baseadas em micro serviços têm contribuído para expansão e eficiência do SaaS [22].

2.4. *Infrastructure as Code*

A gestão e implementação de servidores com alta confiabilidade e disponibilidade sempre foi um procedimento longo e desafiador, especialmente no passado em que era um processo maioritariamente manual [6]. Os servidores eram instalados fisicamente e os componentes de *hardware* configurados manualmente de acordo com os requisitos dos sistemas operativos e das aplicações em execução. Por isso, a instalação, configuração e manutenção da infraestrutura é um processo caro e demorado e com grande probabilidade de erro, sendo normalmente necessárias equipas especializadas para realizar o trabalho. Também os *datacenter*[26] precisam de manutenção, aumentando os custos operacionais, como eletricidade, sistemas de refrigeração e segurança.

A introdução da computação na *cloud* juntamente com a evolução das tecnologias de virtualização surgem como uma solução promissora para muitos destes problemas. A combinação destas tecnologias oferece uma forma mais eficiente de gestão de configurações, que resolve os problemas de escalabilidade e agilidade da infraestrutura. No entanto, muitas organizações enfrentam ainda problemas com a inconsistência destes sistemas, pois tendem a usar processos e ferramentas antigas que não conseguem acompanhar a evolução exigida por estas tecnologias [23].

O conceito de IaC teve início em 2006, quando a *Amazon Web Services* (AWS) lançou o *Elastic Compute Cloud* (EC2) [24]. A partir dessa altura, qualquer aplicação *web* pôde ser implementada em questão de semanas, permitindo que pequenas empresas crescessem rapidamente. Como estas equipas eram pequenas e necessitavam de administrar ambientes complexos, surgiram as primeiras ferramentas de gestão de configuração.

Morris [25] afirma que a computação na *cloud* veio criar duas *«eras»*: a *«era do ferro»* e a atual *«era da cloud»*. Na *«era do ferro»* os sistemas estavam instalados no *hardware* e era necessário trabalho manual para provisionar e gerir a infraestrutura. Na *«era da cloud»* deixa de existir a associação dos sistemas ao *hardware*, e grande parte do trabalho de provisionamento e manutenção da infraestrutura pode ser realizado através de *software*, com um grande ganho de tempo. Segundo Geerling [6], foi a virtualização de servidores que inaugurou a gestão de infraestrutura em larga escala. Ao invés de realizar diversos trabalhos de forma manual, os administradores podem agora criar novos servidores automaticamente e com o mínimo de intervenção.

[25] `https://agilie.com/blog/single-page-apps-vs-multiple-page-web-apps`
[26] local onde estão concentrados os sistemas computacionais de uma empresa ou organização.

A ideia por trás do conceito de IaC é escrever e executar código para definir, implantar, atualizar e destruir a infraestrutura. Isto representa uma mudança importante na mentalidade, agora tudo passa a ser tratado como *software*, mesmo os aspetos que representam *hardware*. Na verdade, esta é uma das filosofias associadas à prática de DevOps [26].

A possibilidade de escrever código para definir a infraestrutura é um mecanismo poderoso. No entanto, implica que os desenvolvedores aprendam novas linguagens de programação e ferramentas. Apesar deste investimento inicial, a conversão das práticas manuais de criação da infraestrutura em código permite obter melhorias drásticas na capacidade de entregar *software*.

Em suma, as principais vantagens da IaC são as seguintes [26]:

- Todo o processo de implantação pode ser automatizado e os desenvolvedores poderão iniciar as suas implantações sempre que necessário;
- Um processo de implantação automático, será significativamente mais rápido devido ao processamento dos computadores. A automatização deste processo conduz à consistência e segurança dos sistemas e garante a sua repetibilidade;
- O estado da infraestrutura pode ser representado em ficheiros de documentação, permitindo que qualquer pessoa da organização compreenda como as coisas funcionam;
- Como a infraestrutura fica definida em código pode também ficar armazenada nos sistemas de controlo de versões, o que significa que todo o histórico da infraestrutura fica registado. Em caso de falha, existe a possibilidade de reverter as alterações para um estado conhecido e de correto funcionamento;
- O processo de revisão de código que normalmente é realizado no ciclo de vida do desenvolvimento de *software* também se aplica ao código da infraestrutura;
- O código utilizado para definir a infraestrutura pode ser empacotado em módulos reutilizáveis, permitindo que novas implantações não necessitem de ser iniciadas do zero.

A IaC apresenta-se como uma melhor alternativa, uma vez que os computadores e os desenvolvedores fazem o que melhor sabem: automação e codificação, respetivamente.

Grandes organizações, como a Amazon, Netflix, Google e Meta, utilizam técnicas de IaC para disponibilizar os seus serviços, o que nos permite comprovar a eficiência e segurança desta prática. Nestas empresas os sistemas não são apenas considerados críticos para o negócio, eles são o próprio negócio, não havendo por isso tolerância para falhas ou indisponibilidades [25].

Apesar das várias vantagens da IaC, também possui algumas desvantagens, tais como:

- Os desenvolvedores precisam de compreender na totalidade os *scripts* de IaC, independentemente da linguagem utilizada. Caso os *scripts* não sejam devidamente compreendidos, a integração e o dimensionamento rápido podem ser mais complexos;
- Embora as técnicas de IaC forneçam mecanismos ótimos de controlar e monitorizar as alterações na infraestrutura, a gestão de todas as configurações e permissões pode tornar-se complexo à medida que o número de pessoas envolvidas aumenta;
- As tecnologias de IaC estão muitas vezes dependentes de *providers* que precisam de cobrir totalmente os recursos que as *clouds* oferecem. Como consequência, os próprios

providers ficam por vezes atrasados no lançamento de novos recursos. O impacto disto é significativo, uma vez que, ou a funcionalidade em questão é estendida por conta da organização ou é necessário esperar pelo lançamento da nova funcionalidade.

2.5. MAM4PRO

O MAM4PRO é um sistema de *ingest* de nível empresarial que oferece uma ampla variedade de operações com *media*, nomeadamente descodificação[27], codificação[28] e transcodificação. Soluções como o MAM4PRO são necessárias para diferentes tipos de clientes que precisam de manipular, uniformizar e transformar a *media* num produto final para distribuir pelos utilizadores. O MAM4PRO oferece um elevado número de transformações e integrações que permitem trabalhar com os mais diversos formatos de *media*. Adicionalmente, o seu constante desenvolvimento garante que está sempre atualizado em relação à compatibilidade de formatos e à qualidade dos recursos disponíveis.

2.5.1. Arquitetura de Alto Nível

O MAM4PRO é baseado numa arquitetura de micro serviços, onde cada serviço possui o seu próprio processo e consegue comunicar com os restantes. Esta abordagem arquitetural permite ter diferentes instalações do MAM4PRO considerando as necessidades do cliente (*i.e.,* pode ser realizada uma instalação completa ou parcial do *software*). Esta flexibilidade ajuda a reduzir o tamanho do *software* e permite instalações mais simples e rápidas.

A arquitetura do sistema (figura A1) é composta por três componentes principais descritos nos seguintes parágrafos: `mCore`, *User Interface* (UI) e *media services*.

2.5.1.1. mCore

O `mCore` é o serviço central do sistema e é o único capaz de ordenar aos *media services* para iniciar ou parar as suas tarefas. Efetivamente, o `mCore` atua como um intermediário que está conectado a cada serviço. Após receber os pedidos *Hypertext Transfer Protocol* (HTTP) da UI, o `mCore` solicita o processamento da *media* aos respetivos serviços. A comunicação entre o `mCore` e os restantes serviços é realizada através de uma fila de mensagens (*message queue*) implementada usando RabbitMQ[29]. Na prática, os serviços estão conectados entre si através do `mCore`. Adicionalmente, o `mCore` é também responsável pelo estado do sistema e por estabelecer a comunicação com a base de dados.

2.5.1.2. *User Interface*

A UI é acessível através do *browser* e procura oferecer um experiência de utilização simples e intuitiva para o utilizador final. Existem páginas acessíveis através de um menu lateral que permitem configurar todo o sistema, *widgets* e *dashboards* que permitem ao utilizador personalizar o *software* de acordo com as suas necessidades (figura A2).

2.5.1.3. *Media Services*

Este componente é composto por vários micro serviços responsáveis por processar e transformar a *media*.

[27] do inglês *decode*, descompactação de vídeo.
[28] do inglês *encode*, preparação do vídeo para saída.
[29] `https://www.rabbitmq.com/`

Os micro serviços que o compõe são:

- **`hOperation`**: responsável pelo *ingest* de ficheiros ou *streams* para formatos de *media* profissionais;
- **`mExtractor`**: responsável por extrair os metadados de ficheiros de *media* existentes num armazenamento local ou na *cloud* para serem indexados pelo MAM4PRO;
- **`rPlayerSdi`**: responsável por reproduzir um ficheiro ou *stream* de *media* produzindo uma transmissão ao vivo do conteúdo em *stream* ou em sinal de vídeo digital (*Serial Digital Interface* (SDI)/*High Definition Multimedia Interface* (HDMI));
- **`rPlayerWeb`**: responsável por ler e transformar ficheiros ou *streams* de *media* profissional de forma a serem visualizados no *browser*. Permite procurar dados em qualquer parte do vídeo;
- **`rPlayout`**: responsável pela automação de instâncias de `rPlayerSdi` para a reprodução de *playlists*[30] complexas em ambientes de distribuição. Os dados são recebidos em tempo real e por isso não é permitido avançar ou retroceder no vídeo;
- **`sCapture`**: responsável pela captura, monitorização e gravação de transmissões ao vivo em *stream* ou em sinal de vídeo digital (SDI/HDMI).

2.5.2. Conteinerização do Sistema

Tal como referido na secção 1.2, o MAM4PRO atualmente é comercializado e instalado em servidores de acordo com as necessidades do cliente, no entanto a MOG Technologies já suporta a implantação do MAM4PRO utilizando *containers* Docker. Cada serviço está disponível como um *container* independente, contendo o seu próprio processo `mAnnouncer` para permitir que o serviço se anuncie para os outros MAM4PRO existentes na rede.

A figura A3 ilustra uma possível implantação do MAM4PRO utilizando *containers*. É de notar que o número de imagens criadas (figura A3D) não é igual ao número de *containers* (figura A3E) uma vez que, algumas delas servem apenas de imagem base para construção de outras (*cf.,* apêndice A.4).

2.6. Metodologias de Investigação

As metodologias de investigação são adotadas pelos investigadores para garantir resultados válidos e confiáveis [27], auxiliando no desenho de um plano de ação, estratégia, processo ou *design* que sustente a escolha dos métodos e implementações utilizadas. Para garantir que uma pesquisa é considerada válida e relevante, tanto no meio académico como na sociedade em geral, esta deve ser desenvolvida com rigor e passível de debate e verificação [28].

Dependendo do âmbito e objetivo da investigação em curso, existem diversas metodologias que podem ser aplicadas [29][30]. No caso desta dissertação foram consideradas duas metodologias: DSR [31] e *Action Research* (AR) [32].

2.6.1. *Design Science Research*

A metodologia DSR cria e avalia artefactos de TI destinados a resolver problemas organizacionais identificados. É vista como uma atividade de pesquisa que constrói, inventa

[30] lista de ficheiros de vídeo/áudio que podem ser reproduzidos num *player* sequencialmente numa ordem aleatória.

ou inova artefactos, para a solução de problemas ou obtenção de melhorias [33]. A DSR também é definida como uma investigação sistemática cujo objetivo é o conhecimento, configuração, incorporação, estruturação, e compreensão do propósito, valor e significado dos sistemas feitos pelo Homem [31].

Hevner *et al.* [34], propuseram um conjunto de diretrizes para a metodologia DSR, que visam ajudar os investigadores a compreender os requisitos para uma investigação científica eficaz. Embora a metodologia seja composta por sete diretrizes distintas, nem todas precisam de ser aplicadas e a ordem pela qual são aplicadas pode diferir:

- **D. 1 -** *Design* como um artefacto: a pesquisa e investigação deve produzir um artefacto na forma de uma construção, uma representação ou uma técnica;
- **D. 2 -** A relevância para o problema: o objetivo básico da pesquisa é desenvolver soluções baseadas em tecnologia para resolver problemas de negócios importantes;
- **D. 3 -** A avaliação do *design*: a utilidade, a qualidade e a eficácia do artefacto devem ser demonstradas rigorosamente através de métodos de avaliação bem executados;
- **D. 4 -** Contribuições: a pesquisa deve resultar em contribuições claras e verificáveis;
- **D. 5 -** Rigor de pesquisa: a pesquisa depende da aplicação de métodos rigorosos tanto na construção como na avaliação do artefacto;
- **D. 6 -** *Design* como um processo de pesquisa: análise do estado da arte e comparação com soluções existentes relacionadas com o problema definido e estudo das tecnologias relacionadas;
- **D. 7:** Comunicação de pesquisa: documentação do trabalho e disseminação dos resultados da investigação.

2.6.2. *Action Research*

Na metodologia AR, os investigadores procuram resolver um problema do mundo real, e em seguida desenhar um plano de ação iterativo de ocorrência simultânea de *«agir»* e *«pesquisar»*, com estratégias que visam alcançar melhores práticas e resultados [35]. Enquanto a maioria dos métodos de investigação empírica tentam observar o mundo como ele está atualmente, no caso da metodologia AR, os investigadores visam intervir nas situações estudadas com o propósito explícito de melhorá-las. Normalmente, esta metodologia pressupõe o envolvimento de um cliente, portanto este processo é dependente do contexto e não se limita apenas ao *design* e desenvolvimento de artefactos tecnológicos[31].

Destacam-se as seguintes características desta metodologia [36]:

- **Participativa e colaborativa**: os investigadores trabalham em conjunto na concretização de um projeto;
- **Situacional**: é realizado o diagnóstico de um problema num contexto específico e tenta-se resolvê-lo nesse mesmo contexto;
- **Cíclica**: a investigação envolve um conjunto de ciclos, nos quais as descobertas iniciais geram possibilidades de mudança, que são implementadas e avaliadas como introdução do ciclo seguinte;

[31] https://medium.com/@3D_Ideation/simplifying-design-science-research-action-research-and-design-research-bf564959402b

- **Autoavaliativa**: modificações são continuamente avaliadas e monitorizadas, numa perspetiva de flexibilidade e adaptabilidade, com vista a produzir novos conhecimentos.

Existem diversas formas de descrever a metodologia AR, que muitas vezes difere no número de fases [32]. Dickens e Watkins [37] definiram quatro principais fases para esta metodologia (figura 2.5).

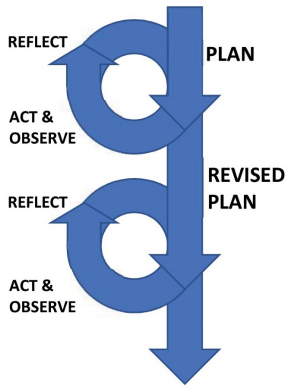


Figura 2.5. Ciclo da metodologia AR: refletir, planear, agir e observar.

Depois de terminada a primeira iteração, os investigadores permanecem neste ciclo até esgotarem o problema que identificaram inicialmente. Um ciclo pode ser suficiente para resolver o problema, no entanto, o mais provável é a equipa de investigação necessitar de várias iterações até o problema ficar devidamente resolvido [32].

Ao contrário da metodologia DSR que pode resultar em *design* e desenvolvimento de artefactos sem a existência de um cliente, a metodologia AR geralmente é iterativa dentro de uma investigação participativa entre o cliente e o investigador e pode não resultar num artefacto tecnológico [38].

2.7. Ferramentas de Análise de Valor

A *Análise de Valor* (AV) pode ser definida como um processo sistemático, formal e organizado aplicado a novos produtos ou projetos, cujo principal objetivo é atender aos requisitos do cliente com o menor custo possível. Não deve ser visto como um processo casual ou informal, mas sim como uma atividade de gestão que requer planeamento, controlo e coordenação [39]. É um processo que procura identificar e eliminar características de produtos e serviços que não agregam valor real ao cliente ou ao produto, mas incorrem em custos para o processo de fabricação ou prestação do serviço. Como tal, o processo de AV é usado para oferecer um produto ou serviço de melhor desempenho e qualidade a um custo mínimo[32]. Antes de iniciar o processo de AV é importante compreender todos os requisitos do cliente para que sejam estabelecidas especificações a fim de avaliar o nível de adequação entre o produto e o valor obtido para o cliente. Erros de planeamento são comuns e difíceis de corrigir, por isso deve ser formada uma equipa multidisciplinar.

De acordo com Rich e Holweg [39] existem cinco fases no processo de AV:

- **Orientação**: seleção e criação da equipa que conduzirá o processo;
- **Análise funcional**: análise do produto ou projeto, identificando quais as funções ou características mais importantes;

[32] `https://www.investopedia.com/terms/p/process-value-analysis-pva.asp`

- **Geração de ideias**: momentos de *brainstorming*[33] com o intuito de todos os membros da equipa contribuírem ativamente com ideias;
- **Análise e avaliação**: todas as alternativas são analisadas e avaliadas de acordo com as funções requeridas. São também avaliados todos os custos envolvidos;
- **Implementação**: na etapa final é quando o produto/projeto é implementado gerando negócio para a empresa e valor para o cliente.

2.7.1. Processo de Inovação

Atualmente, a inovação surge como um fator estratégico das organizações. Se no passado o foco das organizações era aplicar boas práticas de Engenharia, agora é necessário um esforço extra para criar fatores de diferenciação [40].

O conceito de inovação pode traduzir-se na criação de um novo produto, ou na atualização, manutenção ou adição de funcionalidades a um produto existente. Seja qual for o processo em que se verifique inovação, o principal objetivo é acrescentar valor à organização e aos clientes, e por isso existem dois fatores que devem ser tidos em conta: necessidade e valor. A necessidade está relacionada com as reais necessidades e expectativas do cliente a respeito do produto. A definição de valor é uma definição subjetiva uma vez que varia consoante o contexto e a pessoa [39].

O processo de inovação é composto por três fases distintas (figura 2.6):

- ***Fuzzy Front End* (FFE)**: nesta fase inicial o principal objetivo passa por atualizar um produto existente ou lançar um novo, baseando-se numa sequência de passos: identificação e análise da oportunidade, geração de ideias e posterior seleção da ideia a implementar;
- ***New Product Development* (NPD)**: desenvolvimento e testes ao produto;
- **Comercialização:** promoção e comercialização do produto desenvolvido.

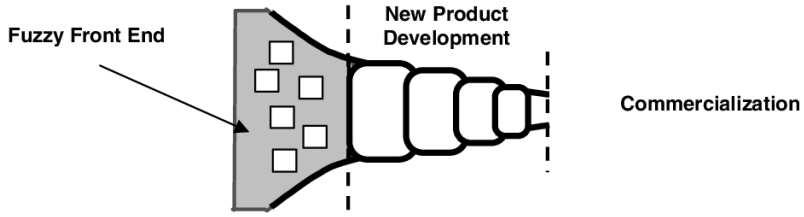


Figura 2.6. Principais fases do processo de inovação: FFE, NPD e comercialização.

2.7.1.1. Modelo *New Concept Development*

O *New Concept Development* (NCD) (figura 2.7) é um modelo cujo objetivo é criar um padrão para a primeira fase do processo de inovação e foi criado em 2002 por Peter Koen [41]. O objetivo do NCD é padronizar linguagens, termos e as melhores práticas associadas à fase FFE. Este modelo é composto por três partes principais [41]:

- **Motor**: representa a liderança, a cultura e as estratégias de negócio de uma organização;

[33] dinâmica individual ou de grupo que visa a exploração da criatividade objetivando a proposta de soluções.

- **Cinco elementos**: identificação e análise da oportunidade, geração e seleção de ideias, e definição do conceito;
- **Fatores de influência**: capacidade organizacional e fatores internos e externos que possam estar envolvidos e que são incontroláveis pela organização.

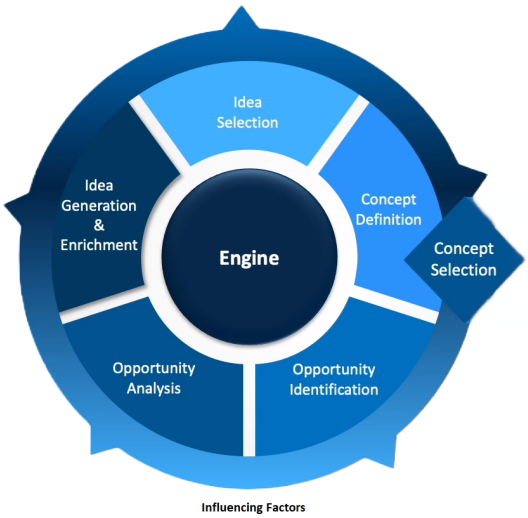


Figura 2.7. Modelo *New Concept Development* (NCD)[34].

Identificação da Oportunidade
Identificação da oportunidade de inovação pela organização, normalmente relacionadas com o objetivo do negócio. A oportunidade pode ser uma resposta de curto prazo a uma ameaça competitiva, uma possibilidade de avanço para conseguir vantagem perante a concorrência ou apenas um meio de simplificar as operações, acelerá-las ou reduzir os custos [41].

Análise da Oportunidade
A análise de oportunidade é um processo cujo objetivo é avaliar se uma oportunidade de negócio deve ser aproveitada. Existem diversas ferramentas que podem auxiliar no processo de análise da oportunidade (*e.g.*, Análise SWOT).

Geração de Ideias
Após a análise da oportunidade é necessário gerar possíveis soluções que possam ser implementadas de forma a alcançar os objetivos associados à oportunidade identificada.

Seleção de Ideias
Após a geração de ideias é necessário avaliá-las e selecionar qual a que servirá de base à solução a ser implementada. Como se trata de uma decisão crucial no processo de inovação, as organizações utilizam métodos de apoio à tomada de decisão nesta fase do processo (*e.g., Analytic Hierarchy Process* (AHP)).

Definição do Conceito
A definição do conceito corresponde à fase final de formalização da ideia selecionada na fase anterior.

Em suma, o NCD inicia-se pela identificação de uma oportunidade que pode interagir com os restantes elementos do modelo. Estes por sua vez podem interagir com os fatores de

[34] https://www.sketchbubble.com/en/presentation-concept-development.html

influência (*i.e.,* capacidade organizacional, fatores internos e externos) e impulsionados pelo motor (*i.e.,* estratégias do negócio, liderança e cultura organizacional). A forma circular do modelo sugere que as ideias e oportunidades devem fluir e interagir entre todos os cinco elementos. Assim, o modelo possui dois pontos de início, porém apenas um de saída: a definição e seleção do conceito, sendo este ponto, a ligação com a fase seguinte do processo de inovação (NPD) [42].

2.7.1.2. Análise SWOT

A análise SWOT (*Strengths, Weaknesses, Opportunities, Threats*) é uma ferramenta de planeamento estratégico usada para avaliar como uma organização se compara à concorrência [43]. Permite fazer uma análise interna da organização, com o quadrante das forças e fraquezas, e uma análise externa relacionada com o meio onde a organização se insere, com o quadrante das oportunidades e ameaças (figura B1). É normalmente utilizada antes de se implementar um projeto ou colocar um novo produto no mercado.

Embora a Análise SWOT seja conhecida por facilitar a criação da estratégia organizacional com a avaliação dos fatores internos e externos, alguns autores acham que a ferramenta é superficial, podendo mesmo prejudicar o desempenho se os resultados forem mal compreendidos ou utilizados [43].

2.7.1.3. *Value Proposition Canvas*

Após a identificação da oportunidade e a seleção da ideia a adotar, torna-se importante perceber de que forma a solução irá contribuir com o acréscimo de valor para a organização e para os clientes. Criado por Osterwalder *et al.* [44], o *Value Proposition Canvas* (figura B2) é uma ferramenta que ajuda a garantir que um produto ou serviço está de acordo com os valores e necessidades dos clientes[35]. O principal objetivo é criar um ajuste entre o produto e o mercado, e por isso destacam-se dois blocos distintos: o perfil do cliente e a proposta de valor[36].

I. Perfil do Cliente

Este bloco ajuda a compreender e a traçar o perfil dos clientes, identificando as principais necessidades e aquilo que procuram no produto ou serviço em questão. Dentro deste bloco destacam-se três categorias:

- **Tarefas do cliente:** é importante compreender quais são as tarefas que o cliente pretende realizar, os problemas que estão a tentar resolver e as necessidades que desejam satisfazer, só assim o produto irá responder às suas necessidades;
- **Dores:** correspondem a tudo aquilo que incomoda os clientes enquanto desenvolvem as suas tarefas, podendo incluir emoções negativas, desafios, riscos envolvidos, consequências e erros;
- **Ganhos:** todos os benefícios que os clientes esperam ou desejam obter, sejam eles funcionais, emocionais, sociais ou financeiros.

[35] https://https://www.strategyzer.com/canvas/value-proposition-canvas
[36] https://businessmodelanalyst.com/value-proposition-canvas/

II. Proposta de Valor

Depois do perfil dos clientes definido é importante descrever e analisar o produto ou serviço em questão de forma a garantir a satisfação dos clientes. Dentro deste bloco destacam-se três categorias:

- **Produtos e serviços:** devem ser incorporados aqui os produtos e serviços que a organização oferece e que podem ajudar os clientes a realizar as tarefas pretendidas. Os produtos e serviços podem ser tangíveis (*e.g.*, produtos manufaturados), digitais/virtuais (*e.g.*, *downloads*, *online*) ou financeiros (*e.g.*, fundos de investimentos, serviços de financiamento);
- **Aliviam as dores:** deve ser descrita a forma como os produtos e serviços eliminam ou reduzem as emoções negativas, custos e situações indesejadas;
- **Criadores de ganho:** deve-se descrever de que forma os produtos e serviços criam ganhos para os clientes.

2.7.1.4. *Analytic Hierarchy Process*

Desenvolvido na década de 1970 pelo Professor Thomas L. Saaty [45], o AHP é um método multicritério de apoio à decisão, baseado em matemática e psicologia que fornece um quadro abrangente e racional de estruturação de um problema de decisão. Segundo Supraja e Kousalya [46], o AHP é uma abordagem para a tomada de decisão que envolve a estruturação de vários critérios de escolha múltipla numa hierarquia, a avaliação da importância relativa desses critérios e a comparação das alternativas para cada critério, determinando uma classificação geral das alternativas.

As três primeiras fases do método AHP consistem em definir os objetivos, os critérios e, por último, as alternativas que resolvem o problema. Assim que estas fases estiverem concluídas é necessário construir a árvore hierárquica de decisão[37] (figura 2.8) que ajuda a perceber a relação entre os critérios definidos.

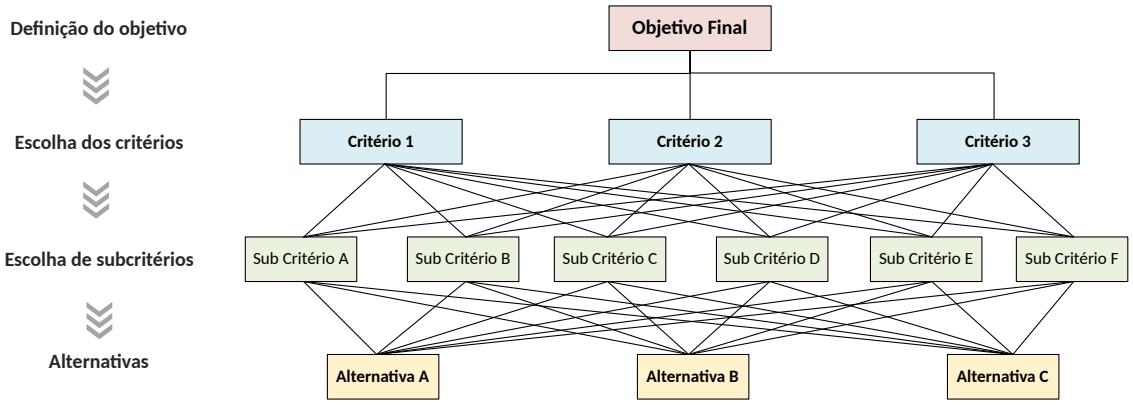


Figura 2.8. Exemplo de uma árvore hierárquica de decisão com critérios e subcritérios.

Posteriormente, os critérios e subcritérios são comparados entre si, sendo normalmente utilizada a escala fundamental de Saaty[38] (tabela C1). Os resultados das comparações

[37] https://vidadeproduto.com.br/framework-ahp/

[38] https://www.researchgate.net/figure/Figura-1-Escala-Fundamental-de-Saaty_fig1_233633014

realizadas devem ser apresentados sob uma matriz quadrada conforme ilustra a matriz A (equação 1).

$$A = \begin{bmatrix} 1 & a_{1,2} & \cdots & a_{1,n} \\ \frac{1}{a_{2,1}} & 1 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{a_{n,1}} & \frac{1}{a_{n,2}} & \cdots & 1 \end{bmatrix} \tag{1}$$

Tendo em conta as seguintes condições:

- $a_{i,j} = \alpha$, em que α corresponde ao valor da importância;
- $a_{i,j} = 1$, as posições da diagonal serão sempre 1, afinal um elemento é igualmente importante a ele mesmo;
- $a_{j,i} = \frac{1}{\alpha}$.

A matriz A é a base da definição de importâncias ou comparações entre os critérios. Assim, a sua definição resulta num vetor próprio de pesos que expressa as importâncias relativas de cada critério [47]. Para calcular este vetor é necessário normalizar a matriz, dividindo cada elemento pelo total da respetiva coluna (s_j) e calcular a média aritmética dos valores de cada linha da matriz normalizada (equação 2). Ao conjunto das prioridades relativas chamamos de vetor próprio ou vetor de prioridades (X) e permite-nos concluir quais os critérios que se revelam mais importantes para o problema em estudo.

$$A = \begin{bmatrix} 1 & a_{1,2} & \cdots & a_{1,n} \\ \frac{1}{a_{2,1}} & 1 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{a_{n,1}} & \frac{1}{a_{n,2}} & \cdots & 1 \\ s_1 & s_2 & s_3 & s_4 \end{bmatrix} \xrightarrow[\text{normalizada}]{\text{matriz}} \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{média}} X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2}$$

À semelhança da comparação dos critérios, é também necessário comparar as alternativas entre si de acordo com cada um dos critérios definidos. Por último, calcula-se a matriz composta de comparação, que relaciona todos os critérios com todas as alternativas, obtendo-se assim a prioridade relativa de cada uma das alternativas. A alternativa que apresentar maior peso será aquela mais indicada para resolução do problema.

Uma vez que a comparação dos critérios e alternativas é um processo subjetivo, pois os valores da decisão são obtidos através de pesquisas e informações recolhidas por diferentes pessoas, é necessário garantir a consistência da decisão. Esta consistência é garantida através do cálculo do *Índice de Consistência* (IC) e *Razão de Consistência* (RC). O IC (equação 3) é dado por:

$$IC = \frac{\lambda_{max} - n}{n - 1}; \quad A * X = \lambda_{max} * X \tag{3}$$

Em que n corresponde ao número de critérios avaliados, que por sua vez, é igual ao número de elementos da matriz.

A RC (equação 4) é calculada pelo quociente entre o IC e o *Índice Aleatório* (IA), em que o IA é dado pelos valores da tabela C2.

$$RC = \frac{IC}{IA} \tag{4}$$

Se o resultado da RC for menor que 0.1 (10%) considera-se que os critérios são consistentes e o método válido. Caso contrário, o método precisa de ser repetido para melhorar a consistência.

2.7.1.5. *Quality Function Deployment*

Atualmente os clientes têm ao seu dispor uma infinidade de opções disponíveis para escolher produtos e serviços. A maioria escolhe com base numa perceção geral de qualidade ou de valor, tentando sempre atingir o maior retorno possível. Para se manterem competitivas, as organizações devem definir quais as características dos produtos ou serviços (*e.g.,* confiabilidade, estilo, desempenho) que foram a perceção de qualidade e valor para o cliente[39].

O *Quality Function Deployment* (QFD) é um processo e conjunto de ferramentas usadas para definir as necessidades do cliente, e convertê-las em especificações de Engenharia detalhadas, e planos para produzir os produtos que atendem a esses requisitos.

Foi desenvolvido pela primeira vez no Japão no final do ano de 1960 por Yoji Akao enquanto trabalhava na Mitsubishi. Mais tarde, no início dos anos 80, passou a ser adotado por outras grandes empresas, como a Toyota[39].

Existem vários benefícios ao utilizar o QFD, nomeadamente:

- **Foco no cliente:** foco nas necessidades do cliente e não naquilo que a organização acredita que o cliente deseja;
- **Análise da concorrência:** permite a comparação direta do *design* ou do produto à concorrência. Essa análise pode ser benéfica permitindo que a organização encontre fatores de destaque;
- **Redução de custos:** redução da probabilidade de existência de alterações tardias no projeto;
- **Estrutura e documentação:** o QFD fornece um método estruturado e ferramentas para registar as decisões tomadas e aprendizagens conseguidas durante o processo de desenvolvimento do produto ou projeto. Esta base de conhecimento pode servir como histórico e auxiliar em projetos futuros.

O QFD é um processo que pode ser desdobrado em quatro fases[40] com atividades que devem ser executadas ao longo do ciclo de desenvolvimento do produto:

- **Definição do produto:** esta fase inicia-se com a recolha e tradução das necessidades dos clientes em especificações do produto. Pode também incluir uma análise competitiva para avaliar a adequação do produto da concorrência às necessidades do cliente;
- **Desenvolvimento do produto:** transformar as características técnicas em componentes reais que irão garantir a qualidade desejada;
- **Desenvolvimento de processos:** após conceção do produto na fase anterior, esta fase procura analisar a maneira mais eficiente de produzir os componentes;

[39] https://quality-one.com/qfd/

[40] https://www.nortegubisian.com.br/blog/qfd-como-utilizar-o-desdobramento-da-funcao-qualidade/

- **Controlo de qualidade do processo:** são estabelecidos os meios que serão utilizados para monitorizar o processo. Nesta fase, é importante ter um programa de gestão total da qualidade corporativo (*e.g., Six Sigma*).

Uma ferramenta habitualmente utilizada na primeira fase desta metodologia é a *House of Quality* (HOQ)[41]. O princípio base da HOQ é a crença que de os produtos devem ser projetados para refletir os desejos e necessidades dos clientes.

Para o preenchimento da HOQ (figura D1) inicialmente é necessário compreender corretamente os requisitos e necessidades do cliente e traduzi-las em características de Engenharia. Posteriormente é necessário atribuir uma importância a cada uma das necessidades do ponto de vista do cliente, sendo normalmente utilizada uma escala numérica de 1 a 5, em que 1 é o nível mais baixo e 5 o nível mais alto. As importâncias devem ser normalizadas de forma a obter a percentagem do grau de importância de cada uma das necessidades.

Após a classificação das importâncias das necessidades e da sua tradução para características de Engenharia é necessário estabelecer a relação entre elas (tabela 2.1A). De seguida, classificam-se as características de Engenharia por importância. Esta importância é calculada tendo em conta a percentagem relativa de importância de cada uma das necessidades e o valor numérico da relação estabelecida previamente.

Por fim, é necessário estabelecer a correlação entre as diferentes características. Esta correlação é avaliada numa escala de cinco níveis (tabela 2.1B), variando entre fortemente positivo e fortemente negativo. Fortemente positivo significa que as características se relacionam de forma complementar e fortemente negativo significa que não existe qualquer relação entre elas.

Tabela 2.1. (A) Escala utilizada para estabelecer a relação entre as necessidades do cliente e as características de Engenharia. (B) Escala utilizada para estabelecer a correlação entre as diferentes características de Engenharia definidas.

A. Relacionamentos		
Forte	9	●
Moderado	3	○
Fraco	1	▽
Sem relação	0	-

B. Correlação	
+ +	Fortemente Positivo
+	Positivo
-	Negativo
- -	Fortemente Negativo
	Sem correlação

Também a análise de concorrentes diretos que possuem o mesmo produto é um ponto de extrema importância no método QFD, porque além de permitir compreender o que os clientes mais apreciam na concorrência é também possível estabelecer vantagens competitivas. Concluída a elaboração da HOQ é possível tirar conclusões sobre qual a necessidade mais importante e menos importante para o cliente. Além disso é também possível perceber a correlação entre as características de Engenharia definidas.

