INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Automatização da implantação de ferramenta de orquestração de contentores

DIOGO FILIPE SOUSA NOGUEIRA
Outubro de 2021

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Automatização da implantação de ferramenta de orquestração de contentores

DIOGO FILIPE SOUSA NOGUEIRA
Outubro de 2021

POLITÉCNICO
DO PORTO

Automatização da implantação de ferramenta de orquestração de contentores

Diogo Nogueira

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

Orientador: Alexandre Bragança
Supervisor: Jorge Oliveira

Porto, 15 de outubro de 2021

Dedicatória

«A todos aqueles que nunca duvidaram de mim e me apoiaram incondicionalmente.»

Resumo

Ao longo do tempo tem-se assistido a uma evolução da computação na *cloud*, a par com o progresso da criação, manutenção e gestão de infraestrutura. Atualmente, são cada vez mais os processos que automatizam e agilizam a criação de infraestrutura de modo consistente.

Apesar de toda a automatização já existente tanto na implantação de projetos e das suas dependências como na instalação de ferramentas, continuam ainda a existir processos realizados manualmente, como a criação de *clusters* para implantar estas soluções.

O objetivo desta dissertação passa por analisar e construir uma solução que colmate o problema descrito, com a automatização da implantação de *clusters* com as suas propriedades e integração dos mesmos com as ferramentas inseridas no meio empresarial.

Para isso, são estudadas e comparadas várias ferramentas relativas ao processo referido, de modo a selecionar a melhor abordagem a utilizar para resolver o problema. Nesta dissertação são discutidas ferramentas de aprovisionamento de infraestrutura, ferramentas de gestão de configurações, ferramentas de orquestração de contentores e ferramentas de CI/CD.

Finalmente, é avaliada a solução construída, percebendo assim o grau de cumprimento dos requisitos e da qualidade de solução desenvolvida. A classificação final revela uma qualidade de grande nível por parte da solução, tendo sido completados, na generalidade, maior parte dos requisitos propostos para a mesma.

Palavras-chave: Automatização, CI/CD, *Cluster*, Implantação, Infraestrutura, Aprovisionamento

Abstract

With the evolution of cloud computing, there has also been a development in the way infrastructure is created and maintained. Today, more and more processes can be automated to create reliable infrastructure more quickly and consistently.

Despite all the automation that already exists, both in the deployment of projects and their dependencies and in the installation of multiple tools, the creation of the clusters necessary to deploy these projects is still performed through manual steps.

The main objective of this dissertation is to analyse and build a solution that solves the described problem, in order to automate the deployment of clusters with their dependencies through software tools and to integrate it with the operation tools.

To accomplish this, several tools regarding the referred process will be explored and compared in order to select the best approach to be used to solve the problem. This dissertation will contemplate infrastructure provisioning tools, configuration management tools, infrastructure testing tools, container orchestration tools and CI/CD tools.

Finally, the built solution is evaluated, thus perceiving the degree of compliance with the requirements and the quality of the developed solution. The final classification reveals a high quality level of the solution, having completed, in general, most of the requirements proposed for it.

Agradecimentos

A realização desta dissertação teve o contributo de um agregado de pessoas sem as quais não teria conseguido concluir este projeto e às quais passo a agradecer.

A todos os docentes do ISEP, especialmente do Departamento de Engenharia Informática por todo o conhecimento que aqui me foi transmitido durante os dois anos de mestrado e os três anos de licenciatura.

Ao meu orientador do ISEP, engenheiro Alexandre Bragança, pela disponibilidade, simpatia, ajuda e conselhos ao longo deste projeto.

Ao meu supervisor da Critical TechWorks, engenheiro Jorge Oliveira, pelo desafio proposto, pela disponibilidade para esclarecimento de dúvidas, pela sabedoria partilhada e apoio desde o dia 1.

À minha equipa da Critical TechWorks por todo o suporte e colaboração durante este período.

A todos os meus amigos e colegas que me acompanharam durante esta jornada e que de alguma forma fizeram com que este momento fosse possível.

À minha namorada pelos conselhos, motivação e por rever este documento e corrigir o português que por vezes mais parecia chinês.

Aos meus pais por todo o incentivo e paciência, pois sem eles não seria possível chegar a esta fase.

A todos eles o meu sincero obrigado.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AHP	*Analytic Hierarchy Process*.
ASG	*Auto Scaling Group*.
AWS	*Amazon Web Services*.
AZ	*Availability Zone*.
BH	*Bastion Host*.
CD	*Continuous Delivery*.
CI	*Continuous Integration*.
CI/CD	*Continuous Integration/Continuous Delivery*.
CN	China.
CTW	Critical TechWorks.
DNS	*Domain Name System*.
EMEA	*Europe, the Middle East and Africa*.
FFE	*Fuzzy Front End*.
HCL	*HashiCorp Configuration Language*.
HPA	*Horizontal Pod Autoscaler*.
IaaS	*Infrastructure as a Service*.
IaC	*Infrastructure as Code*.
IP	*Internet Protocol*.
IT	*Information Technology*.
NCD	*New Concept Development*.
NLB	*Network Load Balancer*.
NPD	*New Product Development*.
PaaS	*Platform as a Service*.
QEF	*Quantitative Evaluation Framework*.
QFD	*Quality Function Deployment*.
RC	Razão de Consistência.
SaaS	*Service as a Service*.
SG	*Security Group*.
SLA	*Service Level Agreement*.

USA *United States of America*.

VC *Value for the customer*.
VPC *Virtual Private Cloud*.

Lista de Blocos de código

Capítulo 1

Introdução

O presente documento visa apresentar toda a pesquisa e trabalho elaborado no âmbito da unidade curricular Tese / Dissertação / Estágio (TMDEI) do mestrado em Engenharia Informática, ramo de Engenharia de Software do Instituto Superior de Engenharia do Porto (ISEP) em conjunto com a Critical TechWorks (CTW).

Neste primeiro capítulo, é realizada a contextualização e descrição do problema, indicados os objetivos a cumprir com a realização desta dissertação, seguido de uma breve abordagem para realizar a solução. Por fim, é apresentada a estrutura e organização do documento.

1.1 Enquadramento

A CTW é uma empresa que surge da parceria entre o BMW Group e a Critical Software. Esta junção, realizada em Setembro de 2018, advém da necessidade do BMW Group aperfeiçoar o processo de desenvolvimento de aplicações para os seus veículos, bem como para uma variedade de áreas chave, de modo a toda a estrutura funcionar da forma mais eficaz possível (Critical Techworks 2020).

Atualmente, a empresa desenvolve soluções tanto para *onboard* como para *offboard*, o que corresponde, respetivamente, a soluções desenvolvidas para o carro (aplicações de entretenimento e navegação) e soluções mais direcionadas para a produção de veículos, área de logística e outros setores (BMW 2020).

Para um variado número de projetos existentes nas equipas da CTW, as aplicações encontram-se hospedadas em *clusters*. Neste projeto em particular, as aplicações encontram-se implantadas em *clusters* geridos por uma ferramenta, nomeadamente Kubernetes. Estas aplicações estão diretamente ligadas à vertente *onboard*, ou seja, são soluções que são utilizadas no carro e que, até à data, estão escritas em Java (*backend*) e React (*frontend*). Para além destes, são também possuídos *clusters* com o objetivo de orquestrar ferramentas de operações, como Kibana e Grafana. Existe, no entanto, a possibilidade no futuro de serem adicionadas soluções que utilizem outras linguagens/tecnologias.

Com o crescendo número de microsserviços derivado da evolução da empresa e da migração de aplicações para equipas instaladas na CTW, existe a necessidade de automatizar a criação de *clusters*, tanto para suportar e orquestrar aplicações como ferramentas de operações.

1.2 Problema

Sendo a CTW uma empresa recente, grande parte das suas equipas ainda se encontram em processo de inicialização e criação de projetos ou em fase de testes. Aquelas que já possuem soluções em produção, contam por vezes com processos manuais no ciclo de entrega do produto, o que, segundo uma metodologia ágil, é considerado como uma má prática.

Atualmente, existe um variado número de *clusters* de Kubernetes partilhados por diversas equipas devido à semelhança dos seus projetos, tanto em contexto como em tecnologias. Estes encontram-se divididos de acordo com a fase de desenvolvimento da solução, existindo *clusters* de testes, integração, *end-to-end* e produção e são ainda distribuídos por uma variedade de regiões: *United States of America* (USA), *Europe, the Middle East and Africa* (EMEA) e China (CN). Para cada *cluster* de aplicações existe um para ferramentas de operações como Kibana e Grafana.

Apesar de já terem sido automatizados vários processos relativos à criação de novos serviços, tais como um gerador de projetos em Java ou *scripts* para criação e aprovisionamento de uma instância Jenkins, existem ainda alguns passos que podem ser melhorados e automatizados, como a implantação dos próprios *clusters*. Devido à urgência de lançar serviços em produção e ao cumprimento de *deadlines*, os *clusters* de Kubernetes existentes foram inicialmente criados através da AWS Management Console por um colaborador já experiente e familiarizado com o procedimento, de forma a agilizar e acelerar todo o processo. Atualmente, tanto a criação (ou recriação, em caso de desastre) como a eliminação dos *clusters* continua a ser efetuada recorrendo à AWS Management Console, o que torna toda esta criação vulnerável à geração de incongruências e criação errónea da infraestrutura pretendida, visto ser um processo manual.

Em situações de contratempo (casos atípicos ou de desastre), a ocorrência de falhas na recriação de infraestrutura tende a aumentar pois existe uma pressão acrescida sob o responsável de efetuar os passos necessários. No caso de um incidente, essa pressão extra pode originar um aumento no tempo de indisponibilidade nos serviços ou originar infraestrutura mal configurada. Para além destas adversidades, a reconstrução manual de um *cluster* tende a ser mais demorada, para além que existe uma maior probabilidade de perda das configurações do *cluster*.

1.3 Objetivos

Para colmatar o problema exposto na Secção 1.2, será criada uma solução cujo o principal propósito passa por automatizar o processo de implantação de *clusters* de Kubernetes. Além do referido, foram ainda delineados mais seis objetivos a alcançar com a realização deste projeto, entre eles:

- A diminuição do tempo de criação da infraestrutura;
- A utilização de ferramentas de aprovisionamento de infraestrutura e/ou ferramentas de gestão de configurações;
- A criação de ficheiros de configuração e aprovisionamento do *cluster* através de infraestrutura como código;
- A redução do custo de manutenção de infraestrutura;

- A utilização de mecanismos de *Continuous Integration/Continuous Delivery* (CI/CD) para implantação de aplicações no *cluster*;
- A integração da solução com as diferentes ferramentas no meio empresarial, incluindo Jenkins, Kubernetes, Vault, entre outras.

1.4 Resultados esperados

Espera-se que com o desenvolvimento deste projeto seja criado um ambiente para execução de aplicações similar ao já existente, mas com todo o seu processo de criação automatizado. Depois de concluída, esta solução afetará positivamente o tempo de criação de *clusters* de Kubernetes, os custos associados ao processo de manutenção e gestão da infraestrutura, a consistência da infraestrutura criada e ainda facilitará o processo de criação da mesma.

1.5 Abordagem

Para alcançar os objetivos propostos na Secção 1.3 de forma metódica, foi fragmentado o desenvolvimento deste projeto em várias fases, de modo a melhor compreender e organizar cada processo. Assim sendo, este procedimento foi divido em onze fases que se encontram a seguir enumeradas:

1. Recolha de informação sobre a arquitetura atual, de modo a conhecer todos os recursos nela existentes, as suas propriedades e dependências;
2. Pesquisa e análise de abordagens utilizadas em problemas semelhantes, entendendo assim os diferentes mecanismos e as várias alternativas que poderão ser utilizadas;
3. Análise de ferramentas com relevância para a resolução do problema;
4. Análise de valor do problema proposto;
5. Definição dos requisitos da solução;
6. Desenho das etapas do processo de implantação;
7. Definição de alternativas para resolução do problema;
8. Seleção da melhor alternativa;
9. Seleção das tecnologias a utilizar;
10. Construção da solução;
11. Avaliação da solução através das grandezas selecionadas.

Esta abordagem é aplicada sob uma *framework* ágil, mais precisamente Scrum (Scrum Portugal 2017). Esta *framework* baseia-se numa melhoria gradual da solução, tendo como propósito aumentar a rapidez de entrega de produto (Scrum Portugal 2017). Paralelamente ao Scrum, é também praticada uma cultura de DevOps, consistindo num conjunto de práticas que facilitam e automatizam o processo de criação, teste, implantação e entrega de soluções (Amazon Web Services 2020; Atlassian 2019; Microsoft Azure 2020). Por fim, e de nível mais técnico, é utilizado um fluxo denominado GitFlow. Este fluxo auxilia o desenvolvimento de *software* em paralelo, facilitando a coordenação entre o código criado pelos vários colaboradores (Atlassian 2021).

1.6 Organização do documento

O presente documento é constituído por 7 capítulos, entre eles a introdução, o estado de arte, a análise de valor, o desenho da solução, a construção da solução, a avaliação da solução e as conclusões.

No Capítulo 1 é referido a contextualização do problema encontrado, seguindo-se os objetivos delineados para o projeto. Consta também uma breve análise da abordagem a seguir durante desenvolvimento da solução.

No Capítulo 2 é feita a contextualização de várias terminologias para uma boa compreensão do contexto do projeto, seguida da representação da atual arquitetura. São também enunciados vários estudos e projetos com relevância para o desenho da solução, bem como a análise de várias ferramentas que poderão integrar a solução final.

No Capítulo 3 é efetuada a análise de valor do projeto, onde é descrita e realizada parte do processo de inovação com a utilização da *framework New Concept Development*. É feita a definição do valor da solução, seguida da proposta de valor. Finalmente, é feita a análise funcional recorrendo ao método *Quality Functional Deployment*.

No Capítulo 4 são enunciados os requisitos propostos para o projeto, descrito o processo de implantação do *cluster* e feita a apresentação e seleção de alternativas para a resolução do problema. Escolhida a alternativa, são definidas as ferramentas a utilizar para criar a solução e, por fim, são apresentadas as várias vistas da mesma.

No Capítulo 5 é descrito o modo como é construída a solução. Inicialmente são pormenorizadas as metodologias utilizadas, seguido da infraestrutura base criada. Nas duas secções seguintes são detalhados os passos necessários para efetuar a criação e gestão de configurações do *Bastion Host* (BH) e o processo de elaboração do *cluster* de Kubernetes. Por fim é efetuada a instalação de uma ferramenta de CI/CD.

No Capítulo 6 são apresentadas as grandezas utilizadas no processo de avaliação da solução, assim como as várias hipóteses a testar. É definida a metodologia de avaliação da solução, sendo de seguida executada.

No Capítulo 7 são enunciadas as conclusões retiradas com a elaboração do projeto e discutido o trabalho a realizar no futuro.

Capítulo 2

Contextualização e estado da arte

Este capítulo visa aumentar o conhecimento sobre ferramentas e projetos que podem ser relevantes durante a fase de desenho e construção da solução.

Primeiramente é realizada a contextualização de diversas terminologias com elevada pertinência para o desenvolvimento e compreensão do projeto. De seguida é feita uma breve descrição do estado atual da infraestrutura e o levantamento de projetos que propõem resolver um problema semelhante ao problema em causa, assim como das abordagens por eles seguidas. Por último, são mencionados os vários tipos de ferramentas que poderão ser utilizadas, bem como as caraterísticas das mesmas.

2.1 Contextualização

Nesta secção são desenvolvidos alguns temas relevantes para a elaboração e compreensão deste projeto, sendo que no final de cada tópico, é feita a sua contextualização de cada um tendo em conta a situação atual.

2.1.1 Cluster

Cluster é a designação dada a um sistema que usufrui da cooperação de dois ou mais computadores (nós) para a execução de tarefas, resultando numa máquina com maior capacidade de processamento (Baker e Buyya 1999; Sloan 2004). Não é exigido que estes nós possuam *hardware* semelhante, nem que estejam fisicamente próximos, podendo ser conectados através de uma rede (Baker e Buyya 1999). No entanto, deve ser garantida a possibilidade de remoção/substituição de nós sem *downtime*, para que as tarefas e aplicações sejam executadas sem interrupção (Baker e Buyya 1999). Na atualidade, existem três categorias de *clusters* (Sloan 2004):

- **Cluster de alta disponibilidade** - Foco em manter as tarefas em execução sem *downtime*;
- **Cluster de alto desempenho** - Foco na performance das tarefas a serem executadas;
- **Cluster de balanceamento de carga** - Foco em distribuir equitativamente a carga das tarefas pelos nós existentes.

A utilização deste tipo de sistemas permite reduzir os custos operacionais, facilita a personalização de configurações através da adição ou remoção de nós, e ainda garante alta disponibilidade para as aplicações nele implantadas, pois caso ocorra algum tipo de falha num nó, os restantes compensam na execução das tarefas (HostGator 2020).

A solução corrente é composta por uma variedade *clusters* distribuídos por várias regiões, entre elas: USA, EMEA e CN.

2.1.2 Aplicações em contentores

O método de implantação de soluções em contentores permite encapsulá-las no seu ambiente de execução, surgindo como uma alternativa ao uso de máquinas virtuais (Luksa 2018; Oliveira 2020). Esta abordagem possibilita a execução de várias aplicações na mesma máquina, cada uma com os seus processos isolados, beneficiando também de um menor consumo de recursos comparativamente com o uso de máquinas virtuais (Oliveira 2020). Isto resulta da necessidade de cada máquina virtual possuir um sistema operativo próprio, o que leva a que, com *hardware* idêntico, seja possível executar um maior número de aplicações em contentores (Luksa 2018). Por limitação de recursos, a utilização de máquinas virtuais resulta, ocasionalmente, na implantação de várias aplicações dentro da mesma máquina virtual, como é exemplificado no diagrama da Figura 2.1 (Luksa 2018):

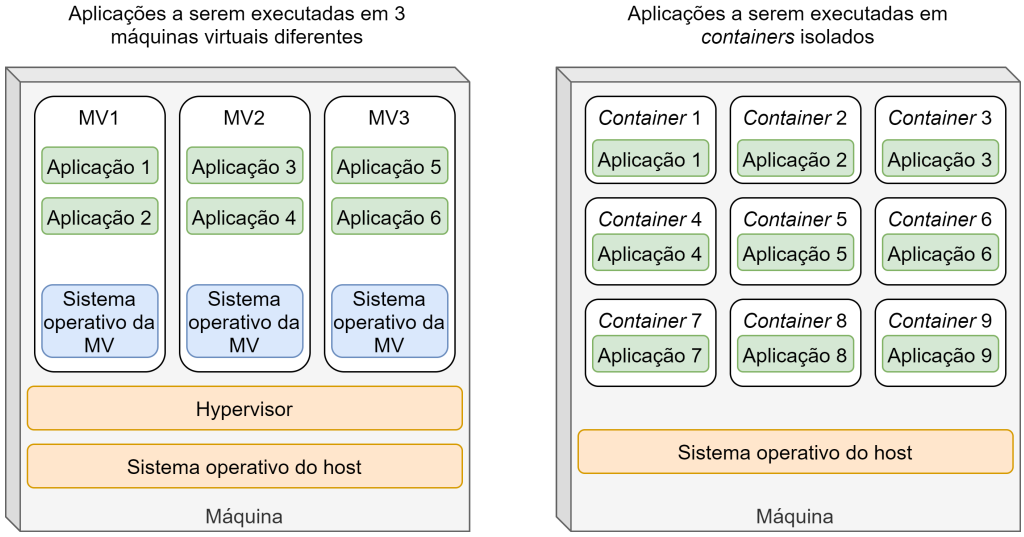


Figura 2.1: Diferença entre o uso de máquinas virtuais e contentores. Fonte: (Luksa 2018)

A implantação de aplicações em contentores permite a sua execução de um modo mais consistente, independente do sistema onde estejam a operar e possibilita a integração da fase de implantação da solução em métodos de desenvolvimento ágil, de modo a acelerar o ciclo de entrega de produto (NetApp 2021). Esta abordagem reduz também a necessidade de manutenção da infraestrutura, devido à diminuição do número de camadas no sistema e do tempo de inicialização da aplicação, resultando num ambiente com maior portabilidade e de fácil escalabilidade (Chamberlain 2018; Luksa 2018). Para além do decréscimo do tempo de inicialização, a existência de ferramentas como Docker fornecem a este mecanismo um acréscimo de portabilidade, pois torna possível executar a mesma solução em máquinas diferentes recorrendo somente a um ficheiro com as configurações e dependências necessárias (Luksa 2018). Estes aspetos melhoram o processo de desenvolvimento, teste e implantação da solução, favorecendo a sua utilização. Todos estes aspetos levam ao crescimento do interesse neste tipo de abordagem, como se pode observar através do gráfico presente na Figura 2.2.

Figura 2.2: Interesse pelo termo *App container* entre 2008 e 2021. Fonte: (Google Trends 2021a)

As soluções existentes na atualidade nos *clusters* estão implantadas em contentores, sendo todos eles geridos através da ferramenta de orquestração Kubernetes.

2.1.3 Arquitetura de microsserviços

A arquitetura de microsserviços consiste num aglomerado de pequenos componentes autónomos com funcionalidades independentes e de responsabilidade única mas com possibilidade de interligação entre si (Luksa 2018; Martin 2019; Thones 2015). Esta arquitetura possibilita a cada microsserviço utilizar a *stack* tecnológica mais adequada para a sua finalidade, sendo, no entanto, permitida a comunicação entre os mesmos através de APIs bem definidas (Martin 2019). Cada serviço é implementado de forma isolada e é responsável pela persistência dos seus dados, possuindo, se necessário, uma base de dados para o efeito (Martin 2019; Thones 2015). Recorrendo a esta separação, na eventualidade de ocorrência de falhas em algum microsserviço, os restantes mantêm a sua execução normal, possibilitando assim a criação de uma solução mais fracionada e distribuída, comparativamente com uma arquitetura monolítica, como pode ser visualizado através da Figura 2.3 (Luksa 2018; Martin 2019; RedHat 2021b):

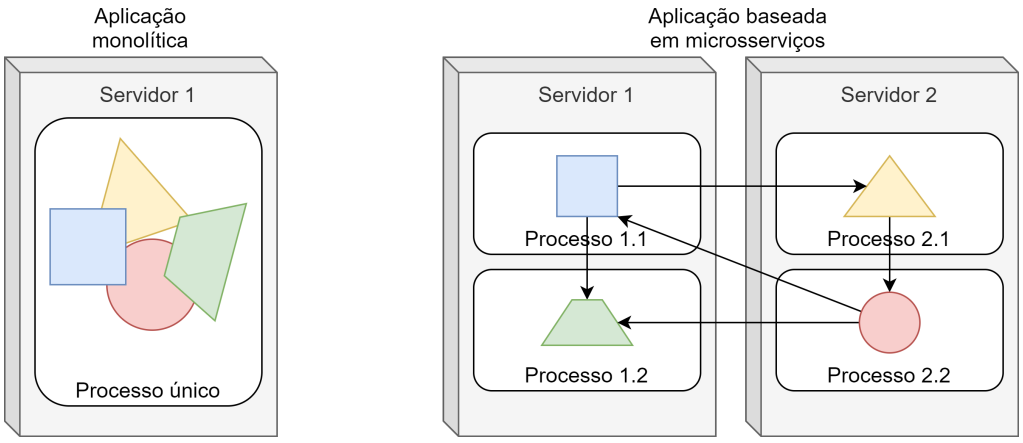


Figura 2.3: Diferenças entre uma arquitetura monolítica e uma arquitetura de microsserviços. Fonte: (Luksa 2018)

Numa arquitetura monolítica, o sistema é escalado como um todo sempre que é necessário incrementar os recursos de uma funcionalidade da aplicação. Por outro lado, com a utilização de microsserviços torna-se possível escalar horizontalmente uma funcionalidade da

aplicação em específico (Luksa 2018). Quando uma aplicação monolítica possuí componentes impossíveis de escalar, a adoção de microsserviços permite escalar as restantes sem que esta seja afetada, como exemplificado na Figura 2.4.

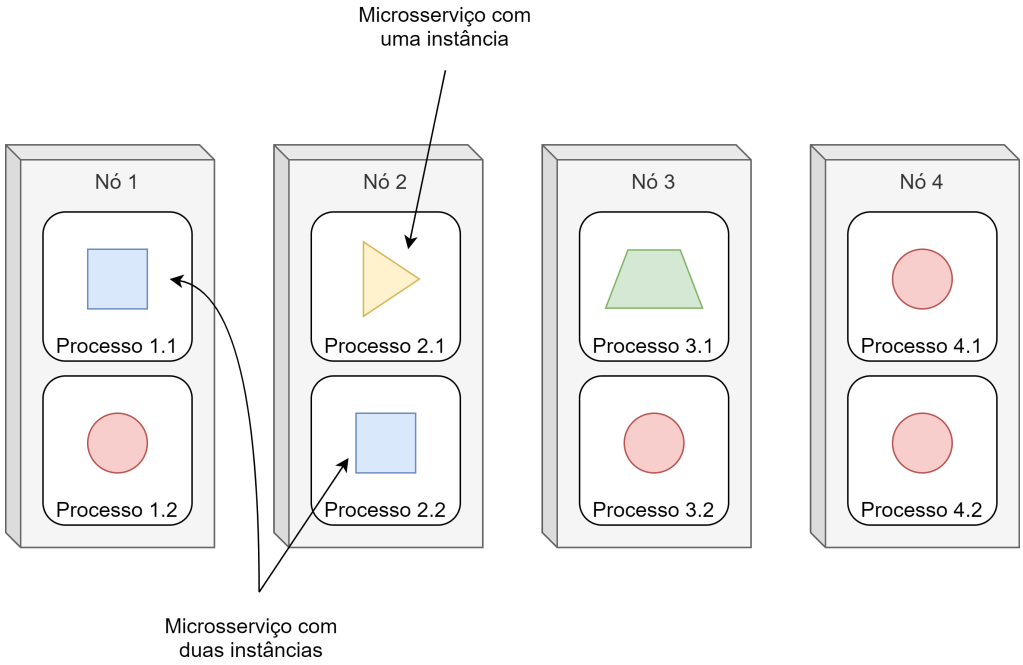


Figura 2.4: Vários níveis de escalabilidade para os diferentes microsserviços.
Fonte: (Luksa 2018)

Esta arquitetura apresenta, portanto, um conjunto de aspetos positivos, entre eles (Luksa 2018; Martin 2019; RedHat 2021b; Thones 2015):

- Atualizações e modificações de cada microsserviço isoladas;
- Redução da quantidade de código em cada projeto;
- Aumento da agilidade de desenvolvimento de novas funcionalidades;
- Facilidade de compreensão de cada microsserviço;
- Isolamento de falhas.

Em contrapartida, esta arquitetura possui também múltiplos obstáculos, tais como o aumento da complexidade do sistema no geral, a necessidade de interligação entre os serviços, o aumento da dificuldade de gestão dos serviços, derivado do acréscimo do número de infraestrutura para gerir, entre outros (Martin 2019; RedHat 2021b).

Não obstante os pontos negativos mencionados, a arquitetura descrita está presente em todas as soluções existentes nos *clusters*.

2.1.4 Continuous Integration/Continuous Delivery

CI/CD é um conjunto de princípios e práticas cujo objetivo é a entrega de produto de um modo consistente e com maior frequência (Purohit 2020; Sacolick 2020). Este é considerado um elemento chave para equipas que pratiquem a cultura DevOps, devido à automatização providencia ao processo de desenvolvimento e à implantação de soluções, resultando no

aumento do foco das equipas na criação do produto (Purohit 2020; Sacolick 2020). Esta prática é constituída por dois processos: *Continuous Integration* (CI) e *Continuous Delivery* (CD).

CI é o processo de armazenamento e junção (num repositório) de todas as alterações efetuadas no código, possibilitando, posteriormente, a automatização da compilação e teste da solução (AWS 2021k; Sacolick 2020). Este procedimento permite a deteção prematura de problemas, resultando na melhoria da qualidade do código produzido e na redução do tempo de lançamento de novas versões (AWS 2021k; Purohit 2020).

CD é o processo que ocorre, normalmente, após a compilação e teste da aplicação (CI). Este resume-se na implantação de novas alterações de forma automática e em ambientes específicos (teste, integração, etc), ainda que, por vezes, requeira um número reduzido de passos manuais durante a implantação de soluções em ambientes de produção, como representado na Figura 2.5 (AWS 2021j).

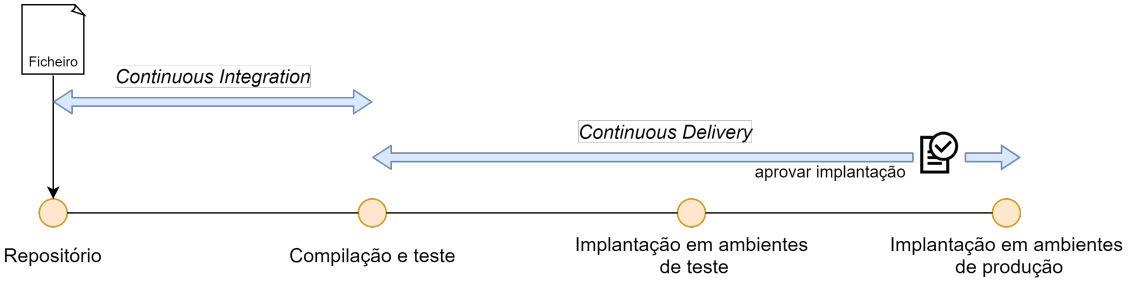


Figura 2.5: Processo de CI/CD. Fonte: (AWS 2021k)

De momento, esta técnica é utilizada recorrendo à ferramenta Jenkins, de modo a implantar as várias soluções existentes no *cluster* como aplicações, ferramentas de monitorização, bases de dados e infraestrutura.

2.1.5 Infrastructure as code

Infrastructure as Code (IaC) ou infraestrutura como código, é uma abordagem que se que se baseia na gestão e aprovisionamento de infraestrutura através de ficheiros, possibilitando o seu armazenamento num repositório (Morris 2020). Posteriormente, estes ficheiros podem ser utilizados para criar, modificar ou eliminar a infraestrutura neles declarada. Este processo vem melhorar inúmeros aspetos essenciais da gestão de infraestrutura, tais como (Morris 2020; Schults 2019):

- **Custo** - Reduz o custo de mão de obra, uma vez que a responsabilidade de gerir a infraestrutura passa a ser de uma equipa de desenvolvimento de produto ao invés de uma equipa especializada.
- **Velocidade** - Aumenta a velocidade de atualização, remoção ou criação de infraestrutura, facilitando a gestão da mesma através da execução de *scripts*.
- **Disponibilidade** - Aumenta a disponibilidade das aplicações integradas na infraestrutura criada, como resultado da automatização de passos manuais, reduzindo assim a introdução de erros na infraestrutura.

- **Consistência** - Permite executar diversas vezes a mesma configuração, resultando sempre na criação de infraestrutura semelhante e diminuindo assim a probabilidade de ocorrência de *environment drifts*.
- **Automatização da gestão de infraestrutura** - Elimina a necessidade de execução dos passos manuais sempre que seja necessário efetuar alguma alteração na infraestrutura, permitindo também a sua integração com ferramentas de CI/CD.
- **Reutilização de componentes** - Permite aplicar a mesma a infraestrutura em múltiplos ambientes (teste, produção, etc) e regiões (USA, EMEA, CN), utilizando código similar.
- **Documentação da arquitetura** - Possibilita gerar documentação e descrições para cada recurso e para as suas respetivas propriedades e dependências, aquando desenhado ou sempre que é modificado, permitindo assim partilha de conhecimento entre os vários colaboradores e facilitando a compreensão da infraestrutura.

A implantação de IaC pode ser realizada segundo duas abordagens: de modo declarativo ou de modo imperativo. No modo declarativo, para executar a criação de um recurso, é necessário especificar o recurso que se pretende criar, as suas propriedades e dependências, sendo a sua elaboração efetuada automaticamente e sem necessidade de ordenação dos recursos (Bellavance 2019; Brooks 2018). Por outro lado, no modo imperativo é necessário especificar todas as instruções necessárias para a criação dos recursos, sendo a ordem pela qual os estes são criados um fator relevante (Bellavance 2019; Brooks 2018).

Atualmente, é possuída IaC em múltiplos projetos, mais precisamente para a criação de bases de dados, de infraestrutura necessária para os projetos e de ferramentas, sendo que para o aprovisionamento de infraestrutura é utilizado a ferramenta Terraform.

2.1.6 Cloud Service Providers

Cloud Service Providers, ou provedores de serviço na nuvem, são organizações que oferecem serviços de computação, como *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) ou *Service as a Service* (SaaS), e ambientes como *private* ou *public clouds* (RedHat 2021c). A utilização de um *Cloud Service Provider* substitui a necessidade de um investimento inicial de capital em servidores, pelo pagamento de uma mensalidade, contemplando unicamente a infraestrutura utilizada. Desta forma, é evitado o desperdício de infraestrutura comprada que poderia não ser utilizada, permitindo também dispensar colaboradores peritos na área (poupança na mão de obra) e usufruir da segurança, rapidez e facilidade de criação de toda a infraestrutura através de IaC.

Atualmente, o *Cloud Service Provider* utilizado é a *Amazon Web Services* (AWS).

2.1.7 Ferramentas de aprovisionamento e ferramentas de gestão de configurações

As ferramentas de aprovisionamento de infraestrutura permitem desenhar os recursos para os criar, sejam eles servidores, bases de dados ou outro tipo de componentes necessários (Yevgeniy Brikman 2016; Yevgenjy Brikman 2019). Ferramentas de gestão de configurações, por outro lado, trabalham normalmente em servidores já aprovisionados, de modo a gerir o *software* neles presente, incluindo a atualização de versões, a instalação de novas ferramentas, entre outros (Yevgeniy Brikman 2016; Yevgenjy Brikman 2019).

2.1.8 Arquitetura atual

Como referido na Secção 1.2, 2.1.1 e 2.1.6, atualmente encontram-se implantados vários *clusters* de Kubernetes no *Cloud Service Provider* AWS, distribuídos por um variado número de regiões. A infraestrutura existente para o exemplo de um *cluster* de operações está representada na Figura 2.6.

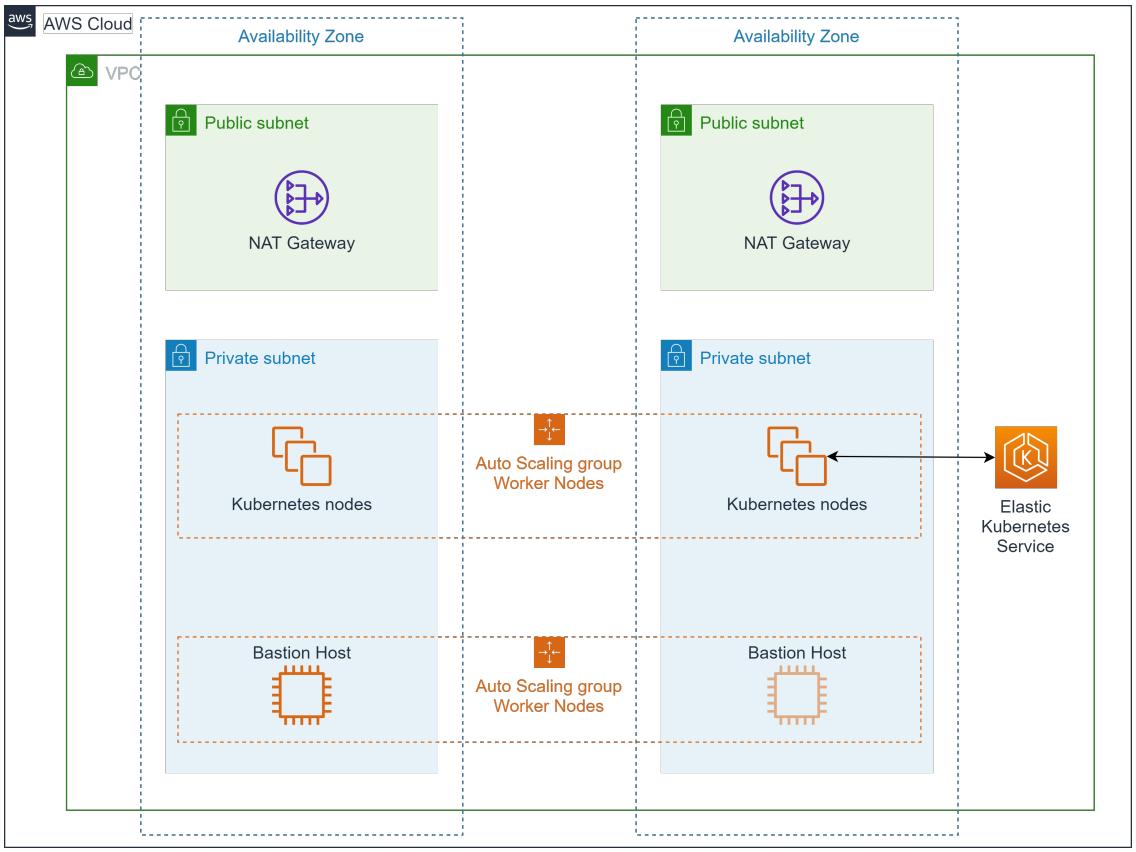


Figura 2.6: Arquitetura atualmente implantada na AWS para um *cluster* de operações.

Esta arquitetura é constituída pelos seguintes componentes:

- **Virtual Private Cloud (VPC)** - Este é um dos componentes mais essenciais da AWS, uma vez que possibilita a criação de recursos numa rede privada virtual e isolada (AWS 2021d). Este serviço declara os intervalos de *Internet Protocol* (IP) a utilizar em determinados componentes, os *subnets* e as *route tables*, para além de permitir criar pontos de conexão com a rede exterior através de *endpoints* (AWS 2021d);
- **Availability Zone (AZ)** - Consiste num ponto isolado onde estão instalados, fisicamente, *datacenters* (AWS 2021n). Dentro de cada região, as múltiplas AZs nela contidas encontram-se conectadas através de redes de baixa latência e elevada largura de banda, de modo a garantir alta velocidade na transferência de informação entre as mesmas (AWS 2021n). Assim, o *Cloud Service Provider* possibilita a formação de redundância na infraestrutura criada, a qual permite alcançar níveis superiores de alta disponibilidade e de tolerância a falhas de aplicações e bases de dados, algo que um *datacenter* isolado não conseguiria (AWS 2021n);

- **Subnet** - De uma forma simplificada, este recurso resume-se a uma rede dentro de outra rede. A utilização de *subnets* permite ao tráfego chegar ao seu destino mais rapidamente e de forma mais eficaz (CloudFlare 2021);

  Public subnet - Consiste numa *subnet* que permite o redirecionamento de tráfego para a *internet* (AWS 2021q);

  Private subnet - É uma *subnet* que impossibilita o acesso à *internet*. Em casos onde seja necessário criar uma conexão à *internet*, é utilizada uma NAT Gateway para o efeito (AWS 2021q);

- **NAT Gateway** - Componente que permite que instâncias inseridas numa *private subnet* estabeleçam uma ligação à *internet*, proibindo, porém, a inicialização da conexão às instâncias por parte da *internet* (AWS 2021i). A NAT Gateway pode ser também utilizada para permitir a conexão a outros VPCs (AWS 2021i). Este recurso é criado dentro de uma *public subnet*;

- **BH** - É uma máquina que permite filtrar a conexão à rede privada, minimizando assim penetrações na mesma (Malaval 2016). Este serviço limita o intervalo de *ips* com permissões para se conectar com os recursos da rede privada, aumentando assim a segurança geral da solução (Malaval 2016);

- **Amazon EC2** - É um serviço que disponibiliza capacidade de computacional, de um modo simples e rápido (AWS 2021a). É possível criar instâncias Amazon EC2 com recurso a uma variedade de combinações de configurações, resultando numa máquina altamente personalizável (AWS 2021a);

- **Auto Scaling Group (ASG)** - Consiste num componente que gere um conjunto de instâncias Amazon EC2. Este é responsável por adaptar o número de máquinas necessário para que os serviços sejam executados normalmente, considerando, para isso, o limite de instâncias e a adaptando esse número consoante a carga aplicada em cada máquina (AWS 2021e).

2.2 Trabalhos relacionados

Com a evolução da tecnologia e constante transição para uma cultura de DevOps, é cada vez mais frequente a elaboração estudos e projetos relacionados com a automatização de processos de implantação de infraestrutura, seja para soluções, ferramentas de monitorização ou até mesmo *clusters*.

A título de exemplo, tem-se o projeto "Implementação de infraestrutura como código para aprovisionamento e *deploy* de aplicações", realizado pelo autor Jones Noll, que é bastante pertinente para este projeto devido à semelhança dos seus propósitos (Noll 2020). A solução descrita foi construída no *Google Cloud Platform* (*Cloud Service Provider*), recorrendo a ferramentas como Kubernetes para orquestrar os contentores e como Docker para a criação dos mesmos. O aprovisionamento da infraestrutura foi realizado por meio de IaC, recorrendo à ferramenta Terraform. Posteriormente, para a gestão das configurações e instalação de ferramentas, foi utilizada a ferramenta Ansible. No final, o autor evidência as vantagens da utilização de IaC para a construção da solução e a rapidez que esta abordagem oferece ao processo. Com a conclusão deste projeto, surge a possibilidade de criar toda a infraestrutura de modo automático, a qual pode ser visualizada a partir do diagrama representado na Figura 2.7.

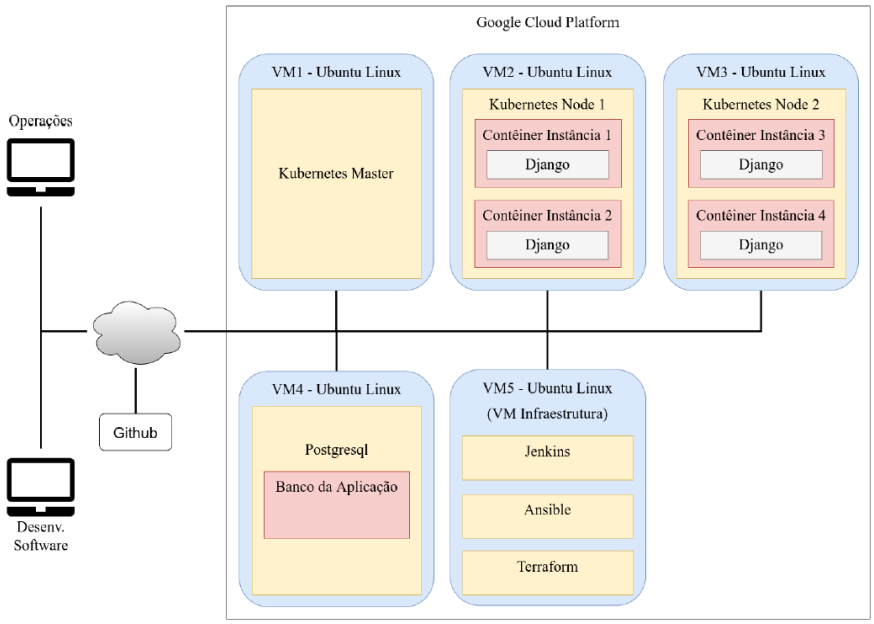


Figura 2.7: Arquitetura criada. Fonte: (Noll 2020)

O projeto apresentado revela ter um problema muito similar ao que se pretende resolver nesta dissertação. Porém, este não aborda a possibilidade de utilização de IaC em múltiplos ambientes, regiões ou para criação de diferentes tipos de *clusters*, da mesma forma que também não refere nenhuma aproximação para testar a infraestrutura criada.

Um outro exemplo, é um artigo elaborado por um grupo de estudantes do Instituto Politécnico de Milão, em parceria com uma empresa de *Information Technology* (IT) baseada em Ljubljana, onde é discutida a interligação entre a cultura de DevOps e a utilização de IaC (Tamburri et al. 2017). No documento, são referidas as razões pelas quais se sentiu necessidade de adotar a prática DevOps, seguida da explicação de IaC ser uma mais valia para equipas onde esta prática é aplicada (Tamburri et al. 2017). Dentro das razões mencionadas, encontra-se a facilidade em melhorar e testar a infraestrutura, a rapidez das operações devido a automatização e a reutilização de código (Tamburri et al. 2017).

O autor Yevgeniy Brikman elaborou, também, um estudo de forma a perceber qual a melhor ferramenta de IaC, para posterior adoção pela sua empresa, a Gruntwork (Yevgeniy Brikman 2016). Nesse estudo, é comparada uma série de ferramentas como Chef, Puppet, Terraform, SaltStack, AWS CloudFormation e Heat. Tendo como base os critérios estabelecidos pela empresa (ferramenta *open source*, sem dependência de um *Cloud Service Provider* em específico, com suporte para infraestrutura imutável, linguagem declarativa e com uma arquitetura de *client-only*), o autor concluiu que a melhor ferramenta a utilizar seria Terraform (Yevgeniy Brikman 2016).

Outro grupo de estudantes, de quatro faculdades distintas, efetuou uma série de entrevistas a vários colaboradores de múltiplas empresas de forma a recolher informação sobre as suas experiências com a utilização de IaC (Guerriero et al. 2019). O objetivo dessas entrevistas, passava por entender o panorama atual desta prática e quais os tópicos com margem para melhoria (Guerriero et al. 2019). Numa primeira fase, foram colocadas perguntas genéricas sobre o trabalho que os colaboradores exerciam, de modo a perceber a sua rotina. Considerando o *feedback* recebido, a maioria dos colaboradores desenvolviam IaC em simultâneo

com desenvolvimento de *software* aplicacional (88.1%) e 61.9% dos inquiridos gasta menos de 5 horas por semana a desenvolver IaC (Guerriero et al. 2019). Seguidamente, foram feitas perguntas de caráter mais técnico, entre elas quais as ferramentas que os desenvolvedores utilizam. Analisando as respostas recebidas, 69% dos entrevistados faziam uso de 3 ou mais ferramentas de IaC, sendo a utilização das mesmas é declarada com maior detalhe na Tabela 2.1 (Guerriero et al. 2019).

Tabela 2.1: Resultados do questionário sobre as ferramentas utilizadas, ordenadas pelo número descendente de frequência. Fonte: (Guerriero et al. 2019)

Ferramenta	Nº menções	Utilização
Docker	26	59.0%
Ansible	23	52.2%
Vagrant	19	43.1%
Kubernetes	18	40.9%
Chef	16	36.3%
Terraform	15	34.1%
Puppet	13	29.5%
Apache Brooklyn	09	20.0%
Packer	09	20.0%
CloudFormation	09	20.0%
TOSCA	08	18.2%
Salt	06	13.6%
Shell scripts	04	09.0%
Cloudify	03	06.8%
Octopus Deploy	01	02.3%
Azure DevOps	01	02.3%

Na etapa final, foram discutidas as dificuldades existentes no manuseamento de IaC. Os desafios mais referidos dirigiam-se para a impossibilidade de testar o código produzido, a dificuldade de compreensão da infraestrutura codificada e a inconsistência da execução do código, incluindo a falta de compatibilidade com versões mais antigas aquando de uma atualização (Guerriero et al. 2019).

Por último, o autor Kevin Luu realizou o aprovisionamento de uma instância Amazon EC2 na AWS recorrendo à ferramenta Terraform (Luu 2020). Conseguiu também, com recurso a uma funcionalidade desta ferramenta, instalar um mecanismo CI/CD, mais propriamente o Jenkins. Como resultado desta experiência, o autor sublinha a facilidade de implantação da infraestrutura referida através de Terraform.

2.3 Tecnologias relevantes

Nesta secção são analisadas diversas ferramentas com possibilidade de serem utilizadas para a realização deste projeto, assim como ferramentas já utilizadas na solução corrente. Dentro de cada categoria de ferramentas, cada ferramenta é descrita tendo em conta diversos critérios, sendo também detalhados os respetivos pontos positivos e negativos.

2.3.1 Ferramentas de aprovisionamento de infraestrutura

Como referido na Secção 2.1.7, as ferramentas de aprovisionamento facilitam a construção de infraestrutura no *Cloud Service Provider* de eleição.

De seguida, são analisadas detalhadamente duas das mais conhecidas ferramentas de aprovisionamento de infraestrutura com integração com o *Cloud Service Provider* AWS: Terraform e AWS CloudFormation.