2.7.2. Técnicas de Análise de Requisitos

A qualidade do *software* é um elemento chave na Engenharia de *Software* [48] e cada vez mais importante no mercado atual. Esta compreende todas as características e funcionalidades de um produto ou a capacidade de satisfazer determinados requisitos. No entanto, ao longo dos anos várias definições de qualidade foram surgindo e com elas também alguns modelos.

[41] https://hbr.org/1988/05/the-house-of-quality

O modelo FURPS+ *(Functionality, Usability, Reliability, Performance, Supportability, +)* criado por Robert Grady na *Hewlett-Packard* (HP) é utilizado para a classificação de atributos de qualidade do *software*. Especificamente, este modelo é normalmente usado para classificar os requisitos de um sistema como funcionais e não funcionais[42]. É uma evolução do modelo FURPS que, em adição às cinco categorias iniciais, inclui uma nova (+) para contemplar necessidades adicionais. Atualmente as seis categorias compreendem:

- **Funcionalidade (*functionality*):** nesta categoria devem constar as principais funcionalidades que o *software* deve apresentar[43], mas que não se relacionam com os casos de uso do sistema. Estas funcionalidades podem estar relacionadas com a segurança, ajudas ao utilizador, entre outros [49];
- **Usabilidade (*usability*):** refere-se ao quão eficaz é o produto do ponto de vista do utilizador, se é visualmente apelativo, de fácil utilização e compreensão. Por norma a usabilidade é avaliada considerando fatores de estética geral e consistência [49];
- **Confiabilidade (*reliability*):** a forma como o sistema lida com as falhas, qual o tempo de inatividade necessário durante uma reposição e de que forma o sistema é recuperado. Este atributo é normalmente avaliado através da medição do tempo médio que decorre entre falhas (*Mean Time Between Failures* (MTBF)) e do tempo médio necessário para reparar uma falha (*Mean Time To Repair* (MTTR)) [49];
- **Desempenho (*performance*):** refere-se ao desempenho do sistema, à velocidade de processamento, ao consumo de recursos e ao tempo máximo de resposta;
- **Suporte (*supportability*):** diz respeito à facilidade de manutenção, legibilidade do código e a possibilidade de monitorização do sistema;
- **+ :** esta categoria contempla necessidades adicionais que surjam, tais como, restrições do projeto e requisitos de implementação, interface ou físicos.

2.8. Tecnologias

Depois de apresentados os conceitos teóricos sob os quais esta dissertação assenta é também importante estudar e apresentar as várias tecnologias capazes de suportar os conceitos introduzidos e que permitem concretizar os objetivos propostos. Esta secção encontra-se dividida em quatro partes: inicialmente é focada nos sistemas de controlo de versões, posteriormente são apresentadas e comparadas as várias tecnologias existentes de orquestração de *containers* e, por último são estudadas as ferramentas de IaC e as tecnologias de gestão de aplicações.

2.8.1. Sistemas de Controlo de Versões

Os sistemas de controlo de versões são ferramentas de *software* que ajudam as equipas a gerir as alterações no código-fonte de um *software* ao longo do tempo, permitindo trabalhar de forma mais rápida. Mantêm um registo de todas as modificações no código, e por isso se um erro for cometido os desenvolvedores podem voltar rapidamente às versões anteriores[44]. Estes sistemas tornam-se também úteis para as equipas de DevOps, pois reduzem o tempo de desenvolvimento e aumentam a eficiência nas implantações do *software*.

[42] https://qualidadebr.wordpress.com/2008/07/10/furps/
[43] https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/
[44] https://www.atlassian.com/br/git/tutorials/what-is-version-control

2.8.1.1. Git

O Git[45] é um sistema de controlo de versões *open-source* criado por Linus Torvalds em 2005 para auxiliar no desenvolvimento do projeto Linux. Além de ter uma arquitetura distribuída, o Git foi desenhado para garantir desempenho, segurança e flexibilidade. Devido à sua capacidade de gerir projetos com grandes equipas de trabalho, o Git acabou por ser adotado por outros projetos tornando-se no sistema de controlo de versões mais utilizado do mundo[46]. Atualmente existem diversas plataformas de controlo de versões que se baseiam na tecnologia Git, como é o caso do GitLab e do GitHub.

I. Principais Conceitos do Git

- **Repositório:** local onde os ficheiros e respetivas cópias ficam armazenados. O repositório pode ser local ou remoto e permite guardar não apenas ficheiros de texto, mas também imagens, áudios e outros elementos relacionados com o projeto;
- ***Branch:*** corresponde às ramificações do código do projeto, ou seja, são cópias do código original que podem ser manipuladas livremente pelos desenvolvedores, sem afetar as funcionalidades em produção. Isto permite que todas as alterações sejam realizadas de forma segura, sem a ocorrência de erros na cópia principal do projeto;
- ***Commit:*** é um conjunto de alterações no código e uma mensagem descritiva sobre o que foi feito;
- ***Merge:*** após a finalização de um trabalho numa ramificação, é necessário realizar o *merge*, isto é, fundir a cópia e ficheiros modificados com a ramificação principal do projeto;
- ***Push request:*** é o envio das modificações após o *merge* para o repositório central;
- ***Pull request:*** é utilizado quando um desenvolvedor altera a ramificação principal no repositório central, puxando as modificações realizadas para o seu computador, fundindo a nova versão com o seu código local;
- ***Fork:*** trata-se da cópia de um repositório remoto para uma máquina local, e é normalmente realizado quando se começa a contribuir para um projeto já existente.

II. Fluxo de Trabalho com Git

Geralmente, não existe só uma *branch* para registar o histórico de um projeto, existem duas: a *master* e a *develop*. A *master* armazena o histórico de lançamento do *software* e a *develop* serve como uma *branch* de integração de novos recursos que vão sendo desenvolvidos pelas equipas. A figura 2.9 representa o fluxo de trabalho recomendado, também conhecido como *Git Flow*[47], e normalmente adotado pelas organizações que utilizam estes sistemas.

Quando um desenvolvedor começa a contribuir para um projeto deve clonar o repositório central e criar uma nova *branch (feature)* a partir de *develop* (figura 2.9A). Sempre que o recurso estiver devidamente implementado a *branch* deve ser integrada em *develop* (figura 2.9B). Quando a *develop branch* estiver pronta para o lançamento de uma nova versão do *software* é criada uma *release branch* (figura 2.9C). A criação desta *branch* inicia o próximo ciclo de lançamento, portanto, nenhum novo recurso deve ser adicionado depois deste ponto, com exceção de correções de *bugs (hotfix)*, geração de documentação e outras

[45] https://git-scm.com/about
[46] https://www.atlassian.com/br/git/tutorials/what-is-git
[47] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

tarefas relacionadas com o lançamento. Quando a *release branch* estiver pronta para ser distribuída pelos clientes, deve ser integrada na *master branch* e na *develop branch* e identificada com um número de versão (figura 2.9D).

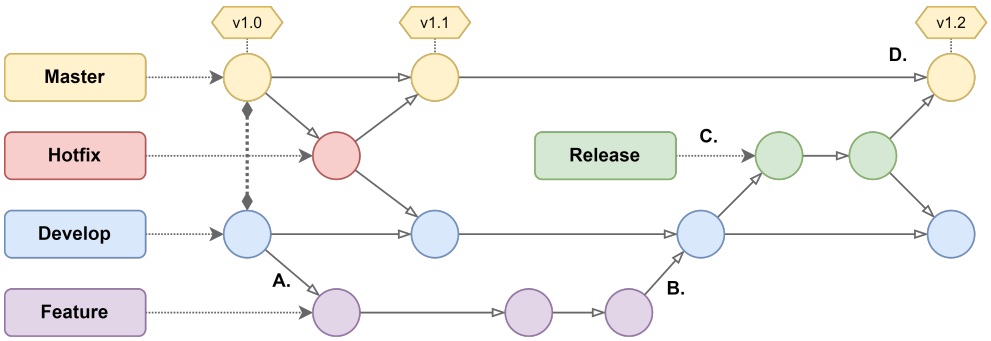


Figura 2.9. Fluxo de trabalho com Git normalmente adotado pelas organizações e conhecido como *Git Flow*[48]. (A) Criação de uma nova *branch* a partir de *develop*. (B) *Merge* da *feature branch* com *develop*. (C) Criação da *release branch* dando início ao próximo ciclo de lançamento do *software*. (D) Integração da *release branch* e nova versão disponível para distribuir pelos clientes.

2.8.1.2. GitLab

O GitLab foi lançado em 2014, porém ficou mais conhecido em 2018 quando o GitHub foi comprado pela Microsoft e os seus utilizadores optaram pelo GitLab alegando não querer colocar o código em plataformas pertencentes à Microsoft.

A MOG Technologies utiliza o GitLab no seu quotidiano como sistema de controlo de versões, e por isso os desenvolvimentos desta dissertação foram lá integrados.

I. GitLab *Pipelines* CI/CDE

Tal como referido na secção 2.1, o processo de integração e entrega contínua é bastante importante no desenvolvimento moderno de *software* e uma das características mais preciosas é a automação que proporciona. As *pipelines* de CI/CDE são compostas por um conjunto de etapas a serem executadas de forma automática para a disponibilização de uma nova versão do *software*. Estas etapas normalmente compreendem ações como compilação do código-fonte, testes e *deployment* do *software*.

O GitLab possui ferramentas de integração e entrega contínua fáceis e intuitivas de configurar que são definidas num ficheiro *YAML Ain't Markup Language* (YAML), normalmente chamado `.gitlab-ci.yaml`. Este ficheiro é composto essencialmente por um conjunto de *jobs*. Na terminologia do GitLab, *jobs* são os elementos mais básicos dentro deste ficheiro e são executados por *runners* que devem ser previamente configurados e atribuídos ao projeto.

2.8.2. Orquestração de *Containers*

Nas aplicações conteinerizadas é possível gerir e manter os *containers* de forma manual, através das interfaces gráficas e APIs disponibilizadas pelos fabricantes. Porém, num sistema com um número elevado de *containers* e com necessidades específicas não é viável

[48] `https://dev.to/jilcimar/git-flow-uma-estrategia-de-sucesso-para-adaptar-ao-processo-do-seu-time-1bah`

efetuar esse trabalho manualmente[49]. A orquestração de *containers* permite definir como implantar, monitorizar e controlar dinamicamente as aplicações.

As tecnologias de orquestração de *containers* oferecem normalmente as seguintes funcionalidades [50]:

Controlo do limite de recursos (*resource limit control*)
Permite reservar uma quantidade específica de recursos para um determinado *container*. Estas restrições podem ser usadas para tomar decisões de agendamento e limitar os recursos a serem utilizados por várias aplicações.

Agendamento (*scheduling*)
Define a política usada para colocar a quantidade desejada de *containers* nos nós desejados num determinado instante de tempo. Por exemplo, um agendador observa os *containers* recém criados que não têm nós atribuídos e toma a decisão de os colocar em execução no nó que parecer mais adequado.

Balanceadores de carga (*load balancing*)
É responsável por distribuir a carga entre as várias instâncias. Normalmente é utilizada a política de *Round-Robin*[50] (*i.e.,* tipo de escalonamento preemptivo que consiste em distribuir uniformemente os recursos de CPU entre todas as instâncias). Políticas mais complexas podem ser implementadas com balanceadores de carga externos.

Verificação de integridade (*health check*)
A integridade pode ser obtida controlando e verificando se um *container* é capaz de responder a solicitações. As implementações atuais fazem uso de conexões *Transmission Control Protocol* (TCP), *User Datagram Protocol* (UDP) e/ou *Secure SHell* (SSH).

Tolerância a falhas (*fault tolerance*)
Pode ser implementada com controlo de réplicas e/ou controlo de alta disponibilidade. O controlo de réplicas permite especificar e manter um número desejado de *containers*. A verificação de integridade é usada para determinar quando um *container* defeituoso deve ser destruído e um novo lançado para manter o número desejado de réplicas. O controlador de alta disponibilidade permite configurar vários orquestradores de *containers* para ter sempre controlo na aplicação, caso um orquestrador falhe ou esteja sobrecarregado.

Escalonamento automático (*auto-scaling*)
O escalonamento automático permite adicionar e remover *containers* automaticamente. As políticas implementadas normalmente baseiam-se em limites de recursos, por exemplo, na utilização do CPU e memória.

Atualmente existem várias tecnologias que permitem realizar esta orquestração de *containers*, que são analisadas e comparadas de seguida.

2.8.2.1. Kubernetes

O Kubernetes, também conhecido como K8s, foi originalmente desenvolvido pela Google, depois de uma década de experiência na implantação de sistemas escaláveis e confiáveis.

[49] https://www.vmware.com/topics/glossary/content/container-orchestration.html
[50] https://deinfo.uepg.br/~alunoso/2016/ROUNDROBIN/

Desde a sua introdução em 2014, o Kubernetes cresceu e tornou-se um dos maiores e mais populares projetos *open-source* do mundo. É uma plataforma de orquestração de *containers* que automatiza muitos dos processos manuais envolvidos na implantação, gestão e dimensionamento de aplicações em *containers*[51]. Atualmente é considerada a API padrão para a construção de aplicações na *cloud*. Está presente em quase todas as *clouds* públicas [17] e neste momento está sobre a alçada da *Cloud Native Computing Foundation* (CNCF).

O Kubernetes garante que todos os *containers* estão a ser executados, ficando atento a *containers* inativos, sem resposta ou falhados, substituindo-os por novos *containers* em correto funcionamento [51]. Esta tecnologia foi criada para mudar radicalmente a forma como as aplicações são implantadas e foi desenhada para oferecer mais velocidade, eficiência e agilidade.

Apontada como a tecnologia padrão para a orquestração de *containers*, o Kubernetes apresenta inúmeras vantagens e benefícios, nomeadamente [17]:

- Possibilita a disponibilização de um serviço altamente confiável com um tempo de atividade constante, mesmo que o *software* em execução esteja em constante mudança;
- Oferece escalabilidade e favorece as arquiteturas desacopladas;
- Deixa de existir a associação das aplicações às máquinas em que estão a ser executadas;
- Múltiplas tarefas podem ser compactadas num menor número de máquinas conduzindo a um aumento da produtividade uma vez que várias aplicações podem estar em execução na mesma máquina;
- Possui uma grande comunidade e documentação, sendo apoiada pela Google;
- Compatível com todos os sistemas operativos e disponível nas principais *clouds* públicas;
- É automático e tem a capacidade de autocorreção, oferecendo suporte ao escalonamento automático;
- Possui monitorização integrada e uma ampla gama de integrações disponíveis;

Existem também algumas desvantagens que devem ser tidas em conta, nomeadamente:

- O processo de instalação é complexo e possui uma curva de aprendizagem acentuada;
- Para projetos pequenos pode não ser justificável o seu uso;
- Requer a instalação e aprendizagem de ferramentas separadas;
- Transição complexa e difícil de gerir.

I. Objetos do Kubernetes

No Kubernetes todos os elementos são considerados objetos e cada um deles possui responsabilidades e funções bem definidas.

Node

É um único *host* e pode ser uma máquina física ou virtual. A sua função é executar *pods*. Cada *node* do Kubernetes executa vários componentes, como o *Kubelet* e o *Kube Proxy*. Os *nodes* são geridos por um *master*.

[51] `https://www.redhat.com/en/topics/containers/what-is-kubernetes`

Master

Também designado por *Control Plane* é composto por vários componentes: *API Server (api)*, *Scheduler* e *Controller Manager*. É responsável pela gestão global dos *pods* e manipulação de eventos. Normalmente consiste num único *host* no entanto, em cenários que requerem alta disponibilidade e *clusters* muito grandes, pode ser necessário utilizar redundância.

Pod

É a unidade de trabalho no Kubernetes. Cada *pod* pode conter um ou mais *containers* e são executados sempre na mesma máquina. Todos os *containers* de um *pod* possuem o mesmo endereço *Internet Protocol* (IP) e podem ter acesso ao armazenamento local partilhado no *node* que hospeda o *pod*. Assim que um *pod* é lançado num *node* do *cluster* este permanece lá até que algo o destrua. Se um *pod* entrar num estado de falha este é eliminado e um novo *pod* é lançado [16].

Services

Os *services*[52] fornecem uma abstração para expor uma aplicação num conjunto de *pods* como um serviço na rede. Com o Kubernetes não é necessário modificar a aplicação para usar mecanismos de descoberta de serviço, uma vez que os *pods* possuem o seu próprio endereço IP e um único *Domain Name System* (DNS).

Volumes

O armazenamento local é destruído juntamente com o *pod*. Os *volumes* servem para persistir os dados mesmo depois de o *pod* ter sido destruído. Quando o objetivo é apenas trocar dados entre os *containers* não é necessária a persistência dos dados, e por isso não é necessária a utilização de *volumes*.

Namespaces

Um *namespace* funciona como um *cluster* virtual. Pode existir apenas um *cluster* físico com vários *namespaces* que são isolados entre si, e apenas se conseguem comunicar através de interfaces públicas.

StatefulSet

O *StatefulSet* é normalmente utilizado para gerir aplicações com estado mantendo uma identidade fixa para cada um dos seus *pods*[53].

Secrets

Os *secrets* são pequenos objetos que contêm informações confidenciais (*e.g.,* credenciais, *tokens* de autenticação). São armazenados como texto simples numa base de dados, acessíveis pelo servidor da API do Kubernetes e podem ser usados pelos *pods*.

II. Arquitetura de um *Cluster* Kubernetes

Na tecnologia Kubernetes um *cluster* é um conjunto de *hosts* de armazenamento e recursos de rede que o Kubernetes usa para executar as várias tarefas. Os principais componentes (figura 2.10) de um *cluster*, são:

Etcd

É uma base de dados distribuída que armazena pares de chave-valor e fornece uma forma de armazenar informação consistente num *cluster*[54].

[52] https://kubernetes.io/docs/concepts/services-networking/service/
[53] https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
[54] https://etcd.io/

API Server
Este é o elemento responsável por expor uma interface *Representational State Transfer* (REST) através da qual todas as interações relacionadas com os objetos do Kubernetes são validadas e efetuadas.

Scheduler
Este elemento é responsável por selecionar o *node* que preenche os requisitos de um *pod* a ser lançado.

Kubelet
O *Kubelet* é o elemento primário de um *node* que recebe instruções do *API Server* e a sua principal tarefa é instruir como operar os *containers* dos *pods*. Ademais, este elemento tem a tarefa de provisionar os *volumes* necessários ao *pod* e de reportar o estado do seu *node* e dos seus *pods* ao resto do sistema.

Kube Proxy
É o elemento responsável por criar a abstração fornecida pelos *services*. Para isso este elemento cria e gere um conjunto de regras de rede permitindo o redirecionamento e balanceamento dos pedidos pelos vários *pods*.

Kubectl
O *Kubectl* é a interface de linha de comandos que executa comandos no *cluster* Kubernetes.

Controller Manager
Este elemento tem a tarefa de controlar e para isso conta com um conjunto de controladores individuais que realizam tarefas em rotina, sendo os mais relevantes:

- ***Node Controller:*** responsável por observar e reagir aos eventos de estado dos *nodes* do *cluster*;
- ***Replication Controller:*** responsável por manter o número de réplicas de cada *pod* correto;
- ***Service Account*** *e* ***Token Controllers****:* responsável por lidar com questões de autenticação e autorização.

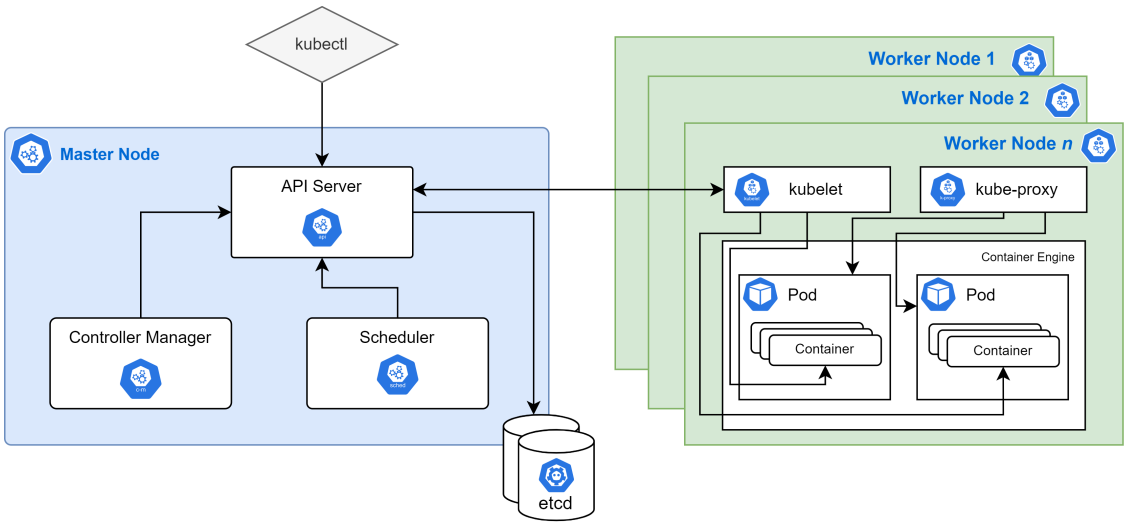


Figura 2.10. Arquitetura de um *cluster* Kubernetes com um *master node* e *n worker nodes*.

III. Comunicação entre os *Pods*

O Kubernetes garante que os *pods* conseguem comunicar uns com os outros independentemente do *node* do *cluster* onde se encontrem. Para que isto aconteça é fornecido a cada *pod* um endereço IP interno ao *cluster* que garante a comunicação entre os *pods* sem necessidade de criar *links* explícitos entre os *pods* ou mapeamento entre as portas dos *containers* e as portas do *host*.

Para implementar o modelo de rede que satisfaz os requisitos de comunicação do Kubernetes existem as seguintes ferramentas: Flannel, Calico, ACI, Cilium, Jaguar, entre outras[55].

IV. *Deployment* de Aplicações com Kubernetes

Para implantar uma aplicação conteinerizada num *cluster* Kubernetes são utilizados ficheiros do tipo YAML. Um objeto do tipo `Deployment` define como criar e atualizar instâncias da aplicação. Depois de criar um `Deployment`, o *master* do Kubernetes agenda as instâncias da aplicação incluídas nesse `Deployment` para ser executado em *nodes* individuais do *cluster*.

Na sua forma mais simples, um `Deployment` pode ser escrito da seguinte forma:

```
1  apiVersion: apps/v1beta1
2  kind: Deployment
3  metadata:
4      name: nginx-deployment
5  spec:
6      replicas: 3
7      template:
8          metadata:
9              labels:
10                 app: nginx
11         spec:
12             containers:
13             - name: nginx
14                 image: nginx:1.7.9
15                 ports:
16                 - containerPort: 80
```

Código 2.4. Exemplo de um objeto do Kubernetes do tipo `Deployment`.

Neste exemplo (código 2.4) é instanciado um `Deployment` com o nome `nginx-deployment`. São criadas três réplicas de *pods* com a imagem `nginx` na versão `1.7.9`. Este `Deployment` expõe a porta 80 para ser utilizada pelos *pods*.

2.8.2.2. Docker Compose

O Docker Compose[56] é uma ferramenta que ajuda a definir e partilhar aplicações de vários *containers*. Com o Compose, é possível escrever um ficheiro YAML para definir os serviços e com um único comando executar todos os *containers*.

[55] https://kubernetes.io/docs/concepts/cluster-administration/networking/
[56] https://docs.docker.com/compose/

```
services:
  backend:
    image: backend
    secrets:
      - dbpassword
    depends_on:
      - db
  db:
    image: postgres
    secrets:
      - dbpassword
    volumes:
      - dbdata:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=example
    expose:
      - 5432
secrets:
  dbpassword:
    file: db/password.txt
```

Código 2.5. Exemplo de um ficheiro `docker compose` composto por dois *containers*.

No código 2.5 estão a ser definidos dois *containers* diferentes: `backend` e `db`, e cada um deles é construído com base numa imagem diferente: `backend` e `postgres`, respetivamente. Através da instrução `depends_on` é possível definir dependências entre os *containers*, como é o caso do *container* `backend` que só será inicializado depois do *container* `db`. Neste exemplo, existe também a definição de *secrets* (`secrets`), variáveis de ambiente (`environment`), exposição de portas (`expose`) e o mapeamento de *volumes* (`volumes`).

2.8.2.3. Docker Swarm

O Docker e o Docker Compose auxiliam e facilitam a orquestração de *containers*, no entanto são usados apenas para uma execução *standalone*, ou seja, numa máquina local ou num único servidor. Quando surge a necessidade de escalar os serviços horizontalmente entre vários servidores existem alguns obstáculos. Em meados de 2016, a Docker criou a sua própria ferramenta de orquestração de *containers*, o Docker Swarm[57].

O Docker Swarm, ou apenas Swarm, é uma ferramenta que permite a criação de *clusters* de Docker, ou seja, podem existir vários *hosts* de Docker associados a uma mesma *pool* de recursos, facilitando assim a orquestração dos *containers*.

À semelhança de outras tecnologias de orquestração de *containers* estudadas nesta dissertação, o Docker Swarm segue uma arquitetura dividida em dois componentes fundamentais: os *Swarm Managers* e os *Workers*. Os *Swarm Managers* funcionam como nós principais responsáveis pela gestão do *cluster* e os *Workers* são os nós responsáveis pela execução dos *containers* através do Docker Engine.

O Docker Swarm apresenta várias vantagens, nomeadamente:

- Simplicidade de instalação e curva de aprendizagem reduzida;
- Integração nativa com o Docker, não sendo necessário nenhum *software* adicional;
- Escalabilidade;

[57] `https://docs.docker.com/engine/swarm/`

- *Design* descentralizado em relação aos nós do *cluster*, não exigindo que os *hosts* do *cluster* sejam do mesmo tipo ou dimensão;
- Uma aplicação pode ser escrita de forma declarativa através de um ficheiro `docker compose`;
- Monitorização do estado *containers* em execução, e caso algum *container* termine inesperadamente, um novo *container* será iniciado;
- Quando um novo serviço é executado no *cluster*, um DNS único é atribuído;
- É simples realizar qualquer tipo de atualização num serviço em execução.

Apesar de todas as vantagens referidas, destacam-se as seguintes desvantagens [52]:

- O Docker é uma tecnologia do sistema operativo Linux e embora seja suportado em ambientes Windows e macOS, internamente a tecnologia utiliza máquinas virtuais para garantir essa compatibilidade. Assim, uma aplicação conteinerizada que necessite de ser executada em *containers* Windows não poderá ser executada no Linux e vice-versa;
- Monitorização limitada: fornece apenas informações básicas sobre o estado dos *containers* em tempo real;
- Não oferece uma maneira fácil de conectar os *containers* ao armazenamento. Os *volumes* de armazenamento exigem muita improvisação e configurações manuais;
- A tolerância a falhas é limitada.

2.8.2.4. OpenShift

O OpenShift[58] é uma plataforma *open-source* desenvolvida pela Red Hat e baseada em Kubernetes e *containers* Linux de maneira independente da plataforma na qual os *containers* são executados. Permite controlar todo o ciclo de vida de uma aplicação baseada em *containers*, desde a sua implantação até à execução propriamente dita[59].

O OpenShift oferece vários serviços, incluindo gestão de processos de CI/CDE, *logs*, monitorização dos estados das aplicações e *containers*. A arquitetura do OpenShift é baseada em micro serviços possuindo assim todas as vantagens que este tipo de arquiteturas oferece (*cf.,* secção 2.2). É executado em cima de um *cluster* baseado em Kubernetes e por isso os componentes do OpenShift ficam em execução num *node master* do Kubernetes. Esses componentes são descritos de seguida:

API/Authentication
: Controla o acesso às APIs do OpenShift e do Kubernetes. Esse processo de autenticação é baseado no padrão OAuth e em certificados *Secure Sockets Layer* (SSL).

Data Store
: Responsável por armazenar o estado e outras informações das aplicações. Geralmente utiliza o *Etcd*, uma estrutura de armazenamento baseada em pares chave-valor.

Scheduler
: Responsável por distribuir as cargas de trabalho entre os *nodes* do *cluster*. É um dos principais componentes do OpenShift.

[58] `https://www.redhat.com/en/technologies/cloud-computing/openshift`
[59] `https://www.treinaweb.com.br/blog/o-que-e-o-openshift`

Management/Replication
Mecanismo responsável pelo processo de replicação e pela recolha de informações sobre o estado dos componentes e elementos do *cluster*. Atua como um ciclo infinito que vai recolhendo dados e atualizando o estado dos componentes, armazenando-os no *Data Store*. Estes processos são controlados através de estruturas chamadas *Controllers*.

2.8.2.5. Nomad

O Nomad[60] é uma ferramenta de orquestração de *containers* simples e flexível que permite às organizações implantar, gerir e dimensionar as aplicações em *containers*. Lançado em 2015 e desenvolvido pela HashiCorp, o Nomad destaca-se do Kubernetes e do Docker Swarm por oferecer suporte para gerir aplicações que não estão conteinerizadas.

Entre as suas vantagens destacam-se também a sua interface simples para gestão de aplicações, compatibilidade com os principais sistemas, e a sua excelente integração com ferramentas que suportam cenários de IaC (*e.g.,* Terraform, Consul e Vault). Devido a estas vantagens, o Nomad é considerado um concorrente do Kubernetes. No entanto, o Nomad pode ser usado como complemento de forma a atender cenários que exigem múltipla orquestração.

2.8.2.6. Marathon

O Marathon[61] é uma plataforma de orquestração de *containers* de nível de produção para o *Datacenter Operating System* (DC/OS) da Mesosphere e Apache Mesos.

As principais características desta tecnologia, são:

- Alta disponibilidade: através da eleição de um *master* consegue garantir elevados tempos de atividade;
- Vários ambientes de execução: para *containers* Mesos (usando `CGroups`) e Docker;
- Aplicações com estado: permite vincular *volumes* de armazenamento persistente às aplicações;
- Interface de utilizador intuitiva;
- Descoberta de serviço e balanceadores de carga;
- Verificação do estados das aplicações;
- Sistema de monitorização.

2.8.2.7. Análise Comparativa

Estudadas as várias tecnologias de orquestração de *containers* que atendem aos requisitos do problema, foram agregadas as principais características de cada uma das tecnologias na tabela 2.2, fornecendo assim uma breve análise comparativa. É de salientar que a tecnologia Docker Compose não foi considerada uma vez que o Docker Swarm é mais poderoso e também utiliza parte dessa tecnologia.

[60] `https://www.nomadproject.io/`
[61] `https://mesosphere.github.io/marathon/`

Tabela 2.2. Análise comparativa das várias tecnologias de orquestração de *containers* estudadas: Kubernetes, Docker Swarm, OpenShift, Nomad e Marathon.

Características/Recursos	Kubernetes [53][54][55] (https://kubernetes.io/)	Docker Swarm [54][55] (https://docs.docker.com/engine/swarm/)	OpenShift [54] (https://docs.openshift.com/)	Nomad [54] (https://www.nomadproject.io/)	Marathon [54] (https://mesosphere.github.io/marathon/)
Gerais					
Custo	Gratuito	Gratuito	€	Gratuito	Gratuito
Sistemas Operativos	Linux - macOS - Windows	Dependente da plataforma	RHCOS	Linux - macOS - Windows	DC/OS
Lançamento	2014	2016	2011	2015	2016
Fabricante	CNCF	Docker	Red Hat	HashiCorp	Apache Mesos
Linguagem	Go	Python	Go, AngularJS	Go	C++
Características Técnicas					
Limite de Recursos (*Resource Limit Control*)	✓	✓	✓	✓	✗
Agendamento *(Scheduling)*	✓	✓	✓	✓	✓
Balanceamento de Carga *(Load Balancing)*	✓	✓	✓	✓	✓
Verificação de Integridade *(Health Check)*	✓	✓	✓	✓	✓
Tolerância a Falhas *(Fault Tolerance)*	✓	✓	✓	✓	✗
Escalonamento Automático *(Auto-Scaling)*	✓	✗	✓	✓	✓
Outras					
Documentação/Apoio da Comunidade	☺	☺	😐	😐	☺
Complexidade/Curva de aprendizagem	😐	☺	😐	☺	😐
Logging e Monitorização	☺	☹	☺	☺	☺
Armazenamento	☺	😐	☺	☺	☺
Interface do Utilizador	☺	☹	☺	☺	☺

☺ Bom (4 - 5) ✓ (Possui)
😐 Médio (2 - 3) ✓ (Com Limitações)
☹ Fraco (0 - 1) ✗ (Não Possui)

2.8.3. *Infrastructure as Code*

Para permitir a implementação de mecanismos de IaC recorre-se normalmente a tecnologias capazes de provisionar e gerir infraestruturas de forma automática recorrendo a ficheiros de código.

2.8.3.1. Vagrant

O Vagrant[62] é um *software open-source* da HashiCorp utilizado para criar e manter ambientes de desenvolvimento virtuais (*i.e.,* máquinas virtuais), utilizando vários *providers*, nomeadamente: VirtualBox, Hyper-V e VMware. O Vagrant reduz o tempo de configuração de ambientes de desenvolvimento, aumenta a velocidade de desenvolvimento e traz a ideia de recursos computacionais descartáveis [56]. No Vagrant aplica-se a seguinte nomenclatura:

Box
: Uma *box* é um sistema operativo que deve ser definido no início do projeto. O Vagrant deve realizar o *download* da imagem do sistema que foi definido e aplicá-la na máquina virtual.

Host* e *Guest
: O *host* é o sistema onde foi desenvolvido o projeto e onde se está a executar o Vagrant. O *guest* é a máquina virtual que será criada.

Provider
: Um *provider* é o *software* responsável pela virtualização. Como dito anteriormente, o Vagrant é compatível com outros *providers*, no entanto o VirtualBox já vem pré-configurado no Vagrant e por isso é bastante simples de usar. Para os outros *providers* poderá ser necessária a instalação de alguns *plugins*.

Vagrantfile
: É o ficheiro de configuração do Vagrant para o projeto em questão. É nele que definimos tudo o que diz respeito à máquina virtual, versão do sistema operativo, memória, endereço IP, portas expostas, entre outras configurações.

Provisioner
: Um *provisioner* é o *software* que irá automatizar as tarefas que serão executadas no ambiente virtual assim que o sistema operativo estiver disponível.

2.8.3.2. Terraform

O Terraform[63] desenvolvido pela HashiCorp é uma ferramenta de IaC que permite definir recursos locais e na *cloud* em ficheiros de configuração legíveis por humanos que podem ser versionados, reutilizados e partilhados. Permite aos desenvolvedores usar uma linguagem de configuração de alto nível chamada *HashiCorp Configuration Language* (HCL) para descrever a infraestrutura. É gerado um plano com as ações que serão executadas, que de seguida é aplicado. Outras características[64] levam os desenvolvedores a optar pelo Terraform como ferramenta de IaC:

- É um *software* livre, e por isso grandes comunidades desenvolvem e disponibilizam *plugins*, extensões e suporte profissional para a tecnologia;

[62] `https://www.vagrantup.com/`
[63] `https://www.terraform.io/`
[64] `https://www.ibm.com/cloud/learn/terraform`

- É uma plataforma independente, o que a torna passível de utilizar com qualquer *provider* de serviços na *cloud*;
- Oferece uma excelente integração com o Kubernetes;
- O Terraform provisiona infraestrutura imutável, o que significa que a cada mudança no ambiente a configuração atual é substituída por uma nova que considera a mudança e a infraestrutura é fornecida novamente.

2.8.4. Gestão de Aplicações

Na secção 2.8.2.1 estudamos como podemos implantar aplicações em Kubernetes através de ficheiros YAML, em que cada objeto configurado desempenha um papel bem definido. Normalmente, uma aplicação requer mais do que um objeto e isso traduz-se em vários ficheiros com centenas de linhas. À semelhança das ferramentas de gestão de configuração, que auxiliam na gestão e configuração de sistemas computacionais, servidores e *softwares*[65], existem também ferramentas de gestão de aplicações, cujo objetivo é facilitar a implantação e gestão das aplicações.

2.8.4.1. Helm

O Helm é apresentado, desde o seu primeiro *commit* no repositório Git em 2015, como um gestor de pacotes do Kubernetes. O seu principal objetivo é empacotar os objetos Kubernetes necessários para implantação de uma aplicação, garantindo que podem ser instalados, atualizados e destruídos juntos [57].

Existem quatro objetivos principais do Helm [57]:

«Do zero até ao Kubernetes»
: Inicialmente o Kubernetes era difícil de configurar, e requeria muitas vezes compilar o código-fonte e utilizar *scripts* para o executar. Depois de o *cluster* Kubernetes estar finalmente ativo, era ainda necessário descrever as aplicações em ficheiros YAML. Assim, o Helm surge com o objetivo de facilitar e agilizar a implantação das aplicações com exemplos prontos para a aplicação em produção.

Sistema de Gestão de Pacotes
: A maioria dos sistemas operativos possui um gestor de pacotes, cujo objetivo é facilitar a localização, instalação, atualização e exclusão do *software*. Comparando o Kubernetes com um sistema operativo, mas que ao invés de *software* executa *containers*, verificou-se a necessidade de um gestor de pacotes.

Segurança
: A instalação de pacotes através do Helm apresenta-se como um mecanismo seguro, uma vez que é possível verificar se o pacote é de uma fonte confiável.

Reutilização e Configurabilidade
: Uma das muitas vantagens dos gestores de pacotes é a possibilidade de instalar algo repetidamente sempre com um resultado previsível. A reutilização é conseguida com o uso de *charts*, uma vez que fornecem um padrão para produzir os mesmos manifestos do Kubernetes. Um Helm *chart* é na verdade um conjunto de ficheiros organizado de forma específica num diretório (*cf.,* apêndice F.1.1). No mundo Linux, cada distribuição tem o seu próprio gestor de pacotes e repositórios. No caso do

[65] https://www.redhat.com/pt-br/topics/automation/what-is-configuration-management

Kubernetes, o Helm foi construído para que todas as distribuições de Kubernetes pudessem compartilhar o mesmo gestor de pacotes. Quando há diferenças entre duas distribuições do Kubernetes, os *charts* contemplam essas diferenças juntamente com as configurações.

O Helm fornece ferramentas para configurar pacotes no momento de instalação e reconfigurar instalações durante as atualizações. Apesar disso, o Helm é um gestor de pacotes e não uma ferramenta de gestão de configurações.

2.9. Sumário

Neste capítulo, foram apresentados e estudados os conceitos teóricos mais relevantes para o desenvolvimento desta dissertação. Foram também analisadas algumas das tecnologias que permitem implementar a orquestração de *containers* (*cf.,* secção 2.8.2) e IaC (*cf.,* secção 2.8.3) para provisionamento e gestão da infraestrutura. No capítulo seguinte, é apresentada a análise de valor da solução a implementar, analisados os requisitos funcionais e não funcionais e definidos os casos de uso do sistema.

Capítulo 3

Análise

Considerando os conceitos teóricos e as tecnologias apresentadas pretende-se neste capítulo analisar a solução do ponto de vista do negócio e do utilizador, averiguando o valor que este acrescenta à organização. Ademais, é apresentada a Engenharia de requisitos com a definição dos requisitos funcionais e não funcionais da solução e a sua relação com os casos de uso do sistema.

3.1. Modelo *New Concept Development*

De forma a compreender e analisar a solução a implementar foi aplicado o modelo *New Concept Development* (*cf.,* secção 2.7.1.1) através da exploração de cada uma das fases que o compõe: identificação e análise da oportunidade, geração e seleção de ideias.

3.1.1. Identificação da Oportunidade

A MOG Technologies identificou a necessidade de evoluir a forma como o *software* é distribuído pelos seus clientes (*cf.,* secção 1.2). Esta transformação tem impacto não apenas no negócio mas também no dia-a-dia dos colaboradores.

A implementação desta oportunidade foi dividida em duas fases. A primeira fase corresponde à criação automatizada de uma infraestrutura alojada nas instalações da MOG Technologies capaz de suportar o *deployment* do *software*, sendo este o principal foco da presente dissertação. Por sua vez, a segunda fase depende dos desenvolvimentos da primeira e corresponde à disponibilização do *software* como um serviço numa *cloud* pública.

Terminada a identificação da oportunidade, realizou-se uma análise ao seu valor a fim de avaliar a viabilidade da ideia.

3.1.2. Análise da Oportunidade

A oportunidade identificada irá permitir simplificar o processo de distribuição do *software*, garantindo que os clientes tenham acesso ao *software* através da internet, de forma rápida e eficaz. Além destas melhorias, também o processo interno da MOG Technologies será otimizado, uma vez que também os colaboradores irão ter acesso ao *software* de forma mais rápida e eficiente.

Deste modo, considerou-se que os pontos positivos que a oportunidade apresenta são suficientes para ser explorada e aproveitada. Adicionalmente elaborou-se uma análise SWOT com objetivo de validar esta premissa.

3.1.2.1. Análise SWOT

A análise SWOT (*cf.,* secção 2.7.1.2) é uma ferramenta que auxilia no processo de análise da oportunidade e que, devidamente interpretada, pode trazer mais valias. Assim, realizou-se uma análise SWOT à solução a implementar e que é apresentada na tabela 3.1.

Tabela 3.1. Análise SWOT da solução a implementar.

A. Fatores Internos	
I. Forças	**II. Fraquezas**
Portabilidade e flexibilidade da solução devido à compatibilidade existente com as principais *clouds* públicas e suporte para os vários sistemas operativos. Maior velocidade de entrega do *software*. Redução dos processos manuais de instalação do *software*, e de criação e gestão da infraestrutura. Redução dos custos relacionados com o licenciamento e *hardware*. Atualização e lançamento de novas versões do *software* mais rápidos e eficientes. Recuperação rápida em caso de desastre através de mecanismos de redundância e alta disponibilidade. Possibilidade de usar ferramentas de controlo de versões, e assim voltar a estados conhecidos em caso de algum comportamento inesperado.	Curva de aprendizagem acentuada que pode obrigar à formação dos colaboradores e maior investimento em equipas específicas (*e.g.,* segurança). O custo relacionado com as *clouds* pode ser imprevisível, pois depende do tráfego e recursos utilizados. Necessidade de alteração do modelo de negócio. A transição entre tecnologias e ferramentas mais tradicionais poderá ser complexa. Necessidade de manutenção e gestão de permissões para garantir a segurança.
B. Fatores Externos	
I. Oportunidades	**II. Ameaças**
Adoção em grande escala de SaaS por várias organizações. Grande utilização das tecnologias envolvidas pela comunidade. As *clouds* e as ferramentas de gestão de configurações estão cada vez mais preparadas para este modelo de negócio. Constante evolução das tecnologias. Crescente necessidade e procura de ferramentas de *broadcast* na *cloud*. Evolução da velocidade e processamento de dados na *cloud* (*e.g.,* largura de banda).	Potenciais quebras de segurança devido a ataques informáticos. Porque ainda está em expansão muitos riscos e *malwares* podem ainda ser desconhecidos.

3.1.2.2. *Value Proposition Canvas*

Para garantir que este projeto está de acordo com os valores e necessidades dos clientes foi elaborado um *Value Proposition Canvas*.

I. Perfil do Cliente

Enquadrada no primeiro bloco desta ferramenta, a tabela 3.2 apresenta a definição do perfil dos clientes considerando as tarefas que desempenham, as dores que sentem e os ganhos que procuram obter.

Tabela 3.2. Definição do perfil dos clientes.

A. Clientes	
Colaboradores da MOG Technologies. Clientes/Utilizadores do MAM4PRO.	
B. Tarefas do Cliente	
Instalação manual do MAM4PRO. Gestão manual dos servidores e infraestrutura necessária. Testes manuais e de usabilidade. Demonstrações a clientes.	
I. Dores	**II. Ganhos**
Processo maioritariamente manual. Processo trabalhoso e demorado. As equipas de desenvolvimento ficam várias vezes dependente de outras equipas para acesso a *hardware* para instalação do MAM4PRO ou versões de *software* específicas. Depuração de erros e resolução de problemas muitas vezes provocados por mudanças subtis nos ambientes.	Tempo e eficiência. Facilidade e agilidade nas tarefas do dia-a-dia. Rapidez na entrega de novas versões do *software*.

II. Proposta de Valor

Relacionado com o segundo bloco do *Value Proposition Canvas*, apresenta-se na tabela 3.3 a proposta de valor que a solução oferece aos seus clientes e de que forma esta alivia as dores e gera ganhos para o cliente.

Tabela 3.3. Definição da proposta de valor.

A. Produtos e Serviços	
Infraestrutura alojada nas instalações da MOG Technologies para *deployment* de aplicações. *Pipeline* de CI/CDE que permite automatizar todo o processo. *Dashboard* que permite o escalonamento dos serviços.	
I. Aliviam as Dores	**II. Criadores de Ganhos**
Com a automatização pretende-se que os clientes deixem de perder tanto tempo em processo manuais. Com a infraestrutura criada pretende-se que a implantação do *software* deixe de ser manual e passe a ser automática e com recurso a *containers*.	Ganho de tempo. Processo mais simplificado no dia-a-dia. Os administradores de sistemas ficam com um trabalho menos aborrecido.

3.1.3. Geração de Ideias

Uma vez tomada a decisão de prosseguir com a oportunidade identificada (*cf.,* secção 3.1.1) e analisada (*cf.,* secção 3.1.2), passou-se à terceira etapa do NCD que consiste na geração de ideias.

As principais ideias que foram debatidas estão relacionadas com as tecnologias para orquestração de *containers* no contexto do problema e com a forma de provisionamento da infraestrutura necessária. Relativamente ao primeiro ponto, foram consideradas várias tecnologias (*i.e.,* Kubernetes, Docker Swarm, OpenShift, Nomad e Marathon) que apresentam características que permitem atingir os objetivos propostos. No que respeita ao

segundo ponto, decidiu-se de forma relativamente unânime recorrer a técnicas de IaC para provisionamento da infraestrutura dadas as suas vantagens e características.

3.1.4. Seleção de Ideias

Na sequência da etapa anterior e dada a complexidade do sistema, percebeu-se que manter e gerir manualmente todos os *containers* necessários para uma instalação completa do MAM4PRO seria algo complexo e inviável. Como tal, e de acordo com as tecnologias estudadas, foi necessário optar por uma tecnologia de orquestração de *containers*. Para esta decisão utilizou-se um método de apoio à tomada de decisão normalmente utilizado nesta fase do modelo NCD: o AHP.

3.1.4.1. *Analytic Hierarchy Process*

O método AHP foi aplicado (*cf.,* apêndice C.3) ao problema em questão (*i.e.,* qual a melhor tecnologia de orquestração de *containers* a adotar) e foram seguidas as seis etapas que o compõe e que foram apresentadas na secção 2.7.1.4.

I. Definição dos Critérios

- **Complexidade:** nível de dificuldade associado à implementação, manutenção e curva de aprendizagem da tecnologia;
- **Documentação:** quantidade de documentação existente na internet e apoio da comunidade;
- **Desempenho:** nível de desempenho da tecnologia, tendo em conta fatores como o tempo de resposta, o consumo de recursos e a estabilidade;
- **Monitorização:** nível de monitorização que a tecnologia oferece;
- **Flexibilidade:** nível de flexibilidade da tecnologia, nomeadamente a compatibilidade de sistemas operativos, integração com as principais *clouds* públicas e serviços de terceiros;
- **Segurança:** nível de segurança e mecanismos de autenticação que a tecnologia oferece.

II. Definição das Tecnologias

Das várias tecnologias estudadas e apresentadas nesta dissertação (*cf.,* secção 2.8) foram selecionadas aquelas que atendem aos requisitos definidos. Para facilitar a análise e aplicação do método foi definida a nomenclatura da tabela 3.4 para os critérios e tecnologias escolhidas.

Tabela 3.4. Nomenclatura definida para os critérios e tecnologias consideradas.

Critério	Nomenclatura
Complexidade	Critério **A**
Documentação	Critério **B**
Desempenho	Critério **C**
Monitorização	Critério **D**
Flexibilidade	Critério **E**
Segurança	Critério **F**

Tecnologia	Nomenclatura
Kubernetes	Tecnologia **1**
Docker Swarm	Tecnologia **2**
OpenShift	Tecnologia **3**
Nomad	Tecnologia **4**
Marathon	Tecnologia **5**

III. Árvore Hierárquica de Decisão

Com o objetivo, critérios e tecnologias definidas, constrói-se a árvore hierárquica de decisão (figura 3.1) associando todos os critérios a todas as tecnologias.

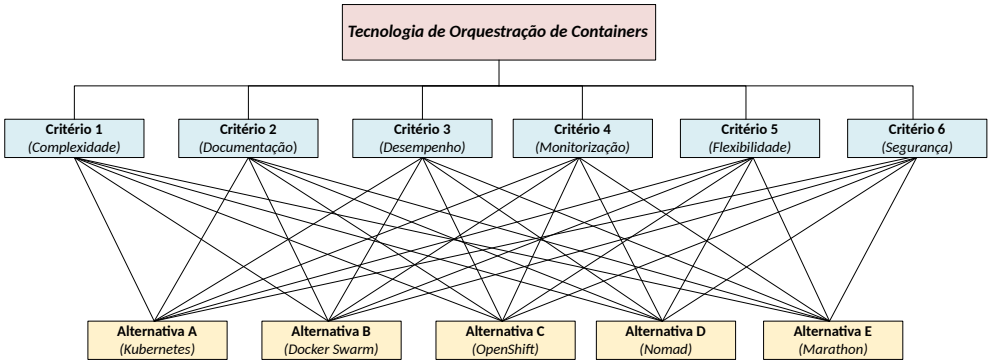


Figura 3.1. Árvore hierárquica de decisão que relaciona todos os critérios com todas as tecnologias.

IV. Comparação dos Critérios

Uma vez definidos os critérios a avaliar, estes são comparados entre si, utilizando a escala fundamental de Saaty (tabela C1). Através do cálculo das prioridades relativas de cada critério (tabela C3) obteve-se a matriz normalizada (P) e o vetor de prioridades relativas associado (X) (equação 5).

$$P = \begin{bmatrix} 0.09 & 0.24 & 0.07 & 0.07 & 0.11 & 0.08 \\ 0.01 & 0.04 & 0.05 & 0.07 & 0.07 & 0.02 \\ 0.45 & 0.28 & 0.36 & 0.36 & 0.34 & 0.38 \\ 0.09 & 0.04 & 0.07 & 0.07 & 0.07 & 0.08 \\ 0.27 & 0.20 & 0.36 & 0.36 & 0.34 & 0.38 \\ 0.09 & 0.20 & 0.07 & 0.07 & 0.07 & 0.08 \end{bmatrix} \quad X = \begin{bmatrix} 0.11 \\ 0.04 \\ \mathbf{0.36} \\ 0.07 \\ \mathbf{0.32} \\ 0.10 \end{bmatrix} \tag{5}$$

Observando o vetor X, concluiu-se que o **critério C** (0.36) e o **critério E** (0.32) são os que se revelam mais importantes para o problema identificado. Estas prioridades relativas justificam-se pelas características que uma solução de implantação de *software* deve possuir.

V. Comparação das Tecnologias

Depois de conhecido o vetor X, é necessário comparar as tecnologias de acordo com cada um dos critérios considerados. É de salientar que esta comparação é subjetiva e baseada apenas em informações recolhidas não tendo sido realizadas experiências práticas com o intuito de as comparar.

Critério A: Complexidade

A matriz A corresponde à matriz normalizada da comparação das tecnologias segundo o critério A e o vetor Y corresponde às prioridades relativas associadas (equação 6).

$$A = \begin{bmatrix} 0.06 & 0.08 & 0.02 & 0.09 & 0.09 \\ 0.29 & 0.38 & 0.44 & 0.35 & 0.45 \\ 0.24 & 0.08 & 0.09 & 0.09 & 0.09 \\ 0.24 & 0.38 & 0.36 & 0.35 & 0.27 \\ 0.18 & 0.08 & 0.09 & 0.12 & 0.09 \end{bmatrix} \quad Y = \begin{bmatrix} 0.07 \\ \mathbf{0.39} \\ 0.12 \\ \mathbf{0.32} \\ 0.11 \end{bmatrix} \tag{6}$$

Para o critério A, concluiu-se que:

- A **tecnologia 2** (0.39) e a **tecnologia 4** (0.32) são as que apresentam uma maior importância, o que nos permite afirmar que são as tecnologias menos complexas;
- A tecnologia que apresenta maior complexidade é a **tecnologia 1** (0.07), devido ao facto de a sua arquitetura ser complexa e a curva de aprendizagem acentuada.

Critério B: Documentação

A matriz B corresponde à matriz normalizada da comparação das tecnologias segundo o critério B e o vetor W corresponde às prioridades relativas associadas (equação 7).

$$B = \begin{bmatrix} 0.50 & 0.62 & 0.36 & 0.36 & 0.46 \\ 0.17 & 0.21 & 0.29 & 0.29 & 0.35 \\ 0.10 & 0.05 & 0.07 & 0.07 & 0.04 \\ 0.10 & 0.05 & 0.07 & 0.07 & 0.04 \\ 0.13 & 0.07 & 0.21 & 0.21 & 0.12 \end{bmatrix} \quad W = \begin{bmatrix} \mathbf{0.46} \\ \mathbf{0.26} \\ 0.07 \\ 0.07 \\ 0.15 \end{bmatrix} \tag{7}$$

Para o critério B, concluiu-se que:

- A **tecnologia 1** (0.46) apresenta maior importância, seguindo-se a **tecnologia 2** (0.26), sendo por isso estas as que apresentam maior apoio da comunidade e mais documentação disponível;
- As **tecnologias 3** e **4** (0.07) são as que apresentam menor importância.

Critério C: Desempenho

A matriz C corresponde à matriz normalizada da comparação das tecnologias segundo o critério C e o vetor Z corresponde às prioridades relativas associadas (equação 8).

$$C = \begin{bmatrix} 0.27 & 0.32 & 0.29 & 0.32 & 0.14 \\ 0.27 & 0.32 & 0.29 & 0.21 & 0.41 \\ 0.09 & 0.11 & 0.10 & 0.04 & 0.27 \\ 0.09 & 0.16 & 0.29 & 0.11 & 0.05 \\ 0.27 & 0.11 & 0.05 & 0.32 & 0.14 \end{bmatrix} \quad Z = \begin{bmatrix} \mathbf{0.27} \\ \mathbf{0.30} \\ 0.12 \\ 0.14 \\ 0.18 \end{bmatrix} \tag{8}$$

Para o critério C, concluiu-se que:

- A **tecnologia 2** (0.30) é aquela que apresenta melhor desempenho, seguindo-se a **tecnologia 1** (0.27);
- A **tecnologia 3** é a que apresenta pior desempenho (0.12).

O critério C foi classificado como sendo um dos mais importantes para a solução e por isso a comparação das tecnologias segundo este critério é bastante importante.

Critério D: Monitorização

A matriz D corresponde à matriz normalizada da comparação das tecnologias segundo o critério D e o vetor K corresponde às prioridades relativas associadas (equação 9).

$$D = \begin{bmatrix} 0.45 & 0.33 & 0.47 & 0.47 & 0.47 \\ 0.09 & 0.07 & 0.05 & 0.05 & 0.05 \\ 0.15 & 0.20 & 0.16 & 0.16 & 0.16 \\ 0.15 & 0.20 & 0.16 & 0.16 & 0.16 \\ 0.15 & 0.20 & 0.16 & 0.16 & 0.16 \end{bmatrix} \quad K = \begin{bmatrix} \mathbf{0.44} \\ 0.06 \\ \mathbf{0.17} \\ \mathbf{0.17} \\ \mathbf{0.17} \end{bmatrix} \tag{9}$$

Para o critério D, concluiu-se que:

- A **tecnologia 1** (0.44) destaca-se das restantes sendo por isso a que apresenta melhores mecanismos de monitorização;
- A que apresenta pior sistema de monitorização é a **tecnologia 2**.

Critério E: Flexibilidade

A matriz E corresponde à matriz normalizada da comparação das tecnologias segundo o critério E e o vetor L corresponde às prioridades relativas associadas (equação 10).

$$E = \begin{bmatrix} 0.40 & 0.43 & 0.41 & 0.38 & 0.41 \\ 0.08 & 0.09 & 0.18 & 0.08 & 0.18 \\ 0.06 & 0.03 & 0.06 & 0.08 & 0.06 \\ 0.40 & 0.43 & 0.29 & 0.38 & 0.29 \\ 0.06 & 0.03 & 0.06 & 0.08 & 0.06 \end{bmatrix} \quad L = \begin{bmatrix} \mathbf{0.41} \\ 0.12 \\ 0.06 \\ \mathbf{0.36} \\ 0.06 \end{bmatrix} \tag{10}$$

Para o critério E, concluiu-se que:

- A **tecnologia 1** (0.41) é a que apresenta maior importância, seguindo-se da **tecnologia 4** (0.36);
- O facto de a **tecnologia 1** ser mais apoiada pela comunidade e apresentar um bom desempenho, faz com que existam inúmeras integrações com sistemas computacionais e que o nível de adaptabilidade a diferentes cenários seja também mais elevado;
- As tecnologias que apresentam menor importância relativa são as **tecnologias 3** e **4**;

À semelhança do critério C, também o critério E foi classificado como sendo um dos mais importantes para a solução a implementar.

Critério F: Segurança

A matriz F corresponde à matriz normalizada da comparação das tecnologias segundo o critério F e o vetor M corresponde às prioridades relativas associadas (equação 11).

$$F = \begin{bmatrix} 0.37 & 0.33 & 0.37 & 0.33 & 0.39 \\ 0.07 & 0.07 & 0.07 & 0.07 & 0.04 \\ 0.37 & 0.33 & 0.37 & 0.33 & 0.39 \\ 0.07 & 0.07 & 0.07 & 0.07 & 0.04 \\ 0.12 & 0.20 & 0.12 & 0.20 & 0.13 \end{bmatrix} \quad M = \begin{bmatrix} \mathbf{0.36} \\ 0.06 \\ \mathbf{0.36} \\ 0.06 \\ 0.15 \end{bmatrix} \tag{11}$$

Para o critério F, concluiu-se que:

- As tecnologias que aparentam responder melhor a esta necessidade são as **tecnologias 1** e **3** (0.36);
- As **tecnologias 2** e **4** são as que apresentam menor segurança.

VI. Matriz Composta de Comparação

Após comparação das tecnologias segundo os vários critérios é possível obter uma matriz (H) que relaciona todas as tecnologias com todos os critérios (equação 12).

$$H = \begin{bmatrix} 0.07 & 0.46 & 0.27 & 0.44 & 0.41 & 0.36 \\ 0.39 & 0.26 & 0.30 & 0.06 & 0.12 & 0.06 \\ 0.12 & 0.07 & 0.12 & 0.17 & 0.06 & 0.36 \\ 0.32 & 0.07 & 0.14 & 0.17 & 0.36 & 0.06 \\ 0.11 & 0.15 & 0.18 & 0.17 & 0.06 & 0.15 \end{bmatrix} \tag{12}$$

Com base na matriz H e no vetor X calcula-se o vetor de prioridades relativas (N) que permite identificar qual a melhor tecnologia para resolução do problema em estudo (equação 13). Estas prioridades podem ser obtidas através da multiplicação da matriz H pelo vetor X.

$$N = \begin{bmatrix} \mathbf{0.32} \\ 0.21 \\ 0.12 \\ 0.22 \\ 0.13 \end{bmatrix} \tag{13}$$

Analisando o vetor N concluí-se que a tecnologia a adotar para orquestração de *containers* deverá ser a **tecnologia 1 (Kubernetes)**, uma vez que é a que apresenta maior importância.

VII. Validação do Método

Para validar o método é necessário garantir a consistência do mesmo. Utilizando a matriz de prioridades dos critérios (P) e o vetor de prioridades relativas de cada um dos critérios (X), calcula-se:

$$P * X = \lambda_{max} * X \iff \lambda_{max} \approx 6.45 \tag{14}$$

Aplicando a equação 3 (*cf.,* secção 2.7.1.4), vem que:

$$IC = \frac{\lambda_{max} - n}{n - 1} \iff \frac{6.45 - 6}{6 - 1} \approx 0.09 \tag{15}$$

Obtido o valor do IC calcula-se então a RC. O valor do IA depende do valor de n (tabela C2).

$$RC = \frac{IC}{IA} \iff RC = \frac{0.09}{1.24} \approx 0.07 \tag{16}$$

Uma vez que $RC < 0.1$, os dados consideram-se consistentes e o resultado do método AHP válido, optando-se assim pela adoção da **tecnologia 1 (Kubernetes)** para resolução do problema identificado.

3.2. *Quality Function Deployment*

Após a identificação da oportunidade e a escolha da melhor tecnologia a utilizar, torna-se importante perceber de que forma esta solução irá contribuir com o acréscimo de valor para a organização e para os clientes. Conforme apresentado na secção 2.7.1.5 foi executada a primeira fase do método QFD (*i.e.,* definição do produto) com o auxílio da ferramenta HOQ.

3.2.1. Necessidades do Cliente

- **Fácil manutenção:** a solução deve ser de fácil manutenção, não apenas para os administradores, mas também para os desenvolvedores;
- **Rápido:** a implantação do *software* e criação da infraestrutura deve ser rápida;

- **Seguro:** devem estar garantidos os mecanismos fundamentais de segurança da infraestrutura;
- **Estável:** a solução deve ser estável e com o mínimo de falhas possíveis;
- **Utilização simples:** qualquer pessoa que utilize a solução deve conseguir usá-la com facilidade;
- **Escalável:** deve ser possível adaptar a solução a cenários complexos sem grande impacto no desempenho;
- **Automático:** a criação da infraestrutura e o processo de implantação do *software* deve ser o mais automático possível;
- **Flexível:** deve ser possível usar a solução em vários ambientes computacionais.

Após a identificação das necessidades do cliente, é necessário classificar a sua importância do ponto de vista do cliente (tabela 3.5). As classificações são apresentadas normalizadas de forma a obter em percentagem o grau de importância de cada uma das necessidades.

Tabela 3.5. Definição das importâncias das necessidades do cliente.

	Importância	**%**
Fácil Manutenção	4	12.5%
Rápido	5	15.63%
Seguro	3	9.38%
Estável	4	12.50%
Utilização Simples	3	9.38%
Escalável	4	12.50%
Automático	5	15.63%
Flexível	4	12.50%

Analisando a tabela 3.5 conclui-se que as necessidades rápido e automático são as mais importantes, enquanto que as necessidades seguro e utilização simples são as menos importantes.

3.2.2. Características de Engenharia

O próximo passo é traduzir as necessidades identificadas em características de Engenharia:

- **Boas práticas de Engenharia**: característica associada às boas práticas aplicadas no desenho da solução e no desenvolvimento do código;
- **Tempo de implantação**: característica associada ao tempo necessário para criação da infraestrutura e implantação do *software*;
- **Autenticação**: característica associada à segurança da solução e mecanismos de autenticação;
- **Estabilidade**: característica associada à estabilidade da solução, tempo de disponibilidade, capacidade de resposta e recuperação de falhas;
- **Usabilidade**: característica associada à capacidade de os utilizadores conseguirem usufruir de toda a solução de forma eficaz, eficiente e satisfatória;

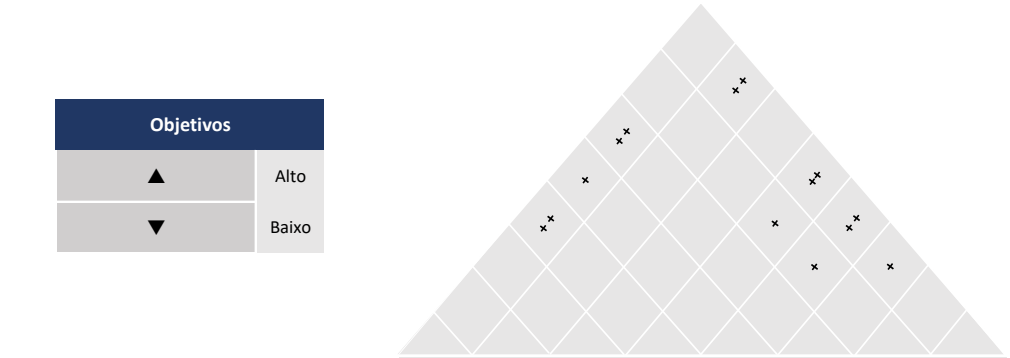
- **Escalabilidade**: característica associada à possibilidade de escalonamento dos serviços, procurando manter a disponibilidade do serviço em cenários de maior uso;
- **Automatização**: característica associada à capacidade de implantação do *software* e criação da infraestrutura de forma mais automática possível;
- **Compatibilidade entre sistemas**: a solução deve ser compatível com o máximo de sistemas computacionais possíveis.

Após a definição das características de Engenharia, é necessário estabelecer uma relação entre estas e as necessidades do cliente, e analisar a correlação entre as características de Engenharia definidas.

Normalmente, é realizada também uma análise da concorrência, no entanto no caso desta dissertação não foi realizada essa análise por não existirem soluções semelhantes que pudesse servir de comparação.

Analisando a HOQ preenchida (figura 3.2) conclui-se que:

- As necessidades do cliente **rápido** e **automático** são as que se relevam mais importantes, com uma percentagem de 15.63%, enquanto que a necessidade seguro e utilização simples são as que apresentam menor importância, com uma percentagem de 9.38%;
- Das características de Engenharia definidas, a **automatização** é a característica que apresenta a maior importância (26,75%), seguindo-se a **escalabilidade** (17.53%) e **compatibilidade entre sistemas** (12.18%). A característica com menor importância é a mecanismos de autenticação (4.98%), que se justifica pelo facto de a necessidade seguro ser a com menor importância do ponto de vista do cliente.



Objetivos	
▲	Alto
▼	Baixo

		Importância para o Cliente Baixa (1); Alta (5)	Importância para o Cliente (%)	Boas Práticas de Engenharia	Tempo de Implantação	Mecanismos de Autenticação	Estabilidade	Usabilidade	Escalabilidade	Compatibilidade entre Sistemas	Automatização (CI/CD)
				Características de Engenharia (Como?)							
				▲	▼	▲	▲	▲	▲	▲	▲
Necessidades do Cliente (O quê?)	Fácil Manutenção	4	12.50%	●	▽	-	▽	○	▽	▽	○
	Rápido	5	15.63%	-	●	-	▽	-	○	-	●
	Seguro	3	9.38%	-	-	●	-	-	-	○	-
	Estável	4	12.50%	-	▽	-	●	-	●	▽	-
	Utilização Simples	3	9.38%	○	▽	-	-	●	-	○	●
	Escalável	4	12.50%	-	▽	-	▽	-	●	▽	○
	Flexível	4	12.50%	○	-	-	-	▽	▽	●	▽
	Automático	5	15.63%	-	-	-	-	-	-	-	●
			100%								
	Importância Relativa			1.78	1.88	0.84	1.53	1.34	2.97	2.06	4.53
	Importância (%)			10.52%	11.07%	4.98%	9.04%	7.93%	17.53%	12.18%	26.75%

Matriz de Correlação		
+ +	Fortemente Positivo	9
+	Positivo	3
-	Negativo	-1
	Sem Correlação	0

Matriz de Relação		
●	Forte	9
○	Médio	3
▽	Fraco	1
-	Sem Relação	0

Figura 3.2. Preenchimento da ferramenta HOQ. As necessidades rápido e automático são as mais importantes do ponto de vista do cliente. As necessidades seguro e utilização simples são as que apresentam menor importância para o cliente. Das características de Engenharia definidas, a automatização é a que apresenta maior importância e a mecanismos de autenticação a que apresenta menor importância.

3.3. Engenharia de Requisitos

A Engenharia de Requisitos é uma disciplina da Engenharia de *Software* e consiste no processo de definir, documentar e manter os requisitos de um sistema atendendo a um determinado negócio [58]. Normalmente divide-se nas seguintes fases: levantamento de requisitos, especificação de requisitos, validação de requisitos e gestão de requisitos.

Os requisitos de um sistema podem ser classificados como funcionais ou não funcionais. Os *Requisitos Funcionais* (RF) resultam da fase de levantamento de requisitos, onde foram analisadas as necessidades da organização de acordo com o problema levantado (*cf.,* secção 1.2), os objetivos formulados (*cf.,* secção 1.3) e especificam as funcionalidades que o sistema deve ser capaz de realizar do ponto de vista do utilizador[1]. Os *Requisitos Não Funcionais* (RNF) são requisitos que não estão diretamente relacionados com funcionalidades do sistema e representam um papel importante no desenvolvimento de um sistema, ajudando a definir estilos de arquitetura e formas de implementação do mesmo.

3.3.1. Atores do Sistema

Em notação *Unified Modeling Language* (UML), define-se como ator do sistema os utilizadores e/ou meios externos que desempenham algum papel em relação ao sistema. Os meios externos representam *hardware* e/ou *software* que, tal como os utilizadores, geram informações para o sistema ou necessitam de informações geradas a partir do sistema[2]. Os atores são os que desempenham os casos de uso e um ator pode estar presente em mais do que um caso de uso. Devem possuir um nome sugestivo, preferencialmente relacionado com a sua função.

No âmbito desta dissertação e na forma de operar da MOG Technologies, definiram-se três atores do sistema:

- **Administrador**: elementos da organização que tenham acesso e autorização para gerir, manter e administrar toda a infraestrutura desta solução;
- **Desenvolvedor**: qualquer elemento das equipas de desenvolvimento que desenvolva requisitos funcionais e não funcionais para a solução;
- **Utilizador**: qualquer pessoa que utilize o MAM4PRO, pode ser um colaborador da MOG Technologies ou um cliente autorizado.

3.3.2. Requisitos do Sistema

Durante a fase de análise deste projeto e através de reuniões com o supervisor da empresa e outros colaboradores da MOG Technologies, definiram-se os seguintes requisitos do sistema:

- Criar toda a infraestrutura através de uma *pipeline* de CI/CDE;
- Bloquear o estado da infraestrutura, garantindo que não é possível executar *pipelines* de CI/CDE que possam destruir a infraestrutura de forma inesperada;
- Destruir toda a infraestrutura através de uma *pipeline* de CI/CDE;
- Implantar o MAM4PRO através de uma *pipeline* de CI/CDE;

[1] https://www.devmedia.com.br/artigo-engenharia-de-software-3-requisitos-nao-funcionais/9525

[2] http://www.linhadecodigo.com.br/artigo/853/uml-unified-modeling-language-atores-atividades-e-componentes.aspx

- Aceder ao MAM4PRO através da internet sem necessidade de instalação de *software*;
- Possibilidade de escalar o número de serviços (*i.e, pods*) de um *deployment*;
- Permitir a visualização de métricas (utilização do CPU, utilização da memória *Random Access Memory* (RAM) e disco) relacionadas com os servidores que compõe a infraestrutura;
- Embora com foco na implantação do MAM4PRO, a solução implementada também deve suportar a implantação de outras aplicações (*e.g.,* Uptime Kuma);
- Devem existir mecanismos de autenticação e segurança no acesso à infraestrutura;
- A solução implementada deve ser fácil de manter, facilitando a adição de novas funcionalidades;
- A solução implementada deve ser simples de usar por todas as pessoas que não tenham total conhecimento de como as funcionalidades estão implementadas;
- A solução deve ser resiliente a lidar com erros, devendo existir mecanismos de redundância e alta disponibilidade;
- A solução implementada deve ser rápida e eficiente;
- O código desenvolvido deve ser escrito em inglês e deve seguir as boas práticas de Engenharia;
- Deve ser desenvolvida documentação de apoio escrita em inglês, para os vários utilizadores da solução;
- O código da solução deve ser enviado para um repositório no GitLab;
- Todos os mecanismos de automação utilizados devem ser implementados com as ferramentas que o GitLab dispõe;
- A solução implementada deve suportar a execução de *containers* Windows e Linux.