2.3.1.1 Terraform

Terraform é uma ferramenta *open source* que viabiliza o aprovisionamento de infraestrutura criada pela Hashicorp. Esta possibilita descrever, planear, criar e até mesmo eliminar infraestrutura para uma variedade de *Cloud Service Providers* (Yevgenjy Brikman 2019; Chan 2018). A declaração dos recursos a gerir é feita através da sua DSL, *HashiCorp Configuration Language* (HCL), que permite a utilização de blocos de código para definição da infraestrutura (Terraform 2020).

Através desta ferramenta, é feita a designação de IaC através de múltiplos elementos Terraform, entre eles:

- **Provider** - Elemento que viabiliza a comunicação entre o código criado e sistemas externos, sendo maioritariamente utilizado para declarar os *Cloud Service Providers* (Terraform 2021a). É normalmente usado como um *plugin*, adicionando a possibilidade de criar uma variedade de novos elementos do tipo *resource*;
- **Resource** - Elemento mais importante da ferramenta (Terraform 2021b). A utilização deste bloco permite definir um ou vários objetos de infraestrutura, como máquinas ou bases de dados, considerando o *provider* definido (Terraform 2021b);
- **Module** - Elemento que é definido através de várias configurações de Terraform, permitindo a reutilização do mesmo código para criação de recursos similares (Yevgenjy Brikman 2019). Este elemento possibilita também o reaproveitamento de módulos criados pela comunidade, através do Terraform Registry;
- **Data** - Elemento onde são inseridos parâmetros de modo a recolher informação sobre recursos já existentes no *Cloud Service Provider*, com a intenção de a utilizar na criação outros;
- **Output** - Elemento que devolve o valor de um componente Terraform, com o propósito de ser utilizado posteriormente;
- **Variable** - Elemento onde são definidos vários parâmetros a serem utilizados aquando da utilização de outros elementos. Assim, é permitido a reutilização dos mesmos recursos para criar infraestrutura semelhante em regiões e ambientes distintos, sendo somente necessário efetuar a declaração das variáveis para cada contexto.

Para utilização destes elementos, a ferramenta possui uma variedade de comandos que permite a sua aplicação, destacando, entre eles:

- *"terraform init"* - Comando que inicializa a configuração descrita e transfere toda a informação necessária para utilizar com sucesso os *providers* definidos (Bellavance 2019);

- *"terraform plan"* - Comando que permite visualizar o plano com as alterações que a ferramenta vai efetuar (Yevgenjy Brikman 2019);
- *"terraform apply"* - Comando que mostra e aplica o plano criado anteriormente, caso o colaborador confirme a execução do mesmo (Yevgenjy Brikman 2019);
- *"terraform destroy"* - Comando utilizado para destruir a infraestrutura criada previamente (Yevgenjy Brikman 2019).

Sempre que são executados comandos de Terraform, a informação da infraestrutura existente nesse preciso o momento, é mapeada para um ficheiro de estado em formato JSON (Yevgenjy Brikman 2019). Esse ficheiro permite comparar a informação nele inserida (infraestrutura corrente) com as configurações pretendidas, determinando as alterações a efetuar para se obter a infraestrutura desejada (Yevgenjy Brikman 2019). Este ficheiro é normalmente guardado remotamente, de modo a permitir o aprovisionamento e modificação da infraestrutura por múltiplos colaboradores.

2.3.1.2 AWS CloudFormation

AWS CloudFormation é um serviço gerido pela AWS que permite o aprovisionamento de infraestrutura através de *templates*. Nesses *templates*, escritos em formato JSON ou YAML, é efetuada a descrição dos recursos que se pretendem criar, bem como as suas propriedades de um modo declarativo. À semelhança da ferramenta apresentada na Secção 2.3.1.1, os recursos não necessitam de ser definidos pela sua ordem de criação, podendo ser utilizado o atributo *dependsOn* para criar uma sequência de eventos e gerar dependência entre os mesmos (AWS 2020b). Os *templates* podem ser reutilizados em mais que um ambiente, sendo somente necessário alterar os valores das variáveis do mesmo consoante o ambiente desejado (AWS 2020b).

A criação dos *templates* pode ser realizada manualmente ou através da ferramenta AWS CloudFormation Designer, disponibilizada na AWS Management Console. Nesta ferramenta é possível gerar *templates* através de *drag-and-drop*, sendo para isso necessário selecionar o recurso que se pretende criar, preencher as suas propriedades e conectar o recurso às suas dependências, caso existam (AWS 2020b). A *interface* gráfica da ferramenta pode ser visualizada na Figura 2.8.

Criado o *template*, este é aplicado de modo a gerar a infraestrutura pretendida. Para tal, podem ser seguidas três abordagens: utilização da AWS Management Console, AWS CLI ou AWS SDK (AWS 2020b). A primeira abordagem, resume-se na utilização dos *templates* através da ferramenta AWS CloudFormation na AWS Management Console, onde é possível fazer o seu *upload* (AWS 2020b). Posteriormente, o *template* fica guardado e disponível num S3 Bucket (serviço de armazenamento de ficheiros) de modo a ser utilizado pela AWS CloudFormation para a criação da infraestrutura (AWS 2020b). Na segunda aproximação, os comandos existentes na AWS CLI são utilizados para administrar a infraestrutura (AWS 2020b). Esta acaba por ser uma alternativa com grande importância, devido à possibilidade de integrar e manusear os *templates* com *pipelines* e outros mecanismos de CI/CD (AWS 2020b). Por último, é também possível utilizar a AWS SDK/API para fazer o aprovisionamento e implantação de uma *stack*, sendo para isso necessário a sua integração no projeto referente (AWS 2021o). No momento da escrita, a AWS SDK tem compatibilidade com as seguintes linguagens de programação: C++, Java, Ruby, JavaScript, Go, .Net, Node.js, PHP e Python (AWS 2021o).

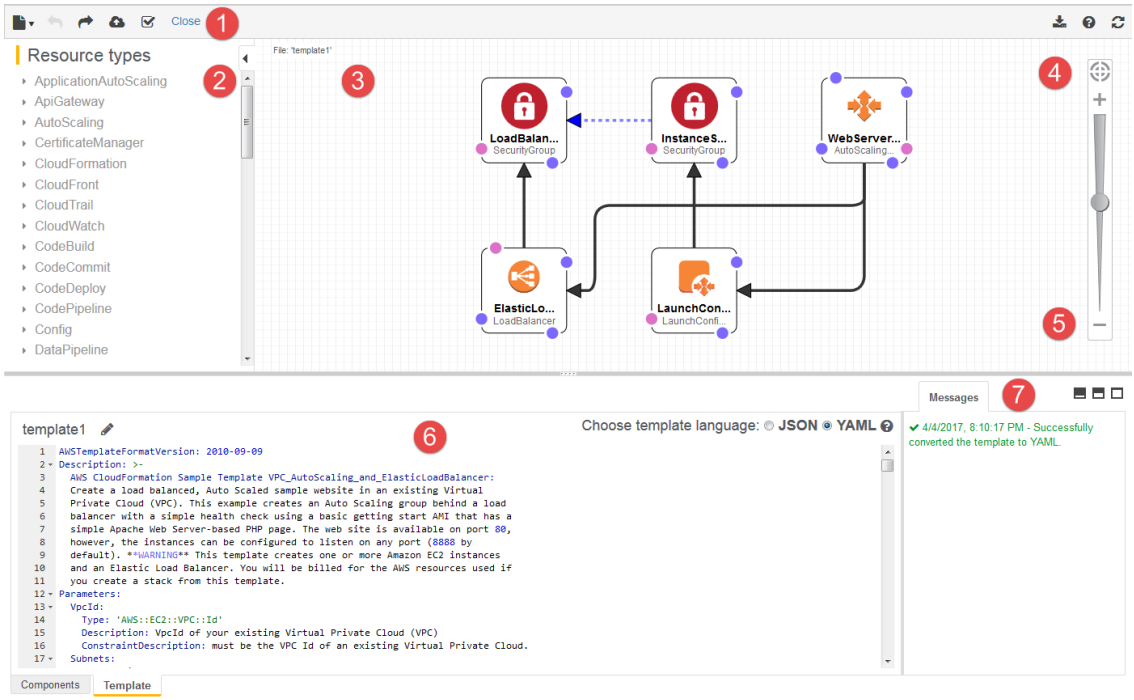


Figura 2.8: *Interface* gráfica da ferramenta AWS CloudFormation Designer.
Fonte: (AWS 2021f)

Esta ferramenta apresenta, no entanto, um aspeto negativo relacionado com a dependência de aprovisionamento da infraestrutura dentro do ecossistema AWS.

2.3.2 Ferramentas de gestão de configurações

Como referido na Secção 2.1.7, ferramentas de gestão de configurações permitem configurar, de modo consistente, um recurso ou sistema em vários ambientes, com o propósito de atingir um estado desejável. Nesta Secção são descritas três ferramentas para o efeito, sendo que estas foram escolhidas com base na sua popularidade e custo. Todas as ferramentas mencionadas são *open source* e possuem capacidade de aprovisionamento de infraestrutura e teste sobre as configurações definidas, mesmo que com a utilização de uma ferramenta externa.

2.3.2.1 Chef

Chef é uma ferramenta de gestão de configurações que, tal como o nome indica, possui terminologias em torno de uma metáfora sobre culinária.

O bloco fundamental para o funcionamento desta ferramenta são as *recipes*, sendo um bloco escrito em Ruby, com configurações e tarefas a executar em uma ou mais máquinas, como exemplificado no Bloco de código 2.1 (Bannan 2019).

```
1 Package "apache" do
2     action: install
3 end
```

Bloco de código 2.1: Exemplo de uma *recipe* para instalar a *package* apache.
Fonte: (Hassan 2016)

As tarefas inseridas numa *recipe* são executadas seguindo a ordem pela qual são declaradas no ficheiro (Bannan 2019), podendo também ser estabelecidas dependências entre elas, de maneira a evitar que as restantes sejam executadas após a ocorrência de uma falha (Bannan 2019). Na definição de uma *recipe*, podem ser consideradas propriedades do sistema inserido, de modo a produzir um *output* diferente dependendo da máquina onde esta é executada (Bannan 2019).

Um *cookbook* é criado a partir de um conjunto de *recipes*, juntamente com outros ficheiros necessários para a sua execução (Hassan 2016). Para além da criação manual de um *cookbook*, também é possível efetuar a transferência de *templates* já com vários recursos predefinidos.

Esta ferramenta possui uma arquitetura baseada em *master-agent*, existindo um servidor que detém a informação que é transmitida para os clientes através de um mecanismo de *pull* (o cliente vai buscar nova informação ao servidor), como está exemplificado na Figura 2.9.

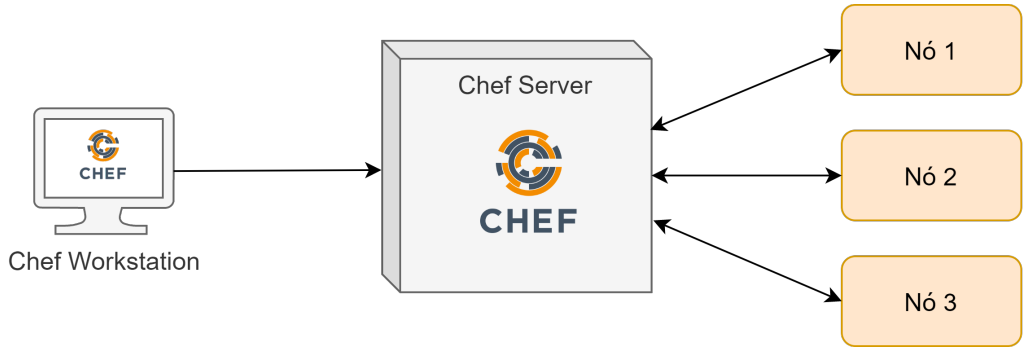


Figura 2.9: Arquitetura da ferramenta Chef. Fonte: (N 2020)

2.3.2.2 Puppet

Puppet é uma ferramenta que permite administrar configurações através da sua DSL, Puppet code, de modo declarativo (Puppet 2021a). Esta ferramenta apresenta uma arquitetura *master-agent*, como representado na Figura 2.10. No servidor são guardados os ficheiros com as configurações que o *agent* transfere para uso posterior, através de um mecanismo de *pull*, tal como a ferramenta descrita na Secção 2.3.2.1 (Puppet 2021a).

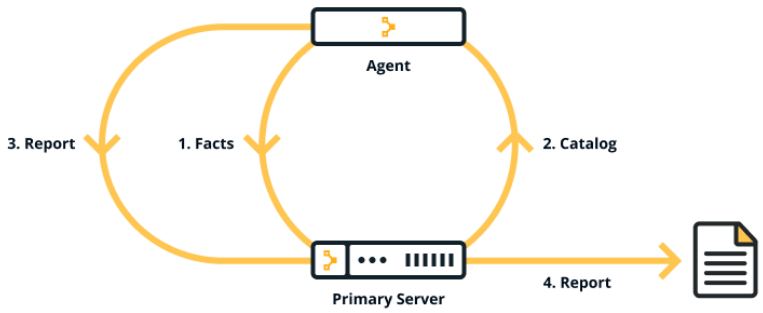


Figura 2.10: Processos realizados na arquitetura da ferramenta. Fonte: (Puppet 2021a)

O estado desejado da infraestrutura é alcançado através de *resources*, sendo este o bloco fundamental desta ferramenta, permitindo a definição das configurações pretendidas para o componente (Puppet 2021b). Os *resources* podem ser subdivididos em diversos tipos, entre eles ficheiros, *packages*, serviços ou até mesmo personalizados (criados manualmente), de modo a satisfazer uma necessidade específica (Puppet 2021b). Apesar de não poder ser criado duas vezes, um *resource* pode ser criado e posteriormente modificado, como exemplificado no Bloco de código 2.2 (Puppet 2021b).

```
$file_ownership = {
    "owner" => "root",
    "group" => "root",
    "mode" => "0644"
}

file { '/etc/passwd':
    ensure => file,
    *      => $file_ownership
}

file['/etc/passwd'] {
    "owner" => "diogo"
}
```

Bloco de código 2.2: Exemplo de um ficheiro de configurações. Fonte: (Puppet 2021b)

Em conformidade com as ferramentas anteriormente apresentadas, esta também possui um repositório a partir do qual podem ser transferidos *templates* com configurações já predefinidas (Puppet 2021a).

2.3.2.3 Ansible

Ansible é uma ferramenta que utiliza ficheiros YAML (na forma de Ansible Playbooks), escritos de modo imperativo, para descrever as configurações pretendidas e correlacionar os sistemas existentes (Ansible 2021a). A utilização desta ferramenta não requer agentes, servidores ou bases de dados extra, sendo apenas necessário os modelos disponibilizados pela ferramenta (Ansible 2021a).

Ansible Modules são pequenos blocos de código criados de forma a serem executados no *host* ou num Ansible Playbook, devolvendo um resultado que posteriormente pode ser utilizado (Ansible 2021a,d). Como estes ficheiros só têm uma função, podem ser facilmente executados através da linha de comandos, com a possibilidade de aceitar argumentos (Ansible 2021d). Para além disso, é possibilitada a utilização de *plugins*, permitindo assim a expansão das capacidades da ferramenta. Os sistemas a configurar devem ser declarados, no ficheiro *inventory*, de maneira a serem orquestrados pela ferramenta, para que mais tarde possam ser alterados através dos Ansible Playbooks. Estes ficheiros permitem descrever as tarefas a concretizar pela ferramenta, nos sistemas declarados no ficheiro *inventory*, segundo a ordem inserida, como exemplificado no Bloco de código 2.3 (Ansible 2021c).

```
- name: update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
```

```
7     yum:
8       name: httpd
9       state: latest
10
11  - name: Write the apache config file
12    template:
13      src: /srv/httpd.j2
14      dest: /etc/httpd.conf
15
16  - name: Update db servers
17    hosts: databases
18    remote_user: root
19
20    tasks:
21    - name: Ensure postgresql is at the latest version
22      yum:
23        name: postgresql
24        state: latest
25    - name: Ensure that postgresql is started
26      service:
27        name: postgresql
28        state: started
```

Bloco de código 2.3: Exemplo de um Ansible Playbook. Fonte: (Ansible 2021c)

2.3.3 Ferramentas de testes de IaC

Ferramentas de testes de IaC permitem garantir que a infraestrutura que se pretende criar se encontra corretamente declarada. Apesar desta ser uma área que recentemente tem tido um maior interesse, em parte devido ao aumento de utilização de IaC, ainda são poucas as soluções existentes que permitem efetuar testes sobre a infraestrutura.

2.3.3.1 Terrascan

Terrascan é uma ferramenta *open source* que permite verificar se a IaC elaborada possui alguma configuração incorreta (Farrell 2021). Esta ferramenta monitoriza possíveis desvios na configuração da infraestrutura, deteta vulnerabilidades de segurança e permite diminuir os riscos na implantação dos recursos configurados (Farrell 2021). Esta ferramenta é compatível com os principais *Cloud Service Providers* (AWS, Azure e GCP) e permite efetuar *scans* a recursos de Terraform, Kubernetes, Helm, Docker e AWS CloudFormation (Accurics 2021; Farrell 2021).

Para além da sua vasta compatibilidade, esta ferramenta pode ser executada tanto localmente como numa ferramenta de CI/CD (Farrell 2021).

2.3.3.2 Terratest

Terratest é uma ferramenta que permite efetuar testes e validações à infraestrutura aquando da sua criação/modificação, sendo que esses testes são criados com recurso à linguagem Go com a *framework testing* (Terratest 2021). Esta ferramenta possibilita testar código de Terraform, *templates* de Packer, imagens de Docker e Charts de Helm (Terratest 2021).

Durante a realização de testes, é efetuada a construção da infraestrutura em ambientes reais (Terratest 2021). Depois de implantada, essa infraestrutura pode ser testada de várias

formas, entre elas: pedidos HTTP, chamadas à API, conexões SSH, etc (Terratest 2021). Por fim, e depois de realizadas todas as verificações, a infraestrutura criada para os testes é destruída (Terratest 2021).

2.3.4 Ferramentas de orquestração de contentores

Ferramentas de orquestração de contentores facilitam todo o processo de criação e gestão das soluções neles contidas. Este tipo de ferramenta permite automatizar a execução de várias tarefas nos ambientes onde os contentores estiverem inseridos, sem necessidade de reestruturação da infraestrutura. Tem-se como exemplo as seguintes tarefas (Red Hat 2021a):

- Aprovisionamento e implantação de aplicações;
- Segurança das interações entre contentores;
- Disponibilidade do contentor;
- Balanceamento de carga e roteamento do tráfego;
- Configuração das aplicações baseado no contentor onde são executadas;
- Configuração e agendamento da implantação de uma aplicação;
- Escalamento da aplicação (horizontalmente e/ou verticalmente) quando necessário;
- Alocação os recursos necessários para a execução da aplicação;
- Monitorização do estado do contentor.

Para usufruir das capacidades destas ferramentas, é necessário criar um ou vários ficheiro(s) de configurações da aplicação, normalmente em formato JSON ou YAML, onde são mencionadas caraterísticas da aplicação, entre as quais a localização da imagem a utilizar na construção do contentor (Red Hat 2021a). Assim, estas ferramentas garantem uma maior portabilidade da solução, devido à possibilidade de escalar um contentor através de um comando sem afetar a execução aplicação. Além disso, é também simplificado o processo de implantação da aplicação, da mesma forma que é melhorada a segurança da mesma, como consequência da sua execução isolada (AVI Networks 2021).

Tendo em consideração a utilização deste tipo de ferramentas, tanto em ambientes de desenvolvimento como em ambientes de produção, é de extrema necessidade que possam ser corrigidos erros sem que toda a solução fique indisponível, pelo que a escolha da ferramenta é crucial para o bom funcionamento da solução.

2.3.4.1 Kubernetes

Desenvolvida em 2014 pela Google, o Kubernetes foi uma das primeiras soluções de orquestração de contentores a existir no mercado, sendo por essa razão a mais utilizada pelas empresas. Esta ferramenta *open source* facilita a configuração e automação de contentores, além de dispor de um grande número de funcionalidades, entre elas (Kubernetes 2020d):

- **Balanceamento de carga e descoberta do serviço** - Caso a carga de um serviço seja elevada, o Kubernetes consegue distribuí-la, estabilizando assim o serviço. Também consegue expor o serviço através de um *Domain Name System* (DNS) ou através do seu IP.

- **Orquestração do armazenamento** - Permite a adição de uma fonte de armazenamento ao serviço.
- **Rollouts e rollbacks automáticos** - Permite que a versão mais recente dos Pods seja revertida de forma automática para uma versão anterior estável (caso haja ocorrência de falhas), permitindo também a atualização automática dos mesmos.
- **Reviver o serviço** - O Kubernetes percebe quando ocorre uma falha nos contentores e testa várias abordagens para os conseguir "levantar", entre as quais: reiniciar contentores, trocar contentores ou remover os que estão a falhar, fazendo com que a conexão com o cliente só ocorra quando os mesmos estiverem a ser executados sem erros.
- **Gestão de segredos e configurações** - Permite que sejam utilizadas informações sensíveis na criação e manutenção dos contentores, sem que essa informação seja partilhada. Também permite a alteração da mesma em *runtime*.
- **Escalamento automático dos serviços** - Caso o aumento de carga sobre um serviço resulte numa utilização dos recursos superior à configurada, afetando o tempo de resposta do mesmo, o Kubernetes escala horizontalmente ou verticalmente o serviço de modo automático.

Ao utilizar a ferramenta é originado um *cluster* constituído no mínimo por um *worker node*, como apresentado na Figura 2.11. É nestas máquinas que podem ser implantadas as aplicações, em Pods. Um Pod é o recurso mais simples do Kubernetes e representa um conjunto de contentores (Kubernetes 2019). Nesta ferramenta é o *Control Plane* que gere os *worker nodes* e por conseguinte os Pods, permitindo que haja tolerância a falhas, resultando também em alta disponibilidade (Kubernetes 2020a).

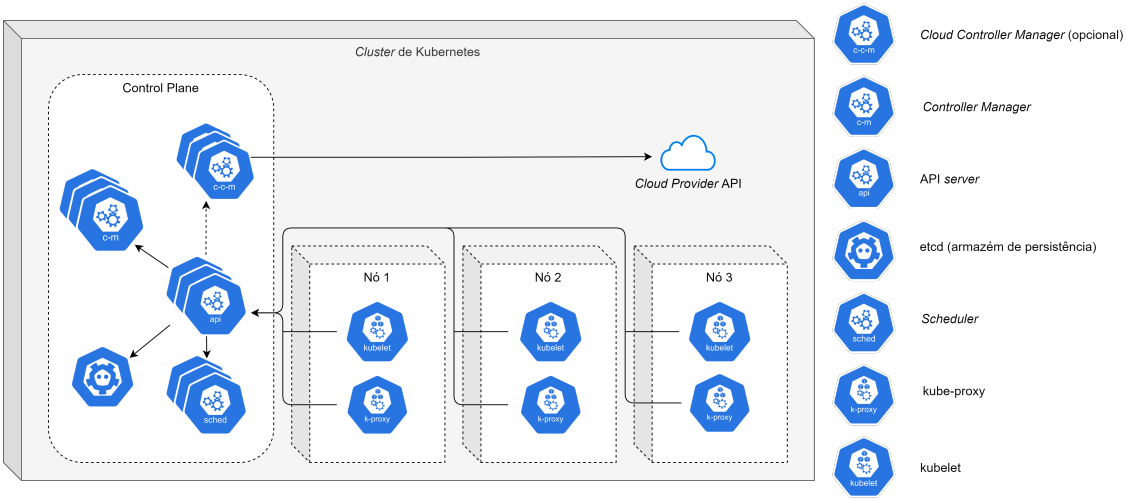


Figura 2.11: *Cluster* de Kubernetes com os respetivos componentes. Fonte: (Kubernetes 2020a)

O *Control Plane* é um componente que gere e toma decisões sobre o *cluster*. Nele estão contidos todos os componentes necessários para que se efetue essa gestão, como já pôde ser observado na Figura 2.11, incluindo o kube-apiserver, componente responsável por disponibilizar a API do Kubernetes, kube-scheduler que verifica a existência de Pods que não tenham sido atribuídos a um nó e os distribui, entre outros (Kubernetes 2020a).