3.3.2.1. Requisitos Funcionais

Analisando a lista de requisitos apresentada anteriormente (*cf.,* secção 3.3.2) é possível verificar que apenas os seguintes são classificados como RF da solução:

- **RF. 1:** Criar toda a infraestrutura através de uma *pipeline* de CI/CDE;
- **RF. 2:** Bloquear o estado da infraestrutura, garantindo que não é possível executar *pipelines* de CI/CDE que possam destruir a infraestrutura de forma inesperada;
- **RF. 3:** Destruir toda a infraestrutura através de uma *pipeline* de CI/CDE;
- **RF. 4:** Implantar o MAM4PRO através de uma *pipeline* de CI/CDE;
- **RF. 5:** Aceder ao MAM4PRO através da internet sem necessidade de instalação de *software*;
- **RF. 6:** Possibilidade de escalar o número de serviços (*i.e.,* pods) de um *deployment*;
- **RF. 7:** Permitir a visualização de métricas (utilização do CPU, utilização da memória RAM e disco) relacionadas com os servidores que compõe a infraestrutura;

- **RF. 8:** Embora com foco na implantação do MAM4PRO, a solução implementada também deve suportar a implantação de outras aplicações (*e.g.,* Uptime Kuma).

3.3.2.2. Requisitos Não Funcionais

Da lista de requisitos do sistema que foram definidos na secção 3.3.2, os RNF foram classificados de acordo com o modelo FURPS+ (*cf.,* secção 2.7.2) e encontram-se descritos na tabela 3.6.

Tabela 3.6. Requisitos não funcionais da solução classificados segundo o modelo FURPS+.

Categoria	Requisito Não Funcional
Funcionalidade	RNF. 1 - Devem existir mecanismos de autenticação e segurança no acesso à infraestrutura.
Usabilidade	RNF. 2 - A solução implementada deve ser fácil de manter, facilitando a adição de novas funcionalidades. RNF. 3 - A solução implementada deve ser simples de usar por todas as pessoas que não tenham total conhecimento de como as funcionalidades estão implementadas.
Confiabilidade	RNF. 4 - A solução deve ser resiliente a lidar com erros, devendo existir mecanismos de redundância e alta disponibilidade.
Desempenho	RNF. 5 - A solução implementada deve ser rápida e eficiente.
Suporte	RNF. 6 - O código desenvolvido deve ser escrito em inglês e deve seguir as boas práticas de Engenharia. RNF. 7 - Deve ser desenvolvida documentação de apoio escrita em inglês, para os vários utilizadores da solução.
Adicional (+)	RNF. 8 - O código da solução deve ser enviado para um repositório no GitLab. RNF. 9 - Todos os mecanismos de automação utilizados devem ser implementados com as ferramentas que o GitLab dispõe. RNF. 10 - A solução implementada deve suportar a execução de *containers* Windows e Linux.

3.3.3. Casos de Uso

Em notação UML um diagrama de casos de uso resume os detalhes dos atores do sistema e as suas interações. A figura 3.3 representa o diagrama de casos de uso do sistema com uma visão dos atores e a sua relação com os RF do sistema definidos. Adicionalmente, cada um dos casos de uso definidos está relacionado com um RF tal como demonstra a tabela 3.7.

Tabela 3.7. Relação entre os casos de uso definidos e os requisitos funcionais da solução.

Caso de Uso	Requisito Funcional (RF)
UC. 1	RF. 1 - Criar toda a infraestrutura através de uma *pipeline* de CI/CDE.
UC. 2	RF. 2 - Bloquear o estado da infraestrutura, garantindo que não é possível executar *pipelines* de CI/CDE que possam destruir a infraestrutura de forma inesperada.
UC. 3	RF. 3 - Destruir toda a infraestrutura através de uma *pipeline* de CI/CDE.
UC. 4	RF. 4 - Implantar o MAM4PRO através de uma *pipeline* de CI/CDE.
UC. 5	RF. 5 - Aceder ao MAM4PRO através da internet sem necessidade de instalação de *software*.
UC. 6	RF. 6 - Possibilidade de escalar o número de serviços (*i.e., pods*) de um *deployment*.
UC. 7	RF. 7 - Permitir a visualização de métricas (utilização do CPU, utilização da memória RAM e disco) relacionadas com os servidores que compõe a infraestrutura.
UC. 8	RF. 8 - Embora com foco na implantação do MAM4PRO, a solução implementada também deve suportar a implantação de outras aplicações (*e.g.,* Uptime Kuma).

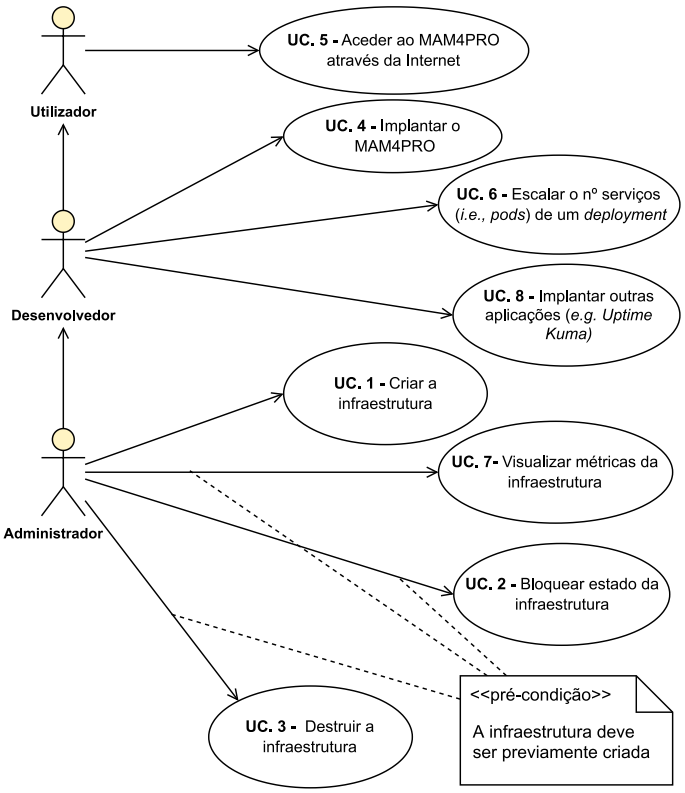


Figura 3.3. Diagrama de casos de uso do sistema com relação aos atores do sistema definidos.

3.4. Sumário

Neste capítulo foi realizada uma análise do ponto de vista do negócio, com destaque nos principais pontos fortes e fracos do projeto recorrendo à Análise SWOT (*cf.,* secção 3.1.2.1) e ao *Value Proposition Canvas* (*cf.,* secção 3.1.2.2) tendo-se percebido que o problema identificado pode ser combatido com a solução apresentada. Foram também aplicados dois métodos de análise de valor: o AHP (*cf.,* secção 3.1.4.1) que permitiu concluir que a tecnologia Kubernetes é a mais adequada como tecnologia de orquestração de *containers* no contexto do problema, e a ferramenta HOQ (*cf.,* secção 3.2) que permitiu concluir que as necessidades rápido e automático são as mais importantes para o cliente. Por último foram definidos os principais requisitos funcionais (*cf.,* secção 3.3.2.1) e não funcionais (*cf.,* secção 3.3.2.2) do sistema relacionando-os com os casos de uso definidos (*cf.,* secção 4.2).

No capítulo seguinte, intitulado Desenho da Solução, é desenhada e projetada toda a solução para o problema identificado.

Capítulo 4

Desenho da Solução

A arquitetura de um sistema corresponde ao conjunto de estruturas fundamentais que compreende os elementos de *software*, as suas propriedades, e as relações entre eles. Assim sendo, a arquitetura da solução tem como objetivo relacionar a abstração dos objetivos do sistema com o sistema em concreto. Uma arquitetura de *software* pode ser desenhada, analisada, documentada e implementada utilizando técnicas que fornecem suporte para atingir os objetivos abstratos e complexos definidos para o sistema, tornando a definição do sistema mais tangível [59].

Pretende-se com este capítulo descrever o desenho da solução implementada apresentando duas perspetivas: uma de mais alto nível, onde se pretende demonstrar e explicar os fluxos de trabalho dos vários atores do sistema, e uma de mais baixo nível, onde se pretende demonstrar de forma mais detalhada e granular as interações entre os diferentes componentes no contexto dos casos de uso definidos (*cf.,* secção 3.3.3).

4.1. Desenho de Alto Nível

Tal como referido na secção 3.3.1, a solução possui vários atores do sistema e cada um deles possui responsabilidades e funções bem definidas e que se enquadram em fluxos de trabalho distintos, nomeadamente:

A. Criação e gestão da infraestrutura: processo de desenvolvimento do código que descreve e provisiona toda a infraestrutura através de técnicas de IaC.

Ator do sistema: Administrador.

B. Codificação e desenvolvimento do MAM4PRO: processo de desenvolvimento do MAM4PRO.

Ator do sistema: Desenvolvedor.

C. Acesso e utilização do MAM4PRO: acesso e utilização do *software* como um serviço.

Ator do sistema: Utilizador.

Os três fluxos de trabalho (A, B e C) descritos permitem visualizar e demonstrar a aplicabilidade de toda a solução. Esta dissertação foca-se essencialmente na implementação do fluxo A (*i.e.,* criação e gestão da infraestrutura). No entanto, foram implementadas algumas melhorias nos restantes fluxos de forma a fomentar uma cultura de DevOps na organização e otimizar a forma como o *software* é distribuído.

Com o objetivo de separar conceitos e aplicar as boas práticas de Engenharia, definiu-se, em conjunto com o supervisor da empresa, a utilização de dois repositórios de Git distintos: o primeiro designado *IaC Repository*, que contém todo o código relacionado com a infraestrutura, e o segundo designado *Apps Repository*, que contém todos os ficheiros necessários para implantação do *software*. De salientar, que o código do MAM4PRO não estará presente em nenhum destes repositórios referidos, mas sim num repositório já existente, adiante designado *Mxfspeedrail Repository*.

4.1.1. Criação e Gestão da Infraestrutura (A)

A infraestrutura que irá suportar o *deployment* do *software* deve ser provisionada recorrendo a técnicas de IaC e tudo deve ficar armazenado em ficheiros de código. Estes ficheiros de código devem ser enviados para um repositório partilhado (*IaC Repository*) e deve ser criada uma *pipeline* de CI/CDE capaz de validar e criar de forma automática a infraestrutura descrita.

O atual processo de desenvolvimento de *software* da MOG Technologies assenta no *Git Flow* (*cf.*, secção 2.8.1.1) e, como tal, todo o processo de codificação da infraestrutura (*development*) deve também seguir este fluxo. Assim que o código está desenvolvido, deve ser revisto por outras pessoas da equipa (*continuous integration*) e integrado na *branch* principal. A *pipeline* de CI/CDE referente a este repositório é responsável por validar todos os ficheiros de código e provisionar toda a infraestrutura (*provisioning*). A figura 4.1 representa este fluxo de trabalho.

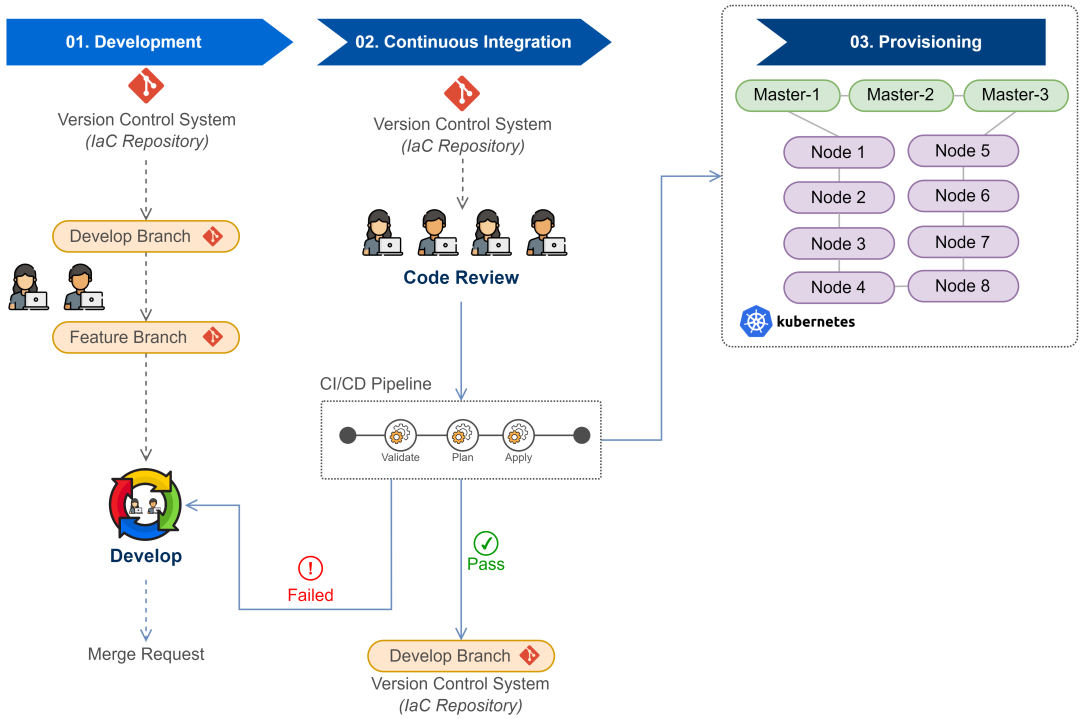


Figura 4.1. Representação do fluxo de trabalho de um administrador, composto por três etapas distintas: *development* (codificação e desenvolvimento do código da infraestrutura), *continuous integration* (integração do código-fonte no sistema de controlo de versões) e *provisioning* (criação da infraestrutura).

4.1.1.1. Rede

A solução a implementar possui uma forte componente de rede, sendo de extrema importância que todos os componentes da infraestrutura comuniquem entre si. Adicionalmente, por ser uma solução para implementar em contexto empresarial, e devendo as aplicações ficar expostas à internet, definiu-se, juntamente com as pessoas envolvidas neste projeto, que se deveriam adotar algumas medidas de segurança.

Para isso foi criada uma zona de rede desmilitarizada[1] ou rede de perímetro em que o principal objetivo é adicionar uma camada extra de proteção à rede [53]. Na prática aplicando o conceito de DMZ foi criada uma *Virtual Local Area Network* (VLAN) isolada pela equipa de IT da MOG Technologies destinada apenas aos servidores desta solução. Estes servidores não conseguem comunicar através da rede interna da MOG Technologies, ou seja, qualquer acesso às aplicações implantadas nesta infraestrutura terá de ser através da internet e nunca pela rede interna.

Esta decisão levanta um desafio ao provisionamento, uma vez que a infraestrutura é provisionada recorrendo a uma *pipeline* de CI/CDE, e não tendo esta VLAN acesso à rede interna da MOG Technologies, o *runner* GitLab não irá conseguir aceder aos servidores da infraestrutura. Para combater este problema, decidiu-se que o servidor `M1-Todd` seria então o *runner* responsável pela execução das *pipelines* de CI/CDE e conseguiria comunicar com as duas redes, através de duas interfaces de rede.

A figura 4.2 ilustra os componentes principais da rede e a comunicação com a infraestrutura provisionada.

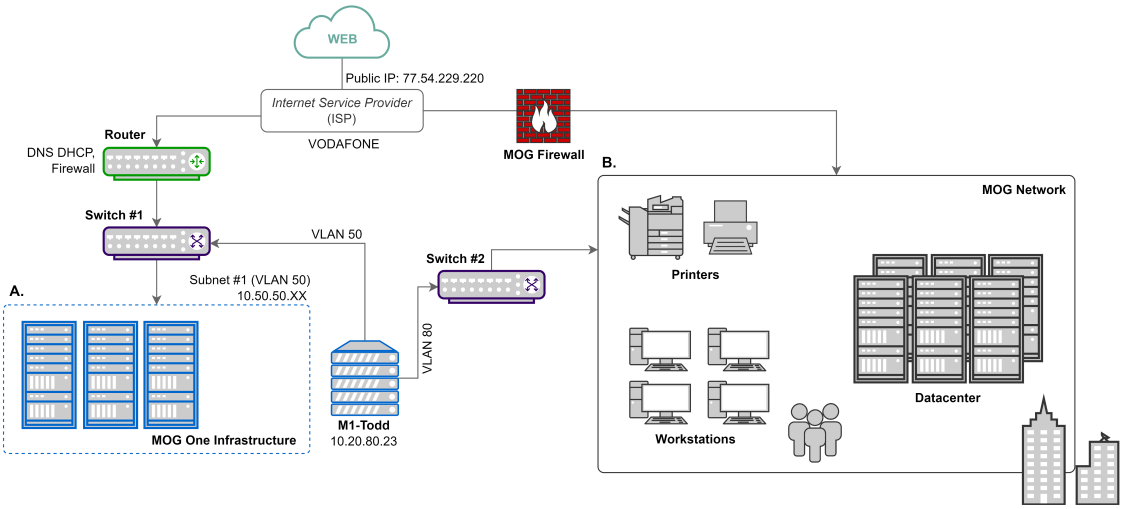


Figura 4.2. Diagrama representativo da rede implementada. (A) Infraestrutura desenvolvida no âmbito desta dissertação. (B) Rede interna da MOG Technologies, com vários tipos de dispositivos e várias sub-redes.

4.1.1.2. Topologia do *Cluster* Kubernetes

Na secção 2.8 foi estudada a arquitetura base e os componentes que um *cluster* Kubernetes possui. No entanto, cada caso de uso tem as suas especificações que obrigam a escolher uma topologia para o *cluster*.

[1] do inglês, *De-Militarized Zone* (DMZ).

Juntamente com o supervisor e a restante equipa da MOG Technologies definiram-se os seguintes pontos a ter em consideração no desenho da solução a implementar:

- Devem ser criados três servidores que desempenham papel de *masters* e implementados mecanismos de redundância e alta disponibilidade, para garantir que a falha de um componente não implica a indisponibilidade do *software*. Para o efeito devem ser utilizados os servidores referidos no apêndice E.3 e as seguintes tecnologias:
  - *HAProxy (load balancing)*: para realizar o balanceamento da carga entre os vários *masters*;
  - *Keepalived (health checker)*: para verificar a cada período de tempo a disponibilidade de cada *master*;
  - *Flannel*: para implementação do modelo de rede sugerido pelo Kubernetes.
- Não existe obrigatoriedade de utilização de máquinas virtuais, no entanto, por simplicidade e para facilitar a criação de *backups*, os *masters* do *cluster* devem ser instanciados recorrendo a VMs;
- Não existem restrições relevantes para os *nodes*, devem ser utilizados os servidores referidos no apêndice E.3.

Cada *master* é composto por vários componentes e cada um deles com responsabilidades bem definidas. A figura 4.3 ilustra a arquitetura dos três *masters* da solução.

Cada servidor Windows hospeda uma VM Linux onde são instanciados os componentes do Kubernetes (*i.e., API Server*, *Controller Manager*, *Scheduler* e *Etcd*), o *Flannel*, o *HAProxy* e o *Keepalived*.

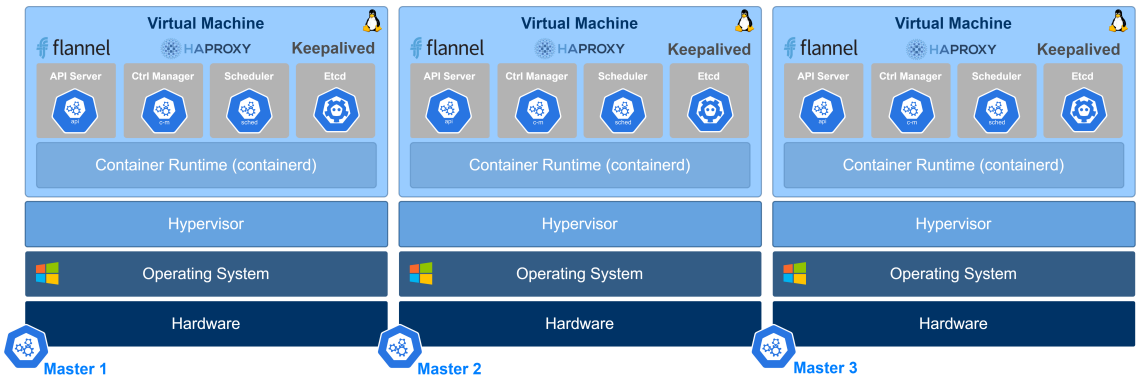


Figura 4.3. Representação de alto nível dos três *masters* que compõe o *cluster*.

Não existem restrições relevantes para os *nodes*, por isso, cada *node* será instanciado diretamente num servidor físico. Dado o requisito de o *cluster* a implementar ser híbrido, isto é, possibilitar a execução de *containers* Windows e Linux, os sistemas operativos dos *nodes* podem ser distintos entre eles.

Cada *node* possui o *Kubelet* e o *Kube Proxy* instalado que permitem gerir e manter os vários *pods* em execução no *cluster* (figura 4.4).

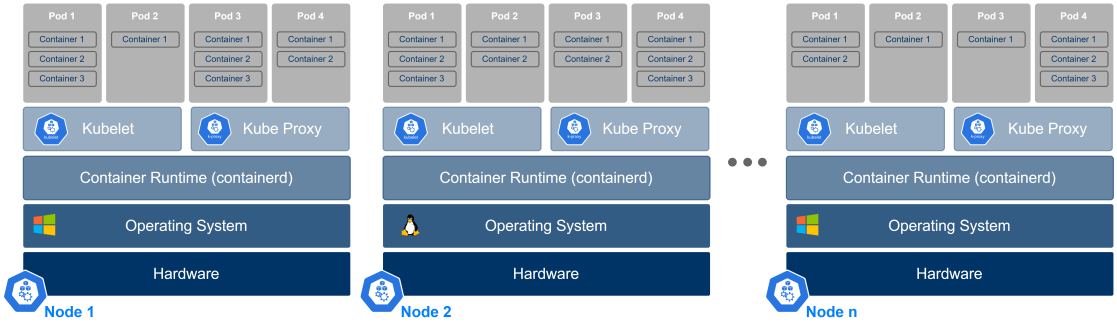


Figura 4.4. Representação de alto nível dos *nodes* que compõe o *cluster* Kubernetes.

4.1.1.3. *Pipeline Continuous Integration/Continuous Delivery*

Para que o provisionamento da infraestrutura seja automatizado e de forma a satisfazer um dos objetivos principais desta dissertação, deve ser criada uma *pipeline* de CI/CDE capaz de provisionar a infraestrutura (figura 4.5). Esta *pipeline* deve ser composta por quatro *stages* e devem existir dependências entres os vários *jobs*, isto é, só faz sentido provisionar os *nodes* do *cluster* caso os *masters* já tenham sido provisionados.

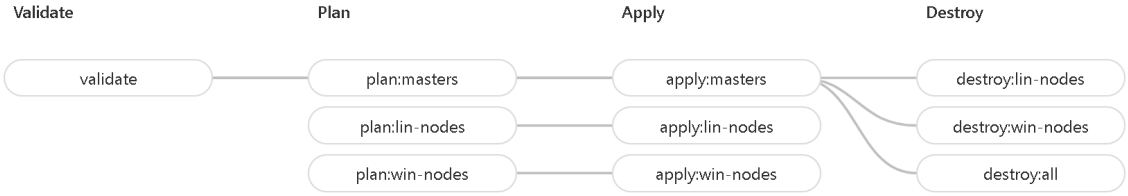


Figura 4.5. *Pipeline* CI/CDE de provisionamento e destruição da infraestrutura criada.

4.1.2. Codificação e Desenvolvimento do MAM4PRO (B)

Composto por quatro fases distintas (*development*, *continuous integration*, *QA* e *deployment*) o processo de codificação e desenvolvimento do MAM4PRO assenta em *Git Flow* (*cf.,* secção 2.8.1.1). Todas as alterações realizadas no código-fonte da aplicação são enviadas para um repositório partilhado, designado *Mxfspeedrail Repository* (*development*), que acionam uma *pipeline* de CI/CDE que além de compilar todo o código, executa testes e valida que o *software* está pronto para entrar na *branch* principal (*continuous integration*). Posteriormente a equipa de *Quality Assurance* garante que não existem comportamentos inesperados e que a nova versão do *software* pode ser colocada em produção (*QA testing*). Por último, a nova versão do *software* é colocada em execução na infraestrutura criada no ponto A (*cf.,* secção 4.1.1) através da execução manual de uma *pipeline* de CI/CDE do *Apps Repository* (*deployment*). A figura 4.6 demonstra todo este processo e como as várias fases se interligam.

À exceção do armazenamento e publicação dos *containers* no *registry*, todo este fluxo já se encontrava implementado na MOG Technologies. Até à data, sempre que existia uma nova versão do *software* pronta para produção a *pipeline* de CI/CDE gerava num instalador pronto a executar. Agora, como o objetivo é disponibilizar o MAM4PRO como um serviço, o resultado final desta *pipeline* são *containers*, que são enviados e armazenados num *registry*.

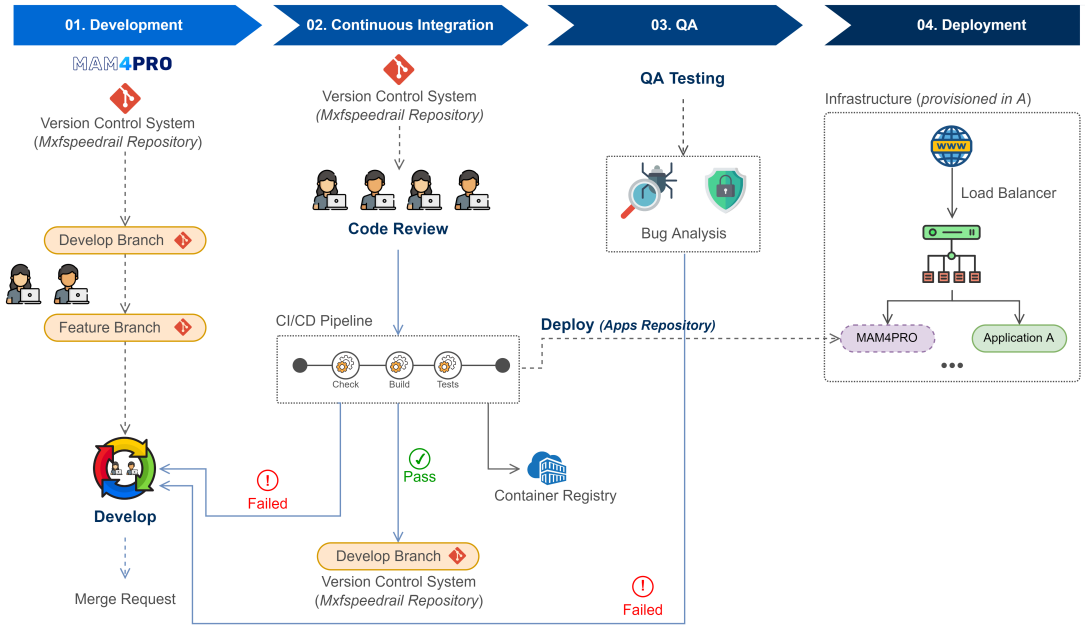


Figura 4.6. Representação do fluxo de trabalho de um desenvolvedor, composto por quatro etapas distintas: *development* (codificação e desenvolvimento de toda a aplicação), *continuous integration* (integração do código-fonte no sistema de controlo de versões garantindo que novas versões estão prontas para produção), *QA testing* (validação e realização de testes pela equipa de *Quality Assurance*) e *deployment* (colocação de novas versões em produção).

4.1.3. Acesso e Utilização do MAM4PRO (C)

Além dos clientes finais da MOG Technologies também alguns colaboradores necessitam de aceder a *deployments* do MAM4PRO, por exemplo a equipa de *Sales* para realizar demonstrações a clientes, ou a equipa de *Quality Assurance* para realizar testes manuais. Atualmente, o processo é semelhante ao que acontece quando uma ordem de encomenda dá entrada na MOG Technologies (*cf.*, secção 1.2).

O *cluster* Kubernetes implementado deve suportar a execução de várias aplicações em simultâneo, sendo a principal forma de isolamento utilizada a criação de *namespaces* para cada *deployment*. Assim, é bastante importante que quando um utilizador acede a um determinado endereço tenha acesso à sua aplicação. Este roteamento realizado pelo Kubernetes que permite que aplicações fiquem expostas num determinado domínio de um mesmo endereço é conseguido pela utilização de um componente designado *Ingress*. O *Ingress* é um objeto do Kubernetes que nos permite expor os *services* de um *cluster*.

4.2. Casos de Uso

Nesta secção é apresentada uma vista arquitetural numa perspetiva de mais baixo nível, recorrendo a diagramas de sequência para dar a entender como as diferentes partes do sistema interagem no contexto dos casos de uso (*cf.*, secção 3.3.3).

Com o objetivo de simplificar a compreensão das interações entre os vários componentes, os casos de uso de uma mesma área de ação foram agrupados num único diagrama de sequência. O UC. 8 (*i.e.*, implantar outras aplicações) não está representado em nenhum diagrama pelo facto de ser muito semelhante à implantação do MAM4PRO.

4.2.1. Criar, Bloquear e Destruir a Infraestrutura

Foram definidos três casos de uso relacionados com o ciclo de vida da infraestrutura: criar, bloquear e destruir infraestrutura. Assim, a figura 4.7 representa o diagrama de sequência referente aos seguintes casos de uso:

- **UC. 1**: Criar a infraestrutura;
- **UC. 2**: Bloquear o estado da infraestrutura;
- **UC. 3**: Destruir a infraestrutura.

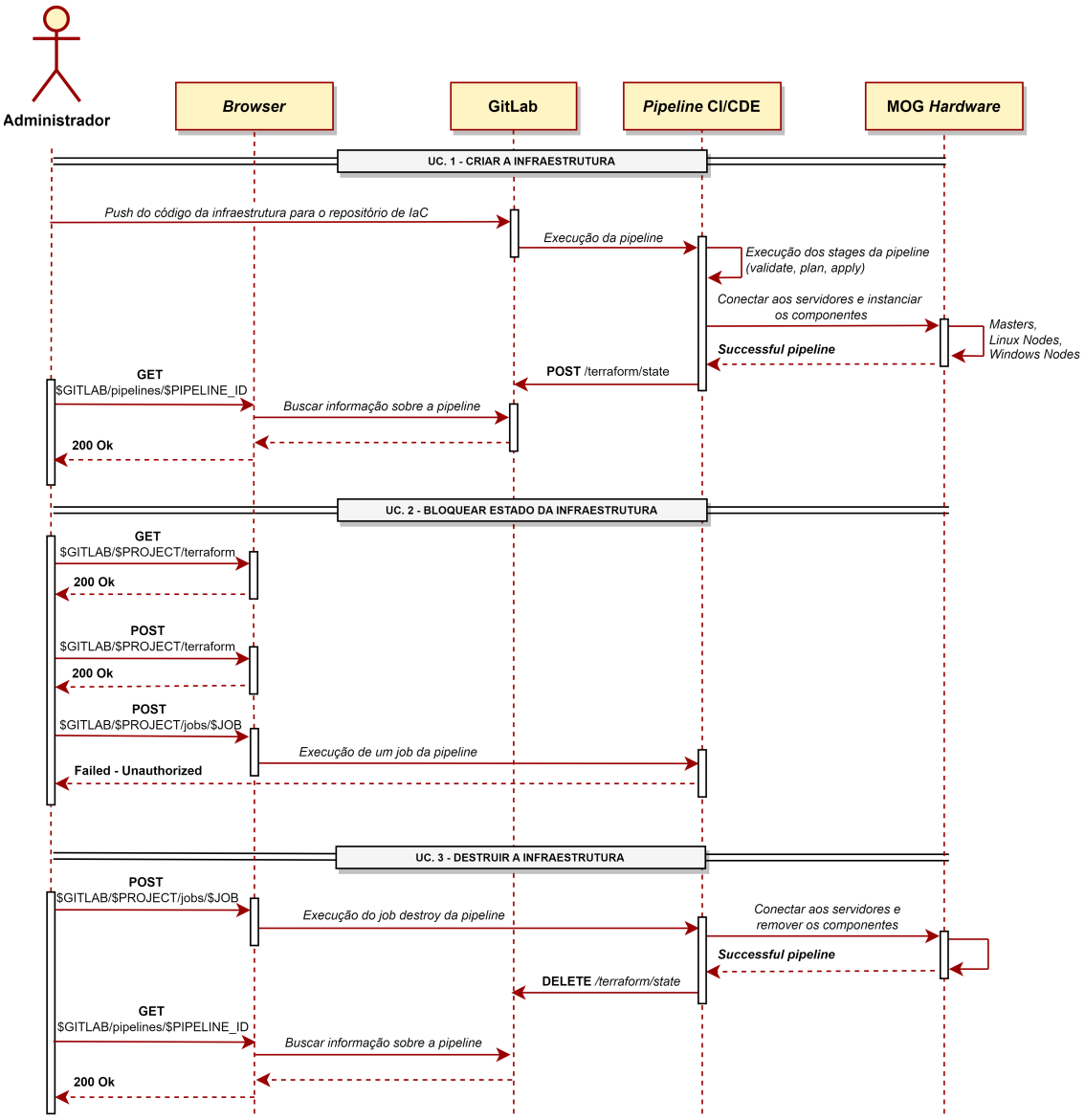


Figura 4.7. Diagrama de sequência que demonstra os UC. 1, 2 e 3.

Por simplificação o URL do GitLab[2] foi substituído por `$GITLAB`, a variável `$PROJECT`, `$JOB` e `$PIPELINE_ID` correspondem ao ID do projeto, do *job* e da *pipeline* de CI/CDE no GitLab, respetivamente.

[2] `https://gitlab.mog-technologies.com`

4.2.2. Implantar e Aceder ao MAM4PRO

O diagrama de sequência da figura 4.8 é referente aos seguintes casos de uso:

- **UC. 4:** Implantar o MAM4PRO;
- **UC. 5**: Aceder ao MAM4PRO através da internet.

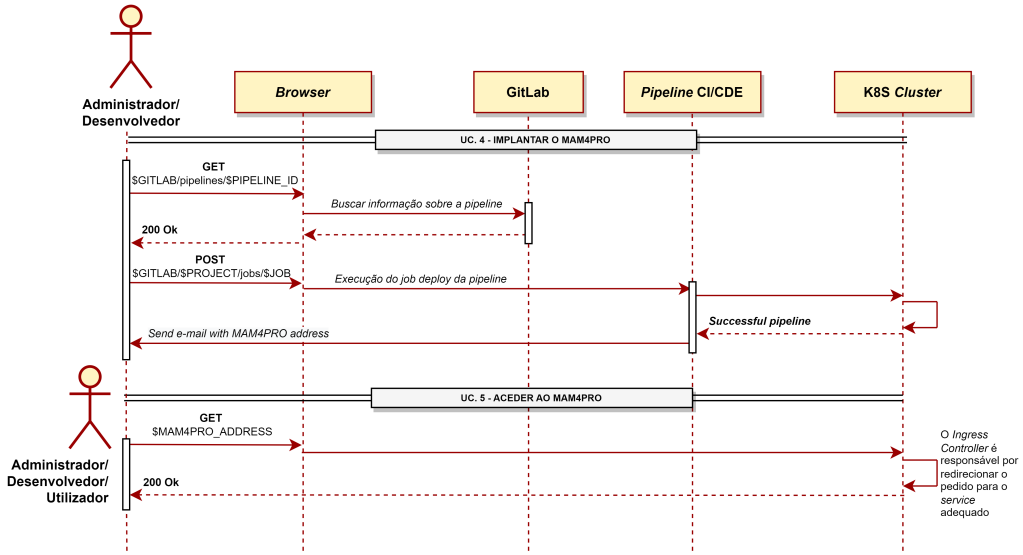


Figura 4.8. Diagrama de sequência que demonstra os UC. 4 e 5.

Por simplificação, o URL do MAM4PRO que é enviado no e-mail após implantação do *software* foi substituído pela variável `$MAM4PRO_ADDRESS`.

4.2.3. Escalar Serviços e Visualizar Métricas

O diagrama de sequência da figura 4.9, é referente aos seguintes casos de uso:

- **UC. 6:** Escalar o número de serviços (*i.e., pods*) de um *deployment*;
- **UC. 7:** Visualizar métricas da infraestrutura.

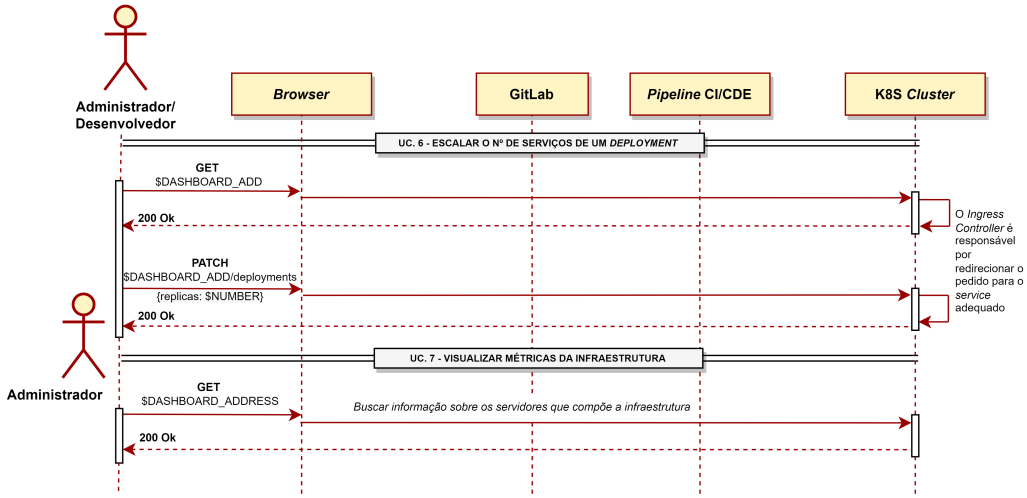


Figura 4.9. Diagrama de sequência que demonstra os UC. 6 e 7.

Por simplificação, o URL de acesso ao *dashboard* Kubernetes foi substituído pela variável `$DASHBOARD_ADD` e a variável `$NUMBER` corresponde ao número de réplicas a escalar no contexto do UC. 6.

4.3. Sumário

Este capítulo apresenta o desenho da solução segundo duas perspetivas: uma de mais alto nível (*cf.,* secção 4.1), onde se pretende demonstrar e explicar os fluxos de trabalho dos vários intervenientes, e uma de mais baixo nível (*cf.,* secção 4.2), onde se pretende demonstrar de forma mais detalhada e granular as interações entre os diferentes componentes no contexto dos casos de uso definidos (*cf.,* secção 3.3.3).

Os diagramas e notas arquiteturais resultantes deste capítulo são o ponto de partida para o capítulo seguinte intitulado Implementação.

Capítulo 5

Implementação da Solução

O presente capítulo documenta a implementação da solução desenhada no capítulo 4. Numa primeira fase é apresentada a *stack* tecnológica utilizada e posteriormente é apresentada a implementação de cada um dos componentes relacionando-os com os casos de uso do sistema definidos na secção 4.2.

5.1. *Stack* Tecnológica

Dada a existência de várias tecnologias que permitem concretizar os objetivos propostos (*cf.,* secção 2.8) foi necessário escolher a mais adequada à resolução do problema. Regra geral, a escolha baseou-se nas características e nas vantagens e desvantagens de cada uma delas. Adicionalmente, no caso da tecnologia de orquestração de *containers*, por ser a tecnologia principal desta dissertação, optou-se também pela aplicação de um método de apoio à decisão (*cf.,* secção 3.1.4.1) que nos permitiu escolher a tecnologia Kubernetes como a mais adequada.

A tabela 5.1 apresenta as tecnologias utilizadas e uma breve descrição do seu propósito.

Tabela 5.1. Tecnologias utilizadas na implementação da solução.

Tecnologia	Descrição
Docker	Tecnologia de conteinerização.
GitLab	Sistema de controlo de versões utilizado para versionamento do código desenvolvido e implementação de *pipelines* CI/CDE.
Helm	Tecnologia de gestão de aplicações.
Hyper-V	Tecnologia de virtualização utilizada para criação das VMs.
Kubernetes	Tecnologia de orquestração de *containers* utilizada para *deployment* das aplicações.
Terraform	Tecnologia de IaC que permite descrever toda a infraestrutura através de ficheiros de código descritivos.
Vagrant	Tecnologia de IaC utilizada para criação das VMs necessárias.

5.2. *Registry* e *Helm Repository*

No âmbito desta dissertação foi implantado um *registry* para armazenar as imagens dos *containers* necessários para implantação do MAM4PRO. Este *registry*, é na verdade um *container* baseado numa imagem oficial disponível no Docker Hub[1] (`registry:2`) e encontra-se em execução no servidor `M1-Todd` (*cf.,* apêndice E.1) (acessível no endereço `https://registry.mog-technologies.com:447`).

[1] `https://hub.docker.com/_/registry`

Uma vez que este *registry* ficará acessível através da internet e de forma a garantir alguma segurança foi criado um mecanismo básico de autenticação. As variáveis `$USER` e `$PASSWD` devem ser substituídas pelo nome de utilizador e *password* desejados, e a variável `$FILEPATH` deve ser substituída pelo caminho onde será armazenado o ficheiro de autenticação.

```
1 docker run htpasswd httpd:2  -Bbn $USER $PASSD > $FILEPATH
2
3 docker run htpasswd httpd:2 -Bbn $USER $PASSWD | Set-Content -Encoding ASCII $FILEPATH
```

Código 5.1. Implementação de mecanismo básico de autenticação no *registry*.

Uma vez criada a autenticação é necessário executar o *container* responsável pelo *registry*. Este *container* suporta várias variáveis de ambiente. Neste caso em particular, com a opção `-v` implantamos os *volumes* de armazenamento necessários, isto é, o local serão armazenadas as imagens dos *containers* (`S:/Registry/Volume`), o local onde estão os certificados *Hyper Text Transfer Protocol Secure* (HTTPS) utilizados (`S:/Registry/Certs`) e o local onde está o ficheiro de autenticação previamente criado (`S:/Registry/Auth`). Com a opção `-p` expomos a porta `5000` do *container* para a porta `447` do *host*.

```
1 docker run -d
2   -v S:/Registry/Volume:/registry -v S:/Registry/Certs/certs -v S:/Registry/Auth:/auth
3   -e REGISTRY_AUTH=htpasswd
4   -e REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm
5   -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd
6   -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt
7   -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key
8   registry:2 -p 447:5000
9   -restart=always -name registry
```

Código 5.2. Execução do *container* responsável por implementar o *registry*.

5.3. Estrutura de Pastas

A figura 5.1 representa a estrutura de pastas do repositório Git criado (*IaC Repository*). Todos os ficheiros relacionados com as *pipelines* CI/CDE encontram-se na pasta `.gitlab`, e todos os ficheiros Terraform necessários para implantação do *cluster* Kubernetes encontram-se na pasta `modules`. O ficheiro `variables.tfvars` é onde se encontram definidas algumas variáveis (*e.g.*, endereços IP dos servidores a utilizar e credenciais de acesso aos servidores). Este ficheiro encontra-se armazenado na raiz do repositório para facilitar no processo de adição e remoção de novos servidores por pessoas com menor conhecimento do código desenvolvido.

```
|__  .gitlab/
   |__ .gitlab-aux.yml
   |__ .gitlab-ci.yml
|__ modules/
   |__ node/
   |__ master/
|__ .gitignore
|__ variables.tfvars
|__ README.md
```

Figura 5.1. Estrutura de pastas do repositório criado.

5.4. Casos de Uso

Nesta secção pretende-se detalhar a implementação de cada um dos casos de uso definidos através da apresentação e explicação de excertos do código desenvolvido. É de salientar que por simplicidade algumas partes do código foram omitidas.

5.4.1. Criar Infraestrutura (UC. 1)

A criação da infraestrutura traduz-se na prática na implantação de um *cluster* híbrido de Kubernetes (*i.e., nodes* Windows e Linux) utilizando os servidores definidos. Esta infraestrutura pode ser criada de forma manual, mas conforme já referido foram aplicadas técnicas de IaC com recurso à tecnologia Terraform.

Este processo de criação do *cluster* é composto por várias partes sequenciais: (i) criação dos três *masters* do *cluster*, (ii) criação dos Windows *nodes* e (iii) criação dos Linux *nodes*.

5.4.1.1. Criação dos *Masters*

Relativamente à criação dos *masters* do *cluster* todos os passos necessários estão presentes no ficheiro `modules/master/main.tf`. Este ficheiro é composto por vários *resources* Terraform que são apresentados e explicados de seguida.

Primeiramente é preciso definir quais os servidores utilizados para desempenhar a função de *masters*. O código 5.3 mostra a declaração das variáveis `masters_hosts` (linhas 1 a 18) e `masters_vm` (linhas 19 a 30), onde são indicados os endereços IP dos *hosts* e das VMs. Além disso são também definidas algumas das suas características (*i.e.,* nome da VM, definições de rede, memória RAM, número de *Virtual Central Processing Unit* (vCPUs) e credenciais).

```
1  masters_hosts = {
2    ips = {
3      address = [ "10.50.50.5", "10.50.50.9", "10.50.50.11" ],
4      hostnames = [ "m1-skinner", "m1-edna", "m1-chalmers" ]
5    },
6    credentials = {          // SSH Credentials
7      username = "mog", password = "_OMITTED_"
8    },
9    folders = {              // Folders will be created in Masters Hosts
10     root                 = "C:\MOG",
11     env                  = "C:\MOG\env",
12     control_plane        = "C:\MOG\control-plane",
13     nodes_files          = "C:\MOG\control-plane\nodes-files",
14     load_balancer        = "C:\MOG\load-balancer",
15     kubernetes_dashboard = "C:\MOG\kubernetes-dashboard",
16     smb_storage_class    = "C:\MOG\smb-storage-class"
17   }
18 }
19 masters_vm = {
20   vm_name                  = "MOG_One_Master",
21   external_virtual_switch  = "'External Switch'",
22   network_adapter          = "'Ethernet 2'",
23   ram                      = "4GB",
24   vcpus                    = "4",
25   address = [ "10.50.50.51", "10.50.50.52", "10.50.50.53" ],
26   master_of_masters = "10.50.50.51",
27   credentials = {        // SSH Credentials
28     username = "vagrant", password = "_OMITTED_"
29   }
30 }
```

Código 5.3. Definição de duas variáveis (`masters_hosts` e `masters_vm`).

Uma vez definidos os servidores e as suas características são preparados e copiados os ficheiros necessários para os servidores *host* onde as VMs serão criadas (código 5.4) e definidos os *hostnames* (código 5.5).

```
resource "null_resource" "prepare_dir" {
  count = length("${var.masters_hosts.ips.address}")
  // ...  Establish SSH connection to each host ...
  provisioner "remote-exec" {
    inline = [
      "mkdir ${var.masters_hosts.folders.root} -Force",
      "mkdir ${var.masters_hosts.folders.control_plane} -Force"
    ]
  }
  provisioner "file" {
    source      = "${path.module}/scripts/change-resources.ps1"
    destination = "${var.masters_hosts.folders.root}/change-resources.ps1"
  }
}
```

Código 5.4. *Resource* responsável por preparar e copiar os ficheiros para os *hosts*.

```
resource "null_resource" "configure_hostname" {
  count = length("${var.masters_hosts.ips.address}")
  // ...  Establish SSH connection to each host ...
  provisioner "remote-exec" {
    inline = [
      "${var.masters_hosts.folders.root}/change-resources.ps1
      CHANGE_HOSTNAME ${var.masters_hosts.ips.hostnames[count.index]}"
    ]
  }
}
```

Código 5.5. *Resource* responsável por definir os *hostnames*.

Posteriormente, a VM é criada com recurso ao Vagrant e por isso é utilizado um ficheiro `Vagrantfile` para o seu provisionamento (código 5.6). Este ficheiro instala todas as dependências necessárias para o correto funcionamento do *master*.

Assim que os três *masters* estiverem disponíveis é alterado o adaptador de rede (código 5.6) e alterados os recursos das VMs de forma a refletir as propriedades definidas nas variáveis (código 5.7).

```
resource "null_resource" "create_vm" {
  count = length("${var.masters_hosts.ips.address}")
  // ...  Establish SSH connection to each host ...
  provisioner "file" {
    source      = "${path.module}/Vagrantfile"
    destination = "${var.masters_hosts.folders.root}/Vagrantfile"
  }
  provisioner "remote-exec" {
    inline = [ "vagrant up" ]
  }
}
resource "null_resource" "set_network_adapter" {
  count = length("${var.masters_hosts.ips.address}")
  // ...  Establish SSH connection to each host ...
  provisioner "remote-exec" {
    inline = [
      "${var.masters_hosts.folders.root}/change-resources.ps1
      CREATE_NET_ADAPTER ${var.masters_vms.external_virtual_switch} ${var.masters_vms.network_adapter}",
      "${var.masters_hosts.folders.root}/change-resources.ps1
      SET_NET_ADAPTER ${var.virtual_machine_name} ${var.masters_vm.external_virtual_switch}"
    ]
  }
}
```

Código 5.6. *Resource* responsável pela criação da VM com recurso ao Vagrant e *resource* responsável pela criação do adaptador de rede.

```
resource "null_resource" "change_machines_resources" {
  count = length("${var.masters_hosts.ips.address}")
  // ...  Establish SSH connection to each host ...
  provisioner "remote-exec" {
    inline = [
      "${var.masters_hosts.folders.root}/change-resources.ps1
      MACHINE_RESOURCES ${var.virtual_machine_name} ${var.masters_vm.ram} ${var.masters_vm.vcpus}",
      "Rename-VM ${var.virtual_machine_name} ${var.masters_vm.vm_name}_${count.index +1}",
      "Start-VM -Name ${var.masters_vm.vm_name}_${count.index +1}"
    ]
  }
}
```

Código 5.7. *Resource* responsável por alterar as características das VMs.

Assim que as VMs estiverem criadas, o Kubernetes e respetivos componentes são inicializados. Por último e tal como demonstra o código 5.8, é realizado o *deployment* do *Ingress*

(linhas 1 a 10), do *dashboard* Kubernetes (linhas 11 a 21) e *Storage Class* (linhas 22 a 36). Uma *Storage Class*[2] está diretamente relacionada com os *volumes* de armazenamento e permite que seja criado um *volume* dinamicamente no momento do *deployment*. No contexto desta dissertação torna-se importante garantir a persistência dos dados do serviço `mCore`.

Estes *deployments* baseiam-se na aplicação de ficheiros YAML fornecidos pelo Kubernetes e que foram ajustados às necessidades da MOG Technologies.

```
1  resource "null_resource" "deploy_ingress" {
2    count = var.control_plane_configs.ingress_controller.use == true ? 1 : 0
3    provisioner "remote-exec" {
4      // ...  Establish SSH connection to master of masters (Master-1) ...
5      inline = [
6        "kubectl apply -f $HOME/nginx-ingress/nginx-ingress.yaml",
7        "kubectl scale deployment --namespace ingress-nginx ingress-nginx-controller --replicas=
       ${var.control_plane_configs.ingress_controller.replicas}"
8      ]
9    }
10 }
11 resource "null_resource" "deploy_kubernetes_dashboard" {
12   count = var.control_plane_configs.dashboard_deployment.use == true ? 1 : 0
13   provisioner "remote-exec" {
14     // ...  Establish SSH connection to master of masters (Master-1) ...
15     inline = [
16       "kubectl apply -f %{var.control_plane_configs.dashboard_deployment.file}",
17       "kubectl apply -f $HOME/kubernetes-dashboard/account.yaml",
18       "kubectl apply -f $HOME/kubernetes-dashboard/dashboard-ingress.yaml"
19     ]
20   }
21 }
22 resource "null_resource" "storage_class_smb" {
23   count = var.control_plane_configs.csi_storage_class.use == true ? 1 : 0
24   provisioner "remote-exec" {
25     // ...  Establish SSH connection to master of masters (Master-1) ...
26     inline = [
27       "kubectl apply -f $HOME/smb-storage-class/rbac-csi-smb-controller.yaml",
28       "kubectl apply -f $HOME/smb-storage-class/csi-smb-driver.yaml",
29       "kubectl apply -f $HOME/smb-storage-class/csi-smb-controller.yaml",
30       "kubectl apply -f $HOME/smb-storage-class/csi-smb-node.yaml",
31       "kubectl apply -f $HOME/smb-storage-class/csi-smb-node-windows.yaml",
32       "kubectl apply -f $HOME/smb-storage-class/smb-storage-creds.yaml",
33       "kubectl apply -f $HOME/smb-storage-class/smb-storage-class.yaml"
34     ]
35   }
36 }
```

Código 5.8. *Resource* responsável pelo *deployment* do *Ingress*, do *dashboard* Kubernetes e *Storage Class*.

Todos os *resources* mostrados nesta secção são implantados nos servidores através de uma *pipeline* de CI/CDE e não manualmente. O *job* responsável por executar estes passos é adiante designado por `apply:masters`. Assim que este *job* esteja concluído é possível aceder ao *dashboard* Kubernetes (figura 5.2).

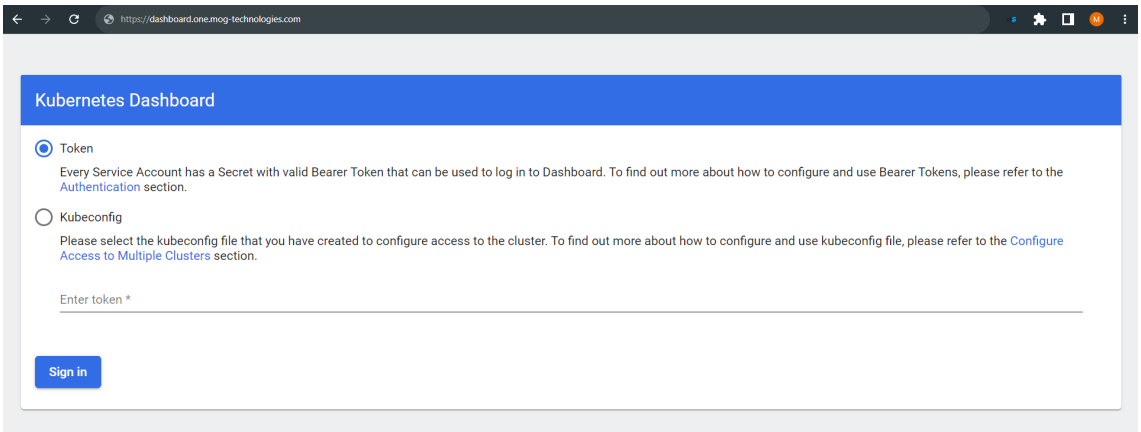


Figura 5.2. *Dashboard* Kubernetes.

[2] `https://kubernetes.io/docs/concepts/storage/storage-classes/`

5.4.1.2. Criação dos Windows *Nodes*

À semelhança da criação dos *masters* do *cluster* também o primeiro passo da criação dos Windows *nodes* consiste na criação e preparação dos diretórios necessários em cada um dos servidores (código 5.9). Os excertos de código apresentado de seguida encontram-se presentes no ficheiro `modules/node/windows/main.tf`.

```
1  resource "null_resource" "prepare_dir" {
2    count = length("${var.win_nodes.ips.address}")
3    // ...  Establish SSH connection to each Windows node ...
4    provisioner "remote-exec" {
5      inline = [
6        "mkdir ${var.win_nodes.folders.root} -Force",
7        "mkdir ${var.win_nodes.folders.root}/scripts -Force"
8      ]
9    }
10   provisioner "file" {
11     source      = "${path.root}/../nodes-files/"
12     destination = "${var.win_nodes.folders.root}"
13   }
14   // Missing code ...
15 }
```

Código 5.9. *Resource* responsável pela criação e preparação dos diretórios nos Windows *nodes*.

O código 5.10 é responsável pela instalação do *container runtime* utilizado nesta solução (linha 5) e pela instalação do Kubernetes e respetivos componentes (linha 12).

```
1  resource "null_resource" "install_containerd" {
2    count = length("${var.win_nodes.ips.address}")
3    // ...  Establish SSH connection to each Windows node ...
4    provisioner "remote-exec" {
5      inline = [ "C:\MOG\Install-Containerd.ps1 -ContainerDVersion 1.6.4" ]
6    }
7  }
8  resource "null_resource" "install_kubernetes" {
9    count = length("${var.win_nodes.ips.address}")
10   // ...  Establish SSH connection to each Windows node ...
11   provisioner "remote-exec" {
12     inline = [ "C:\MOG\PrepareNode.ps1 -KubernetesVersion v1.24.0 -ContainerRuntime containerD" ]
13   }
14 }
```

Código 5.10. *Resource* responsável pela instalação do Kubernetes e respetivos componentes.

Por último, é necessário juntar este *node* aos *masters* que foram anteriormente criados (código 5.11). Para isso é necessária a execução de um comando num dos *masters* que retorna outro comando com o *token* de autenticação que deve ser posteriormente executado no *node* que se pretende adicionar. Assim numa primeira fase o comando é executado no *master* (linha 6) e enviado por SSH para o *node* (linha 7), e posteriormente é executado no *node* (linha 16).

```
1  resource "null_resource" "get_join_command" {
2    count = length("${var.win_nodes.ips.address}")
3    // ...  Establish SSH connection to master of masters (Master-1) ...
4    provisioner "remote-exec" {
5      inline = [
6        "echo $(kubeadm token create --print-join-command) --cri-socket npipe:////./pipe/containerd-containerd > join-
       command-${element(var.win_nodes.ips.address, count.index)}",
7        "sudo sshpass -p ${var.win_nodes.credentials.password} scp -o 'StrictHostKeyChecking no' $HOME/join-command-
       ${element(var.win_nodes.ips.address, count.index)} ${var.win_nodes.credentials.username}@
       ${element(var.win_nodes.ips.address, count.index)}:${var.win_nodes.folders.root}"
8      ]
9    }
10 }
11 resource "null_resource" "add_node" {
12   count = length("${var.win_nodes.ips.address}")
13   // ...  Establish SSH connection to each Windows node ...
14   provisioner "remote-exec" {
15     inline = [
16       "Get-Content ${var.win_nodes.folders.root/join-command-${element(var.win_nodes.ips.address, count.index) |
       Invoke-Expression"
17     ]
18   }
19 }
```

Código 5.11. Execução do comando de criação do *token* para adição do *node*.

5.4.1.3. Criação dos Linux *Nodes*

O processo de criação dos Linux *nodes* é bastante semelhante à criação dos Windows *nodes*, existem apenas algumas particularidades relacionadas com a própria arquitetura do sistema operativo. O primeiro passo corresponde à criação e preparação dos diretórios necessários para a execução da solução (código 5.12), seguindo-se a definição dos *hostnames* (código 5.13).

Os excertos de código apresentados de seguida encontram-se presente no ficheiro `modules/node/linux/main.tf`.

```
resource "null_resource" "prepare_dir" {
  count = length("${var.linux_nodes.ips.address}")
  // ...  Establish SSH connection to each Linux node ...
  provisioner "file" {
    source      = "${path.module}/scripts/"
    destination = "${var.linux_nodes.folders.root}"
  }
}
```

Código 5.12. *Resource* responsável por preparar os diretórios.

```
resource "null_resource" "configure_hostname" {
  count = length("${var.linux_nodes.ips.address}")
  // ...  Establish SSH connection to each Linux node ...
  provisioner "remote-exec" {
    inline = [
      "${var.linux_nodes.folders.root}/config-host.sh ${var.linux_nodes.ips.hostnames[count.index]} ${var.linux_nodes.
      credentials.password}"
    ]
  }
```

Código 5.13. *Resource* responsável por definir os *hostnames*.

De seguida é executado um *script* (`prepare-node.sh`) responsável pela instalação do containerd, camada de rede, Kubernetes e respetivos componentes (código 5.14).

```
resource "null_resource" "prepare_node" {
  count = length("${var.linux_nodes.ips.address}")
  // ...  Establish SSH connection to each Linux node ...
  provisioner "remote-exec" {
    inline = ["echo ${var.linux_nodes.credentials.password} | ${var.linux_nodes.folders.root}/prepare-node.sh"]
  }
}
```

Código 5.14. *Resource* responsável pela execução de um *script* de instalação do containerd, camada de rede, Kubernetes e respetivos componentes.

Por último, e semelhante ao processo de criação dos Linux *nodes*, um comando é executado num dos *masters* do *cluster*, por sua vez retornando outro comando com o *token* de autenticação. Este deve ser posteriormente executado no *node* que se pretende adicionar (código 5.15). Assim, numa primeira fase é então executado o comando no *master* (linha 6) e enviado por SSH para o *node* (linha 7) e posteriormente é executado o comando no *node* (código 5.16).

```
resource "null_resource" "get_join_command" {
  count = length("${var.linux_nodes.ips.address}")
  // ...  Establish SSH connection to master of masters (Master-1) ...
  provisioner "remote-exec" {
    inline = [
      "echo \"$(kubeadm token create --print-join-command)\" '--cri-socket \"unix:///var/run/containerd/containerd.
      sock\"' > join-command-${element(var.linux_nodes.ips.address, count.index)}",
      "sudo sshpass -p ${var.linux_nodes.credentials.password} scp -o \"StrictHostKeyChecking no\" $HOME/join-command-
      ${element(var.linux_nodes.ips.address, count.index)} mog@${element(var.linux_nodes.ips.address, count.index)}:${
      var.linux_nodes.folders.root}/join-command-${element(var.linux_nodes.ips.address, count.index)}",
      "rm $HOME/join-command-${element(var.linux_nodes.ips.address, count.index)}"
    ]
  }
}
```

Código 5.15. Execução do comando de criação do *token* para adição do *node*.

```
resource "null_resource" "add_node" {
  count = length("${var.linux_nodes.ips.address}")
  // ...  Establish SSH connection to each Linux node ...
  provisioner "remote-exec" {
    inline = [
      "echo ${var.linux_nodes.credentials.password} | sudo -S ${var.linux_nodes.folders.root}/join-command-${element(
     var.linux_nodes.ips.address, count.index)}"
    ]
  }
}
```

Código 5.16. Execução do comando de adição do *node*.

Uma vez provisionada a infraestrutura, fica cumprida a pré-condição apresentada anteriormente no diagrama de casos de uso (*i.e.,* a infraestrutura deve ser previamente criada) e podem assim ser executados os casos de uso apresentados nas secções seguintes.

5.4.2. Bloquear Estado da Infraestrutura (UC. 2)

Após o provisionamento da infraestrutura, a execução de um novo provisionamento pode resultar em alterações inesperadas que afetam o correto funcionamento da infraestrutura. Como tal, definiu-se um caso de uso em que deve ser possível bloquear o estado da infraestrutura garantindo que não é possível modificá-la até que este estado se encontre desbloqueado.

O Terraform armazena o estado da infraestrutura e configurações aplicadas num ficheiro (local ou remoto) chamado `terraform.tfstate`. Antes de qualquer operação o Terraform faz uma atualização para garantir que o estado descrito neste ficheiro é o que está aplicado na infraestrutura real.

Para implementar este caso de uso definiu-se que este estado deveria ser armazenado remotamente recorrendo ao GitLab e à sua integração com o Terraform[3]. Assim, todos os ficheiros `main.tf` já apresentados possuem uma configuração semelhante ao código 5.17.

```
terraform {
    backend "http" {}
}
```

Código 5.17. Definição do estado Terraform.

Embora o código 5.17 seja necessário, por si só não é suficiente. O Terraform irá armazenar os estados remotamente mas ainda é necessário indicar o local de armazenamento. No ficheiro `.gitlab-ci.yaml` na definição dos comandos a executar para aplicar os vários *resources* já apresentados é definido o local onde ficam armazenados (código 5.18).

```
terraform init
  -backend-config="address=$ADDRESS_TF_STATE$MASTERS"
  -backend-config="lock_address=$ADDRESS_TF_STATE$MASTERS/lock"
  -backend-config="unlock_address=$ADDRESS_TF_STATE$MASTERS/lock"
  -backend-config=username="$USERNAME" -backend-config=password="$ACCESS_TOKEN"
  -backend-config=lock_method="POST" -backend-config=unlock_method="DELETE"
```

Código 5.18. Configuração do local onde os estados Terraform serão armazenados.

Após o provisionamento da infraestrutura no painel do GitLab (*i.e., Infrastructure - Terraform*) são visíveis os estados armazenados (figura 5.3). Note que aqui estão disponíveis algumas ações, nomeadamente o bloquear (*lock*).

[3] `https://docs.gitlab.com/ee/user/infrastructure/iac/terraform_state.html`

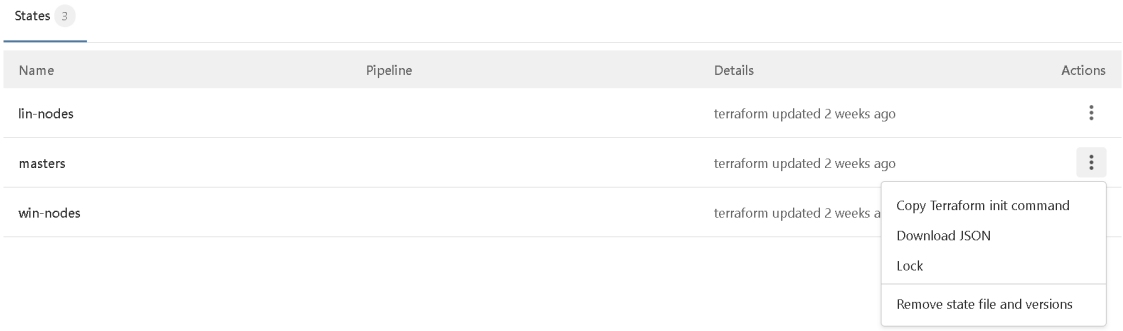


Figura 5.3. Estados Terraform armazenados no GitLab.

Conforme ilustrado no diagrama de sequência deste caso de uso (figura 4.7), se após o bloqueio da infraestrutura for executada uma nova *pipeline* de provisionamento será visível o erro do código 5.19.

```
1  Acquiring state lock. This may take a few moments...
2   Error: Error acquiring the state lock
3   Error message: HTTP remote state already locked:
4   ID=52cdc7e0-1c24-4b66-8f85-d3fca14f710b
5   Lock Info:
6     ID:        5f71db2d-46f1-ec03-fc8f-dc508e781baf
7     Operation: OperationTypeApply
8     Who:       NT AUTHORITY\SYSTEM@M1-TODD
9     Version:   1.2.6
10    Created:   2022-10-03 10:17:20.227213 +0000 UTC
11  Terraform acquires a state lock to protect the state from being written by multiple users at the same time. Please
        resolve the issue above and try again. For most commands, you can disable locking with the "-lock=false" flag,
        but this is not recommended.
12 Cleaning up project directory and file based variables 00:00
13 ERROR: Job failed:  exit status 1
```

Código 5.19. Erro exibido quando é executada uma *pipeline* de provisionamento se os estados estiverem bloqueados.

5.4.3. Destruir Infraestrutura (UC. 3)

Com o comando `terraform destroy` é possível remover as ações que foram aplicadas durante a execução do comando `terraform apply`. No entanto, isto só é possível quando os *resources* utilizados tem alguma função explícita, e não quando são utilizados `null_resources` em que apenas são executados comandos. Esta é uma das limitações desta solução, como não existe um *provider* que ofereça a implantação de *clusters* Kubernetes *on-premise*[4], foi necessária a criação de `null_resources`.

Assim, a destruição da infraestrutura implica a execução de *resources* para remover algumas das configurações realizadas durante o provisionamento. O código 5.20 é responsável pela destruição dos *masters* do *cluster* através da eliminação da VM criada (linha 6 e 7) e ficheiros armazenados no *host* (linha 8).

```
1  resource "null_resource" "destroy_masters" {
2    count = var.run_destroy == true ? length("${var.masters_hosts.ips.address}") : 0
3    provisioner "remote-exec" {
4      // ...  Establish SSH connection to each master host ...
5      inline = [
6        "Stop-VM -Name ${var.masters_vm.vm_name}\_${count.index + 1}",
7        "Remove-VM -Name ${var.masters_vm.vm_name}\_${count.index + 1} -Force",
8        "Remove-Item -Force -Path ${var.masters\_hosts.folders.root}"
9      ]
10   }
11 }
```

Código 5.20. *Resource* responsável pela destruição dos *masters*.

[4] quando o *software* ou infraestrutura são implantados nas instalações da pessoa ou organização

A destruição dos Windows *nodes*, corresponde à execução de um *script* responsável por parar e eliminar os serviços relacionados com a solução, remover os diretórios criados e remover as configurações criadas pelo próprio Kubernetes (código 5.21). A destruição dos Linux *nodes* é semelhante, porém aplicada ao sistema operativo em questão (código 5.22).

```
resource "null_resource" "destroy_windows_nodes" {
  count = var.run_destroy == true ? length("${var.win_nodes.ips.address") : 0
  // ...  Establish SSH connection to each Windows node ...
  provisioner "remote-exec" {
    inline = [ "${var.win_nodes.folders.root}\scripts\prepare-environment.ps1" ]
  }
}
```

Código 5.21. *Resource* responsável pela destruição dos Windows *nodes*.

```
resource "null_resource" "destroy_linux_nodes" {
  count = var.run_destroy == true ? length("${var.linux_nodes.ips.address") : 0
  provisioner "remote-exec" {
    // ...  Establish SSH connection to each Linux node ...
    inline = [
      "echo ${var.credentials.password} | sudo -S kubeadm reset -f",
      "sudo rm -rf ~/.kube /etc/cni /etc/kubernetes /var/lib/etcd /var/lib/kubelet /var/run/kubernetes",
      "sudo iptables -F && sudo iptables -X",
      "sudo iptables -t nat -F && sudo iptables -t nat -X",
      "sudo iptables -t raw -F && sudo iptables -t raw -X",
      "sudo iptables -t mangle -F && sudo iptables -t mangle -X",
      "sudo systemctl restart containerd.service kubelet",
      "sudo apt purge -y --allow-change-held-packages kubeadm kubectl kubelet",
      "sudo apt autoremove -y kubeadm kubectl kubelet"
    ]
  }
}
```

Código 5.22. *Resource* responsável pela destruição dos Linux *nodes*.

5.4.4. Implantar o MAM4PRO (UC. 4)

O *deployment* do MAM4PRO é realizado recorrendo à tecnologia de gestão de aplicações estudada, o Helm. Assim, o primeiro passo para a concretização deste caso de uso foi a criação de um Helm *chart*, que descreve um *deployment* genérico de MAM4PRO (*cf.*, apêndice F.1.1).

Depois de criado o Helm *chart* e colocado no Helm *repository* referido anteriormente, cada *deployment* realizado faz uso de um ficheiro `values.yaml` (código 5.23) que permite configurar alguns parâmetros no momento da instalação.

```
global:
  image:
    registry: registry.mog-technologies.com:447
    tag: latest
ingress:
  hostname: falcon.one.mog-technologies.com
  clusterIssuer: letsencrypt-prod
  mcore:
    replicas: 1
    persistence:
      enabled: true
      storageClass: smb
  messagequeuing:
    replicas: 1
  rplayerweb:
    replicas: 1
  rplayersdi:
    replicas: 1
  hoperation:
    replicas: 1
  scapture:
    replicas: 1
  rplayout:
    replicas: 1
```

Código 5.23. Ficheiro `values.yaml` utilizado para configurar alguns parâmetros no momento do *deployment* do MAM4PRO.

Nas linhas 3 e 4 do código 5.23 são definidas as imagens de *container* a utilizar. Neste caso serão usadas as imagens do *registry* criado com a *tag* `latest`. A *tag* `latest` normalmente

é sempre a última versão do *software*. Na linha 6 é definido o URL onde ficará acessível o MAM4PRO.