Um nó possui vários componentes que tornam possível a orquestração dos contentores (Kubernetes 2020a), incluindo:

- **Kubelet** - Componente que verifica se os Pods estão a ser executados corretamente;
- **Kube-proxy** - Componente que faz a manutenção das regras de *network* nos nós;
- **Container runtime** - Componente responsável por executar os contentores.

Dentro de cada *worker node*, são inseridos vários objetos de Kubernetes. Estes objetos podem descrever os contentores e os nós onde estão a ser executados. Além disso, podem também representar os recursos utilizados para gerir os contentores como *Horizontal Pod Autoscaler* (HPA), regras de reinício, entre outros. Por fim, podem também especificar componentes como ConfigMap, volumes ou outros exemplificados na Figura 2.12 (Kubernetes 2020c). A gestão destes objetos pode ser efetuada através da API disponibilizada pela ferramenta (Kubernetes 2020c).

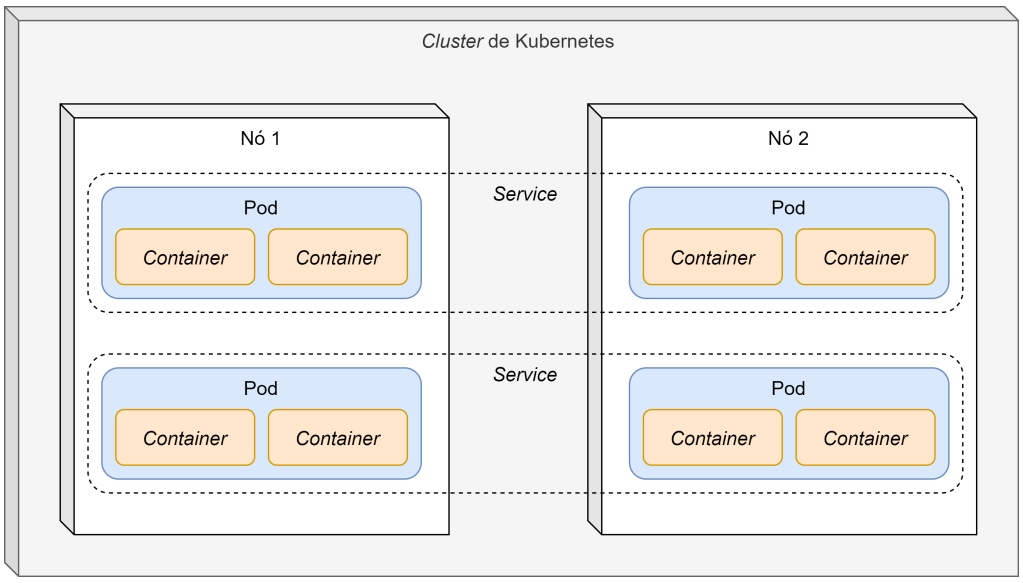


Figura 2.12: Exemplificação dos múltiplos objetos de Kubernetes. Fonte: (Julian 2021)

De modo a criar um objeto dentro de um *worker node*, é necessário realizar a sua definição num ficheiro YAML ou JSON. Nele são declaradas algumas propriedades desse recurso, tais como o tipo de objeto que se pretende criar, a informação de identificação do objeto, o *namespace* inserido e, em alguns casos, a *spec* que corresponde ao estado desejável do objeto declarado aquando da sua criação (Kubernetes 2020c). O estado, por sua vez, indica a situação em que o objeto se encontra no momento, possuindo assim um papel fundamental para a gestão dos recursos criados (Kubernetes 2020c). Sempre que o estado não for igual ao *spec* (estado desejável), o *Control Plane* executa as ações necessárias para que o estado iguale a situação desejável (Kubernetes 2020c). Estes objetos podem ser inseridos dentro de um *namespace*, permitindo a separação das várias aplicações por contextos (Kubernetes 2020b). Um objeto só pode pertencer a um *namespace*.

São várias as alternativas que podem ser utilizadas no que diz respeito à utilização da ferramenta Kubernetes (mesmo considerando apenas soluções na AWS). Uma das alternativas

passa por criar uma instância Amazon EC2 e instalar o Kubernetes na mesma. Outra alternativa é a utilização do serviço Amazon EKS com instâncias EC2 e a última é a utilização do serviço Amazon EKS em conjunto com o AWS Fargate.

No caso de ser criada uma instância Amazon EC2 com a ferramenta instalada na mesma, todo o aprovisionamento de máquinas fica a cargo do colaborador. Felizmente, existem ferramentas que ajudam no aprovisionamento do *cluster*, facilitando a instalação de Kubernetes na máquina. Uma dessas ferramenta é o Kops que ajuda a criar, atualizar e destruir um *cluster* de Kubernetes, para além de tratar de aspetos fundamentas como alta disponibilidade e infraestrutura (Kops 2021). Com apenas um comando é possível gerar todas as configurações necessárias para um criar um *cluster* a partir do Kops.

Kops é uma ferramenta *open source* com uma grande comunidade de desenvolvedores, existindo, por essa razão, fácil acesso a informação relacionada com a mesma, incluindo diversos tutoriais (Arora 2020). Por outro lado, existem diversas desvantagens associadas ao optar por esta solução, como por exemplo: a atualização da versão do Kubernetes tende a ser um processo complexo e demorado e a gestão do *cluster* e da máquina Amazon EC2 é efetuada pela equipa. Por outro lado, o processo referido no último ponto pode também ser considerado um ponto positivo, já que existe um maior poder sobre a solução.

Para além desta abordagem, pode também ser optado por utilizar o serviço Amazon EKS, com as suas variantes exemplificadas na Figura 2.13:

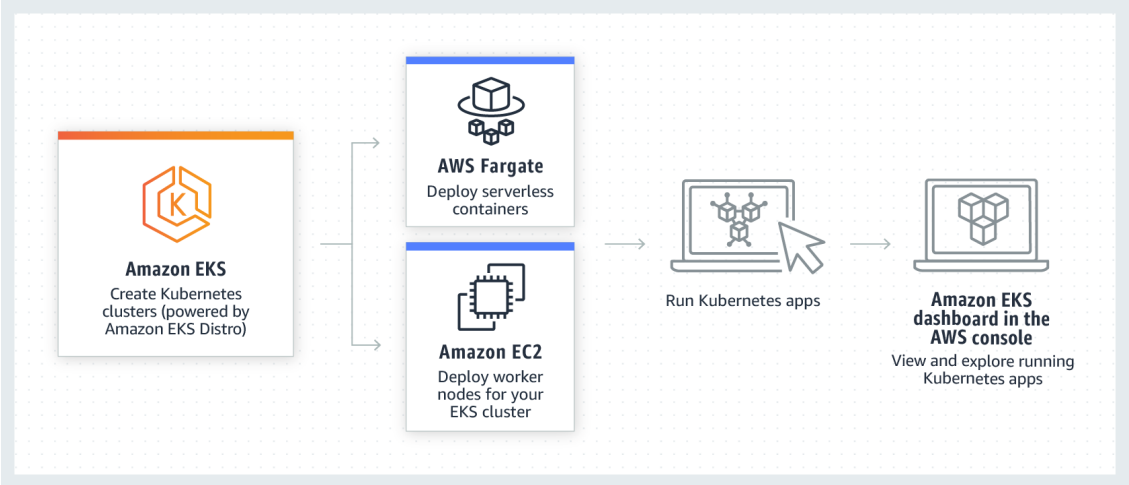


Figura 2.13: Exemplificação das possibilidades ao utilizar Amazon EKS.
Fonte: (AWS 2021g)

Uma dessas abordagens passa por utilizar instâncias Amazon EC2 em conjunto com o serviço Amazon EKS (solução utilizada atualmente), associando assim todas as vantagens e desvantagens referentes à utilização destes dois serviços. O serviço Amazon EKS é completamente gerido pela AWS, permitindo, de um modo mais fluído, sua a integração com outras ferramentas do ecossistema, como o Amazon CloudWatch, AWS Identity and Access Management, entre outros (AWS 2021b). A utilização deste serviço tem associada uma série de aspetos positivos, tais como: o *Control Plane* é gerido pela AWS; garantia de alta disponibilidade através da execução do *master node* em múltiplas regiões diferentes, eliminando assim possíveis pontos de falha; aplicação automática dos últimos *patches* de

segurança, etc (AWS 2021b,m). Por outro lado, a utilização deste serviço tem associado um custo extra, em comparação com ferramenta Kops (AWS 2021m).

Por fim, existe também a possibilidade de seguir a abordagem onde se utiliza o serviço AWS Fargate em conjunto com o Amazon EKS, associando assim o potencial do Amazon EKS às capacidades computacionais do AWS Fargate. O AWS Fargate é um mecanismo de computação sem servidor para contentores que pode operar em conjunto com o Amazon EKS. Este permite que sejam criados servidores sem a necessidade de os gerir ou aprovisionar, aumentando também a segurança ao isolar as aplicações nele contidas, sendo cada Pod (no caso de ser utilizado com EKS) executado num ambiente individual, com recursos de CPU, memória, armazenamento ou rede individuais (AWS 2021g). À semelhança do Amazon EKS, também fornece integração com os vários serviços da AWS e só é cobrado pelos recursos que se utilizam (AWS 2021g).

Na Figura 2.14 estão representadas as diferenças entre o uso de AWS Fargate e Amazon EC2.

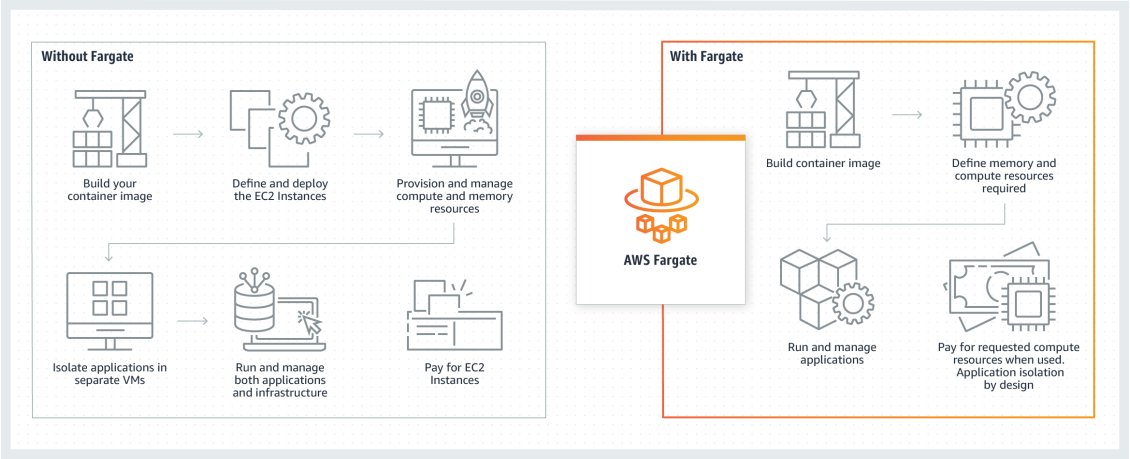


Figura 2.14: Diferenças entre o uso de AWS Fargate e Amazon EC2. Fonte: (AWS 2021g)

2.3.4.2 Docker Swarm

Criado em 2013, o Docker revolucionou a maneira como se interage com contentores. Para além de ser uma ferramenta *open source* e com grande popularidade desde o seu lançamento, o Docker facilita a construção de aplicações em contentores e a comunicação entre as mesmas. Estas caraterísticas permitem a modularização e separação de algumas funcionalidades da aplicação por vários contentores, possibilitando escalar e atualizar cada componente de forma independente (Kasireddy 2016). Uma das funcionalidades mais importante do Docker são as suas imagens, sendo cada uma um conjunto de todas as configurações necessárias para executar uma aplicação no contentor, sejam dependências, comandos de inicialização ou variáveis ambiente (Kasireddy 2016). As imagens podem ser geradas através de um Dockerfile com recurso do comando "docker build" (Kasireddy 2016). Pode ser visualizado o exemplo de um Dockerfile através do Bloco de código 2.4.

```
1 FROM python:3
2
3 WORKDIR /usr/src/app
4
5 # Copia os ficheiros para o contentor
```

```
6  COPY . .
7
8  # Instala dependencias
9  RUN pip install --no-cache-dir -r requirements.txt
10
11 # Define a porta para o contentor expor
12 EXPOSE 5000
13
14 # Corre o comando
15 CMD ["python", "./app.py"]
```

Bloco de código 2.4: Exemplo de um Dockerfile. Fonte: (Kasireddy 2016)

Uma funcionalidade bastante utilizada aquando da criação de um Dockerfile é o uso de uma imagem base com algumas dependências e configurações já predefinidas. Isto é possível com recurso ao Docker Hub, um repositório com milhares de imagens Docker já geradas e prontas a utilizar (Kasireddy 2016). No caso do Bloco de código 2.4 , está a ser utilizado uma imagem com a versão 3 de Python como base.

A utilização do Docker em *swarm mode*, permite que sejam ativadas as funcionalidades de gestão e orquestração de contentores. O *swarm mode* permite a ligação e coordenação de várias máquinas. Existem dois tipos de nós no Docker Swarm: o *manager node*, responsável pela distribuição de tarefas e os *worker nodes*, responsáveis pela execução das tarefas, como esquematizado na Figura 2.15 (Docker 2021a). Os serviços implantados, à semelhança dos objetos de Kubernetes, possuem um estado desejável mantido pelo Docker.

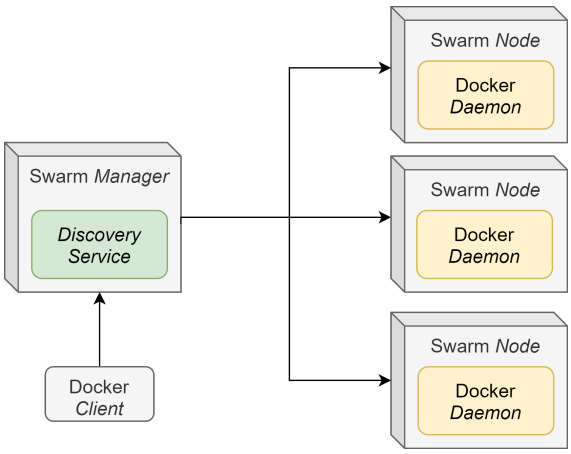


Figura 2.15: Arquitetura do Docker Swarm. Fonte: (Alabonte 2019)

Uma das principais vantagens do Docker Swarm é a possibilidade de modificar uma aplicação em execução (tamanho do armazenamento, número de réplicas, etc) sem necessidade de reinício da mesma (Docker 2021a). Para além dessa vantagem, a gestão de *clusters* já se encontra integrada com a ferramenta Docker, não sendo necessário a instalação de *software* adicional. Esta ferramenta possui um desenho descentralizado, podendo qualquer nó passar, sempre que necessário, de *worker* para *manager* (Docker 2021b).

2.3.5 Ferramentas de CI/CD

Como já referido na Secção 2.1.4, CI/CD é um conjunto de práticas que permitem automatizar vários processos de implantação de soluções. Atualmente, são várias as ferramentas

que o permitem, tal como se pode observar na Figura 2.16, embora nesta secção sejam apenas analisadas ferramentas *open source* e independentes de plataforma.

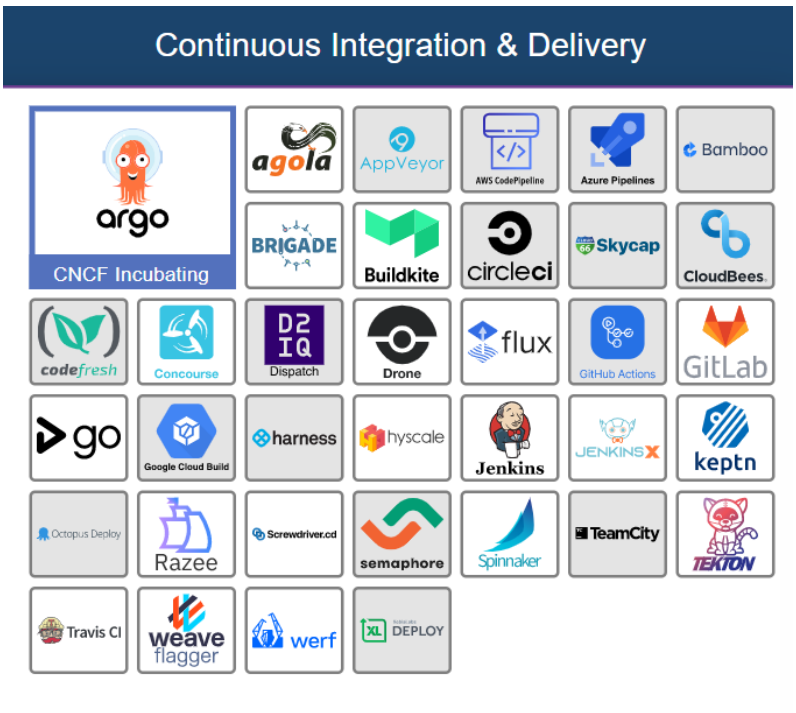


Figura 2.16: Ferramentas de CI/CD atualmente existentes no mercado.
Fonte: (Cloud Native Landscape 2021)

2.3.5.1 Jenkins

Jenkins é uma ferramenta *open source* baseada em Java e desenhada para ser um ponto de automação de todo o processo de CI/CD (Katalon 2021). Esta ferramenta está disponível em plataformas como Windows, MacOs e Linux e também em ferramentas como o Docker, permitindo assim a sua execução em qualquer ambiente (Jenkins 2021a).

O Jenkins possui uma *interface* bastante intuitiva e de fácil utilização, mesmo para quem possui pouca ou nenhuma experiência com a mesma, como pode se pode constatar pela Figura 2.17. A partir da mesma, é possível fazer a criação de novos projetos, gestão de credenciais, gestão do próprio Jenkins, entre outros.

Como referido anteriormente, através do painel principal é possível efetuar a criação de novos projetos. Um projeto é uma tarefa configurada pelo desenvolvedor que é executada pelo Jenkins. Existe uma diversidade de projetos que podem ser criados, entre os quais se destacam:

- **Freestyle project** - Projeto totalmente configurável que transfere do repositório declarado o código nele existente, executando a *pipeline* segundo o caminho especificado;
- **Multibranch pipeline** - Projeto que deteta os *branches* existentes num repositório especificado, criando um projeto para cada uma. Este é utilizado para repositórios com vários *branches*.

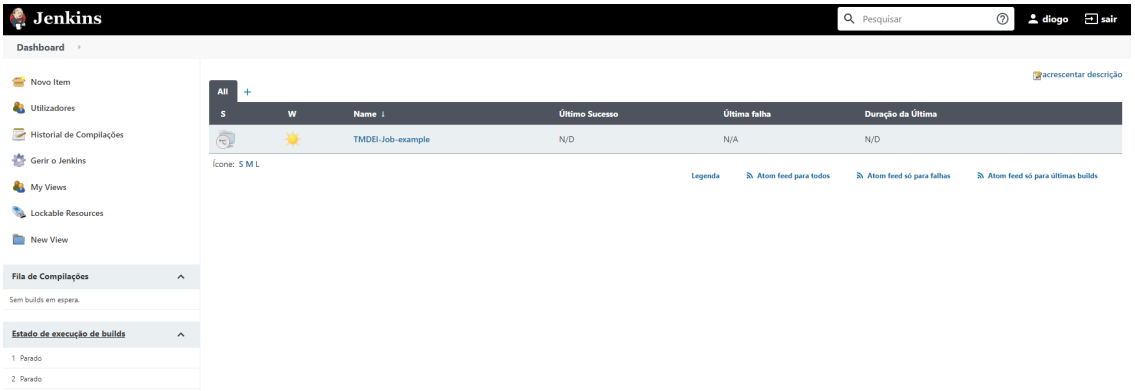


Figura 2.17: Página inicial do Jenkins após a realização de *login*.

Uma *pipeline* representa conjunto de instruções a executar pelo Jenkins (Jenkins 2021b). Atualmente, as *pipelines* podem ser criadas a partir da *interface* do Jenkins ou através de um ficheiro denominado Jenkinsfile, normalmente versionado num repositório (Jenkins 2021b). Com a utilização de Jenkinsfile, é possível efetuar incrementos à *pipeline*, guardando também o seu histórico caso seja necessário utilizar uma versão mais antiga. Como este é um ficheiro versionado, é possível efetuar sua revisão e permite a sua modificação por toda a equipa de desenvolvimento (Jenkins 2021b).

Durante a criação de um Jenkinsfile, é possível seguir uma das duas abordagens existentes: a geração de uma *pipeline* declarativa ou de uma *pipeline scripted*. A principal diferença entre cada abordagem resume-se na sintaxe da escrita. A pipeline declarativa segue as regras da sintaxe Groovy, estando envolvida num bloco *pipeline* (Jenkins 2021c). A utilização desta abordagem permite, ainda assim, a definição de código *scripted* através da utilização do *step script*, como representado no Bloco de código 2.5 (Jenkins 2021c). É possível definir ações após a execução da *pipeline* considerando o resultado final.

```
pipeline {
    agent any
    stages {
        stage('Maven Install') {
            steps {
                sh 'mvn clean install'
            }
        }

        stage('Comando scripted') {
            steps {
                script {
                    def numeroDeExecucoes = 5
                    for (int i = 0; i < numeroDeExecucoes; i++) {
                        echo 'Echo executado dentro de um step script'
                    }
                }
            }
        }
    }

    post {
        always {
            echo 'Executado sempre que a build acaba'
        }
```

```
26          success {
27              echo 'Executado se a build for concluida com sucesso'
28          }
29          failure {
30              echo 'Executado se a build falhar'
31          }
32      }
33  }
```

Bloco de código 2.5: Exemplo de uma *pipeline* declarativa.

Por outro lado, uma *pipeline scripted* utiliza uma DSL construída em Groovy, sendo esta a principal diferença entre as duas abordagens. Um exemplo da sintaxe neste tipo de *pipeline* pode ser visualizado através do Bloco de código 2.6.

```
1  node {
2      stage('Maven Install') {
3          try {
4              sh 'mvn clean install'
5          } catch (exc) {
6              echo 'Ocorreu uma falha'
7          } finally {
8              echo 'Passo sempre executado'
9          }
10     }
11 }
```

Bloco de código 2.6: Exemplo de uma *pipeline scripted*.

Devido às suas funcionalidades, o Jenkins é uma ferramenta que apresenta várias vantagens, entre elas (Katalon 2021):

- Ferramenta *open source*;
- Apresenta uma *interface* simples e intuitiva;
- Disponibiliza um vasto mercado de *plugins*;
- Possibilita o uso para arquiteturas *master-slave*;
- Permite a criação de notificações de estados das execuções;
- Possui mecanismos de autorização.

Esta ferramenta possui, no entanto, uma desvantagem. Devido à simplicidade do uso das *pipelines*, torna-se complicado suportar *pipelines* mais complexas.

2.3.5.2 Jenkins X

Jenkins X é uma ferramenta *open source* introduzida em 2018, com o propósito resolver o problema de automatização dos processos de implantação na *cloud* (Strachan 2018). Esta ferramenta é utilizada num contexto de Kubernetes e permite a criação de imagens de Docker e de recursos Kubernetes automaticamente, através da geração dos ficheiros de configuração necessários. Este processo inclui também a geração de uma *pipeline* de complexidade reduzida comparativamente a uma *pipeline* de Jenkins, possibilitando também a personalização dos ficheiros que são guardados com o projeto num repositório. É possível realizar a configuração de um Docker Registry externo, onde são guardadas as imagens de

Docker (Morgan 2020). Inicialmente foi utilizado o Jenkins como base desta ferramenta, sendo agora utilizada a *framework* Tekton para a execução das *pipelines*, como pode ser observado pela arquitetura apresentada na Figura 2.18 (Morgan 2020).