Foi criado um parâmetro `persistence` (linha 10) que permite habilitar ou desabilitar a persistência, isto é, pode ser realizado um *deployment* com ou sem persistência do serviço `mCore`. Caso esteja ativa, na linha 12 é configurada a *StorageClass* a utilizar para provisionamento de *volumes* de armazenamento no momento do *deployment*. Por último, são definidos o número de réplicas de cada serviço (linhas 13 a 24).

A implantação do MAM4PRO com base neste *chart* ocorre assim que for executado o comando `helm install` (código 5.24).

```
1 helm install -create-namespace m4p-falcon -n m4p-falcon ../mam4pro.tgz -f values.yaml
```

Código 5.24. *Deployment* do MAM4PRO recorrendo ao Helm.

É assim criado um *namespace* chamado `m4p-falcon` e implantado o MAM4PRO com os valores definidos no ficheiro `values.yaml`. O ficheiro `tgz` utilizado corresponde ao *chart* que foi conseguido através da execução do comando `helm pull`.

Embora esta seja uma forma de *deployment* do MAM4PRO, foi criada uma alternativa mais conveniente através da *pipeline* de CI/CDE existente no *Mxfspeedrail Repository*. Tal como referido na secção (*cf.*, secção 4.1.2), sempre que é executada uma *pipeline* deste repositório são gerados *containers*. Posto isto, foi adicionado um novo *job* no *stage deploy* já existente, que permite implantar o MAM4PRO no *cluster* Kubernetes.

Conforme referido na análise ao código 5.23 se a persistência do serviço `mCore` estiver ativa, os *volumes* de armazenamento serão automaticamente provisionados no servidor definido (figura 5.4):

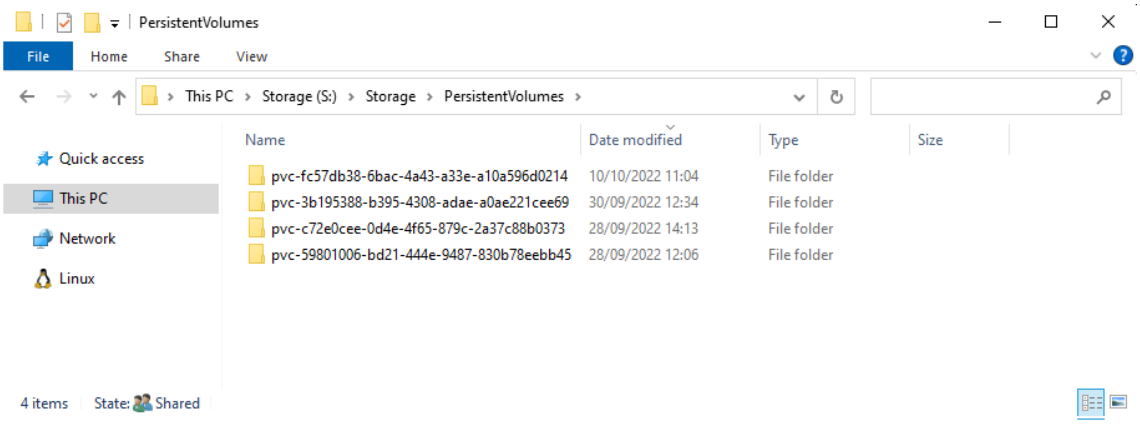


Figura 5.4. Provisionamento automático de *volumes* conseguido com a implantação da *StorageClass*.

5.4.5. Aceder ao MAM4PRO através da Internet (UC. 5)

Após execução do UC. 4 o MAM4PRO pode ser acedido através da internet (figura 5.5) no endereço definido no ficheiro `values.yaml`. Desta forma, elimina-se a necessidade atual de execução de um instalador do MAM4PRO.

Quando o utilizador coloca no *browser* o endereço o pedido é encaminhado ao *Ingress*, que o redireciona para o serviço adequado.

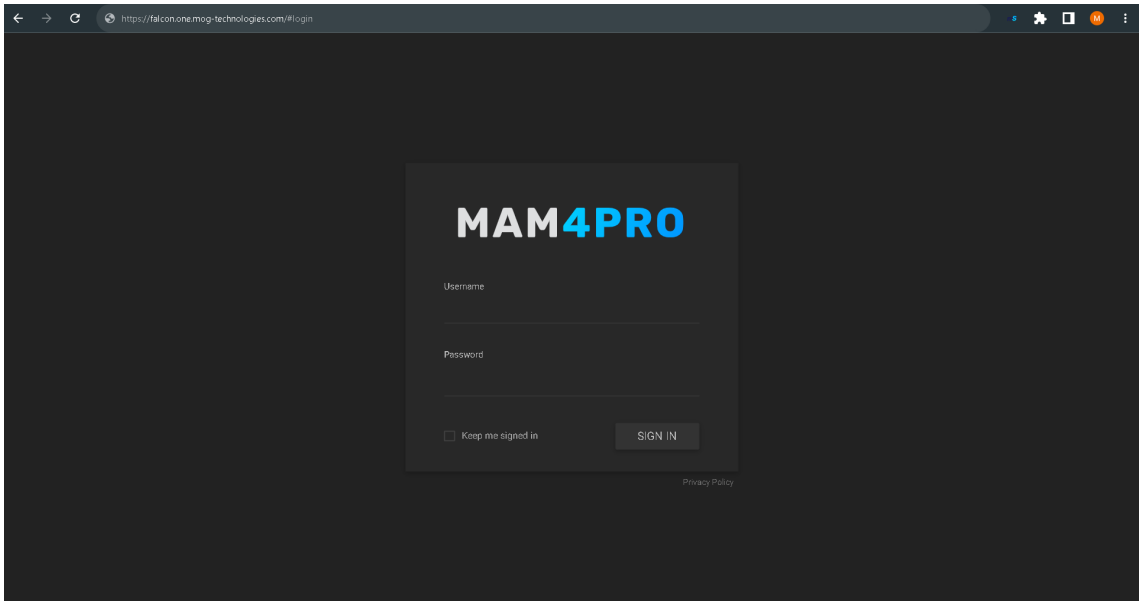


Figura 5.5. Acesso ao MAM4PRO através da internet.

5.4.6. Escalar Serviços de um *Deployment* (UC. 6)

O escalonamento e elasticidade dos serviços na prática traduz-se no aumento e diminuição do número de réplicas dos *deployments*. Esta tarefa pode ser realizada através do *dashboard* Kubernetes ou pode ser realizada recorrendo à linha de comandos utilizando o `kubectl`.

A figura 5.6 ilustra o escalonamento do serviço `hOperation` através do *dashboard* Kubernetes. Existiam 0 *pods* deste *deployment* e aumentaram-se o número de réplicas desejadas para 2.

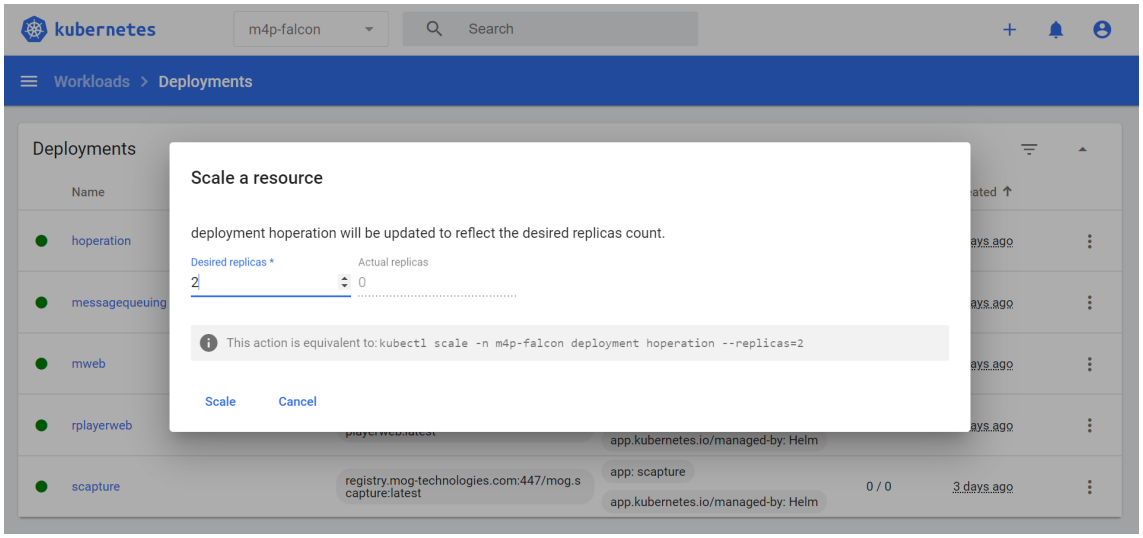


Figura 5.6. Escalonamento das réplicas do serviço `hOperation` através do *dashboard* Kubernetes.

Em alternativa, é possível com o `kubectl` escalar o número de réplicas (código 5.25).

```
1 kubectl scale deployments/hoperation -replicas=2 -n m4p-falcon
```

Código 5.25. Utilização do `kubectl` para escalar o número de réplicas do serviço `hOperation`.

5.4.7. Visualizar Métricas da Infraestrutura (UC. 7)

A visualização das métricas relacionadas com os servidores que compõe a infraestrutura pode ser realizada através do *dashboard* Kubernetes. É visível informação sobre cada *master* (figura 5.7) e *nodes* do *cluster* (figura 5.8).

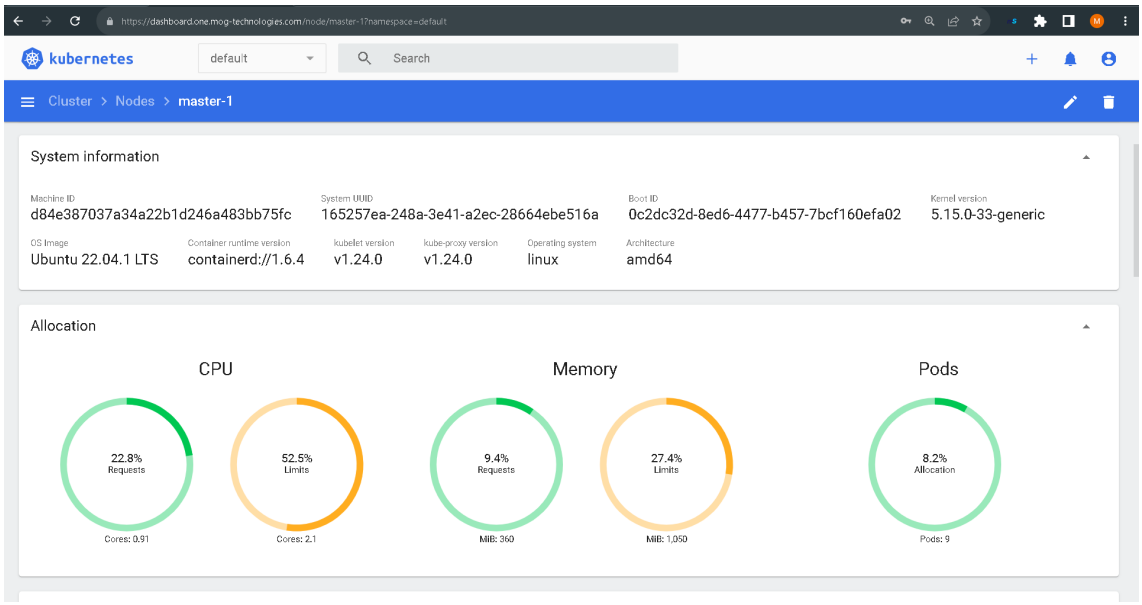


Figura 5.7. Métricas de utilização de um dos *masters* do *cluster*.

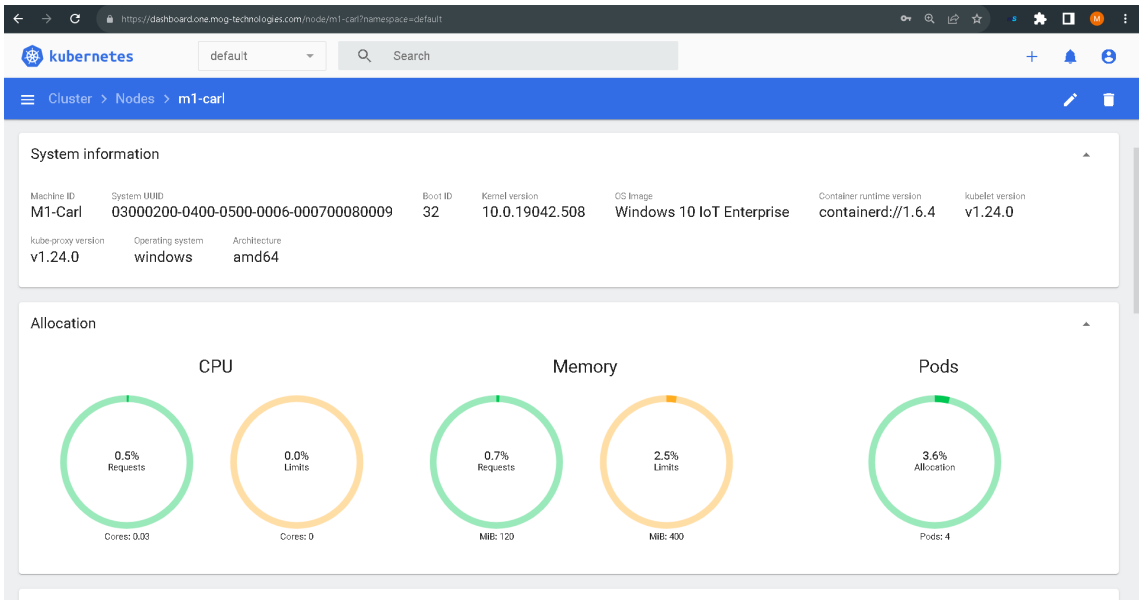


Figura 5.8. Métricas de utilização de um dos *nodes* do *cluster*.

Devido à compatibilidade do Kubernetes com serviços de terceiros este caso de uso poderia ser satisfeito através da utilização de duas ferramentas bastantes conhecidas nestas situações: o Prometheus[5] e o Grafana[6].

[5] https://prometheus.io/
[6] https://grafana.com/

5.4.8. Implantar Outras Aplicações (UC. 8)

Apesar de o foco desta dissertação ser a implementação do *software* MAM4PRO, este tipo de soluções e infraestruturas nunca pressupõe apenas a implantação de uma única aplicação. Desta forma, definiu-se que além do MAM4PRO outras aplicações devem ser implantadas. Conveniente, o facto de ser um *cluster* Kubernetes híbrido, previne restrições relacionadas com os sistemas operativos.

A título de exemplo, optou-se pela implantação de uma aplicação utilitária chamada Uptime Kuma[7] (figura 5.9). Esta aplicação é desenvolvida pela comunidade e fornece uma visão simples e rápida de monitorização.

À semelhança do MAM4PRO também foi elaborado um Helm *chart* para permitir a implantação do Uptime Kuma. O ficheiro `values.yaml` utilizado encontra-se descrito no código 5.26.

```
1 replicas: 1
2 ingress:
3   enabled: true
4   hostname: uptime.one.mog-technologies.com
5 persistence:
6   enabled: false
```

Código 5.26. Ficheiro `values.yaml` utilizado para configurar alguns parâmetros no momento do *deployment* do Uptime Kuma..

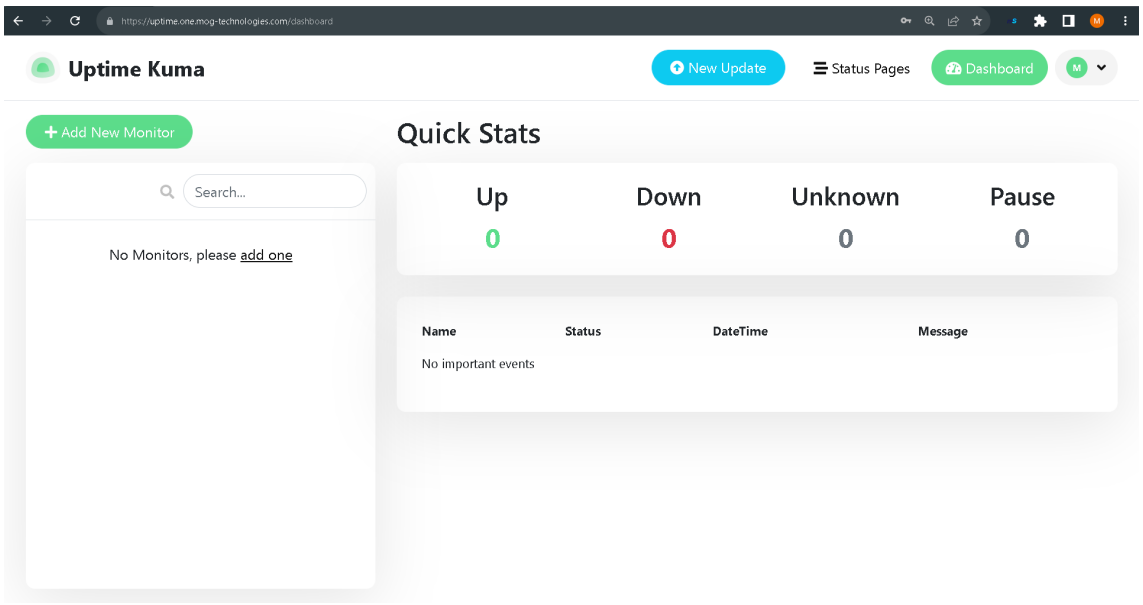


Figura 5.9. Página inicial de acesso ao Uptime Kuma.

5.5. *Pipeline Continuous Integration/Continuous Delivery*

Conforme referido na secção 4.1.1.3 a *pipeline* de CI/CDE de provisionamento e destruição da infraestrutura criada é composta por quatro *stages*: *validate*, *plan*, *apply* e *destroy*.

O primeiro *stage*, denominado *validate*, é responsável por validar o código Terraform através do comando `terraform validate`. O segundo *stage*, designado *plan*, é responsável por criar o plano das ações que serão aplicadas à infraestrutura. Este *stage* é composto por

[7] `https://github.com/louislam/uptime-kuma`

três *jobs* distintos: `plan:masters`, `plan:lin-nodes` e `plan:win-nodes`. Esta divisão visa permitir apenas o provisionamento de uma parte da infraestrutura (*e.g.,* Windows *nodes*) sem necessidade de reconstruir outros componentes.

O terceiro *stage* da *pipeline*, denominado *apply*, é um *stage* também compostos por três *jobs* pela mesma razão do *stage* anterior. Estes *jobs* possuem a particularidade de apenas serem executados através de uma ação humana. Optou-se por colocar esta ação manual, porque pode ocorrer situações que uma alteração no código não necessite de provisionar novamente a infraestrutura.

O último *stage*, denominado *destroy*, é responsável pela execução do UC. 3 (*i.e.,* destruir infraestrutura). A infraestrutura pode ser destruída na totalidade, através da execução do *job* manual `destroy:all`, ou podem ser destruídos apenas os Windows *nodes* (`destroy:win-nodes`) ou Linux *nodes* (`destroy:linux-nodes`).

5.6. Sumário

O presente capítulo apresenta os detalhes mais técnicos da implementação da solução desenhada no capítulo 4 no contexto dos casos de uso definidos. A solução implementada deverá ser avaliada face aos objetivos inicialmente propostos (*cf.,* capítulo 6), com base na questão (*cf.,* secção 1.5) e hipóteses de investigação (*cf.,* secção 1.6) previamente estabelecidas permitindo desta forma, que sejam obtidas conclusões sobre as limitações e trabalho futuro (*cf.,* capítulo 7).

Capítulo 6

Experimentação e Avaliação

Terminada a implementação da solução é necessário experimentá-la a fim de compreender se os objetivos e requisitos formulados foram alcançados.

No presente capítulo são inicialmente relembradas as hipóteses de investigação definidas, apresentada a metodologia de avaliação utilizada e descritos os indicadores de avaliação tidos em consideração. Posteriormente apresentam-se os resultados obtidos na execução de testes funcionais e experiências à solução e é realizada uma avaliação global através da verificação das hipóteses de investigação no contexto dos indicadores definidos.

6.1. Hipóteses de Investigação

Na secção 1.6 foram definidas quatro hipóteses de investigação que devem ser tidas em conta no momento da experimentação e avaliação da solução. Como tal, encontram-se listadas abaixo:

- **H. 1:** Todas as tarefas de provisionamento da infraestrutura devem ser automatizadas;
- **H. 2:** Todas as tarefas de instalação do MAM4PRO devem ser automatizadas;
- **H. 3:** Deve ser possível implantar o MAM4PRO em até um minuto;
- **H. 4:** Depois de implantado o MAM4PRO deve ser possível alterar algumas configurações.

A metodologia de avaliação envolveu a escolha de um conjunto de critérios e a eleição de indicadores consistentes que permitem avaliar o desempenho de algo mediante os padrões estabelecidos. Para avaliar as quatro hipóteses de investigação definidas foram realizados vários testes funcionais à solução implementada com o objetivo de analisar o comportamento da solução implementada no contexto dos indicadores de avaliação definidos.

6.2. Indicadores de Avaliação

Os indicadores de avaliação pretendem auxiliar na avaliação de um projeto. No contexto desta dissertação elegeram-se os seguintes indicadores de avaliação:

- **I. 1:** Número de tarefas manuais necessárias para provisionamento da infraestrutura e implantação do MAM4PRO;
- **I. 2:** Tempo de implantação do MAM4PRO;
- **I. 3:** Possibilidade de alterar algumas configurações depois de implantado o *software*.

Todos os indicadores de avaliação estão relacionados com as hipóteses de investigação: o primeiro indicador (I. 1) relaciona-se com as duas primeiras hipóteses de investigação definidas (H. 1 e H. 2.), o segundo indicador (I. 2) relaciona-se com a terceira hipótese de investigação definida (H. 3) e o terceiro indicador (I. 3) permite verificar a quarta hipótese de investigação (H. 4). Em relação a este último indicador, após implantação do MAM4PRO deve ser possível alterar as seguintes configurações:

- Alterar a versão do *software* implantado (*i.e.,* atualizar as imagens dos *containers* utilizados);
- Alterar o endereço *web* de acesso ao MAM4PRO sem necessidade de implantar novamente o *software*;
- Adicionar dois serviços `rPlayerWeb` ao *deployment*.

6.3. Resultados

Uma vez apresentadas as hipóteses de investigação, a metodologia de investigação aplicada e os indicadores considerados, o próximo passo é apresentar os resultados alcançados com base nos testes funcionais realizados à solução implementada. Assim, inicialmente apresenta-se a análise dos resultados obtidos (*i.e.,* duração do processo de instalação do MAM4PRO) durante a utilização da abordagem anterior, e de seguida para a solução implementada (*i.e.,* disponibilização do MAM4PRO como um serviço).

Por fim, apresenta-se a avaliação global das hipóteses de investigação considerando os indicadores definidos.

6.3.1. Abordagem Anterior

Inicialmente recolheram-se os tempos de duração do processo de instalação do MAM4PRO de duas formas: totalmente manual através da execução do instalador na máquina pretendida (tabela 6.1A) e de forma automática através da execução de uma *pipeline* de CI/CDE (tabela 6.1B). A execução do instalador de forma automática apenas está disponível para implantações realizadas pela equipa de desenvolvimento do MAM4PRO, uma vez que a *pipeline* de CI/CDE possui um *job* para esse propósito. As instalações do MAM4PRO realizadas em máquinas para enviar para os clientes ou realizadas por outros colaboradores da MOG Technologies utilizam sempre a abordagem A (manual).

A instalação foi realizada três vezes em três servidores diferentes (detalhes na tabela E6) para as duas formas apresentadas (*i.e.,* manual e automático) perfazendo assim um total de dezoito testes.

Pela análise da tabela 6.1 e da figura 6.1 concluí-se que a abordagem A (manual) se revela mais rápida, apresentando uma média ($\bar{x}$) de 6 minutos e 25 segundos de tempo de execução. Embora esta abordagem seja totalmente manual e mais inoportuna (por obrigar a que uma pessoa esteja conectada ao servidor de forma síncrona para execução do instalador), ela é mais rápida porque consiste apenas na execução do instalador. No caso da abordagem B (automático) como se traduz na execução de um *job* da *pipeline* de CI/CDE existem tarefas que são realizadas antes da própria execução do instalador). Além disso, a abordagem B (automático) apenas está disponível para um número reduzido de colaboradores da empresa e por isso não apresenta muita relevância.

Nos três testes realizados em cada um dos servidores, os tempos são semelhantes ($\sigma < 1$ minuto). Na verdade, os três servidores utilizados apresentam características de *hardware* e *software* semelhantes, no entanto servidores que apresentem um melhor desempenho (*i.e.,* mais e melhores recursos físicos) poderão apresentar maior velocidade na instalação do MAM4PRO.

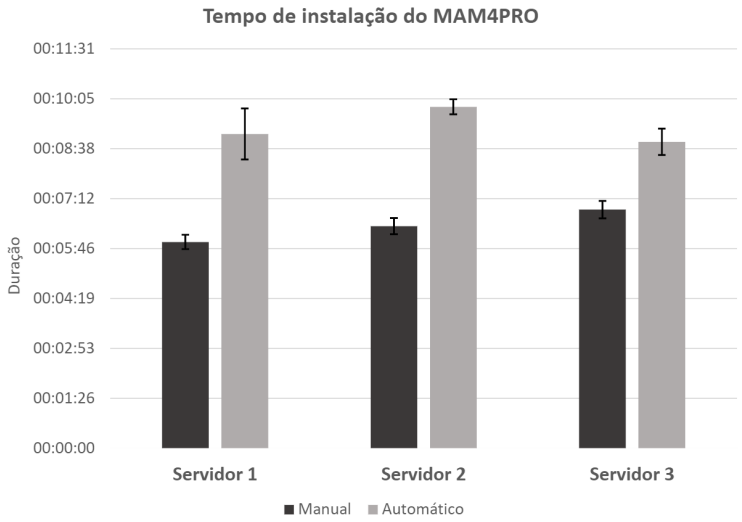


Figura 6.1. Tempo de instalação do MAM4PRO nas duas abordagens apresentadas (manual e automático), por servidor utilizado.

Tabela 6.1. Tempo de instalação do MAM4PRO. (A) Instalação totalmente manual, através da execução do instalador. (B) Instalação automática, através da execução de uma *pipeline* de CI/CDE. Note que $\bar{x}_s$ e σ_s corresponde à média e desvio padrão dos tempos de instalação para cada servidor (s). $\bar{x}$ e σ corresponde à média global dos tempos de instalação da abordagem anterior.

	A. Manual			**B. Automático**		
	Servidor 1	**Servidor 2**	**Servidor 3**	**Servidor 1**	**Servidor 2**	**Servidor 3**
Teste 1	00:05:41	00:06:09	00:06:34	00:09:55	00:10:07	00:08:53
Teste 2	00:06:11	00:06:43	00:07:11	00:09:10	00:09:35	00:08:21
Teste 3	00:05:59	00:06:21	00:06:54	00:08:07	00:09:51	00:09:17
$\bar{x}_s$	00:05:57	00:06:24	00:06:53	00:09:04	00:09:51	00:08:50
σ_s	00:00:12	00:00:14	00:00:15	00:00:44	00:00:13	00:00:23
$\bar{x}$	**00:06:25**			**00:09:15**		
σ	00:00:23			00:00:26		

6.3.2. Solução Implementada

Ao contrário do que acontece na abordagem anterior em que os servidores de teste utilizados são os próprios servidores onde o *software* é instalado, na solução implementada (*i.e.,* disponibilização do MAM4PRO como um serviço) os servidores que processam o trabalho são os *runners* que executam a *pipeline* de CI/CDE. Assim, foi realizado um teste em três *runners* diferentes, perfazendo um total de três testes.

6.3.2.1. Criação da Infraestrutura

Executados os *stages validate*, *plan* e *apply* nos três *runners* obteve-se os valores apresentados na tabela 6.2.

Tabela 6.2. Tempo de criação de toda a infraestrutura.

Stage	***Job***	***Runner* 1**		***Runner* 2**		***Runner* 3**	
Validate	`validate`	00:00:18		00:00:14		00:00:14	
Plan	`lin-nodes`	00:00:09	00:00:29	00:00:09	00:00:30	00:00:07	00:00:24
	`masters`	00:00:10		00:00:10		00:00:08	
	`win-nodes`	00:00:10		00:00:11		00:00:09	
Apply	`lin-nodes`	00:01:44	00:11:55	00:03:38	00:13:25	00:01:28	00:11:07
	`masters`	00:07:29		00:07:11		00:07:02	
	`win-nodes`	00:02:42		00:02:36		00:02:37	
Total		00:12:42		00:13:55		00:11:45	
$\bar{x}$		**00:12:47**					
σ		00:00:53					

A criação da infraestrutura demora em média ($\bar{x}$) cerca de 12 minutos e 47 segundos. Nos três *runners* utilizados os valores foram semelhantes, no entanto o desvio padrão (σ) calculado é de 53 segundos, justificado pelas características dos *runners* e pela velocidade de conexão à internet.

No geral, o valor obtido (12 minutos e 47 segundos) parece bastante promissor, dada a complexidade da infraestrutura criada e que provisionada de forma manual demoraria bastante mais tempo uma vez que seria necessário conectar a cada um dos servidores e executar todos os comandos de provisionamento em cada um deles. Além disso, devido ao fator humano poderiam ocorrer erros e os resultados não seriam previsíveis.

6.3.2.2. Destruição da Infraestrutura

Conforme referido na secção 5.5, a destruição da infraestrutura pode ser realizada de uma única vez, através da execução do *job* `all`, ou parcialmente através da execução dos *jobs* `lin-nodes` e/ou `win-nodes`. A tabela 6.3 apresenta os valores obtidos na execução dos vários *jobs* que compõe o *stage destroy*.

Tabela 6.3. Tempo de destruição da infraestrutura.

Stage	***Job***	***Runner* 1**	***Runner* 2**	***Runner* 3**	$\bar{x}$	σ
Destroy	`all`	00:02:26	00:02:27	00:02:30	**00:02:28**	00:00:02
	`lin-nodes`	00:01:13	00:01:10	00:01:15	**00:01:13**	00:00:02
	`win-nodes`	00:00:23	00:00:19	00:00:21	**00:00:21**	00:00:02

Pela análise da tabela 6.3 verifica-se que dos três testes realizados em três *runners* diferentes, a destruição completa da infraestrutura dura em média ($\bar{x}$) 2 minutos e 28 segundos (`destroy:all`). Nos três *runners* utilizados os valores são semelhantes, obtendo-se um desvio padrão (σ) de 2 segundos.

6.3.2.3. Implantação do MAM4PRO

Conforme referido na secção 5.4.4 a implantação do MAM4PRO pode ser realizada através da execução de um comando (*i.e.,* `helm install`) e por isso mais manual ou pode ser através de uma *pipeline* de CI/CDE. Para os resultados apresentados na tabela 6.4 não foi considerada a primeira forma de implantação referida.

Analisando a tabela 6.4 verifica-se que os tempos de implantação do MAM4PRO através da solução implementada se revelam bastante interessantes e promissores. Uma implantação do MAM4PRO dura, em média, 22 segundos. Para os três *runners* utilizados os tempos revelaram-se bastante semelhantes, tendo-se obtido um desvio padrão irrelevante.

É de salientar que estes valores refletem o tempo de execução da *pipeline* de CI/CDE existente no repositório, no entanto existe um espaço temporal necessário para que o Kubernetes consiga iniciar os *containers*. Este tempo pode ser maior ou menor dependendo das imagens de *container* utilizadas, isto é, se no *node* onde executar o *pod* já possuir a imagem do *container* este processo será significativamente mais rápido uma vez que o *download* já não precisa de ser realizado.

Tabela 6.4. Tempo de instalação do MAM4PRO segundo a solução implementada. Note que $\bar{x}_r$ e σ_r corresponde à média e desvio padrão dos tempos de instalação para cada *runner* (r). $\bar{x}$ e σ corresponde à média global dos tempos de instalação segundo a solução implementada.

	***Runner* 1**	***Runner* 2**	***Runner* 3**
Teste 1	00:00:21	00:00:23	00:00:20
Teste 2	00:00:23	00:00:25	00:00:22
Teste 3	00:00:20	00:00:21	00:00:19
$\bar{x}_r$	00:00:21	00:00:23	00:00:20
σ_r	00:00:01	00:00:02	00:00:11
$\bar{x}$	**00:00:22**		
σ	00:00:00		

6.3.3. Avaliação Global

Por fim, é necessário comprovar ou rejeitar as hipóteses de investigação definidas com base nos resultados obtidos.

6.3.3.1. Hipótese 1

Em relação à primeira hipótese de investigação definida e com base nos resultados obtidos é visível que a tarefa de acionar os *jobs* (`apply:masters`, `apply:lin-nodes` e `apply:win-nodes`) na *pipeline* de CI/CDE permanece manual (*cf.,* secção 5.5). Poderia ter sido adicionado o mecanismo de provisionamento automático caso fossem detetadas alterações no código-fonte do repositório. No entanto, optou-se por não implementar esse automatismo porque nem sempre é desejável que uma alteração no código-fonte produza efeitos na infraestrutura de forma imediata. Assim, conclui-se que a primeira hipótese não se verifica.

6.3.3.2. Hipótese 2

Relativamente à segunda hipótese, o número de tarefas manuais na implantação do MAM4PRO é semelhante ao comportamento obtido no provisionamento da infraestrutura. Existe pelo menos uma tarefa manual, que é o acionar o *job* da *pipeline* de CI/CDE no painel do GitLab. Posto isto, conclui-se que a segunda hipótese de investigação definida não se comprova.

6.3.3.3. Hipótese 3

No que respeita à terceira hipótese de investigação, a implantação do MAM4PRO demora em média 22 segundos, sendo bastante menor que o tempo estabelecido de um minuto. Assim, pode-se afirmar que a terceira hipótese de investigação se verifica.

6.3.3.4. Hipótese 4

Por último, e de acordo com o definido, a verificação da quarta hipótese implica a realização de algumas tarefas, nomeadamente:

Alterar a versão do *software* implantado
Cada versão do MAM4PRO que é lançada armazena no *registry* novas imagens dos *containers* com uma nova *tag*. Assim, sempre que uma nova versão for lançada, e caso exista a necessidade de atualizar um *deployment* já existente, basta editar a *tag* das imagens dos *containers* que estão a ser utilizadas. Esta tarefa pode ser realizada diretamente editando os *deployments* no *dashboard* Kubernetes, porém não é uma prática recomendada porque em caso de falha esta alteração é perdida uma vez que não ficou descrita em código. Assim, recomenda-se que a atualização e um *deployment* para uma versão mais recente implique alteração do ficheiro `values.yaml`. Assim que o ficheiro for editado deve ser executado um comando `helm upgrade` para atualizar o *deployment*.

Alterar o URL de acesso ao MAM4PRO sem necessidade de implantar novamente o *software*
O endereço de acesso ao MAM4PRO é definido através do *ingress* e depois de implantado o MAM4PRO é possível alterar esse endereço ou através do *dashboard* Kubernetes ou através da alteração do parâmetro `ingress.hostname` do ficheiro `values.yaml` utilizado no *deployment*. A segunda alternativa implica que seja executado um comando de `helm upgrade` para atualizar o *deployment*.

Adicionar dois serviços `rPlayerWeb` ao *deployment*
Com o cumprimento do UC. 6 (*i.e.,* escalar o número de serviços de um *deployment*) é possível escalar o número de réplicas do `rPlayerWeb`.

Executadas estas três tarefas com sucesso pode concluir-se que a quarta hipótese de investigação definida se verifica.

6.4. Sumário

Foram definidos três indicadores de avaliação distintos que permitem avaliar o projeto desenvolvido: o número de tarefas manuais necessárias para provisionamento da infraestrutura e implantação do MAM4PRO (I. 1), o tempo necessário para implantar o MAM4PRO (I. 2) e a possibilidade de alterar algumas configurações depois de implantado o MAM4PRO (I. 3). Estes indicadores permitem comprovar ou rejeitar as quatro hipóteses de investigação definidas (*cf.* secção 1.6). A avaliação de cada uma das hipóteses foi conseguida através da realização de testes e experiências à solução implementada.