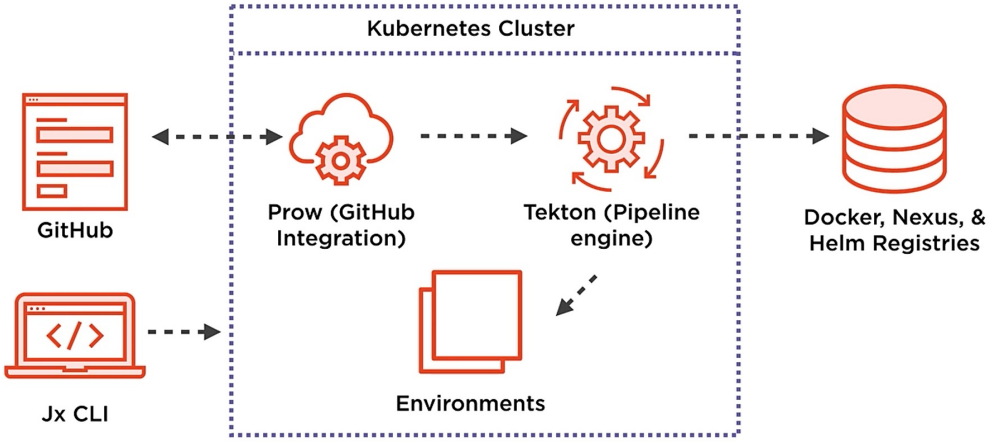


Figura 2.18: Arquitetura do Jenkins X. Fonte: (Morgan 2020)

Esta ferramenta é utilizada através da linha de comandos (jx), onde são realizados todos os processos necessários. É possível criar *templates* para novos projetos numa grande variedade de linguagens, já com Charts para o Helm, *pipelines* e imagem do Docker predefinidos. Esta linha de comandos permite, por exemplo, visualizar o estado de uma execução e importar um projeto, de modo a gerar os ficheiros de configuração que são utilizados para o implantar automaticamente (Jenkins X 2021).

O Jenkins X permite também a execução automática de uma *pipeline* aquando da criação de um *Pull Request*, sem necessidade de utilizar *plugins*, executando testes de forma a assegurar que o novo código se encontra sem falhas e gerando uma pré-visualização num ambiente de teste (Strachan 2018). Depois de *merged*, a nova versão do projeto é gerada e aplicada no *cluster* (Strachan 2018).

2.3.5.3 Travis CI

Travis CI é um serviço de CI/CD baseado na *cloud*, que apresenta uma integração fluída e rápida com o controlo de versões utilizado, sendo atualmente apenas suportado o GitHub e com versões Beta para BitBucket, GitLab e Assembla (Stars 2017; Travis CI 2021b). Estando ligado ao controlo de versões, este serviço deteta automaticamente a existência de ficheiros com o formato YAML e de extensão ".travis.yml", ficheiros esses com os comandos e as configurações necessárias para o projeto ser executado, como exemplificado no Bloco de código 2.7 (Sirota 2019; Stars 2017). Desta forma é inicializada a execução do projeto, permitindo que o mesmo seja testado e implantado (Sirota 2019; Stars 2017).

```
1 languague: python
2 # Versoes a utilizar
3 python:
4   - "2.7"
5   - "3.5"
6 # Comando para instalar as dependencias
7 install:
8   - pip install -r requirements.txt
```

```
9  # Comando para executar testes
10 script:
11   - pytest
```

Bloco de código 2.7: Exemplo de um ficheiro de execução. Fonte: (Travis CI 2021a)

A execução dos comandos pode ser visualizada através da *interface* intuitiva, apresentada na Figura 2.19, ou através da Travis CLI (Travis CI 2020). A Travis CLI é uma ferramenta que permite a utilização da linha de comandos para monitorizar o estado das execuções, verificar possíveis erros no ficheiro de configuração, cancelar e realizar outras ações sobre as execuções (Travis CI 2020).

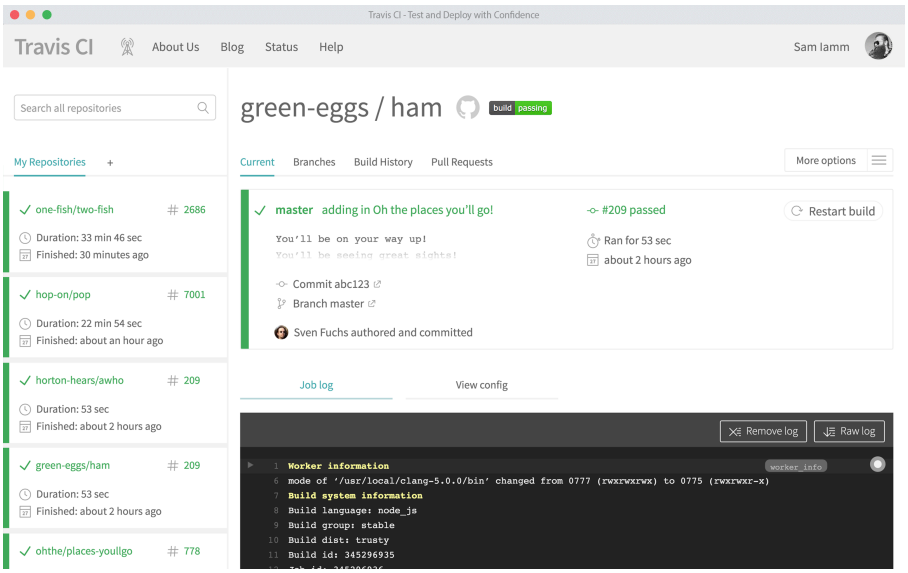


Figura 2.19: Página inicial depois de feito *login* no Travis CI. Fonte: (Travis CI 2021c)

Esta ferramenta tem também uma funcionalidade que a destaca, o Build matrix. Esta funcionalidade permite a execução de comandos equivalentes para diferentes versões da linguagem associada ao projeto, garantindo assim a compatibilidade com várias versões existentes (Sirota 2019; Stars 2017). A utilização deste serviço, oferece um conjunto de outras vantagens às equipas que o utilizem, entre elas (Stars 2017):

- Não existe necessidade de possuir um servidor dedicado para a execução de *builds*;
- Possibilidade de utilização de Build matrix sem configurações prévias;
- Plano gratuito para projetos *open source*, contando já com 900 mil projetos (Travis CI 2021c);
- Possibilidade de notificação do estado da execução.

Ainda assim, este serviço apresenta também algumas desvantagens, incluindo a inexistência de um plano gratuito para empresas e a limitação na personalização das execuções, sendo por vezes necessário recorrer a aplicações externas (Stars 2017).

2.3.5.4 Concourse

Concourse é uma ferramenta *open source* de CI/CD baseada em *pipelines* declarativas (Seroter 2019). Esta ferramenta pode ser implantada através de Docker, Helm Charts no Kubernetes ou em servidores (Seroter 2019). Pode ser utilizada recorrendo à sua API, através da fly CLI, permitindo assim a execução de instruções a partir da linha de comandos, como *login*, execução de *pipelines* e tarefas, verificação do estado das *pipelines*, entre outras ações (Seroter 2019).

Esta ferramenta possui um mecanismo de autenticação e autorização, baseado em equipas. Cada equipa é constituída por múltiplos membros e cada membro possui a sua função (Seroter 2019). Existem 5 funções que podem ser definidas (Concourse CI 2021e):

- **Administrador** - Tem total controlo sobre a ferramenta, desde criação e destruição de equipas à criação e destruição de execuções;
- **Dono da equipa** - Tem total controlo sobre os recursos de uma equipa;
- **Membro da equipa** - Pode realizar alterações às execuções da equipa mas não pode modificar configurações da equipa;
- **Operador da pipeline** - Pode executar *pipelines* mas não pode fazer alterações;
- **Visualizador** - Permite apenas visualizar as execuções das *pipelines*.

Estruturalmente, uma *pipeline* no Concourse é composta por um conjunto de *jobs* e recursos que utilizará para a sua execução (Concourse CI 2021c; Seroter 2019). Os recursos são um componente fundamental da *pipeline*, pois, ao contrário de ferramentas como Jenkins onde é possível usufruir de *plugins*, no Concourse todos os recursos necessários para a execução da *pipeline* são descritos na própria (Seroter 2019).

Os *jobs* são conjuntos de ações a serem executadas na *pipeline*, segundo uma sequência, podendo uma dessas ações ser do tipo Task, como descrito no Bloco de código 2.8 (Concourse CI 2021b). Uma Task é o bloco mais pequeno configurável de uma pipeline. Este bloco é executado isoladamente num contentor, através do comando "fly execute", sendo o mesmo apagado no final da execução da Task (Concourse CI 2021d; Seroter 2019). Tendo isto em consideração, sempre que for necessário utilizar recursos gerados por outras Tasks, estes devem ser incluídos explicitamente como *inputs* (Concourse CI 2021d; Seroter 2019).

```
---
jobs:
  - name: job
    public: true
    plan:
      - task: simple-task
        config:
          platform: linux
          image_resource:
            type: registry-image
            source: { repository: busybox }
          run:
            path: echo
            args: ["Hello, world!"]
```

Bloco de código 2.8: Exemplo de uma *pipeline* com apenas uma Task. Fonte: (Concourse CI 2021a)

Para além das funcionalidades já referenciadas, esta ferramenta também possui algumas desvantagens. Devido à obrigatoriedade de toda a configuração ser feita na pipeline, elas acabam por se tornar bastante complexas e de difícil leitura e entendimento.

Capítulo 3

Análise de valor

Neste capítulo é definida e realizada a análise de valor do projeto que se pretende desenvolver, enunciados os cinco elementos do modelo *New Concept Development* (NCD), definidos os vários conceitos de valor e é também apresentada a proposta de valor recorrendo ao modelo de Proposta de Valor de Osterwalder. Finalmente, é realizada a análise funcional com recurso ao método *Quality Function Deployment* (QFD).

3.1 Definição

Segundo os autores Nick Rich e Matthias Holweg, a análise de valor é um processo sistemático que tem como objetivo oferecer o melhor produto ou serviço ao menor preço possível, conseguindo, simultaneamente, preencher os requisitos dos clientes (Rich e Holweg 2000). A aplicação deste processo pode resultar na adição de uma nova funcionalidade ao produto original, na redução de custos de produção do produto atual através da otimização dos materiais ou dos processos utilizados ou a remoção de funcionalidades que oferecem pouco ou nenhum valor ao produto final (Rich e Holweg 2000).

A análise de valor é um processo que requer várias etapas para, efetivamente, ser executado. Estas etapas passam pela orientação, análise funcional, criação de alternativas, análise e avaliação de alternativas e pela implementação, como esquematizado na Figura 3.1.

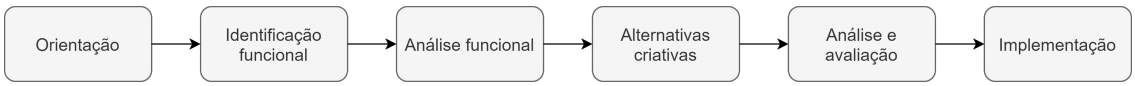


Figura 3.1: Etapas da análise de valor. Fonte: (Rich e Holweg 2000)

A fase de orientação inicia-se com a criação de uma equipa cuja função passa por conduzir o processo de análise de valor. Essa equipa deve ser composta por colaboradores de diferentes ramos do negócio e que tenham funções relevantes para o produto/serviço (Rich e Holweg 2000). Assim que a equipa estiver composta, pode-se dar inicio ao processo de seleção do(s) projeto(s)/serviço(s) para análise (Rich e Holweg 2000).

3.2 Processo de inovação

Depois de selecionado o produto, é necessário perceber quais as funcionalidades (ou incrementos de valor) que podem ser implementadas e perceber quais excluir ou priorizar, para as desenvolver e posteriormente lançar no mercado. Para isso, existe o processo de inovação, o qual pode ser dividido em três partes: *Fuzzy Front End* (FFE), *New Product Development*

(NPD) e a comercialização, como representado na Figura 3.2 (Peter A. Koen, Bertels e E. Kleinschmidt 2014):

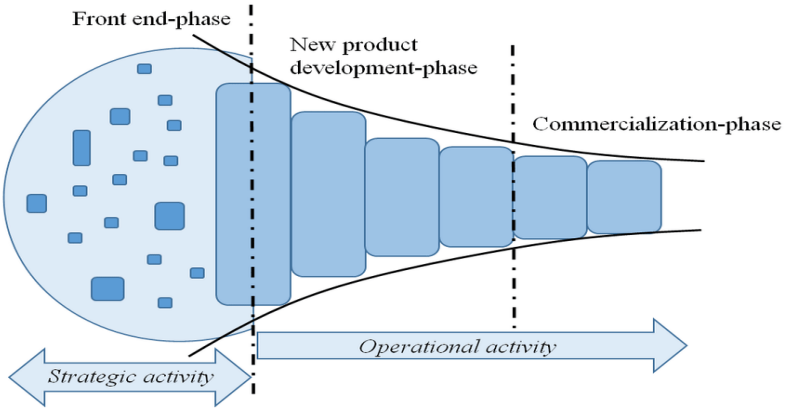


Figura 3.2: Processo de inovação. Fonte: (Martikainen 2017a)

A primeira fase, *Front End*, pode também ela ser dividida em três partes: uma primeira parte, onde são descobertas novas oportunidades, a segunda parte, onde são feitos vários testes rápidos e de baixo custo e onde é visualizado o potencial do projeto e uma terceira parte, onde é construído o caso de uso detalhadamente (Peter A. Koen, Bertels e E. Kleinschmidt 2014). Esta é uma fase muito experimental, sendo que nesta ainda é desconhecida a data de comercialização e ainda existem múltiplas incertezas quanto ao grau de lucro do mesmo (Peter A Koen 2004). No entanto, esta é uma das fases mais importantes deste processo, pois só as funcionalidades escolhidas nesta parte do processo são consideradas no NPD e na comercialização (apesar dos poucos estudos a favor), comparativamente com as restantes fases (Peter A. Koen, Bertels e E. Kleinschmidt 2014).

A segunda fase, NPD, é uma fase já mais disciplinada, com um objetivo em concreto e com uma expectativa sobre a data de lançamento no mercado (Peter A Koen 2004). Esta fase já tem um orçamento associado e é aqui que o incremento ao produto/criação do produto/serviço é realizado.

Por fim, mas não menos importante, a fase da comercialização, onde o novo produto/incremento é disponibilizado para o consumidor.

3.3 New Concept Development

NCD é uma *framework* que veio revolucionar o modo de trabalho da fase de *Front End* do processo de inovação, permitindo definir os componentes essenciais para a realização desta fase numa linguagem comum (Peter A Koen 2004). Esta *framework* divide a fase de *Front End* em 3 partes: o motor, a roda e o aro. Este esquema pode ser inteligível pelo esquema da Figura 3.3 (Peter A. Koen, Bertels e E. Kleinschmidt 2014).

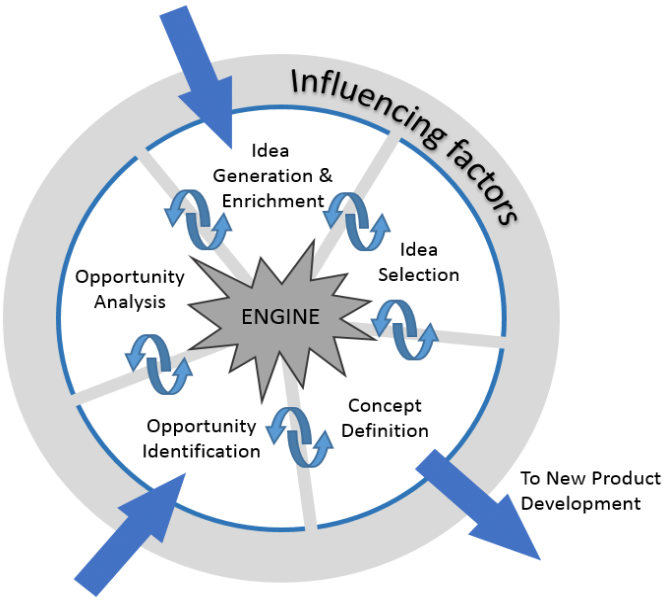


Figura 3.3: Modelo NCD. Fonte: (Martikainen 2017b)

O motor, no centro do modelo, é a fonte de energia para a fase de *Front End* e consiste em dois segmentos: atributos organizacionais e equipas e colaboração (Peter A. Koen, Bertels e E. Kleinschmidt 2014). A roda, na parte interior do modelo, define as 5 atividades chave do *Front End*: identificação da oportunidade, análise da oportunidade, geração de ideias, seleção de ideias e definição do conceito (Peter A. Koen, Bertels e E. Kleinschmidt 2014). O aro, por sua vez, representa os fatores ambientais que influenciam a execução do motor e a forma dos 5 elementos definidos na roda, incluindo a capacidade de organização da empresa, ameaças da competição, modas mundiais e dos consumidores, etc (Rich e Holweg 2000). A forma circular deste modelo dá a perceção que as ideias vão fluindo entre as 5 atividades ao mesmo tempo que estas vão sendo iteradas, de modo a refinar a definição de conceito. Com a utilização deste modelo, a fase de *Front End* acaba por ser mais demorada, tendo como resultado uma definição de conceito mais aprimorada. Normalmente o tempo consumido nesta fase é recuperado e até compensado durante o ciclo de desenvolvimento do produto e comercialização (P A Koen et al. 1996). As atividades iniciais são definidas pelas setas que entram no circulo e, quando o conceito está definido, é iniciada a parte do New Product Development (Peter A. Koen, Bertels e E. Kleinschmidt 2014).

3.3.1 Identificação da oportunidade

Uma oportunidade nasce quando se deteta uma diferença entre o estado do negócio/tecnologia no momento e o que se deseja obter, resultando numa possível vantagem competitiva, numa resolução de um problema ou numa resposta a uma ameaça (Peter A Koen 2004; Peter A. Koen, Bertels e E. J. Kleinschmidt 2014). A atividade de identificação de oportunidades permite elaborar um conjunto de oportunidades das quais a organização pode tirar partido (P A Koen et al. 1996).

Durante o ciclo de vida de um projeto de *software*, são vários os custos associados ao mesmo e, segundo o autor Nexhati Alija, um dos processos mais dispendiosos é a manutenção da solução (Alija 2017). Este termo (manutenção) é referente a todos os pequenos ajustes, correções de falhas e atualizações que têm que ser efetuadas ao longo da vida de um projeto

(Alija 2017). Na Figura 3.4 encontra-se o gráfico circular da distribuição de custos de um projeto de *software*.

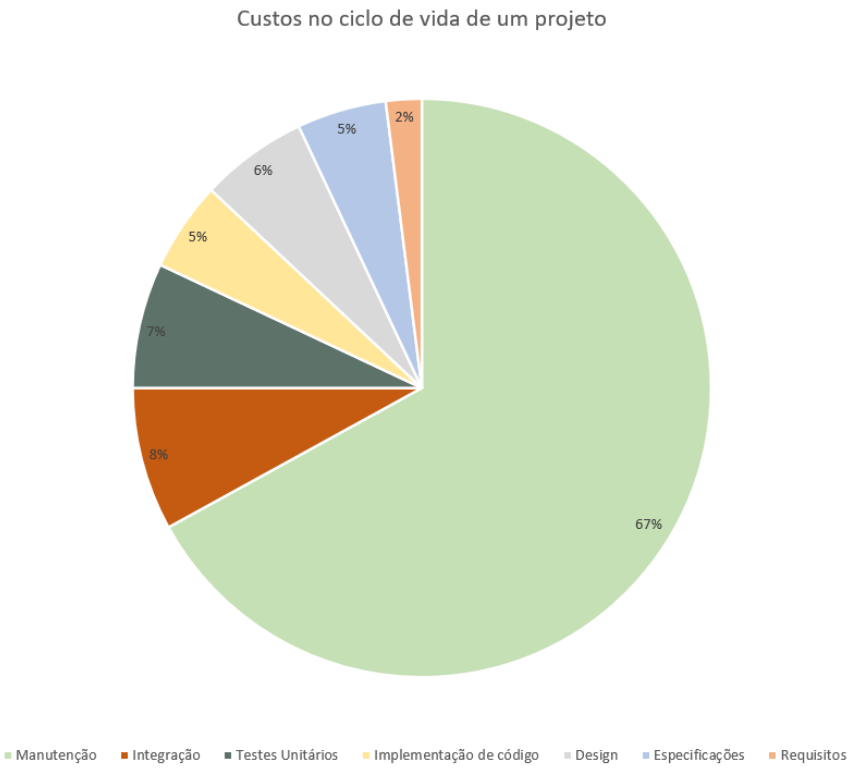


Figura 3.4: Distribuição de custos de um *software*. Fonte: (Alija 2017)

Também outros autores já investigaram sobre o tópico, tendo-se chegado a várias conclusões, devido ao facto dos diferentes autores utilizarem algoritmos de cálculo diferentes. No entanto, todos os valores apresentados possuem uma percentagem de custo de manutenção elevada, como se pode confirmar na Tabela 3.1 (Freitas 2019).

Tabela 3.1: Variação dos custos de manutenção calculados em diferentes artigos tendo em conta o valor total do produto. Fonte: (Freitas 2019)

Autor	% do custo da manutenção
Daniel D. Galorath	75%
Stephen R. Schach	67%
Thomas M. Pigoski	>80%
Robert L. Glass	40% – 80%
Jussi Koskinen	>90%

O processo de manutenção é vital e necessário em todos os projetos, sendo estas as aplicações mais comuns (Alija 2017):

- Correção de falhas técnicas, de desenho ou de execução existentes;
- Manutenção da funcionalidade do *software*;
- Manutenção da execução do *software* em vários ambientes;

- Manutenção da evolução do produto.

Como também já referido na Secção 1.2, quanto maior for o número de etapas manuais na criação de infraestrutura, maior é a probabilidade de ocorrência de erros, o que por sua vez leva a um aumento no custo e tempo gasto na manutenção do projeto.

3.3.2 Análise da oportunidade

Depois de identificada, a oportunidade é analisada de modo a perceber se esta deve ser explorada ou se deve ser excluída, realçando que no futuro existe a possibilidade de voltar a ser analisada. A análise de uma oportunidade ocorre através de iterações, que em grande escala demoram normalmente entre 60 a 90 dias e pode incluir múltiplos processos entre os quais (P A Koen et al. 1996):

- Verificação de compatibilidade entre a oportunidade e o mercado e tecnologia da empresa;
- Análise de moda de tecnologias;
- Descrição detalhada do segmento do mercado;
- Análise da competição;
- Identificação de necessidades do cliente que não estão a ser entregues com os produtos atuais.

Apesar de toda a análise, irão continuar a existir incertezas quanto às assunções realizadas relativamente às tecnologias e ao mercado, sendo essencial iterá-las para reduzir as incertezas ao mínimo possível (P A Koen et al. 1996).

Como referido na Secção 1.5, a empresa e este projeto estão assentes sob uma cultura de DevOps, a qual se foca na automatização e redução de processos manuais. Esta cultura ganhou destaque nos últimos anos já que esta apresenta diversas vantagens, como se comprova pela Figura 3.5, podendo este ser um aspeto importante no que toca à seleção de ideias.

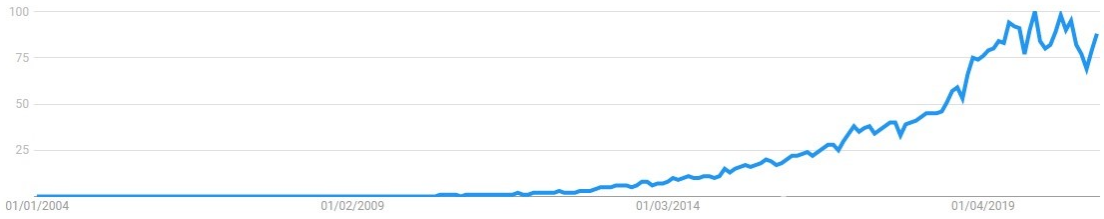


Figura 3.5: Análise do interesse ao longo do tempo da terminologia DevOps.
Fonte: (Google Trends 2021d)

Relativamente ao cliente e às suas necessidades, atualmente sente-se que o processo de criação de *clusters* é demasiado lento, o que obriga a ter por perto um colaborador com mais experiência para que, em caso de falha (ou outra ocorrência), este consiga recriar o *cluster* com maior prontidão. Para além do processo demorar várias horas ou até dias, se a criação for efetuada do zero, torna-se dispendiosa a mão de obra e é mais propício à introdução de erros na infraestrutura devido à intervenção manual.

3.3.3 Geração de ideias

Um atividade que pode também servir como ponto de entrada do processo de inovação é a geração de ideias, uma fase onde são criadas, desenvolvidas e aprimoradas ideias de modo a obter uma ideia mais refinada no final (P A Koen et al. 1996). É um processo iterativo, de forma a que as ideias tenham espaço para amadurecerem, utilizando por vezes informações de outras atividades pertencentes ao NCD (P A Koen et al. 1996). A geração de ideias pode ser feita de diversas formas, incluindo sessões de *brainstorm*, experiências e relações com outras organizações incluindo fornecedores. Por vezes ao gerar ideias, identificam-se novas oportunidades, provando assim a interatividade entre as atividades do NCD (P A Koen et al. 1996).

De modo a criar formas de usufruir desta oportunidade, foram executadas sessões de *brainstorm* com o objetivo de discutir as várias alternativas possíveis, tendo sido geradas as seguintes ideias:

- Sessões de *workshops* e treino para colaboradores que pretendam usufruir de *clusters*;
- Criação de IaC para implantação do *cluster*;
- Documentação de processos efetuados manualmente.

Inicialmente, foi pensado na criação de sessões de *workshop* e treino por parte das equipas que trabalham com os *clusters* de Kubernetes. Esta ideia permite aumentar o conhecimento dos elementos das equipas tornando possível chegar a um nível em que não fosse necessária a ajuda de um colaborador com experiência para efetuar operações de grande impacto no *cluster*, o que eventualmente iria reduzir os custos de manutenção do *software*. Ainda que esta ideia possa reduzir os custos de manutenção, não terá um grande impacto e inicialmente pode até gerar prejuízo comparando com a situação atual, devido à inexperiência dos membros das equipas. Os possíveis erros que podem acontecer sempre que é necessário efetuar alterações ou fazer a criação de um *cluster* continuam a poder existir, tal como *environment drift*, entre outros.

Uma outra ideia seria automatizar um processo que é feito manualmente de momento, a criação dos *clusters* de Kubernetes. Esta ideia passa por utilizar ferramentas de aprovisionamento e gestão de configurações de infraestrutura através de IaC. Esta abordagem, para além de remover a necessidade de auxílio por parte de um colaborador com experiência, provém de uma quantidade de vantagens já referidas nas Secções 2.1.5 e 2.3.1.

Por fim, mas não menos importante, outra ideia identificada foi a possibilidade de documentar os processos efetuados manualmente de modo a reduzir a possibilidade de ocorrência de erros, diminuição do tempo de criação da infraestrutura, resultando na possibilidade de execução desta ações por parte da equipa.

3.3.4 Seleção de ideias

Geradas as ideias, é necessário realizar a seleção das que apresentam maior potencial e valor para a organização, já que nem todas as ideias podem ser executadas. Decidir que ideias podem avançar para uma fase mais avançada é um dos trabalhos mais críticos, tanto para o NCD como para o futuro da organização. No entanto, não existem processos que garantam que a melhor ideia seja a escolhida, sendo normalmente o julgamento individual o ponto de partida (P A Koen et al. 1996).

No processo de tomada de decisão, é utilizado um método de apoio à decisão multicritério. A análise multicritério permite auxiliar a seleção de ideias dentro de um conjunto de alternativas tendo em conta os objetivos a alcançar (Moura Da Silva, Carmen e Belderrain 2005). Este tipo de métodos é normalmente executado através dos seguintes passos (Moura Da Silva, Carmen e Belderrain 2005):

1. Definição das alternativas (já realizada na Secção 3.3.3);
2. Definição dos critérios para escolha das alternativas;
3. Avaliação das alternativas em relação a cada critério;
4. Avaliação da importância relativa de cada critério;
5. Cálculo da melhor alternativa.

Nesta análise em específico, é utilizado o método de *Analytic Hierarchy Process* (AHP) criado por Thomas Saaty (Moura Da Silva, Carmen e Belderrain 2005). Neste método, tal como o próprio nome indica, é estruturada uma relação hierárquica entre o objetivo, os critérios e as alternativas a escolher, de modo a facilitar a sua compreensão. Na Figura 3.6 encontra-se representada a estrutura hierárquica criada para análise.

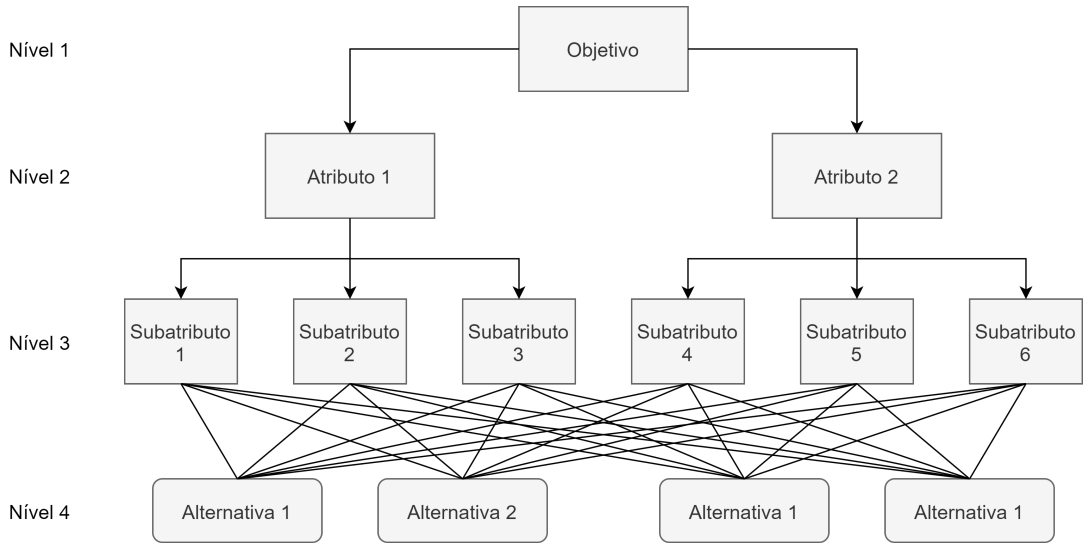


Figura 3.6: Exemplo da estrutura de uma árvore hierárquica criada para análise. Fonte: (Matsumota 2018)

De modo a avaliar cada alternativa, são utilizados os seguintes critérios:

- Tempo de implementação;
- Automatização do processo de implantação do *cluster*;
- Custo da solução.

Com base nesta informação, é possível iniciar o desenvolvimento e análise das alternativas segundo o método selecionado. Inicialmente, é criada a árvore hierárquica a partir dos critérios, objetivo e alternativas, como representado na Figura 3.7.

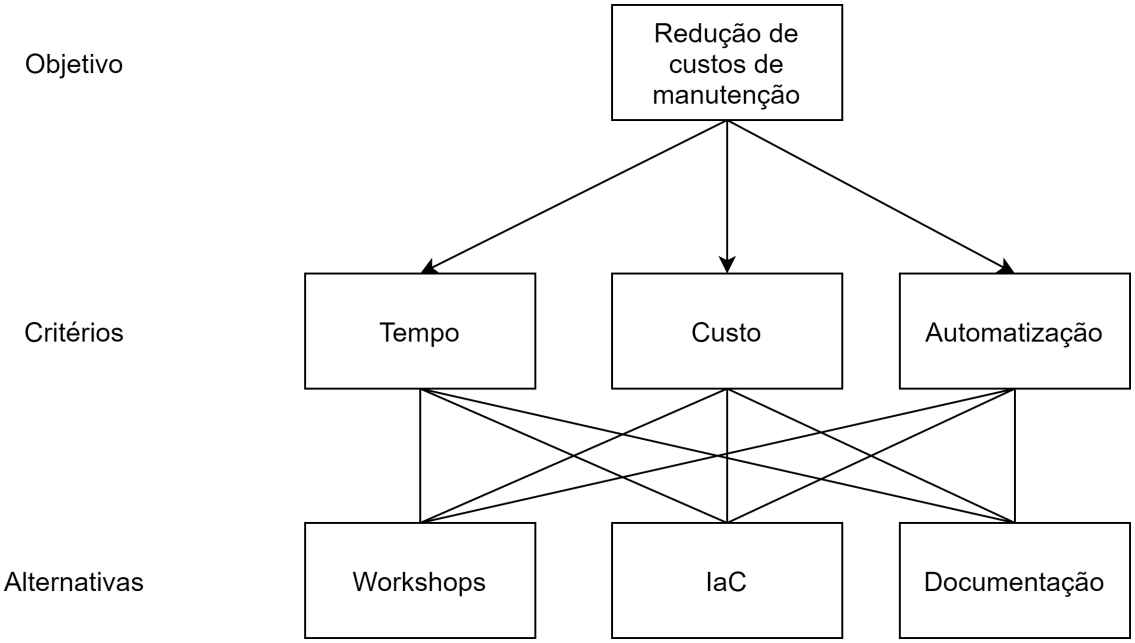


Figura 3.7: Árvore hierárquica criada para análise. Fonte: (Matsumota 2018)

A partir desta hierarquia é possível efetuar a comparação dentro dos diversos níveis da mesma, começando pela comparação dos critérios, seguida da normalização dos valores e por fim, a obtenção do vetor de prioridades. Apesar de não ter sido inserida a matriz normalizada, esta mas foi necessária para se obter o peso de cada critério, podendo ser visualizada através da Tabela 3.2:

Tabela 3.2: Comparação entre critérios e peso de cada critério.

	Tempo	Automatização	Custo	Peso
Tempo	1	1/5	1/2	0,1222
Automatização	5	1	3	0,6479
Custo	2	1/3	1	0,2299

De modo a efetuar a avaliação da consistência dos valores inseridos, como forma de perceber o grau de coerência ou grau de aleatoriedade entre eles, foi calculado a Razão de Consistência (RC). A RC é um valor calculado a partir da equação 3.1, onde o IC representa o Índice de Consistência e o IR o Índice Aleatório. É desejado que os valores RC sejam inferiores a 10%, indicando assim a consistência dos dados inseridos (Andrade Trevizano 2005).

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

O Índice Aleatório é um valor que pode ser retirado da Tabela 3.3 e o Índice de Consistência é calculado através da equação 3.2.

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

Tabela 3.3: Valores para matrizes quadradas de ordem n. Fonte: (Dorneles 2015)

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

Com os cálculos efetuados e com um Índice Aleatório de 0.58 (com n=3 nesta avaliação, pelo facto serem utilizados 3 critérios), foi obtido uma RC de 0.003. Sendo este valor menor que 0.10, é comprovado que os dados fornecidos são consistentes e pode-se prosseguir para a próxima fase, onde são criadas as matrizes das alternativas tendo em conta cada critério. Neste caso, são criadas 3 matrizes em específico, uma para cada critério enunciado (tempo, automatização e custo).

Na Tabela 3.4 encontra-se a comparação entre as alternativas tendo em conta o critério tempo.

Tabela 3.4: Comparação entre as alternativas tendo em conta o critério tempo.

Tempo	*Workshops*	IaC	Documentação	Peso
Workshops	1	5	2	0,5813
IaC	1/5	1	1/3	0,1096
Documentação	1/2	3	1	0,3092

O valor de RC calculado para a matriz de alternativas, tendo em conta o critério tempo, foi de 0.003, o que comprova que estes valores são consistentes, visto ser menor que 0.10.

Na Tabela 3.5 encontra-se a comparação entre as alternativas tendo em conta o critério automatização.

Tabela 3.5: Comparação entre as alternativas tendo em conta o critério automatização.

Automatização	*Workshops*	IaC	Documentação	Peso
Workshops	1	1/9	1/2	0,0790
IaC	9	1	6	0,7775
Documentação	2	1/6	1	0,1435

De modo a fazer a avaliação, foi calculado a RC = 0.007 < 0.10, o que garante a consistência das comparações fornecidas.

Na Tabela 3.6 encontra-se a comparação entre as alternativas tendo em conta o critério custo.

Tabela 3.6: Comparação entre as alternativas tendo em conta o critério custo.

Custo	*Workshops*	IaC	Documentação	Peso
Workshops	1	1/3	1/4	0,1244
IaC	3	1	2/3	0,3586
Documentação	4	3/2	1	0,5171

De modo a fazer a avaliação, foi calculado a RC = 0,001 < 0.10, o que garante consistência nas avaliações realizadas.

O próximo passo é elaborar a tabela com as alternativas e critérios, constituída pelas prioridades globais e pelos pesos de cada critério, como se pode visualizar na Tabela 3.7. De modo a efetuar os cálculos para perceber qual a alternativa com maior valor, apresentado na Tabela 3.8.

Tabela 3.7: Prioridades globais.

	Tempo	Automatização	Custo	Peso de cada critério
Workshops	0,5813	0,0790	0,1244	0,1222
IaC	0,1096	0,7775	0,3586	0,6479
Documentação	0,3092	0,1435	0,5171	0,2299

Tabela 3.8: Resultados de cada alternativa.

	Resultado
Workshops	0,150785262
IaC	**0,599620895**
Documentação	0,249593843

Como pode ser observado, a criação de IaC é a alternativa com maior potencial para a solução, seguida da documentação e por fim os *workshops*. Como tal, é a criação de IaC que é considerada na definição de conceito.

3.3.5 Definição de conceito

A definição de conceito é a última atividade a ser realizada no NCD e resulta num documento onde constam as razões pelas quais a organização devia investir na ideia apresentada (P A Koen et al. 1996). Este documento deve conter diversas informações, tais como: objetivos; a ligação do conceito com a organização; tamanho da oportunidade (impacto financeiro); necessidades dos clientes e do mercado e os seus benefícios; um plano de negócio com a preposição de valor; fatores de risco (ambientais, saúde ou segurança); patrocínios; um plano do projeto para ser executado no NPD (pode variar de empresa para empresa) (P A Koen et al. 1996). Caso o conceito seja rejeitado, o processo volta novamente ao início do NCD, em vez de avançar para o NPD, com o objetivo de melhorar o conceito e podendo, num caso mais extremo, ser descartado (P A Koen et al. 1996).

Tendo em conta a ideia selecionada na Secção 3.3.4, o objetivo é criar IaC de modo a automatizar o processo de implantação de *clusters* de Kubernetes. Primeiramente, é realizado o desenho da solução e análise de alternativas que podem vir a ser utilizadas para o efeito. Só depois desse passo é que é possível realizar uma estimativa do impacto financeiro da abordagem. No entanto, sabe-se à partida que esta alternativa permite reduzir os custos de manutenção, os custos em mão de obra, reduzir o tempo gasto durante a atividade e a possibilidade de criação de erros.

Concluída a fase de decisões e de construção da ideia, são realizados testes à abordagem criada, nomeadamente testes onde é contabilizado o tempo necessário de criação de infraestrutura, é verificada a disponibilidade da solução e também os custos provenientes da solução, de modo a perceber se é uma alternativa viável.

3.4 Valor da solução

Nesta secção são abordadas várias terminologias referentes ao valor da solução. Primeiramente, é realizada a descrição de cada conceito e no final é apresentada a proposta de valor para o projeto descrito neste documento.

3.4.1 Valor do produto

O valor de um produto reflete o desejo que o consumidor/cliente tem para obter o produto/serviço (Neap e Celik 1999). O "nível de desejo" do consumidor varia dependendo de um conjunto de valores que ele próprio associa ao produto ou serviço, sendo o resultado de uma análise entre os benefícios e os sacrifícios (Neap e Celik 1999).

3.4.2 Valor para o cliente

O *Value for the customer* (VC) ou valor para o cliente pode ser visto como o benefício que o cliente obtém pela utilização do produto/serviço e como o que o cliente pode devolver à organização. Este conceito é dividido em 5 categorias (Woodall 2003):

- *Net* VC - Balanço de benefício / sacrifício;
- *Marketing* VC - Atributos do produto;
- *Derived* VC - Experiência de uso;
- *Sale* VC - Valor relacionado com o preço do produto;
- *Rational* VC - Diferença do preço objetivo.

Neste projeto, o valor atribuído para o cliente é uma combinação da automação de ações manuais, sem custo adicional, apenas com o custo de mão de obra, o que facilitará a experiência no caso de necessidade de criação de um novo *cluster*.

3.4.3 Valor percecionado

O valor percecionado é um valor que é medido através da experimentação de cada indivíduo com o produto/serviço, variando consequentemente de indivíduo para indivíduo (Bernardete e Fernandes 2018). Este valor vai variar consoante as expectativas do indivíduo aquando a

compra do produto/serviço e dependendo da finalidade em mente no momento da compra (Bernardete e Fernandes 2018).

Neste contexto, o valor percecionado é definido a cada criação de infraestrutura e pela redução da manutenção da mesma, esperando-se uma redução da pressão por parte do colaborador aquando da execução de alterações da infraestrutura.

3.4.4 Proposta de valor

A proposta de valor descreve em que aspetos o novo produto/serviço se destaca no meio da concorrência, incentivando a compra do mesmo (Lindič e Silva 2011). Esta descrição é sobre o cliente mas é normalmente utilizada internamente, de maneira perceber quais os objetivos do produto/serviço na vida do consumidor.

No caso deste projeto, a proposta de valor passa por facilitar o processo de criação de um *cluster* para o colaborador, libertando-o também da pressão que seria aprovisionar um *cluster* manualmente, dentro de tempo limitado. Foi elaborado o modelo de proposta de valor, inicialmente introduzido por Osterwalder, como pode ser observado na Figura 3.8:

Figura 3.8: Modelo de Proposta de Valor de Osterwalder.

3.5 Análise Funcional

Concluída a fase de orientação, é executada a análise funcional, fase essa onde são analisadas as principais funcionalidades do projeto a desenvolver (Rich e Holweg 2000). Inicialmente são declaradas e descritas as funções da solução, recorrendo a sessões de *brainstorm* e outros exercícios, de modo a identificar o número apropriado de funcionalidades da solução. Depois de enunciadas e agrupadas, as funcionalidades são comparadas segundo um método de comparação (Rich e Holweg 2000). O método utilizado para efetuar a análise funcional é o QFD.

3.5.1 Quality Functional Deployment

O QFD é um método criado com o objetivo de identificar e avaliar os requisitos do cliente para o projeto (Rich e Holweg 2000). A utilização deste método resulta numa visão mais clara sobre as funcionalidades existentes na solução e aquelas que mais importância possuem (Rich e Holweg 2000). Este método pode ser exemplificado através da Figura 3.9.

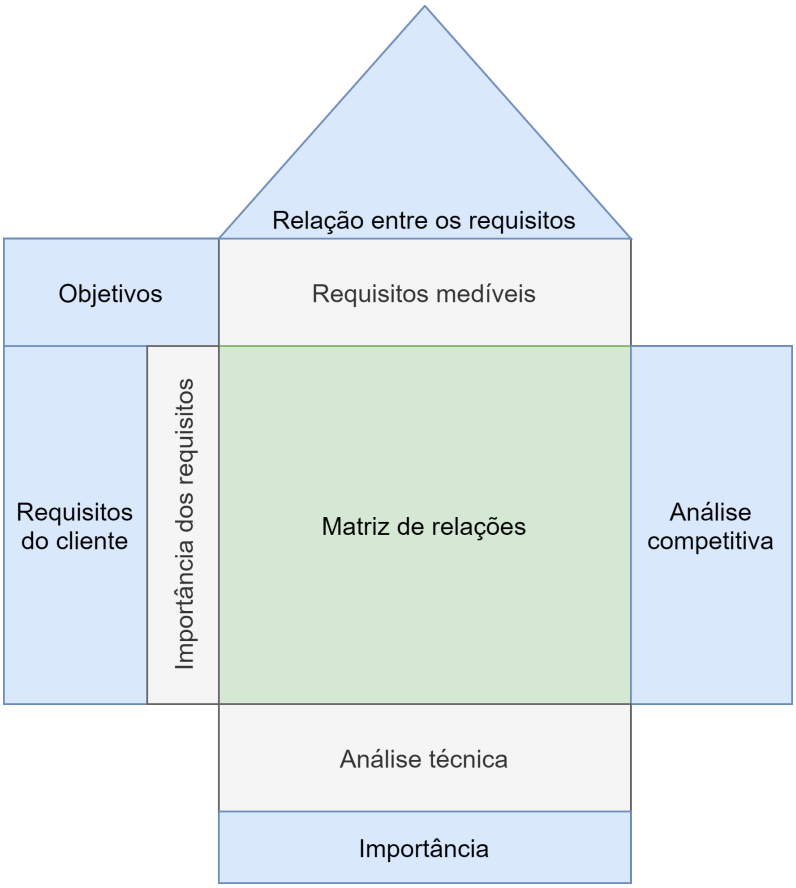
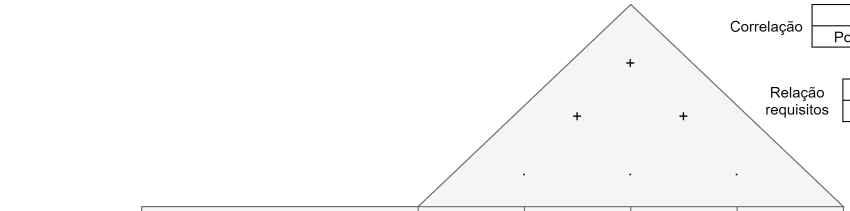


Figura 3.9: Exemplo da estrutura do método QFD. Fonte: (Hallowell 2021)

Através da aplicação do método QFD ao projeto, é obtido o resultado mostrado através da Figura 3.10:



Correlação			
+	.	-	
Positiva	Sem relação	Negativa	

Relação requisitos			
9	3	1	
Forte	Média	Fraca	N/A

1: alto; 6: baixo	Requisitos funcionais →	Automatização da criação de infraestrutura	Gestão de *clusters* através de um BH	Documentação da infraestrutura criada	Mecanismo de autorização	1: baixo; 5: alto		
Importância para o cliente	Requisitos do cliente ↓					*Rating* de satisfação 1	*Rating* de satisfação 2	*Rating* de satisfação 3
1	Rapidez de implantação do *cluster*	9		3		3	2	2
2	Disponibilidade da solução	3		1	1	5	3	3
3	Modularidade de adaptação em vários ambientes	9		1		3	3	3
4	Consistência dos recursos criados	9		1		3	4	4
5	Gestão de infraestrutura por qualquer desenvolvedor	9	9	9	3	4	3	4
6	Teste da infraestrutura criada	3	1	1		4	2	3
	Resultado da importância técnica	141	51	63	17			
	% de importância	52%	19%	23%	6%			
	Rank de prioridades	1	3	2	4			
	Objetivo	Automatiza as etapas de criação de infraestrutura	Cria uma instância que permite gerir o *cluster*	Documenta os recursos criados	Diferenciação de permissões para os diferentes colaboradores			
	Avaliação técnica	3	2	1	1	1: baixo; 5: alto		
	Avaliação técnica de competidor 1	2	3	2	2			
	Avaliação técnica de competidor 2	2	4	3	4			

Figura 3.10: Método QFD aplicado ao projeto.

Com a utilização deste método, pode ser garantida a existência de um requisito funcional com maior destaque em comparação com os restantes: a automatização da criação da infraestrutura. Para além disso, é possível observar a baixa importância da criação do mecanismo de autorização para o projeto.

Capítulo 4

Análise e desenho da solução

Neste capítulo são abordados temas técnicos de modo a desenhar a solução para posterior conceção. Inicialmente são apresentados os requisitos propostos para a solução, seguindo-se de uma exemplificação do processo previsto de implantação do *cluster*. Posteriormente são apresentadas as alternativas para a solução e é feita a comparação das mesmas. Por fim são selecionadas as ferramentas a utilizar na construção da solução e apresentados vários diagramas da solução final.

4.1 Análise do problema

Como referido na Secção 1.2, de momento existe a necessidade de automatizar o processo de implantação da solução existente no *Cloud Service Provider* AWS, representada no gráfico da Figura 2.6.

Contudo, para automatizar a implantação do BH e do *cluster*, é necessário inicialmente criar as bases de *network* para que a solução possa ser executada de forma segura e estar simultaneamente ligada à *internet*. Em termos de poder computacional, é criado o *cluster* e, para efetuar a manutenção dos artefactos no mesmo, é criado um BH.