Relativamente aos resultados obtidos, verificou-se que as duas primeiras hipóteses de investigação foram rejeitadas porque existem tarefas, que por escolha ou imposição, permanecem manuais. A terceira e quarta hipóteses foram comprovadas.

No capítulo seguinte intitulado Conclusões apresentam-se as conclusões do trabalho desenvolvido de acordo com os objetivos que delineados.

Capítulo 7

Conclusões

Esta dissertação permitiu provisionar de forma automatizada uma infraestrutura nas instalações da MOG Technologies com o objetivo de disponibilizar o *software* como um serviço. Este desenvolvimento permite à MOG Technologies evoluir a forma como o *software* é distribuído e procura servir como ponto de partida para a implantação do *software* numa *cloud* pública.

No presente capítulo, pretende-se apresentar as conclusões de acordo com os objetivos inicialmente definidos na secção 1.3:

- Provisionamento de uma infraestrutura (*i.e., cloud* privada alojada na MOG Technologies) capaz de suportar o *deployment* de aplicações, nomeadamente do MAM4PRO;
- Aplicação dos conceitos de SaaS e IaC;
- Garantir a portabilidade e a escalabilidade dos serviços para que rapidamente a infraestrutura possa ser utilizada para o desempenho de outras funções;
- Integrar o desenvolvimento numa *pipeline* de CI/CDE para que o processo seja automatizado;
- Desenvolvimento de um *dashboard* capaz de estabelecer comunicação com a API da infraestrutura para permitir o escalonamento e a elasticidade dos serviços.

Após identificação e análise o problema seguiram-se as filosofias e diretrizes recomendadas pela metodologia de investigação DSR que permitiram levar a cabo esta dissertação. Numa primeira fase, foram estudados todos os conceitos teóricos subjacentes a esta dissertação de forma a perceber o estado atual da arte e das tecnologias envolvidas (*cf.,* capítulo 2). Além disso foi realizada uma análise de valor a fim de identificar e aproveitar a oportunidade identificada (*cf.,* capítulo 3) e aplicando o método AHP percebeu-se que a tecnologia de orquestração de *containers* mais adequada para resolução do problema era o Kubernetes.

A escolha das tecnologias para implementação da solução e a definição dos requisitos funcionais e não funcionais da solução permitiram desenhar a solução (*cf.,* capítulo 4) da forma mais adequada à resolução do problema identificado para posterior implementação (*cf.,* capítulo 5).

Relativamente aos três primeiros objetivos, como se comprova pelo capítulo 5, foi possível a implementação de um *cluster* Kubernetes provisionado com recurso a IaC e à tecnologia Terraform. Este *cluster* suporta a implantação automatizada do MAM4PRO pondo em prática os conceitos de SaaS. Ademais, a portabilidade e escalabilidade dos serviços características da tecnologia Kubernetes foram utilizadas para satisfazer o terceiro objetivo.

O quarto objetivo diz respeito à automatização da solução que foi conseguida pela criação de uma *pipeline* de CI/CDE como se comprova pela secção 5.5. Por último, apesar de inicialmente se ter definido que deveria ser desenvolvido um *dashboard* que permitisse realizar o escalonamento dos serviços, durante o desenvolvimento desta dissertação percebeu-se que tal não seria necessário. Alternativamente, optou-se por utilizar o *dashboard* Kubernetes que cumpre com os requisitos e propósito.

Além dos objetivos propostos, todos os casos de uso do sistema foram implementados, permitindo assim cumprir com os requisitos funcionais da solução definidos na secção 3.3.2.1. No que respeita aos requisitos não funcionais da solução, embora não exista nenhuma forma concisa e objetiva de os comprovar conclui-se que na globalidade também foram cumpridos.

Com o objetivo de experimentar e avaliar a solução implementada, procedeu-se à avaliação da solução com base em alguns indicadores, seguindo uma metodologia baseada em experiências e testes funcionais à solução (*cf.,* capítulo 6).

7.1. Questões de Investigação

No início desta dissertação, o problema em estudo foi traduzido numa única questão de investigação para tornar a investigação mais objetiva e precisa.

De que forma se conseguirá automatizar a criação de uma infraestrutura capaz de suportar o *deployment* do MAM4PRO?

Partindo da análise do problema identificado (*cf.,* secção 1.2) e com a solução implementada é possível verificar que recorrendo a técnicas de IaC com recurso à tecnologia Terraform é possível automatizar a criação de um *cluster* Kubernetes capaz de suportar a implantação do *software* MAM4PRO.

7.2. Contribuições

Além da implementação da solução apresentada e do presente documento, o desenvolvimento desta dissertação resultou também na escrita de documentação para uso interno na MOG Technologies (*cf.,* apêndice G.1) e em duas apresentações internas de disseminação de conhecimento, intituladas *Cloud Deploys - Olhar para o Futuro* e *Desmistificado Kubernetes no Mundo MOG* (*cf.,* apêndice G.2).

Adicionalmente encontra-se em curso a escrita de um artigo sobre o estudo e solução apresentada nesta dissertação, para submissão no Simpósio de Engenharia Informática 2022 (SEI'22)[1], iniciativa dinamizada pelo Instituto Superior de Engenharia do Porto.

7.3. Limitações e Dificuldades

Na sua generalidade, a implementação da solução foi bem sucedida, o que se comprova pela experimentação e avaliação dos resultados conseguidos. No entanto, identificaram-se algumas limitações que dificultaram o desenvolvimento da presente dissertação, nomeadamente:

[1] `http://sei.dei.isep.ipp.pt/index.html`

- Extensão da dissertação: a extensão do tema desta dissertação levou a que alguns conceitos não pudessem ser tão aprofundados como inicialmente previsto;
- Necessidade de criação de uma VLAN isolada dificultou a arquitetura da solução implementada;
- Atualização da versão do Kubernetes: durante a implementação da solução foi lançada a versão `1.24` do Kubernetes, e após análise das notas de lançamento considerou-se que as alterações incorporadas nesta versão eram significativas, e por isso era mais benéfico adaptar a solução desde o início;
- A não existência de um *provider* Terraform que possibilite a implantação de um *cluster* Kubernetes *on-premise* dificultou o desenvolvimento.

7.4. Trabalho Futuro

A solução implementada representa uma alteração significativa na forma de distribuição do *software* da MOG Technologies. Devido à extensão da dissertação não foi possível explorar alguns conceitos interessantes. Existem ainda bastantes possibilidades de implementação de melhorias e de evolução, sendo as mais relevantes:

- Alargar o conhecimento a mais pessoas da organização;
- Otimização de alguns aspetos relacionados com o provisionamento da infraestrutura;
- Implementação de *workflows* de GitOps;
- Implantação dos *runners* GitLab no *cluster* objetivando a eliminação da necessidade de máquinas dedicadas para o efeito;
- Explorar e tirar partido da integração do GitLab com o Kubernetes.

7.5. Observações Pessoais

Terminada a realização deste projeto posso afirmar que completei uma das etapas mais importantes para mim. Esta dissertação permitiu-me alargar os meus conhecimentos a uma área que ainda não me era muito familiar, como é o caso dos conceitos de DevOps, SaaS, IaC e da orquestração de *containers*. A curiosidade e a motivação proporcionada pelo problema em análise foi, sem dúvida, um dos fatores chave ao longo desta dissertação, que na sua globalidade, cumpriu com os objetivos propostos.

As pesquisas e os desenvolvimentos realizados permitiram alavancar conceitos e práticas ainda muito embrionárias na MOG Technologies, e que servirão como ponto de partida para uma disponibilização do *software* como um serviço numa *cloud* pública.

Por último, quero expressar o meu sentimento de satisfação e orgulho no projeto desenvolvido e ter contribuído para a melhoria de algo numa organização onde exerço funções há quatro anos.

Bibliografia

[1] Grass Valley Canada, «Disrupting Traditional Broadcasting Through Digital Transformation and Social/Digital Production,» p. 4, URL: https://wwwapps.grassvalley.com/docs/WhitePapers/media_workflow/gv_alyve/GV_Alyve_WP-PUB-2-0881A-EN.pdf (acedido em 04/04/2022).

[2] A. M. Tekalp, *Digital Video Processing*, 2nd ed. New York: Prentice-Hall, 2015, ISBN: 978-0-13-399100-0.

[3] B. M. Taveira Pereira, «Virtualização de Estúdios Móveis na Produção de Conteúdos Audiovisuais em Direto,» tese de mestrado, Faculdade de Engenharia da Universidade do Porto, Porto, 2017. URL: https://repositorio-aberto.up.pt/bitstream/10216/102556/2/179801.pdf (acedido em 22/04/2022).

[4] W. Sun, X. Zhang, C. J. Guo, P. Sun e H. Su, «Software as a Service: Configuration and Customization Perspectives,» em *2008 IEEE Congress on Services Part II (services-2 2008)*, 2008, pp. 18–25. DOI: 10.1109/SERVICES-2.2008.29.

[5] I. Sommerville, *Software Engineering*, 9th ed. Boston: Pearson, 2011, ISBN: 978-0-13-703515-1.

[6] J. L. Noll e F. Pretto, «Implementação de infraestrutura como código para provisionamento e deploy de aplicações,» pt, *Revista Destaques Acadêmicos*, vol. 12, n.º 4, dez. de 2020, ISSN: 2176-3070. DOI: 10.22410/issn.2176-3070.v12i4a2020.2760.

[7] D. Sato, *DevOps: Na Prática: entrega de software confiável e automatizada.* 1st ed. Casa do Código, 2014, ISBN: 978-8-56-625040-4.

[8] B. Beyer, C. Jones, J. Petoff e N. R. Murphy, *Engenharia de Confiabilidade do Google - Como o Google administra seus sistemas de produção*, 1st ed. Brasil: Novatec, 2016, ISBN: 978-8-57-522517-2.

[9] S. Goericke, ed., *The Future of Software Quality Assurance*, Cham: Springer International Publishing, 2020. DOI: 10.1007/978-3-030-29509-7.

[10] *The Main Pillars of the DevOps Toolchain.* URL: https://www.whitesourcesoftware.com/rc-content/wp/the-main-pillars-of-the-devops-toolchain.pdf (acedido em 24/04/2022).

[11] J. Humble e D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test and Deployment Automation.* 1st ed. Addison-Wesley Professional, 2010, ISBN: 978-0-32-167025-0.

[12] M. Shahin, M. Ali Babar e L. Zhu, «Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,» *IEEE Access*,

vol. 5, pp. 3909–3943, 2017, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2685629.

[13] S. Bobrovskis e A. Jurenoks, «A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment,» p. 9, URL: http://ceur-ws.org/Vol-2218/paper31.pdf (acedido em 13/10/2022).

[14] M. Rosenblum, «The Reincarnation of Virtual Machines: Virtualization Makes a Comeback.,» *Queue*, vol. 2, n.º 5, pp. 34–40, 2004, ISSN: 1542-7730. DOI: 10.1145/1016998.1017000.

[15] A. Mouat, *Using Docker - Developing and Deploying Software With Containers*, 1st ed. United States of America: O'Reilly Media, Inc., 2015, ISBN: 978-1-49-191576-9.

[16] D. C. dos Santos Reis, «Execução e Gestão de Aplicações Conteinerizadas,» tese de mestrado, Faculdade de Ciências da Universidade do Porto, Porto, 2017. URL: https://repositorio-aberto.up.pt/handle/10216/109650.

[17] Brendan Burns, Joe Beda e Kelsey Hightower, *Kubernetes: Up & Running*, 2nd ed. United States of America: O'Reilly Media, Inc., 2019, ISBN: 978-1-49-204653-0.

[18] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, Inc., 2021, ISBN: 978-1-49-195035-7.

[19] R. Chen, S. Li e Z. Li, «From Monolith to Microservices: A Dataflow-Driven Approach,» em *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 466–475. DOI: 10.1109/APSEC.2017.53.

[20] O. Al-Debagy e P. Martinek, «A Comparative Review of Microservices and Monolithic Architectures,» em *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2018, pp. 000 149–000 154. DOI: 10.1109/CINTI.2018.8928192.

[21] L. Wang, G. von Laszewski, A. Younge et al., «Cloud Computing: a Perspective Study,» *New Generation Computing*, vol. 28, n.º 2, pp. 137–146, abr. de 2010, ISSN: 1882-7055. DOI: 10.1007/s00354-008-0081-5.

[22] P. Buxmann, T. Hess e S. Lehmann, «Software as a Service,» *WIRTSCHAFTSINFORMATIK*, vol. 50, n.º 6, pp. 500–503, dez. de 2008, ISSN: 1861-8936. DOI: 10.1007/s11576-008-0095-0.

[23] S. Naziris, «Infrastructure as Code - Towards Dynamic and Programmable IT Systems,» tese de mestrado, Faculty of Electrical Engineering, Mathematics e Computer Science of University of Twente, Netherlands, 2019. (acedido em 22/04/2022).

[24] S. Nelson-Smith, *Test-Driven Infrastructure with Chef*, 2nd ed. United States of America: O'Reilly Media, Inc., 2015, ISBN: 978-1-44-937220-0.

[25] K. Morris, *Infrastructure as Code - Managing Servers in the Cloud*, 1st ed. United States of America: O'Reilly Media, Inc., 2016, ISBN: 978-1-49-192435-8.

[26] Y. Brikman, *Terraform: Up & Running: Writing Infrastructure as Code*, 2nd ed. United States of America: O'Reilly Media, Inc., 2019, ISBN: 978-1-49-204690-5.

[27] B. N. Gupta e N. Gupta, *Research Methodology*, 1st ed. SBPD Publications, 2021.

[28] D. P. Lacerda, A. Dresch, A. Proença e J. A. V. Antunes Júnior, «Design Science Research: método de pesquisa para a engenharia de produção,» *Gestão & Produção*, vol. 20, n.º 4, pp. 741–761, 26 de nov. de 2013, ISSN: 1806-9649, 0104-530X. DOI: 10.1590/S0104-530X2013005000014.

[29] S. Easterbrook, J. Singer, M.-A. Storey e D. Damian, «Selecting Empirical Methods for Software Engineering Research,» em *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer e D. I. K. Sjøberg, eds. London: Springer London, 2008, pp. 285–311, ISBN: 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_11.

[30] W. Hasselbring e S. Giesecke, *Research Methods in Software Engineering*. Berlin, Germany, 2006, ISBN: 978-3-93-677157-2.

[31] N. Bayazit, «Investigating Design: A Review of Forty Years of Design Research,» *Design Issues*, vol. 20, n.º 1, pp. 16–29, jan. de 2004, ISSN: 0747-9360. DOI: 10.1162/074793604772933739.

[32] M. Staron, «Action Research as Research Methodology in Software Engineering,» em *Action Research in Software Engineering*. Cham: Springer International Publishing, 2020, pp. 15–36. DOI: 10.1007/978-3-030-32610-4_2.

[33] D. B. Bisandu, *Design Science Research Methodology in Computer Science and Information Systems*, en, 2019. URL: https://www.researchgate.net/publication/330041672_Design_Science_Research_Methodology_in_Computer_Science_and_Information_Systems.

[34] A. R. Hevner, S. T. March, J. Park e S. Ram, «Design Science in Information Systems Research,» en, p. 33, 2004.

[35] *Action Research*. URL: https://ctl.oregonstate.edu/sites/ctl.oregonstate.edu/files/action-research.pdf (acedido em 09/05/2022).

[36] K. H. Oliveira da Fonseca, «Investigação-Ação: Uma metodologia para prática e reflexão docente,» *Revista Onis Ciência, Braga*, 2012, ISSN: 2182-598X. URL: https://revistaonisciencia.com/wp-content/uploads/2020/02/2ED02-ARTIGO-KARLA.pdf (acedido em 05/05/2022).

[37] L. Dickens e K. Watkins, «Action Research: Rethinking Lewin,» *Management Learning*, vol. 30, n.º 2, pp. 127–140, 1999. DOI: 10.1177/1350507699302002.

[38] J. Iivari e J. Venable, «Action research and design science research - Seemingly similar but decisively dissimilar,» jan. de 2009, pp. 1642–1653. URL: https://www.researchgate.net/publication/221407297_Action_research_and_design_science_research_-_Seemingly_similar_but_decisively_dissimilar (acedido em 13/10/2022).

[39] N. Rich e M. Holweg, «Value Analysis Value Engineering,» en, Lean Enterprise Research Centre, Cardiff, United Kingdom, rel. téc., jan. de 2000, p. 32. URL: https://www.urenio.org/tools/en/value_analysis.pdf.

[40] L. A. Bonini e R. Sbragia, «O Modelo de Design Thinking como Indutor da Inovação nas Empresas: Um Estudo Empírico,» pt, *Revista de Gestão e Projetos*, vol. 2, n.º 1, p. 23, 2011. DOI: https://doi.org/10.5585/gep.v2i1.36.

[41] P. A. Koen, G. M. Ajamian, S. Boyce et al., «Fuzzy Front End: Effective Methods, Tools, and Techniques,» *The PDMA ToolBook for New Product Development*, p. 32, URL: https://media.wiley.com/product_data/excerpt/13/04712061/0471206113.pdf (acedido em 13/08/2022).

[42] P. Teza, V. Miguez, R. Fernandes e G. Dandolini, «Direcionadores do processo de inovação: o papel da estratégia, liderança e cultura,» *Navus: Revista de Gestão e Tecnologia*, vol. 3, nov. de 2013. DOI: 10.18815/navus.v3i2.156.

[43] D. Teoli, T. Sanvictores e J. An, *SWOT Analysis*, 2022. URL: http://europepmc.org/books/NBK537302 (acedido em 10/03/2022).

[44] A. Osterwalder, Y. Pigneur, G. Bernarda e A. Smith, *Value Proposition Design: How to Create Products and Services Customers Want*, 1st ed. New Jersey: John Wiley & Sons, Inc., 2014, ISBN: 978-1-11-896805-5.

[45] T. L. Saaty, *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World*, 3rd Ed. University of Pittsburgh, 2012, ISBN: 978-0-96-203170-0.

[46] S. Supraja e P. Kousalya, «A comparative study by AHP and TOPSIS for the selection of all round excellence award,» em *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 314–319. DOI: 10.1109/ICEEOT.2016.7755271.

[47] B. A. Rodrigues Soares, «Estudo da Aplicação de uma Metodologia Multicritério na Seleção de Fornecedores: Método AHP,» tese de mestrado, Lisbon School of Economics & Management - Univerisidade de Lisboa, Lisboa, 2021. URL: https://www.repository.utl.pt/handle/10400.5/21622.

[48] Suman e M. Wadhwa, «A Comparative Study of Software Quality Models,» em *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, 2014, p. 5.

[49] B. Behkamal, M. Kahani e M. K. Akbari, «Customizing ISO 9126 quality model for evaluation of B2B applications,» *Information and Software Technology*, vol. 51, n.º 3, pp. 599–609, 2009, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2008.08.001.

[50] E. Casalicchio e S. Iannucci, «The state-of-the-art in container technologies: Application, orchestration and security,» *Concurrency and Computation: Practice and Experience*, vol. 32, n.º 17, e5668, 2020, e5668 cpe.5668. DOI: https://doi.org/10.1002/cpe.5668.

[51] G. Sayfan, *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes*, 1st ed. Birmingham: Packt Publishing Ltd., 2017, ISBN: 978-1-78-646100-1.

[52] J. E. Leitão Freire, «Orquestração de Containers Usando Kubernetes e Docker Swarm,» tese de mestrado, Universidade da Beira Interior, Covilhã, 2021. URL: https://ubibliorum.ubi.pt/bitstream/10400.6/11091/1/7913_17373.pdf (acedido em 22/04/2022).

[53] S. K. Mondal, R. Pan, H. M. D. Kabir, T. Tian e H.-N. Dai, «Kubernetes in IT administration and serverless computing: An empirical study and research challenges,» *The Journal of Supercomputing*, vol. 78, n.º 2, pp. 2937–2987, fev. de 2022, ISSN: 0920-8542, 1573-0484. DOI: 10.1007/s11227-021-03982-3.

[54] A. A. Shah, G. Piro, L. A. Grieco e G. Boggia, «A Qualitative Cross-Comparison of Emerging Technologies for Software-Defined Systems,» en, em *2019 Sixth International Conference on Software Defined Systems (SDS)*, Rome, Italy: IEEE, jun. de 2019, pp. 138–145, ISBN: 978-1-72-810722-6. DOI: 10.1109/SDS.2019.8768566.

[55] J. Shah e D. Dubaria, «Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform,» em *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0184–0189. DOI: 10.1109/CCWC.2019.8666479.

[56] M. Hashimoto, *Vagrant Up and Running: Create and Manage Virtualized Development Environments*, 1st ed. United States of America: O'Reilly Media, Inc., 2013, ISBN: 978-1-44-933583-0.

[57] M. Butcher, M. Farina e J. Dolitsky, *Learning Helm - Managing Apps on Kubernetes*, 1st ed. United States of America: O'Reilly Media, Inc., 2021, ISBN: 978-1-49-208365-8.

[58] L. A. Macaulay, *Requirements Engineering*. Springer, ISBN: 978-1-44-711005-7.

[59] L. Bass, P. Clements e R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston: Addison-Wesley Professinal, 2003, ISBN: 978-0-32-115495-8.

Apêndice A

MAM4PRO

A.1. Desenho de Alto Nível

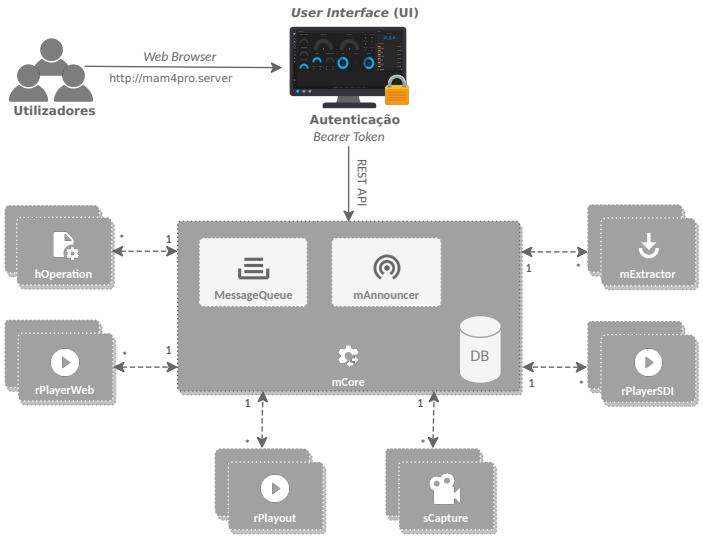


Figura A1. Arquitetura de alto nível do MAM4PRO: UI, `mCore` e *media services*.

A.2. *User Interface*

Figura A2. Componentes principais da UI: (A) *pages*, (B) *widgets* e (C) *dashboards*.

A.3. Conteinerização do MAM4PRO

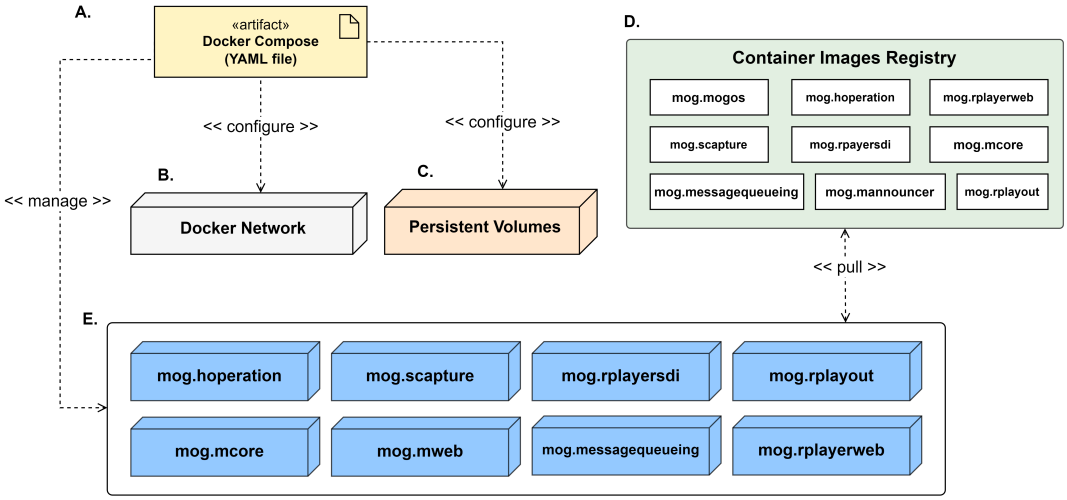


Figura A3. Conteinerização do MAM4PRO. (A) Ficheiro necessário para implantação do MAM4PRO. (B) Rede criada pelo Docker para estabelecer a comunicação entre os *containers*. (C) *Volume* de armazenamento necessário para garantir a persistência do sistema. (D) Imagens Docker necessárias e alojadas no *registry*. (E) *Containers* que serão instanciados assim que for implantado o MAM4PRO.

A.4. Imagens dos *containers*

De seguida apresentam-se todas as imagens de *containers* necessárias para suportar uma instalação do MAM4PRO:

- **`mogos`**: imagem de *container* baseada no Windows Server Core com componentes *Microsoft Foundation Classes* (MFC);
- **`mog.mannouncer`**: imagem de *container* contendo o serviço `mAnnouncer`. Todas as imagens de serviços contêm uma instância do `mAnnouncer`;
- **`mog.mcore`**: imagem de *container* com o `mCore` (base de dados e um *proxy* responsável por redirecionar pedidos da UI para outros *containers*);
- **`mog.messagequeueing`**: imagem de *container* composta pelo RabbitMQ, que é instalado em tempo de execução para ultrapassar algumas das limitações do Erlang;
- **`mog.hoperation`**: imagem de *container* composta pelo serviço `hOperation`;
- **`mog.rplayerweb`**: imagem de *container* composta pelo serviço `rPlayerWeb`;
- **`mog.rplayersdi`**: imagem de *container* composta pelo serviço `rPlayerSdi`;
- **`mog.scapture`**: imagem de *container* composta pelo serviço `sCapture`;
- **`mog.rplayout`**: imagem de *container* composta pelo serviço `rPlayout`, `rPlayerSdi` e `sCapture`.

Apêndice B

Ferramentas de Análise de Valor

B.1. Análise SWOT

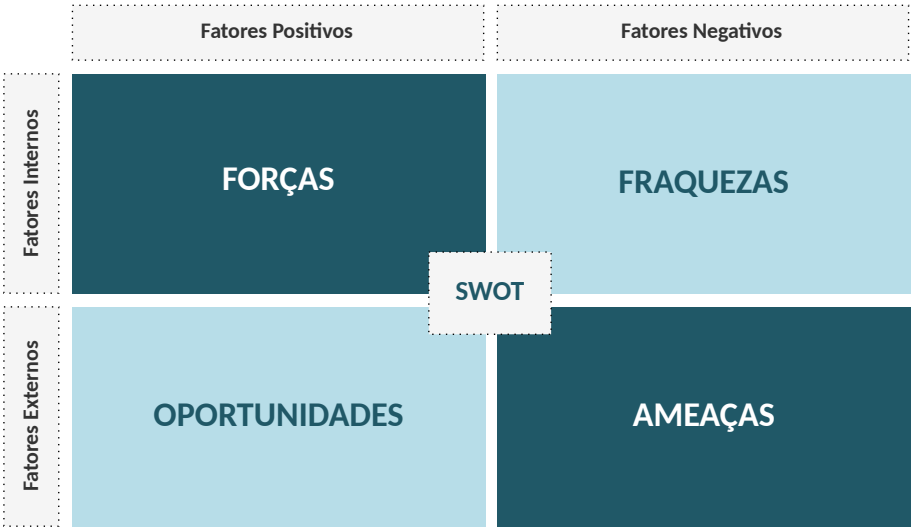


Figura B1. Análise SWOT *(Strengths, Weaknesses, Opportunities, Threats).*

B.2. *Value Proposition Canvas*

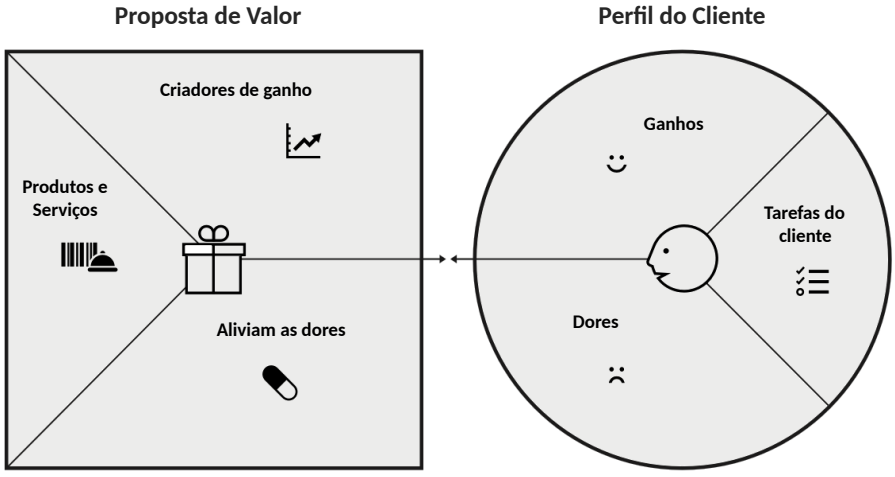


Figura B2. *Value Proposition Canvas.*

Apêndice C

Analytic Hierarchy Process (AHP)

C.1. Escala Fundamental de Saaty

Tabela C1. Escala fundamental de Saaty.

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

C.2. Valores do Índice Aleatório

Tabela C2. Valores do IA.

Ordem da Matriz	2	3	4	5	6	7	8	9
IA	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45

C.3. Aplicação do Método AHP

As tabelas seguintes resultam da aplicação do método AHP.

C.3.1. Priorização dos Critérios

Tabela C3. Comparação par-a-par de todos os critérios considerados.

	Critério A	**Critério B**	**Critério C**	**Critério D**	**Critério E**	**Critério F**
Critério A	**1**	6	0.20	**1**	0.33	**1**
Critério B	0.17	1	0.14	**1**	0.20	0.20
Critério C	5	7	**1**	5	**1**	5
Critério D	**1**	**1**	0.20	**1**	0.20	**1**
Critério E	3	5	**1**	5	**1**	5
Critério F	**1**	5	0.20	**1**	0.20	**1**

Tabela C4. Dados normalizados resultantes da comparação par-a-par de todos os critérios considerados e respetivas prioridades relativas (P_r).

	Critério A	**Critério B**	**Critério C**	**Critério D**	**Critério E**	**Critério F**	P_r
Critério A	0.09	0.24	0.07	0.07	0.11	0.08	**0.11**
Critério B	0.01	0.04	0.05	0.07	0.07	0.02	**0.04**
Critério C	0.45	0.28	0.36	0.36	0.34	0.38	**0.36**
Critério D	0.09	0.04	0.07	0.07	0.07	0.08	**0.07**
Critério E	0.27	0.20	0.36	0.36	0.34	0.38	**0.32**
Critério F	0.09	0.20	0.07	0.07	0.07	0.08	**0.10**
Total	**1**	**1**	**1**	**1**	**1**	**1**	**1**

C.3.2. Critério A - Complexidade

Tabela C5. Comparação par-a-par de todas as tecnologias consideradas segundo o critério A (complexidade).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	0.20	0.25	0.25	1
Tecnologia 2	5	**1**	5	1	5
Tecnologia 3	4	0.20	**1**	0.25	1
Tecnologia 4	4	1	4	**1**	3
Tecnologia 5	3	0.20	1	0.33	**1**

Tabela C6. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério A (complexidade) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.06	0.08	0.02	0.09	0.09	**0.07**
Tecnologia 2	0.29	0.38	0.44	0.35	0.45	**0.39**
Tecnologia 3	0.24	0.08	0.09	0.09	0.09	**0.12**
Tecnologia 4	0.24	0.38	0.36	0.35	0.27	**0.32**
Tecnologia 5	0.18	0.08	0.09	0.12	0.09	**0.11**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.3. Critério B - Documentação

Tabela C7. Comparação par-a-par de todas as tecnologias consideradas segundo o critério B (documentação).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	3	5	5	4
Tecnologia 2	0.33	**1**	4	4	3
Tecnologia 3	0.20	0.25	**1**	1	0.33
Tecnologia 4	0.20	0.25	1	**1**	0.33
Tecnologia 5	0.25	0.33	3	3	**1**

Tabela C8. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério B (documentação) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.50	0.62	0.36	0.36	0.46	**0.46**
Tecnologia 2	0.17	0.21	0.29	0.29	0.35	**0.26**
Tecnologia 3	0.10	0.05	0.07	0.07	0.04	**0.07**
Tecnologia 4	0.10	0.05	0.07	0.07	0.04	**0.07**
Tecnologia 5	0.13	0.07	0.21	0.21	0.12	**0.15**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.4. Critério C - Desempenho

Tabela C9. Comparação par-a-par de todas as tecnologias consideradas segundo o critério C (desempenho).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	1	3	3	1
Tecnologia 2	1	**1**	3	2	3
Tecnologia 3	0.33	0.33	**1**	0.33	2
Tecnologia 4	0.33	0.5	3	**1**	0.33
Tecnologia 5	1	0.33	0.5	0.3	**1**

Tabela C10. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério C (desempenho) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.27	0.32	0.29	0.32	0.14	**0.27**
Tecnologia 2	0.27	0.32	0.29	0.21	0.41	**0.30**
Tecnologia 3	0.09	0.11	0.10	0.04	0.27	**0.12**
Tecnologia 4	0.09	0.16	0.29	0.11	0.05	**0.14**
Tecnologia 5	0.27	0.11	0.05	0.32	0.14	**0.18**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.5. Critério D - Monitorização

Tabela C11. Comparação par-a-par de todas as tecnologias consideradas segundo o critério D (monitorização).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	5	3	3	3
Tecnologia 2	0.20	**1**	0.33	0.33	0.33
Tecnologia 3	0.33	3	**1**	1	1
Tecnologia 4	0.33	3	1	**1**	1
Tecnologia 5	0.33	3	1	1	**1**

Tabela C12. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério D (monitorização) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.45	0.33	0.47	0.47	0.47	**0.44**
Tecnologia 2	0.09	0.07	0.05	0.05	0.05	**0.06**
Tecnologia 3	0.15	0.20	0.16	0.16	0.16	**0.17**
Tecnologia 4	0.15	0.20	0.16	0.16	0.16	**0.17**
Tecnologia 5	0.15	0.20	0.16	0.16	0.16	**0.17**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.6. Critério E - Flexibilidade

Tabela C13. Comparação par-a-par de todas as tecnologias consideradas segundo o critério E (flexibilidade).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	5	7	1	7
Tecnologia 2	0.20	**1**	3	0.20	3
Tecnologia 3	0.14	0.33	**1**	0.20	1
Tecnologia 4	1	5	5	**1**	5
Tecnologia 5	0.14	0.33	1	0.20	**1**

Tabela C14. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério E (flexibilidade) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.40	0.43	0.41	0.38	0.41	**0.41**
Tecnologia 2	0.08	0.09	0.18	0.08	0.18	**0.12**
Tecnologia 3	0.06	0.03	0.06	0.08	0.06	**0.06**
Tecnologia 4	0.40	0.43	0.29	0.38	0.29	**0.36**
Tecnologia 5	0.06	0.03	0.06	0.08	0.06	**0.06**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.7. Critério F - Segurança

Tabela C15. Comparação par-a-par de todas as tecnologias consideradas segundo o critério F (segurança).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**
Tecnologia 1	**1**	5	1	5	3
Tecnologia 2	0.20	**1**	0.20	1	0.33
Tecnologia 3	1	5	**1**	5	3
Tecnologia 4	0.20	1	0.20	**1**	0.33
Tecnologia 5	0.33	3	0.33	3	**1**

Tabela C16. Dados normalizados resultantes da comparação par-a-par de todas as tecnologias consideradas segundo o critério F (segurança) e respetivas prioridades relativas (P_r).