O BH consiste numa máquina (Amazon EC2, no caso de utilização da AWS) com uma imagem de Linux nela instalada. Esta máquina atua como intermediário entre os colaboradores e o *cluster* de Kubernetes, permitindo a execução de operações sob o *cluster* e a sua gestão, diminuindo simultaneamente os pontos de vulnerabilidade no acesso ao mesmo. Sabendo que esta máquina é utilizada por um baixo número de colaboradores, não existe necessidade da mesma possuir elevada capacidade computacional, sendo para isso utilizado apenas uma instância do tipo t2.micro. A AWS garante um *Service Level Agreement* (SLA) de 99.99% para instâncias Amazon EC2 por região, assegurando a existência de alta disponibilidade (AWS 2021l). Este SLA permite prever, no máximo, *downtime* durante 52 minutos e 36 segundos por ano (um número bastante grande, que no entanto equivale a 8.65 segundos por dia).

Durante o processo de criação do BH, é necessário também efetuar a criação de um *Security Group* (SG) para o mesmo. O SG, definido no BH, determina se é possível ou não estabelecer conexão com o mesmo, agindo como uma *firewall* virtual, controlando assim o tráfego de entrada e saída nos recursos onde está associado (pode estar vinculado a vários recursos), como demonstrado na Figura 4.1 (AWS 2021p).

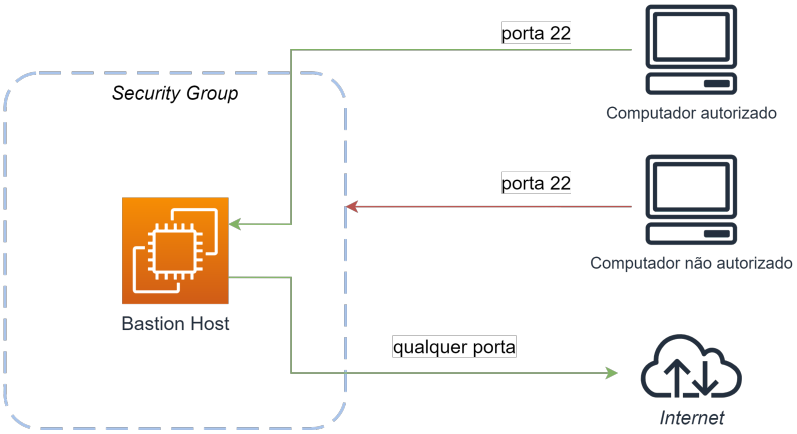


Figura 4.1: Funcionalidade de um SG. Fonte: (Dashora 2019)

Neste caso em particular, o SG vai permitir a conexão de máquinas da rede interna do BMW Group ao BH. Assim, são garantidas as permissões necessárias para os colaboradores usufruírem da máquina.

O BH encontra-se implantado num variado número de regiões, incluindo EMEA, CN e USA, distribuído em todos os ambientes disponíveis atualmente (teste, integração, *end-to-end* e produção) e tanto para *clusters* de aplicações como de operações.

Para além disto, o BH necessita de estar conectado ao respetivo *cluster* e de dispor de várias ferramentas que permitam a gestão do mesmo, como Kubectl, Helm, Git, entre outros.

Outro lado fundamental da solução são os *clusters* existentes. No contexto onde se encontram inseridos, estes recursos possuem caraterísticas mistas entre um *cluster* de alta disponibilidade e um *cluster* de balanceamento de carga. A alta disponibilidade é garantida pela utilização do serviço EKS, oferecendo um SLA de 99.95% (AWS 2020a). À semelhança dos BHs, os *clusters* também estão presentes em várias regiões e ambientes, sendo que a sua função pode passar por gerir aplicações ou, por outro lado, gerir ferramentas. São várias as configurações que necessitam de ser alteradas dependendo da função do *cluster*, como por exemplo: *drivers* de armazenamentos, variáveis de ambiente, acesso à *internet*, etc.

Cada *worker node* pertencente ao *cluster* tem associado um SG, como forma de reforçar a sua segurança. O SG é constituído por duas regras de ingresso: uma delas para permitir a conexão por SSH de máquinas inseridas na VPC e outra para permitir o acesso e conexão entre os vários serviços existentes no *cluster*. Atualmente os nós são geridos manualmente, tornando a atualização e manutenção do *cluster* uma tarefa complexa e demorada. Esta atividade acaba por estar associada múltiplas vezes a *downtime* nas aplicações existentes no respetivo *cluster*, derivado da quantidade de passos manuais e da possibilidade de ocorrência de erros durante a mesma.

Um dos tópicos de maior importância relativo ao *cluster* é relacionado com a carga a que este é sujeito no dia-a-dia. Atualmente, em ambientes de produção o número de pedidos aos serviços inseridos nos *clusters* são elevados, atingindo, durante os picos, valores próximos de mil pedidos por segundo, como pode ser observado através da Figura 4.2.

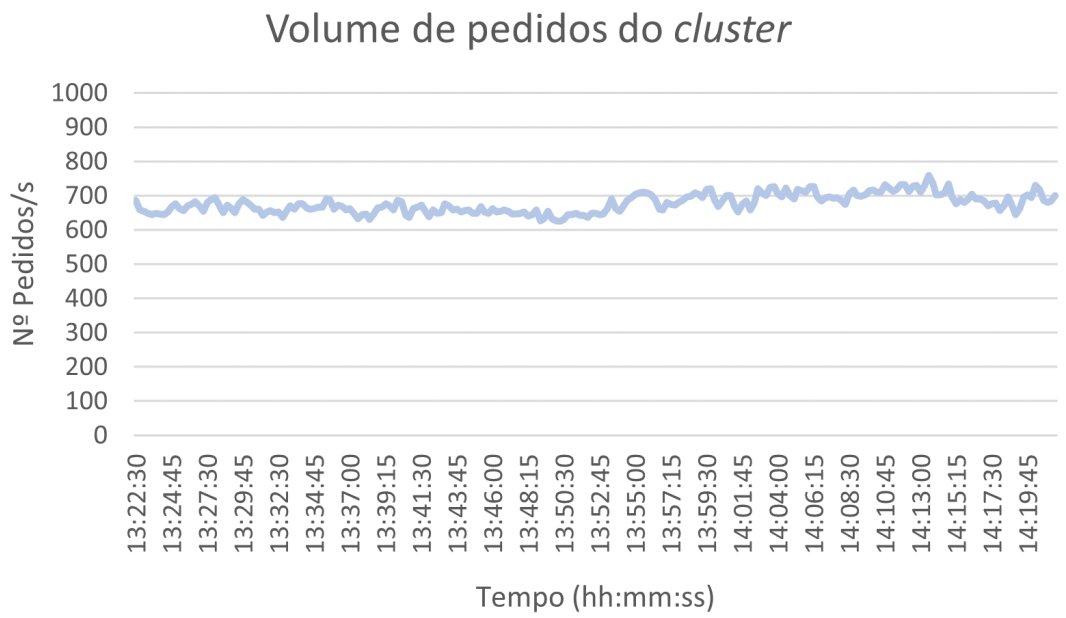


Figura 4.2: Gráfico de pedidos por segundo em um dos *clusters* existentes, retirado da ferramenta Grafana.

4.1.1 Modelo de Domínio

Para melhor explicar a relação entre os vários conceitos do problema, foi elaborado um modelo de domínio como pode ser visualizado na Figura 4.3:

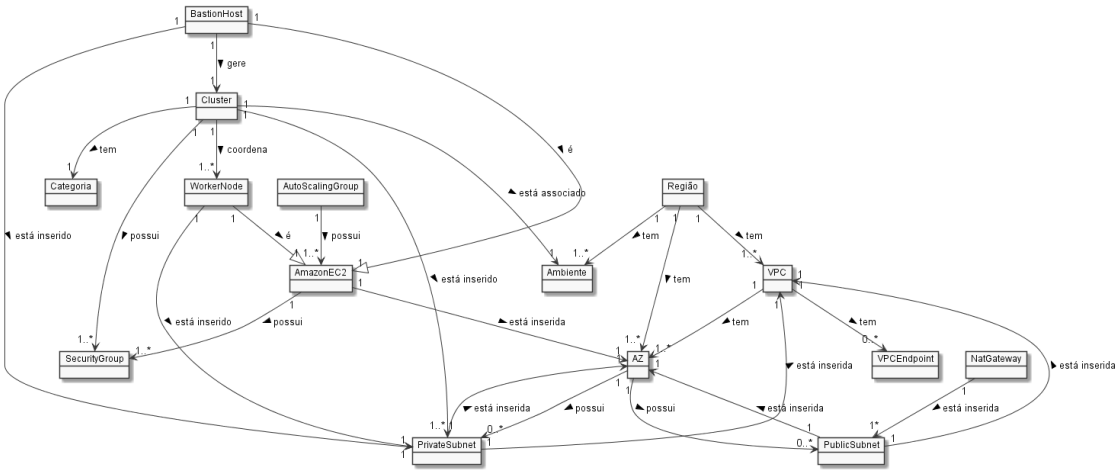


Figura 4.3: Modelo de domínio da solução.

Neste diagrama podem ser visualizados alguns conceitos ainda pouco abordados ao longo do documento. Um desses conceitos é o ambiente do *cluster*. O ambiente está diretamente associado à fase das aplicações que estão a ser executadas no mesmo. Cada região possui múltiplos ambientes, entre eles:

- Teste - Ambiente onde as aplicações são testadas unitariamente e funcionalmente, sendo por vezes necessário recorrer a *mocks* para emular respostas de sistemas externos.
- Integração - Ambiente onde é testada a integração entre várias ferramentas (como o exemplo da conexão entre o *backend* e *frontend*).
- *End-to-end* - Ambiente onde são inseridas e testadas as aplicações de ponta a ponta, sendo para isso utilizados carros de teste e *racks*.
- Produção - Ambiente onde as aplicações são executadas com conexão com os carros de utilizadores reais.

Para além dos vários ambientes onde os *clusters* vão estar inseridos, cada um também terá uma função predefinida. Atualmente, é possível identificar os *clusters* segundo duas categorias: operações e aplicações. Os *clusters* de aplicações suportam as soluções que recebem pedidos diretamente dos veículos e as ferramentas para as monitorizar. Os *clusters* de operações permitem a implantação de ferramentas destinadas a recolher informação sobre as soluções que estão a ser executadas no *cluster* de aplicações, de modo a apresentá-las aos colaboradores através de gráficos e de forma mais legível.

4.1.2 Processo de implantação da solução

De modo a clarificar as etapas necessárias para a implantação da solução a partir do zero, foi elaborado um diagrama de processos, como esquematizado na Figura 4.4:

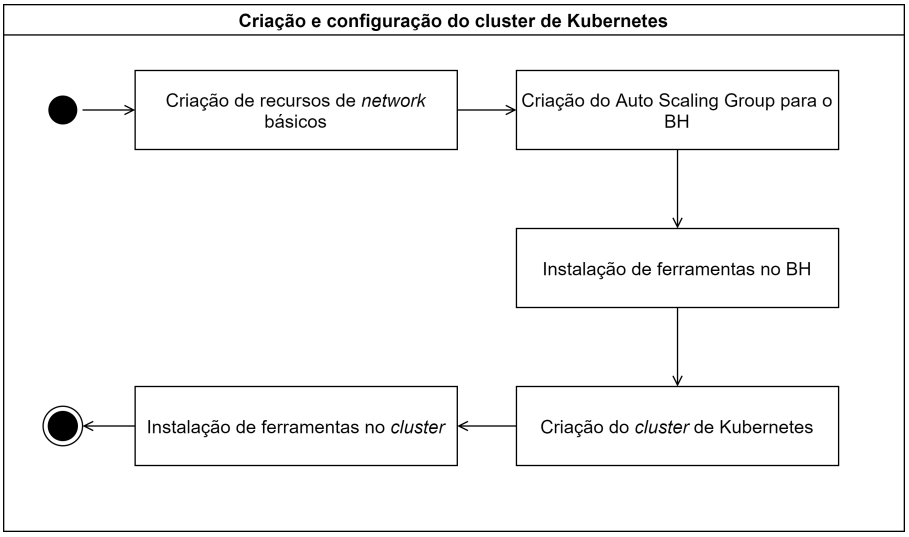


Figura 4.4: Processo de implantação do *cluster*.

O processo começa com a etapa de geração dos recursos de *network* básicos, uma vez que sem esses componentes é impossível efetuar qualquer tipo de implantação, seja do BH ou do *cluster*, porque como identificado na Figura 4.3, tanto o BH como o EKS estão inseridos dentro de uma *private subnet*. De seguida, é necessário efetuar a criação do BH para que aquando da criação do *cluster*, a máquina que o gere já esteja disponível, de modo a efetuar a sua ligação automaticamente.

O objetivo principal é que este processo possa ser executado na totalidade por qualquer colaborador, mesmo com pouco conhecimento sobre processos de DevOps, de modo a eliminar a dependência de pessoas externas ou de uma equipa especializada em operações para efetuar a tarefa.

4.1.3 Criação dos recursos de network básicos

Esta é a primeira fase para a construção da solução é durante esta fase que são criados recursos de *network* onde são inseridas as máquinas Amazon EC2 e outros recursos. Nesta parte são criados componentes como SGs, *Network Load Balancer* (NLB), NAT Gateways, *route tables*, VPC *endpoints* e *subnets*.

4.1.4 Criação do ASG para o BH

Nesta fase é efetuado o aprovisionamento da infraestrutura para criar o BH. É efetuada a construção da máquina com a imagem definida na *subnet* privada.

4.1.5 Instalação de ferramentas no BH

Nesta parte são instaladas todas as ferramentas necessárias no BH para que este consiga gerir o *cluster* de Kubernetes.

4.1.6 Criação do cluster de Kubernetes

Nesta etapa é realizado o aprovisionamento do *cluster* de Kubernetes. São também configurados os *drivers* e dependências necessárias de modo a obter um *cluster* pronto a utilizar. Esta etapa necessita de ser efetuada depois da criação do BH devido à utilização do mesmo para criar o *cluster* e para se conectarem automaticamente.

4.1.7 Instalação de ferramentas no cluster de Kubernetes

Por último, esta é a fase onde são instaladas ferramentas ou aplicações no *cluster* de modo a usufruir das vantagens que este disponibiliza.

4.2 Especificação de requisitos

É necessário, como ponto de partida para o desenho da solução, apresentar os requisitos a cumprir com a construção da mesma, de modo a alinhar as expectativas do cliente com o produto final. Nesta secção são enunciados os requisitos funcionais, não funcionais e restrições através do modelo FURPS+.

O levantamento destes requisitos foi efetuado pelo meio de múltiplas reuniões decorridas ao longo de várias semanas. Ao longo das várias reuniões, de duração variável, estiveram presentes vários *stakeholders*, entre eles: *Scrum Master*, *Tech Lead* e *Product Owner*.

4.2.1 Modelo FURPS+

Existem vários sistemas de classificação de requisitos, entre eles o modelo FURPS. Este modelo baseia-se no seu acrónimo para definir os requisitos, tanto funcionais como não funcionais, e tem como significado a funcionalidade, usabilidade, confiabilidade (*reliability*),

performance e suportabilidade da solução (Eeles 2001; Eeles e Cripps 2009). Da evolução deste modelo surge o FURPS+, adicionando ao modelo inicial tópicos sobre restrições de desenho, de implementação, de *interface* e físicas (Eeles 2001; Eeles e Cripps 2009).

4.2.2 Requisitos funcionais

Requisitos funcionais descrevem os comportamentos e funcionalidades que o produto deve possuir (Eeles 2001). Na idealização este projeto foram delimitados os seguintes:

4.2.2.1 F01 - Automatização do processo de criação infraestrutura

Como engenheiro de DevOps, pretendo automatizar a criação da infraestrutura, de modo a permitir a sua construção por qualquer colaborador, diminuindo também o tempo de construção e os erros introduzidos na mesma.

4.2.2.2 F02 - Gestão de clusters através de um BH

Como engenheiro de DevOps, pretendo gerir o *cluster* de Kubernetes através de uma máquina adjacente, podendo assim efetuar as modificações necessárias no *cluster*, sem comprometer a segurança da solução.

4.2.2.3 F03 - Documentação da infraestrutura

Como engenheiro de DevOps, pretendo que seja gerada documentação para a infraestrutura criada, de modo a descrever a informação necessária sobre os recursos. Assim, torna-se possível a partilha de conhecimento sobre a infraestrutura criada com os vários colaboradores do projeto e facilita a compreensão da mesma.

4.2.2.4 F04 - Mecanismo de autorização

Como *Product Owner*, pretendo implementar um mecanismo de autorização, de modo a aplicar diferentes níveis de permissões aos diversos colaboradores.

4.2.3 Requisitos de usabilidade

Requisitos de usabilidade descrevem caraterísticas relativas à estética e facilidade de uso da solução (Eeles 2001).

4.2.3.1 NF01 - Criação de infraestrutura por qualquer colaborador

Como *Product Owner*, pretendo que a infraestrutura possa ser criada por qualquer colaborador, eliminando a dependência de pessoas externas para efetuar este procedimento.

4.2.4 Requisitos de confiabilidade

Requisitos de confiabilidade descrevem caraterísticas da solução relacionadas com a disponibilidade e tempo de recuperação em caso de desastre (Eeles 2001).

4.2.4.1 NF02 - A solução criada deve possuir 99.9% de disponibilidade

Como *Product Owner*, pretendo que a solução gerada tenha alta disponibilidade, de modo a hospedar aplicações em ambientes de produção sem ocorrência de falhas.

4.2.4.2 NF03 - A solução criada deve reduzir em pelo menos 50% o tempo de recuperação do cluster em caso de desastre

Como engenheiro de DevOps, pretendo que haja uma redução substancial do tempo de recuperação do *cluster* em caso de desastre face ao cenário atual, de modo a limitar o tempo de indisponibilidade dos serviços.

4.2.4.3 NF04 - Construção consistente de infraestrutura

Como engenheiro de DevOps, pretendo que a infraestrutura seja criada de forma consistente, de modo a originar sempre a mesma infraestrutura.

4.2.5 Requisitos de performance

Requisitos de performance descrevem caraterísticas relativas ao tempo de resposta, tempo de inicialização e tempo de *shutdown* da solução (Eeles 2001).

4.2.5.1 NF05 - A solução criada não deve deteriorar o tempo de resposta considerando os valores praticados atualmente

Como *Product Owner*, pretendo que a infraestrutura produzida não deteriore o tempo de resposta comparativamente com a infraestrutura existente de momento, de modo a executar as aplicações com sucesso.

4.2.6 Requisitos de suportabilidade

Requisitos de suportabilidade descrevem caraterísticas relativas à testabilidade, manutenção, compatibilidade, configurabilidade, instalabilidade, escalabilidade e localização da solução (Eeles 2001).

4.2.6.1 NF06 - Escalamento automático do número de nós do cluster

Como engenheiro de DevOps, pretendo que seja acessível escalar a solução, permitindo adicionar ou remover *worker nodes* sempre que necessário. Com esta abordagem torna-se possível ajustar os recursos da solução em horas de pico e em alturas de menor tráfego.

4.2.6.2 NF07 - Testes funcionais

Como engenheiro de DevOps, pretendo que sejam criados testes funcionais para a solução, de modo a garantir que a solução é construída corretamente e se encontra operacional.

4.2.6.3 NF08 - Facilidade de atualização das versões de Kubernetes

Como engenheiro de DevOps, pretendo que a atualização da versão de Kubernetes seja um processo simples, de modo a diminuir o tempo gasto no processo e permitir a realização do mesmo por qualquer colaborador.

4.2.6.4 NF09 - Facilidade de atualização das versões das ferramentas instaladas no BH

Como engenheiro de DevOps, pretendo que a atualização da versão de ferramentas instaladas no BH seja um processo simples, de modo a diminuir o tempo gasto no processo e permitir a realização do mesmo por qualquer colaborador.

4.2.6.5 NF10 - Solução modular adaptar de modo a ser utilizada em várias regiões e ambientes

Como engenheiro de DevOps, pretendo que a solução seja modular, de modo a possibilitar a reutilização da mesma infraestrutura em diversas regiões e ambientes, garantindo a criação dos recursos necessários.

4.2.6.6 NF11 - Redução de custos de manutenção

Como *Product Owner*, pretendo que sejam reduzidos os custos de manutenção da infraestrutura face à solução atual.

4.2.7 Restrições de desenho

Restrições de desenho especificam imposições que se devem ter em conta aquando do desenho da solução (Eeles 2001).

4.2.7.1 NF12 - Infraestrutura criada no Cloud Service Provider AWS

Como *Product Owner*, pretendo que a infraestrutura seja criada no *Cloud Service Provider* AWS, de modo a replicar a existente.

4.2.7.2 NF13 - Utilização da ferramenta EKS

Como engenheiro de DevOps, pretendo que o *cluster* utilize a ferramenta EKS, à semelhança dos existentes atualmente, de modo a facilitar a gestão dos nós do *cluster*.

4.2.8 Restrições de implementação

Restrições de implementação especificam imposições na codificação e construção da solução (padrões, tecnologias, etc) (Eeles 2001).

4.2.8.1 NF14 - Definição da infraestrutura em código

Como engenheiro de DevOps, pretendo que a infraestrutura seja declarada como código, de modo a permitir a consistência da infraestrutura criada e também de modo a possibilitar a criação por parte de qualquer colaborador.

4.2.9 Restrições de interface

Restrições de interface especificam imposições nas interações com sistemas externos (Eeles 2001).

Não foram definidas restrições de interface.

4.2.10 Restrições físicas

Restrições físicas especificam imposições no *hardware* utilizado pela solução (Eeles 2001).

Não foram criadas restrições físicas devido à infraestrutura ser criada num *Cloud Service Provider*. Da mesma forma, não foram definidos requisitos relativos à máquina que executa os *scripts* de criação e gestão da infraestrutura.

4.3 Análise das alternativas

Nesta secção são apresentadas as alternativas encontradas que permitem efetuar o aprovisionamento e gestão da infraestrutura pretendida. Finalmente, é realizada uma comparação entre as mesmas.

4.3.1 Utilização de uma ferramenta multifuncional

Como pode ser verificado na análise e pesquisa sobre trabalhos relacionados com o tema na Secção 2.2, é possível utilizar apenas uma ferramenta (seja de aprovisionamento ou de gestão de configurações) para obter uma solução que colmate o problema.

Como pode ser visualizado na Figura 4.5, podemos comprovar que as ferramentas de gestão de configurações apresentadas na Secção 2.3.2, permitem também de certo modo aprovisionar infraestrutura. Por outro lado, como referido anteriormente, ferramentas de aprovisionamento também permitem a instalação de ferramentas no *cluster* através da execução de comandos aquando da sua criação.

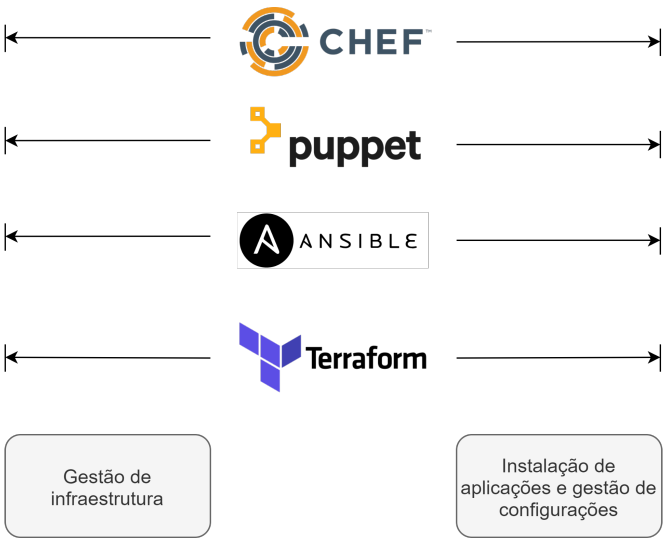


Figura 4.5: Funções das várias ferramentas. Fonte: (Co 2020)

Esta solução permite aumentar a velocidade de desenvolvimento porque as configurações permanecem todas juntas, sendo apenas necessário configurar e aprender como utilizar uma ferramenta.

O ponto negativo desta abordagem passa pela utilização de uma ferramenta para dois trabalhos distintos, quando poderia ser utilizada a melhor ferramenta para cada um dos trabalhos

a ser realizado (aprovisionamento e gestão de configurações). Esta alternativa diminui também a qualidade geral da solução e dificulta a leitura e compreensão do código criado.

4.3.2 Utilização de ferramentas dedicadas

Como a solução inclui tanto fases de aprovisionamento de infraestrutura como de configuração dos sistemas existentes, outra abordagem possível seria a utilização de duas ferramentas, cada uma apropriada para cada fase.