	Tecnologia 1	**Tecnologia 2**	**Tecnologia 3**	**Tecnologia 4**	**Tecnologia 5**	P_r
Tecnologia 1	0.37	0.33	0.37	0.33	0.39	**0.36**
Tecnologia 2	0.07	0.07	0.07	0.07	0.39	**0.06**
Tecnologia 3	0.37	0.33	0.37	0.33	0.39	**0.36**
Tecnologia 4	0.07	0.07	0.07	0.07	0.04	**0.06**
Tecnologia 5	0.12	0.20	0.12	0.20	0.13	**0.15**
Total	**1**	**1**	**1**	**1**	**1**	**1**

C.3.8. Priorização das Tecnologias

Tabela C17. Prioridades relativas de cada uma das tecnologias para cada um dos critérios considerados.

	Critério A	**Critério B**	**Critério C**	**Critério D**	**Critério E**	**Critério F**
Tecnologia 1	0.07	**0.46**	**0.27**	**0.44**	**0.41**	**0.36**
Tecnologia 2	**0.39**	**0.26**	**0.30**	0.06	0.12	0.06
Tecnologia 3	0.12	0.07	0.12	0.17	0.06	**0.36**
Tecnologia 4	**0.32**	0.07	0.14	**0.17**	**0.36**	0.06
Tecnologia 5	0.11	0.15	0.18	**0.17**	0.06	0.15
Total	**1**	**1**	**1**	**1**	**1**	**1**

Apêndice D

Quality Function Deployment (QFD)

D.1. *House of Quality* (HOQ)

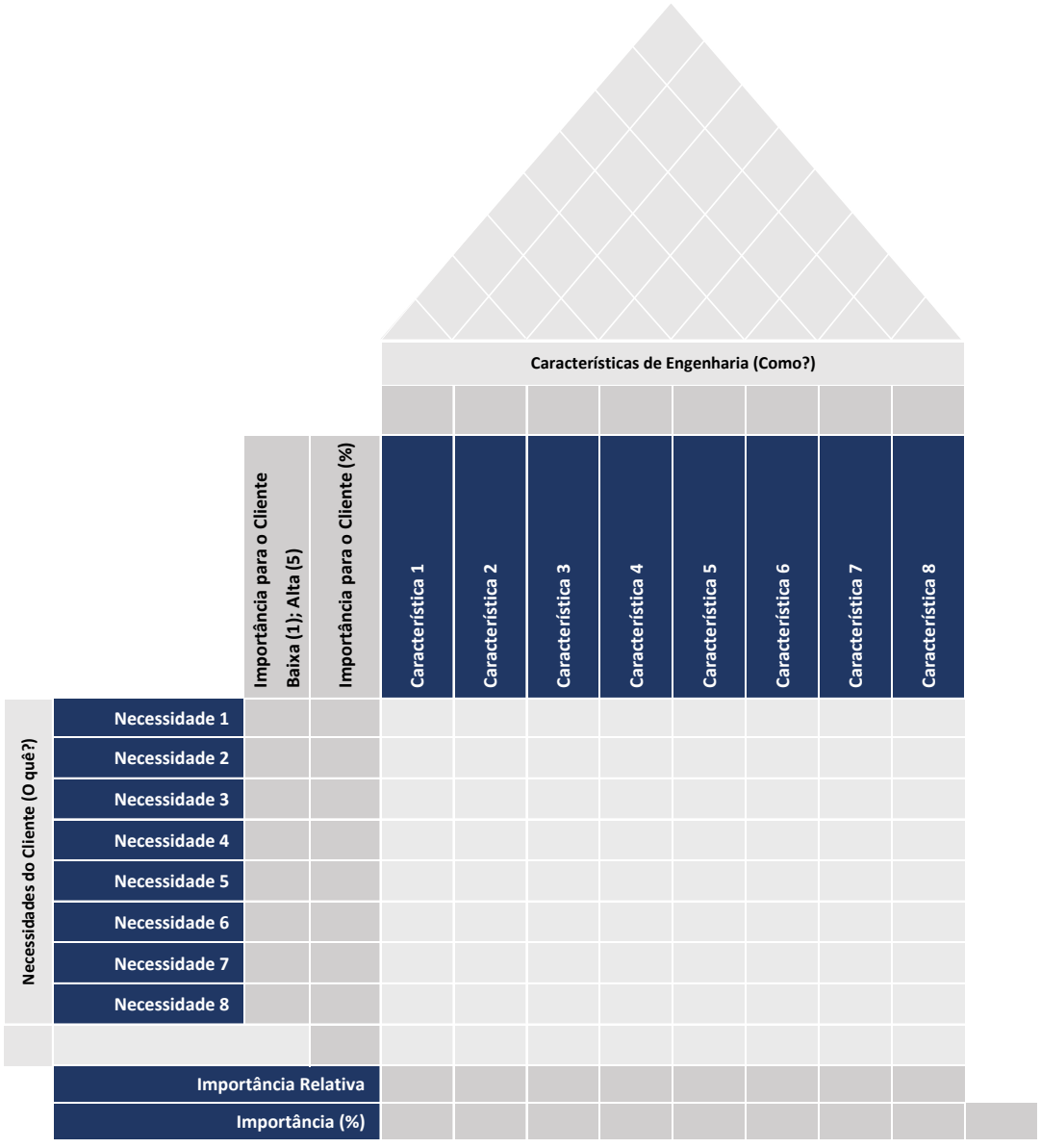


Figura D1. Ferramenta HOQ.

Apêndice E

Características dos Servidores

E.1. Características - Servidor M1-Todd

Tabela E1. Características (sistema operativo, CPU, armazenamento e memória RAM) do servidor M1-Todd responsável por executar as *pipelines* CI/CDE do projeto (*GitLab runner*), alojar o *registry* e *storage* dos *deployments.*

M1-Todd - 10.50.50.5			
Windows 10 IoT Enterprise 20H2	Intel Core i5-8600 @ 3.10GHz	SSD 120GB	16GB RAM

E.2. Características dos *Masters* do *Cluster*

Tabela E2. Características (sistema operativo, CPU, armazenamento e memória RAM) da máquina virtual Master-01 e do *host* M1-Skinner.

***Host* (M1-Skinner - 10.50.50.5)**			
Windows 10 IoT Enterprise 20H2	Intel Core i5-8600 @ 3.10GHz	SSD 120GB	16GB RAM
Máquina Virtual (Master-01 - 10.50.50.51)			
Ubuntu 22.04.2 LTS	4 vCPUs	-	4GB RAM

Tabela E3. Características (sistema operativo, CPU, armazenamento e memória RAM) da máquina virtual Master-02 e do *host* M1-Edna.

***Host* (M1-Edna - 10.50.50.9)**			
Windows 10 IoT Enterprise 20H2	Intel Core i7-7700 @ 3.60GHz	SSD 250GB	16GB RAM
Máquina Virtual (Master-02 - 10.50.50.52)			
Ubuntu 22.04.2 LTS	4 vCPUs	-	4GB RAM

Tabela E4. Características (sistema operativo, CPU, armazenamento e memória RAM) da máquina virtual Master-03 e do *host* M1-Chalmers.

***Host* (M1-Chalmers - 10.50.50.11)**			
Windows 10 IoT Enterprise 20H2	Intel Core i7-4790S @ 3.20GHz	SSD 120GB	8GB RAM
Máquina Virtual (Master-03 - 10.50.50.53)			
Ubuntu 22.04.2 LTS	4 vCPUs	-	4GB RAM

E.3. Características dos *Nodes* do *Cluster*

Tabela E5. Características (sistema operativo, CPU, armazenamento e memória RAM) dos nós do *cluster*.

Nome	Sistema Operativo	CPU	Armazenamento	RAM
M1-Homer (10.50.50.2)	Ubuntu 20.04.3 LTS	Intel Xeon E3-1270 V2 @ 3.50 GHz x 8	HDD 500GB	8GB
M1-Marge (10.50.50.3)	Windows 10 Enterprise 20H2	Intel Core i7-3770T @ 2.50GHz	SSD 120GB	8GB
M1-Maggie (10.50.50.4)	Ubuntu 20.04.4 LTS	Intel Core i7-3770T @ 2.50GHz	SSD 120GB	8GB
M1-Herman (10.50.50.6)	Windows 10 IoT Enterprise 20H2	Intel Xeon E3-1270 V2 @ 3.50GHz	HDD 466GB	8GB
M1-Lewis (10.50.50.7)	Ubuntu 20.04.04 LTS	Intel Xeon E3-1270 V2 @ 3.50GHz	HDD 466GB	8GB
M1-Charlie (10.50.50.8)	Windows 10 IoT Enterprise	Intel Xeon E3-1270 V3 @ 3.50GHz	HDD 450GB	8GB
M1-Selma (10.50.50.12)	Windows 10 Enterprise 20H2	Intel Core i7-4790S @ 3.20GHz	SSD 120GB	8GB
M1-Patty (10.50.50.13)	Ubuntu 20.04.4 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Lisa (10.50.50.14)	Ubuntu 20.04.3 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Sarah (10.50.50.15)	Ubuntu 20.04.3 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Dave (10.50.50.16)	Windows 10 IoT Enterprise 20H2	Intel Xeon E3-1270 V2 @ 3.50GHz	HDD 466GB	8GB
M1-Roger (10.50.50.17)	Ubuntu 20.04.3 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Jimbo (10.50.50.18)	Ubuntu 20.04.3 LTS	Intel Xeon X3470 @ 2.93GHz	HDD 250GB	12GB
M1-Gloria (10.50.50.19)	Ubuntu 20.04.3 LTS	Intel Xeon X3470 @ 2.93GHz	HDD 466GB	12GB
M1-Luigi (10.50.50.20)	Ubuntu 20.04.3 LTS	Intel Xeon X3470 @ 2.93GHz	HDD 466GB	12GB
M1-Carl (10.50.50.21)	Windows 10 IoT Enterprise 20H2	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Martin (10.50.50.22)	Windows 10 IoT Enterprise 20H2	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Bart (10.50.50.23)	Ubuntu 20.04.3 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB
M1-Arnie (10.50.50.24)	Ubuntu 20.04.3 LTS	Intel Core i5-9600 @ 3.10GHz	SSD 256GB	16GB

E.4. Características dos Servidores de Teste

Tabela E6. Características (sistema operativo, CPU, armazenamento e memória RAM) dos três servidores de testes utilizados.

Nome	Sistema Operativo	CPU	Armazenamento	RAM
Servidor 1 (10.20.40.82)	Windows 10 IoT Enterprise 20H2	Intel Xeon X3470 @ 2.93GHz	HDD 250GB	8GB
Servidor 2 (10.20.45.116)	Windows 10 IoT Enterprise 20H2	Intel Xeon X31270 @ 3.40GHz	HDD 250GB	8GB
Servidor 3 (10.20.45.115)	Windows 10 IoT Enterprise 20H2	Intel Xeon E3-1270 V2 @ 3.50GHz	HDD 150GB	8GB

Apêndice F

Detalhes de Implementação

F.1. Helm *Charts*

F.1.1. Estrutura de um Helm *Chart*

Um Helm *Chart* é um conjunto de ficheiros organizados dentro de um diretório. O nome do diretório corresponde ao nome do *chart* (sem indicação da versão). Assim um *chart*, chamado `ChartTest` organizar-se-ia da seguinte forma:

```
|__ Chart.yaml
|__ LICENSE
|__ README.md
|__ values.yaml
|__ values.schema.json
|__ charts/
|__ crds/
|__ templates/
   |__ NOTES.txt
```

Figura F1. Estrutura de pastas de um Helm *chart*.

F.1.2. MAM4PRO

O Helm *chart* do MAM4PRO é composto pelos seguintes ficheiros:

```yaml
apiVersion: v2
appVersion: placeholder
description: A Helm chart for Kubernetes
name: mam4pro
type: application
version: 1.1.1
```

Código F1. Ficheiro `Chart.yaml`.

```yaml
global:
  licensingIP: ""
  image:
    registry: docker.gitlab.mog-technologies.com:446
    repositoryPrefix: common/mxfspeedrail/
ingress:
  enabled: true
  hostname: ""
  tls: []
  clusterIssuer: "letsencrypt-prod"
mcore:
  replicas: 1
  image:
    repository: mog.mcore
  persistence:
    enabled: true
    storageClass: ""
messagequeuing:
  host: ""
  replicas: 1
  image:
    repository: mog.messagequeuing
rplayerweb:
  replicas: 0
  image:
```

```
    repository: mog.rplayerweb
hoperation:
  replicas: 0
  image:
    repository: mog.hoperation
scapture:
  replicas: 0
  image:
    repository: mog.scapture
```

Código F2. Ficheiro `values.yaml`.

```
# mCore StatefulSet
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mcore
  labels:
    app: mcore
spec:
  serviceName: "mam4pro"
  podManagementPolicy: "OrderedReady"
  replicas: {{ .Values.mcore.replicas }}
  selector:
    matchLabels:
      app: mcore
  template:
    metadata:
      labels:
        app: mcore
      name: mcore
    spec:
      containers:
        - name: mcore
          image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.mcore.image
    .repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
          imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
          env:
            - name: NAMESPACE
              value: {{ .Release.Namespace }}
            - name: MOG_LOGGING_CONSOLE
              value: "true"
            - name: MOG_MCORE_MIGRATEDATABASE
              value: "true"
            - name: LICENSE_SERVER
              value: {{ .Values.global.licensingIP }}
            - name: RABBITMQ_SERVER
              {{- include "messagequeuing.hostname" . | indent 14 }}
          resources:
            limits:
              memory: "2Gi"
            requests:
              memory: "1Gi"
          volumeMounts:
            - mountPath: "C:/ProgramData/MOG/MAM4PRO/DB"
              name: mcoredb
{{- include "global.imagePullSecrets" . | indent 6 }}
      nodeSelector:
        kubernetes.io/os: windows
{{- if not .Values.mcore.persistence.enabled }}
      volumes:
      - name: mcoredb
        emptyDir: {}
{{- else }}
  volumeClaimTemplates:
    - metadata:
        name: mcoredb
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: {{ .Values.mcore.persistence.storageClass }}
        resources:
          requests:
            storage: 2Gi
{{- end }}
---
# mWeb Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mweb
  labels:
    app: mweb
spec:
  replicas: {{ .Values.mweb.replicas }}
  selector:
    matchLabels:
      app: mweb
  template:
    metadata:
      labels:
        app: mweb
      name: mweb
    spec:
      containers:
```

```
        - name: mweb
          image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.mweb.image.repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
          imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
          env:
            - name: NAMESPACE
              value: {{ .Release.Namespace }}
            - name: MOG_LOGGING_CONSOLE
              value: "true"
            - name: INGRESS
              value: "true"
            - name: JANUS_SERVER
              value: {{ .Values.mweb.mlive.uri }}/janus
          resources:
            limits:
              memory: 256Mi
            requests:
              memory: 256Mi
{{- include "global.imagePullSecrets" . | indent 6 }}
      nodeSelector:
        kubernetes.io/os: windows
---
{{- if not .Values.messagequeuing.host }}
# MessageQueuing Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: messagequeuing
  labels:
    app: messagequeuing
spec:
  replicas: {{ .Values.messagequeuing.replicas }}
  selector:
    matchLabels:
      app: messagequeuing
  template:
    metadata:
      labels:
        app: messagequeuing
      name: messagequeuing
    spec:
      containers:
        - name: messagequeuing
          image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.messagequeuing.image.repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
          imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
          resources:
            limits:
              memory: "1Gi"
            requests:
              memory: "1Gi"
{{- include "global.imagePullSecrets" . | indent 6 }}
      nodeSelector:
        kubernetes.io/os: windows
{{- end }}
---
# rPlayerWeb Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rplayerweb
  labels:
    app: rplayerweb
spec:
  replicas: {{ .Values.rplayerweb.replicas }}
  selector:
    matchLabels:
      app: rplayerweb
  template:
    metadata:
      labels:
        app: rplayerweb
      name: rplayerweb
    spec:
      containers:
        - name: rplayerweb
          image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.rplayerweb.image.repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
          imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
          ports:
          - containerPort: 8731
            name: http
          env:
            - name: NAMESPACE
              value: {{ .Release.Namespace }}
            - name: MOG_LOGGING_CONSOLE
              value: "true"
            - name: LICENSE_SERVER
              value: {{ .Values.global.licensingIP }}
            - name: RABBITMQ_SERVER
              {{- include "messagequeuing.hostname" . | indent 14 }}
            - name: ZEROCONF_DATAGRAM_UNICAST_HOSTS
              value: "mam4pro-discovery.{{ .Release.Namespace }}.svc.cluster.local"
          resources:
            limits:
```

```
174              memory: "1Gi"
175            requests:
176              memory: "256Mi"
177 {{- include "global.imagePullSecrets" . | indent 6 }}
178       nodeSelector:
179         kubernetes.io/os: windows
180 ---
181 # hOperation Deployment
182 apiVersion: apps/v1
183 kind: Deployment
184 metadata:
185   name: hoperation
186   labels:
187     app: hoperation
188 spec:
189   replicas: {{ .Values.hoperation.replicas }}
190   selector:
191     matchLabels:
192       app: hoperation
193   template:
194     metadata:
195       labels:
196         app: hoperation
197       name: hoperation
198     spec:
199       containers:
200         - name: hoperation
201           image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.hoperation.
      image.repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
202           imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
203           env:
204             - name: NAMESPACE
205               value: {{ .Release.Namespace }}
206             - name: MOG_LOGGING_CONSOLE
207               value: "true"
208             - name: LICENSE_SERVER
209               value: {{ .Values.global.licensingIP }}
210             - name: RABBITMQ_SERVER
211               {{- include "messagequeuing.hostname" . | indent 14 }}
212             - name: ZEROCONF_DATAGRAM_UNICAST_HOSTS
213               value: "mam4pro-discovery.{{ .Release.Namespace }}.svc.cluster.local"
214           resources:
215             limits:
216               memory: "1Gi"
217             requests:
218               memory: "256Mi"
219 {{- include "global.imagePullSecrets" . | indent 6 }}
220       nodeSelector:
221         kubernetes.io/os: windows
222 ---
223 # sCapture Deployment
224 apiVersion: apps/v1
225 kind: Deployment
226 metadata:
227   name: scapture
228   labels:
229     app: scapture
230 spec:
231   replicas: {{ .Values.scapture.replicas }}
232   selector:
233     matchLabels:
234       app: scapture
235   template:
236     metadata:
237       labels:
238         app: scapture
239       name: scapture
240     spec:
241       containers:
242         - name: scapture
243           image: {{ .Values.global.image.registry }}/{{ .Values.global.image.repositoryPrefix }}{{ .Values.scapture.
      image.repository }}:{{ .Values.global.image.tag | default .Chart.AppVersion }}
244           imagePullPolicy: {{ .Values.global.image.pullPolicy | quote }}
245           ports:
246           - containerPort: 8731
247             name: http
248           env:
249             - name: NAMESPACE
250               value: {{ .Release.Namespace }}
251             - name: MOG_LOGGING_CONSOLE
252               value: "true"
253             - name: LICENSE_SERVER
254               value: {{ .Values.global.licensingIP }}
255             - name: RABBITMQ_SERVER
256               {{- include "messagequeuing.hostname" . | indent 14 }}
257             - name: ZEROCONF_DATAGRAM_UNICAST_HOSTS
258               value: "mam4pro-discovery.{{ .Release.Namespace }}.svc.cluster.local"
259           resources:
260             limits:
261               memory: "1Gi"
262             requests:
263               memory: "256Mi"
264 {{- include "global.imagePullSecrets" . | indent 6 }}
265       nodeSelector:
266         kubernetes.io/os: windows
```

Código F3. Ficheiro `templates/deployment.yaml`.

```
{{/*
Return the proper Docker Image Registry Secret Names
*/}}
{{- define "global.imagePullSecrets" -}}
{{- if .Values.global.image.pullSecrets }}
imagePullSecrets:
{{- range .Values.global.image.pullSecrets }}
  - name: {{ . }}
{{- end }}
{{- end }}
{{- end -}}

{{/*
Return the templated default or defined messagequeuing host
*/}}
{{- define "messagequeuing.hostname" -}}
{{- if .Values.messagequeuing.host }}
value: {{ .Values.messagequeuing.host }}
{{- else }}
value: "mam4pro-messagequeuing.{{ .Release.Namespace }}.svc.cluster.local"
{{- end }}
{{- end -}}
```

Código F4. Ficheiro `templates/_helpers.tpl`.

```
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mam4pro-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: {{ .Values.ingress.clusterIssuer }}
spec:
{{- if .Values.ingress.tls }}
  tls:
{{ tpl (toYaml .Values.ingress.tls) $ | indent 4 }}
{{- end }}
  rules:
    - host: {{ .Values.ingress.hostname }}
      http:
        paths:
          - pathType: Prefix
            path: "/sr/"
            backend:
              service:
                name: mam4pro
                port:
                  number: 8731
          - pathType: Prefix
            path: "/ws"
            backend:
              service:
                name: mam4pro
                port:
                  number: 8741
          - pathType: Prefix
            path: "/proxy/"
            backend:
              service:
                name: mam4pro
                port:
                  number: 8731
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: mam4pro-gui
                port:
                  number: 80
{{- end }}
```

Código F5. Ficheiro `templates/ingress.yaml`

```
# MAM4PRO Service
apiVersion: v1
kind: Service
metadata:
  name: mam4pro
  labels:
    app: mam4pro
spec:
  ports:
    - name: mogservices
      port: 8731
      targetPort: 8731
```

```
    - name: mogproxy0
      port: 8734
      targetPort: 8734
    - name: mogproxy1
      port: 8735
      targetPort: 8735
    - name: websocket
      port: 8741
      targetPort: 8741
    - name: db
      port: 27017
      targetPort: 27017
  selector:
    app: mcore
  type: ClusterIP
---
# MAM4PRO Service
apiVersion: v1
kind: Service
metadata:
  name: mam4pro-gui
  labels:
    app: mam4pro
spec:
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    app: mweb
  type: ClusterIP
---
# MAM4PRO Discovery Service
apiVersion: v1
kind: Service
metadata:
  name: mam4pro-discovery
  labels:
    app: mam4pro
spec:
  ports:
    - name: zeroconfunicast
      port: 8782
      targetPort: 8782
      protocol: UDP
  selector:
    app: mcore
  type: ClusterIP
---
{{- if not .Values.messagequeuing.host }}
# MAM4PRO Messagequeuing Service
apiVersion: v1
kind: Service
metadata:
  name: mam4pro-messagequeuing
  labels:
    app: mam4pro
spec:
  ports:
    - name: rabbitmqamqp1
      port: 5671
    - name: rabbitmqamqp2
      port: 5672
    - name: rabbitmqhttp
      port: 15672
      targetPort: 15672
  selector:
    app: messagequeuing
  type: ClusterIP
{{- end }}
```

Código F6. Ficheiro `templates/service.yaml`.

F.1.3. Uptime Kuma

O Helm *Chart* do Uptime Kuma é composto pelos seguintes ficheiros:

```
apiVersion: v2
appVersion: "1"
description: A Helm chart for Uptime Kuma
name: uptime-kuma
type: application
version: 1.0.0
```

Código F7. Ficheiro `Chart.yaml`

```yaml
replicas: 1
image:
  repository: louislam/uptime-kuma
  pullPolicy: IfNotPresent
ingress:
  enabled: false
  hostname: ""
persistence:
  enabled: true
  storageClass: ""
```

Código F8. Ficheiro `values.yaml`.

```yaml
{{- if not .Values.persistence.enabled }}
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: uptime-kuma
  name: uptime-kuma
spec:
  replicas: 1
  selector:
    matchLabels:
      app: uptime-kuma
  template:
    metadata:
      labels:
        app: uptime-kuma
    spec:
      containers:
      - image: {{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        name: uptime-kuma
{{- end }}
```

Código F9. Ficheiro `templates/deployment.yaml`.

```yaml
{{- if not .Values.persistence.enabled }}
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: uptime-kuma-ingress
spec:
  ingressClassName: nginx
  rules:
    - host: {{ .Values.ingress.hostname }}
      http:
        paths:
        - pathType: Prefix
          path: "/"
          backend:
            service:
              name: uptime-kuma-svc
              port:
                number: 3001
{{- end }}
```

Código F10. Ficheiro `templates/ingress.yaml`.

```yaml
{{- if .Values.serviceMonitor.enabled }}
apiVersion: v1
kind: Service
metadata:
  labels:
    app: uptime-kuma
  name: uptime-kuma-svc
spec:
  ports:
  - name: http
    port: 3001
    protocol: TCP
    targetPort: 3001
  selector:
    app: uptime-kuma
  type: ClusterIP
```

Código F11. Ficheiro `templates/service.yaml`

```yaml
{{- if .Values.persistence.enabled }}
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app: uptime-kuma
  name: uptime-kuma
```

```
8  spec:
9    serviceName: "uptime-kuma-svc"
10   podManagementPolicy: "OrderedReady"
11   replicas: {{ .Values.replicas }}
12   selector:
13     matchLabels:
14       app: uptime-kuma
15   template:
16     metadata:
17       labels:
18         app: uptime-kuma
19     spec:
20       containers:
21       - image: {{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}
22         imagePullPolicy: {{ .Values.image.pullPolicy }}
23         name: uptime-kuma
24         volumeMounts:
25         - name: data
26           mountPath: /app/data
27   volumeClaimTemplates:
28     - metadata:
29         name: data
30       spec:
31         accessModes: [ "ReadWriteOnce" ]
32         storageClassName: {{ .Values.persistence.storageClass }}
33         resources:
34           requests:
35             storage: 2Gi
36 {{- end }}
```

Código F12. Ficheiro `templates/statefulset.yaml`

Apêndice G

Contribuições

G.1. Documentação Desenvolvida

Kubernetes Cluster

1. Requirements

- Some knowledge on:
  - Hyper-V;
  - PowerShell;
  - Linux Commands;
  - Vagrant;
  - Terraform;
  - Docker;
  - Kubernetes;
- Patience and effort! 💪

Note: The repository referred to in this documentation is available at: https://gitlab.mog-technologies.com/one/internal/duster

2. Cluster Architecture

According to the architecture of a Kubernetes Cluster and in order to satisfy the necessary requirements for the MOG use case, there are 3 main components:

- **Terraform Master:**
  - Machine responsible for executing the terraform commands that will create the duster. The goal is that this machine is never a local machine but a GitLab Runner (`CICD-RUNNER`) responsible for executing pipelines.
- **Master Nodes + Load Balancers:**
  - By default, 3 masters (`Master-1`, `Master-2`, `Master-3`) are created that correspond to a VM (`Linux 20.04.3`) hosted on each Master Host. Each one of these VMs, in addition to the dependencies necessary to play the role of Control Plane of the duster, also have a load balancer mechanism to guarantee redundancy and high availability.
- **Worker Nodes:**
  - The duster nodes that will run the containers of the various applications are called worker nodes and can be Linux (`lin-nodes`) or Windows (`win-nodes`).

Each of the machines, depending on the role it will play, must have some pre-installed dependencies:

Terraform Master	Terraform installed and command line access.
Master Nodes + Load Balancers	Each host that will host the Master VM must have Hyper-V active and Vagrant installed. The VM will be created automatically through Terraform and Vagrant.
Worker Nodes	They must have Hyper-V disabled to avoid network problems.

Notes:

- All machines involved, with the exception of Terraform Master, are on vLAN 50 (`10.20.50.XX`);
- Machine names have the prefix `M1-`, which stands for MOG One, followed by a Simpsons character name
- These developments were tested on Linux machines with `Ubuntu Focal 20.04.4` LTS and `Windows 10 20H2` machines.

Figura G1. Excerto da documentação interna desenvolvida acessível na Wiki do GitLab.

G.2. Disseminação de Conhecimento

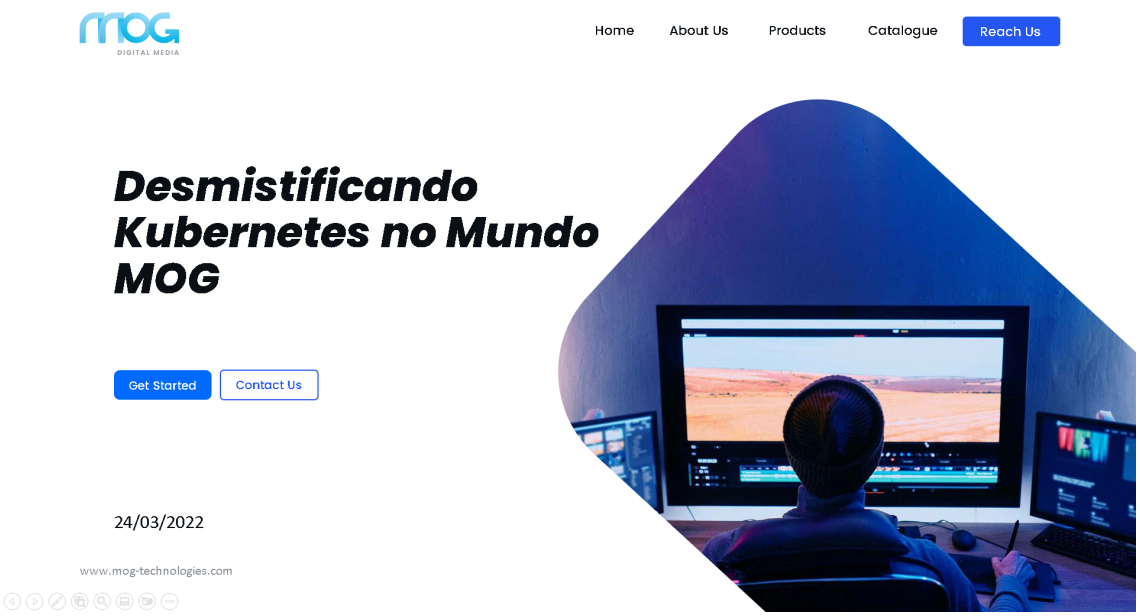


Figura G2. Primeira apresentação de disseminação de conhecimento realizada na MOG Technologies.

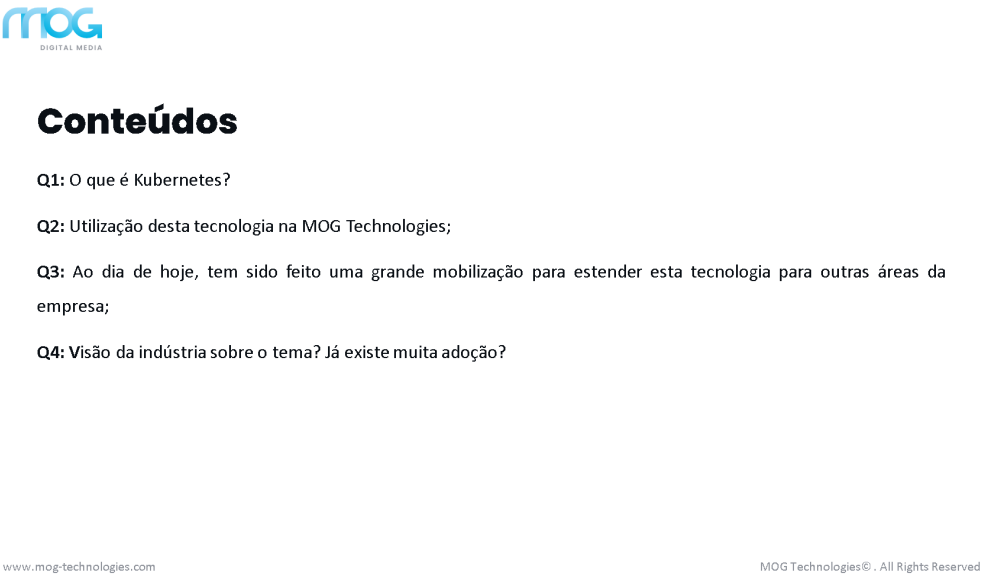


Figura G3. Conteúdos abordados na primeira apresentação de disseminação de conhecimento realizada na MOG Technologies.

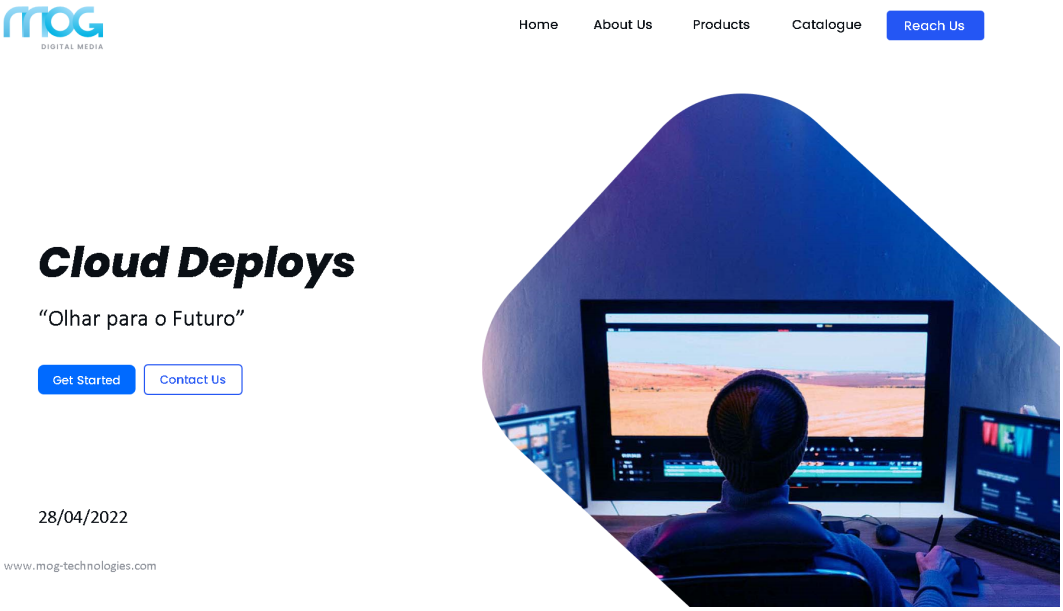


Figura G4. Segunda apresentação de disseminação de conhecimento realizada na MOG Technologies.

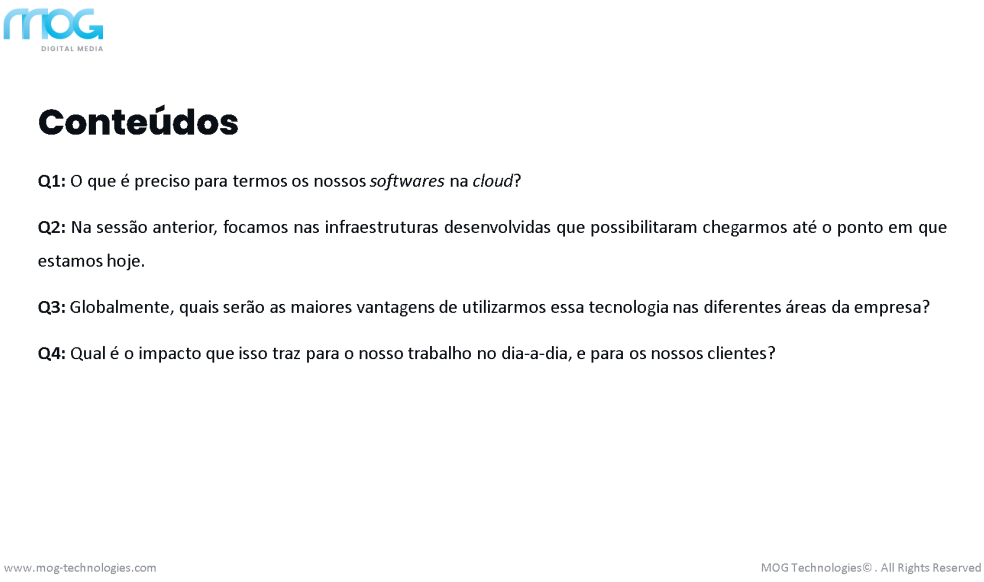


Figura G5. Conteúdos abordados na segunda apresentação de disseminação de conhecimento realizada na MOG Technologies.