Esta abordagem permite a utilização de cada ferramenta para o seu propósito, facilitando assim a separação de funções e contextos. Com esta abordagem seria utilizado uma ferramenta de aprovisionamento para criar o BH e posteriormente o *cluster* de Kubernetes. Por outro lado teria a utilização de uma ferramenta de gestão de configurações para efetuar a instalação das ferramentas necessárias no BH e posteriormente, se necessário, no *cluster* de Kubernetes.

4.3.3 Comparação das alternativas

Considerando os pontos apresentados nas secções anteriores, foi elaborada a Tabela 4.1 para comparação das duas alternativas, com as caraterísticas ordenadas por prioridade descendente. As caraterísticas utilizadas para efetuar a comparação das abordagens foram selecionadas numa reunião com o *Scrum Master*.

Tabela 4.1: Comparação entre as duas alternativas.

	Utilização de 1 ferramenta	Utilização de 2 ferramentas
Qualidade da solução	-	+
Rapidez de desenvolvimento	+	-
Separação de funções	-	+
Curva de aprendizagem	+	-

Analisando a tabela, é percetível a utilização de duas ferramentas resulta numa solução de maior qualidade e com maior separação de funções. Esta abordagem possui, no entanto, uma curva de aprendizagem mais elevada e consecutivamente um processo de desenvolvimento da solução mais demorado. Apesar dos pontos negativos, considera-se a melhor abordagem a seguir devido à qualidade da solução final.

4.4 Seleção das ferramentas

Nesta secção é realizada a comparação entre as várias ferramentas possíveis de realizar as etapas necessárias considerando a alternativa escolhida.

4.4.1 Ferramenta de aprovisionamento de infraestrutura

Considerando o diagrama criado na Secção 4.1.2, inicialmente é necessário realizar o aprovisionamento de recursos de *network* básicos como VPCs e *subnets* e posteriormente do ASG para o BH. Como ferramentas de aprovisionamento, foram referidas apenas duas na Secção 2.3.1: AWS CloudFormation e Terraform. A ferramenta selecionada, é também

utilizada para fazer o aprovisionamento dos *clusters* de Kubernetes. De modo a selecionar a melhor ferramenta a utilizar, foi realizado uma comparação entre as duas na Tabela 4.2 (Ethridge 2021; Terraform 2021c):

Tabela 4.2: Comparação entre as ferramentas de aprovisionamento.

	Terraform	AWS CloudFormation
Agnóstica a *Cloud Service Provider*	+	-
Combinação de múltiplos *Cloud Service Providers*	+	-
Definição da infraestrutura	modo declarativo	modo declarativo
Número de ferramentas de teste existentes	+	-
Reutilização de recursos	+	+
Linguagem de configuração	DSL (HSL)	YAML/JSON

Foi também realizada uma análise em termos de interesse das ferramentas ao longo do tempo, apresentada na Figura 4.6:

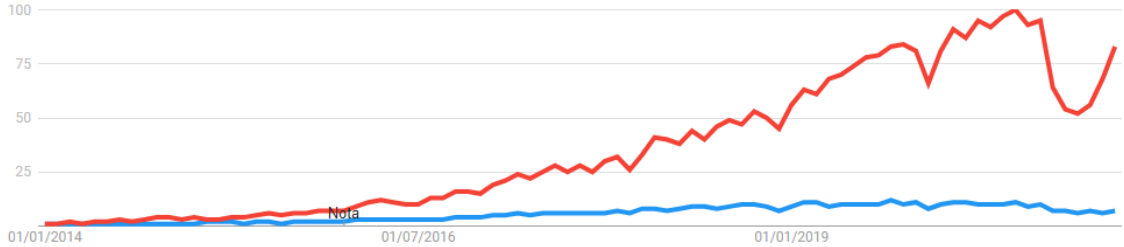


Figura 4.6: Interesse ao longo do tempo dos termos Terraform (vermelho) e AWS CloudFormation (azul). Fonte: (Google Trends 2021b)

Analisando a Tabela 4.2 e a Figura 4.6, conclui-se que a ferramenta Terraform tem observado um grande crescimento ao longo dos anos e, a longo prazo, pode ser uma opção viável devido a ser agnóstica a *Cloud Service Provider*. Esta caraterística é de extrema importância devido à constante inovação existente no meio atual e à possibilidade de serem utilizados outros *Cloud Service Providers* no futuro, derivado do surgimento de novos projetos. Assim, pode-se concluir que a utilização da ferramenta Terraform é a abordagem mais vantajosa.

4.4.1.1 Ferramenta para armazenamento do estado da infraestrutura

A utilização de ferramentas de aprovisionamento de infraestrutura como Terraform, requerem uma estratégia de armazenamento do estado dos recursos aquando a sua criação ou modificação. Para tal, existem três abordagens que permitem o seu armazenamento (Yevgenjy Brikman 2019):

- Armazenamento local - É a abordagem mais simples das três mencionadas, funcionando perfeitamente para um projeto pessoal. No entanto, seguir esta aproximação num projeto em equipa leva a que nunca existe um alinhamento entre os vários ficheiros de estado espalhados pelos múltiplos computadores.
- Armazenamento num repositório de controlo de versões - É uma abordagem bastante utilizada pela comunidade, permitindo o acesso e modificação do ficheiro sempre que

necessário, para além da centralização da informação. Esta perspetiva é, no entanto, propicia a erros manuais derivados da possibilidade de não utilização do ficheiro com as últimas atualizações e não permite a execução de comandos simultâneamente em janelas de tempo curtas (Yevgenjy Brikman 2019).

- Armazenamento na *cloud* - Através da configuração de um *remote backend*, é possível efetuar a gravação automática do ficheiro de estado remotamente. Esta funcionalidade permite agilizar o processo de edição e armazenamento do ficheiro, não necessitando de manutenção.

Considerando o requisito de utilização do *Cloud Service Provider* AWS, é utilizado o serviço S3 *Bucket* para guardar o ficheiro de estado da solução, dispondo assim de 99.999999999% de durabilidade dos objetos e 99.99% disponibilidade, minimizando o risco de perca de dados ou de *outages* (AWS 2021c; Yevgenjy Brikman 2019).

4.4.2 Ferramenta de gestão de configurações

Depois de aprovisionado, o BH deve possuir inicialmente ferramentas como Helm, Git, Tar, Kubectl e outras necessárias. Para esse efeito, é utilizada uma ferramenta de gestão de configurações. De modo a realizar a comparação entre as ferramentas já apresentadas, foi elaborada a Tabela 4.3 com os vários aspetos das mesmas (Johari 2019; Srivastava 2021; Veritis 2021):

Tabela 4.3: Comparação entre ferramentas de gestão de configurações.

	Chef	Puppet	Ansible
Facilidade de instalação	-	-	+
Facilidade de gestão	-	-	+
Disponibilidade da solução	+	+	+
Mecanismo	*Pull*	*Pull*	*Pull* e *Push*
Linguagem de configuração	DSL (Ruby)	DSL (PuppetDSL)	YAML
Arquitetura	*Master-agent*	*Master-agent*	*Master (agentless)*

Foi também elaborado, à semelhança das ferramentas de aprovisionamento, um gráfico de interesse das várias ferramentas, representado pela Figura 4.7:

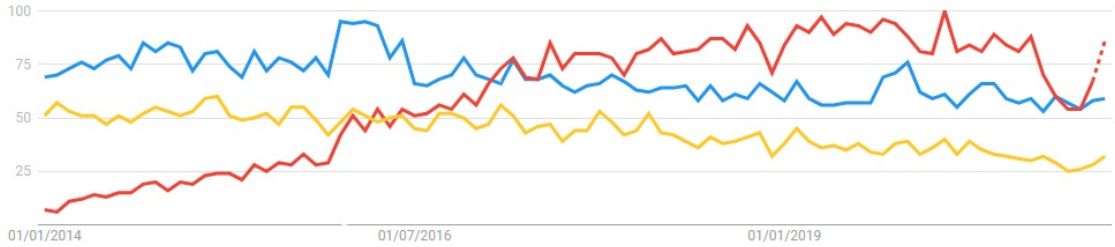


Figura 4.7: Interesse ao longo do tempo dos termos Chef (azul), Ansible (vermelho) e Puppet (amarelo). Fonte: (Google Trends 2021c)

Analisando o gráfico de interesses, podemos verificar um pequeno favorecimento da ferramenta Ansible comparativamente com as restantes. Considerado os dados levantados

na Tabela 4.3, podemos concluir que a ferramenta Ansible para além de possuir facilidade na sua instalação, também permite uma gestão mais simples, derivada da sua arquitetura, linguagem de configuração e tipos de mecanismo utilizados para transferir as configurações.

Com a realização da análise destas ferramentas, pode-se concluir que a melhor ferramenta para gestão de configurações é o Ansible.

4.4.3 Ferramenta para teste da infraestrutura

Para além das ferramentas para a conceção da solução, é necessário também selecionar a ferramenta que melhor permite cumprir o requisito de testar a solução em termos de funcionalidade. Como já analisado na Secção 2.3.3, atualmente são raras as ferramentas que permitem testar a infraestrutura, e mesmo entre as mencionadas, várias possuem limitações.

A ferramenta Terrascan permite melhorar o processo de desenvolvimento de IaC devido à deteção de falhas na segurança e de vulnerabilidades, permitindo fazer *scan* a múltiplos ficheiros de várias tecnologias. No entanto, para o contexto e requisito em questão, esta ferramenta não permite efetuar o teste da funcionalidade da infraestrutura criada. Em alternativa, a ferramenta Terratest, também abordada na Secção 2.3.3, permite a validação e execução de testes sobre a infraestrutura. O factor diferenciador entre esta e Terrascan resume-se à possibilidade de criação de testes escritos em Go que permitem criar toda a infraestrutura necessária, efetuar vários tipos de validação da infraestrutura, seja com pedidos HTTP, chamadas à API ou outros. Considerando os pontos referidos, a única abordagem capaz de resolver o requisito definido é com a utilização da ferramenta Terratest.

4.4.3.1 Abordagem de testes

Após a seleção da ferramenta de testes, é necessário adaptar a abordagem de validação da infraestrutura de modo a alinhar os objetivos com a ferramenta. Para a infraestrutura criada, é necessário principalmente testar a funcionalidade de dois componentes: o BH e o *cluster* de Kubernetes.

No processo de teste do BH, pode ser utilizada a funcionalidade da ferramenta que permite a conexão à máquina através de SSH. Se a ligação for estabelecida com sucesso, é assumido que o componente está a trabalhar conforme suposto.

Por fim, para efetuar testes sobre o *cluster*, pode ser utilizada a RESTful API definida pela ferramenta Kubernetes. A ferramenta Terratest permite, através desta API, efetuar pedidos HTTP para os *health endpoints* do *cluster*: livez e readyz (Kubernetes 2021). Assim, é possível garantir o bom funcionamento do componente.

4.5 Arquitetura da solução

Nesta secção são apresentados e clarificados vários artefactos que melhor descrevem a idealização da solução. Primeiramente é demonstrado o diagrama de implantação e de seguida o diagrama de componentes.

4.5.1 Vista lógica de componentes

Para o desenho do diagrama de componentes, foram utilizados vários recursos já declarados no diagrama presente na Figura 4.9, com alguma abstração. Para além desses recursos,

foram adicionadas também todas as dependências necessárias para construção da solução. Essas dependências contemplam as ferramentas necessárias a instalar na máquina local para efetuar todo o processo de geração de infraestrutura e também componentes como o Terraform Registry, que permite a utilização de módulos criados pela comunidade para facilitar a criação de recursos Terraform.

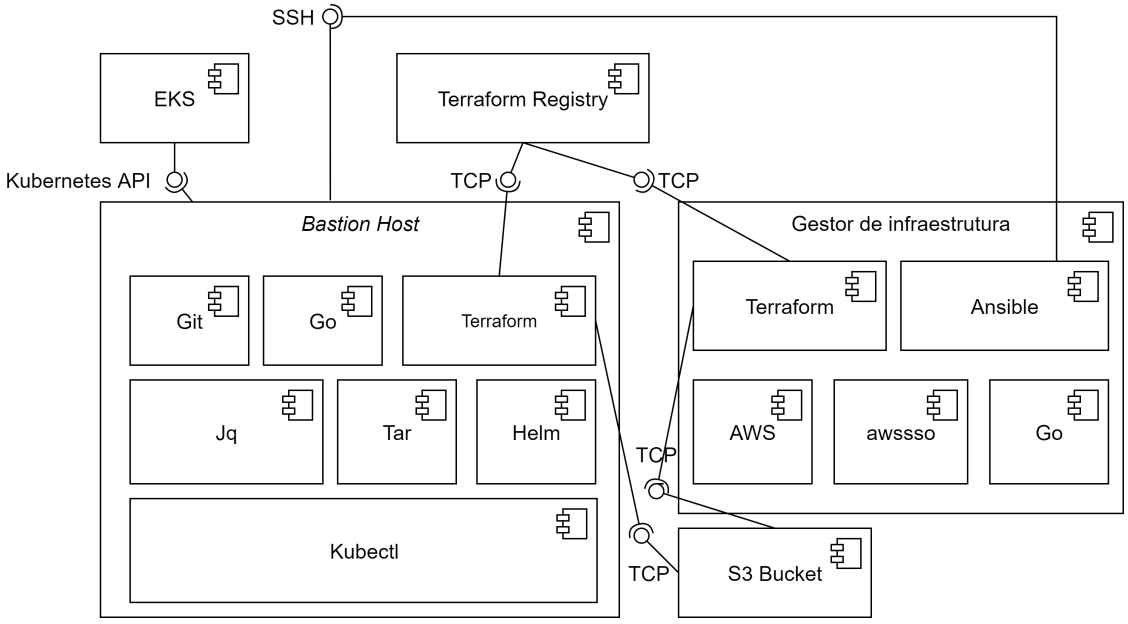


Figura 4.8: Diagrama de componentes da solução.

Todas as ferramentas inseridas dentro do componente Gestor de infraestrutura permitem a criação e gestão do BH e a execução dos testes sobre a infraestrutura. Na mesma nota, os componentes existentes no BH permitem a criação, gestão e teste do *cluster* de Kubernetes. Estes componentes encontram-se melhor detalhados na Secção 5.2.

4.5.2 Vista de implantação

Considerando que este projeto tem como objetivo apenas a automatização da infraestrutura já criada anteriormente, não existe a necessidade de analisar as várias arquiteturas possíveis para solucionar o problema. No entanto, de modo a clarificar a arquitetura da solução, foi criado um diagrama de implantação representativo do estado final, exibido na Figura 4.9.

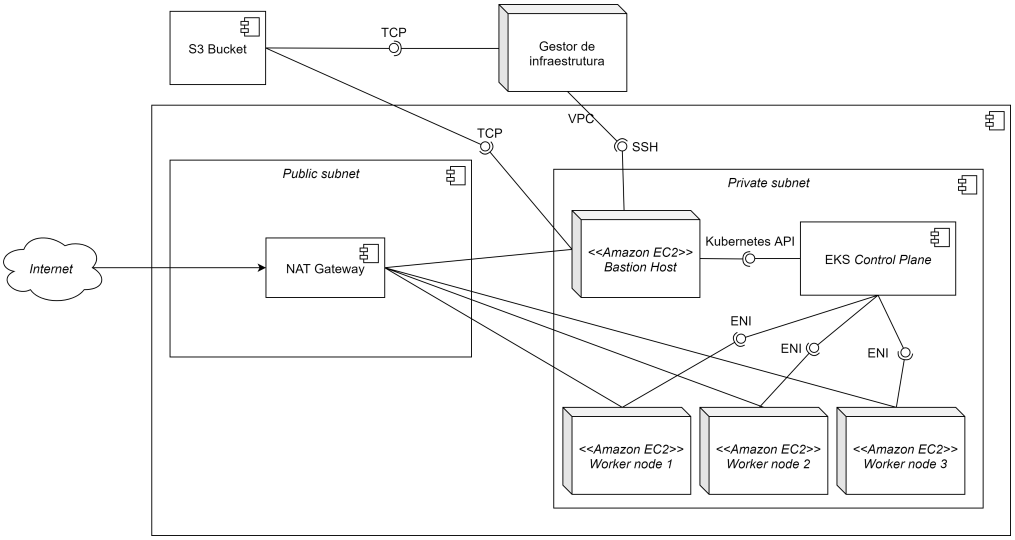


Figura 4.9: Diagrama de implantação da solução.

Capítulo 5

Construção da solução

Neste capítulo são documentados todos os passos necessários para a construção da solução e justificadas as decisões tomadas durante o processo. Inicialmente são descritos todos os pré requisitos necessários para a criação da solução, seguido da infraestrutura base necessária para o bom funcionamento da solução. Posteriormente é detalhada a construção e instalação de configurações do BH e do *cluster* e por fim é efetuada a instalação de uma ferramenta de CI/CD no *cluster*.

5.1 Metodologias de trabalho

Durante o processo de desenho e construção da solução foram utilizadas várias abordagens de modo a efetuar a criação da solução do modo mais eficiente e eficaz possível.

Todo este projeto foi desenvolvido segundo uma *framework* ágil, mais precisamente o Scrum (Scrum Portugal 2017). Scrum é uma *framework* que permite flexibilizar e aumentar a rapidez de entrega e de desenvolvimento de novos produtos, cimentando-se na construção e melhoria gradual do produto, através de Sprints (Scrum Portugal 2017). Sprints consistem em intervalos com duração entre 1 a 4 semanas onde são definidos objetivos e tarefas a realizar, resultando num incremento de valor ao produto final. As tarefas a serem realizadas durante o Sprint são transferidas do Product Backlog, onde estão inseridas e organizadas por prioridade, para o Sprint Backlog (Scrum Portugal 2017). A gestão do Product Backlog é feita pelo Product Owner através do Refinement, uma ação onde são adicionadas, estimadas e detalhadas tarefas. No decurso de uma Sprint, existe um conjunto de eventos a serem executados, incluindo (Schwaber e Sutherland 2013):

- **Planeamento** - Evento que decorre no início de uma Sprint, onde são organizadas, priorizadas e selecionadas as tarefas que irão ser feitas durante a mesma.
- **Revisão** - Evento onde são apresentadas as tarefas realizadas durante a Sprint aos *stakeholders*. Normalmente este evento ocorre no fim da Sprint.
- **Retrospetiva** - Evento com o objetivo de refletir e analisar os pontos positivos e negativos da Sprint. Normalmente este evento ocorre no fim da Sprint.
- **Reunião diária** - Evento diário onde todos os elementos da equipa descrevem sucintamente o que realizaram no dia anterior, o que irão realizar no próprio dia e as dificuldades que tiveram. Esta reunião tem uma duração máxima de 15 minutos.

Todo o fluxo entre os eventos e artefactos desta *framework* são descritos na Figura 5.1:

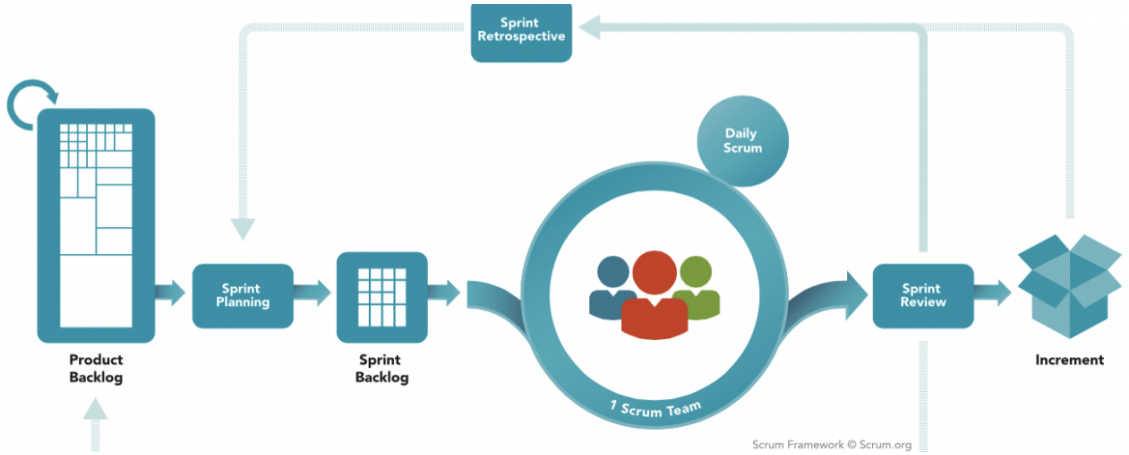


Figura 5.1: Fluxo da framework Scrum. Fonte: (Scrum Portugal 2017)

Neste projeto em específico foram realizadas Sprints com a duração de 2 semanas e o Product Backlog é mantido recorrendo à ferramenta Jira. É também utilizado um sistema de controlo de versões, neste caso um servidor no Bitbucket fornecido pela organização, onde estão inseridos os repositórios criados no decorrer deste projeto.

Para complementar a *framework* Scrum, foi também praticada uma cultura de DevOps (Amazon Web Services 2020; Atlassian 2019; Microsoft Azure 2020; Purohit 2020). DevOps é um conjunto de práticas com o objetivo de aumentar a rapidez de criação, teste, implantação e entrega de soluções, unindo o processo de desenvolvimento do produto (Dev) às operações (Ops) recorrendo sempre que possível à automação de processos manuais (Amazon Web Services 2020; Purohit 2020). Com a junção das várias fases do produto, representadas na Figura 5.2, DevOps fomenta uma maior necessidade de comunicação entre equipas, aumentando a partilha de conhecimento entre elas e as responsabilidades de cada parte envolvida.

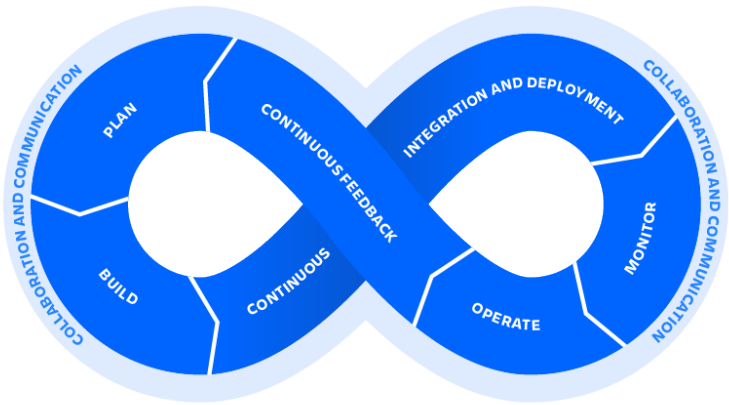


Figura 5.2: Sequência de processos numa cultura de DevOps. Fonte: (Atlassian 2019)

Esta cultura tem aliado um leque de vantagens para as equipas que a praticam, tais como (Amazon Web Services 2020; Atlassian 2019; Microsoft Azure 2020; Purohit 2020):

- Aumento da frequência e velocidade de lançamento de novos incrementos do produto;
- Aumento do conhecimento geral do produto;
- Automatização de processos manuais sempre que possível;
- Melhoria na colaboração entre equipas.

De modo a facilitar a adoção e prática da cultura de DevOps, foi utilizado um fluxo denominado Gitflow. Gitflow é um fluxo de trabalho que facilita a criação contínua de *software* através da existência de vários *branches* e da relação entre os mesmos. (Atlassian 2021). Segundo este fluxo, em deterioramento da existência de apenas um *branch master*, são utilizados dois *branches* para gerir o histórico do projeto: master, onde é mantido o histórico de *releases* e *tags* do projeto, e *develop*, onde são integradas novas funcionalidades do projeto (Atlassian 2021). Estes *branches* são utilizados em conjunto com outros três: *feature*, *release* e *hotfix*. Como o nome indica, o *branch feature*, gerado a partir da *branch develop*, é criado com o objetivo de adicionar valor ao projeto, sendo que, depois da nova funcionalidade estar concluída, este é integrado no *branch develop* (Atlassian 2021). Assim que o *branch develop* possuir conteúdo suficiente para ser gerada uma nova *release*, é criado um *branch release* para o efeito. Tal como foi referido, este *branch* é criado a partir do *branch develop* e tem como objetivo integrar as novas funcionalidades implementadas no *branch master*, marcando o mesmo com uma nova *tag* (Atlassian 2021). Por fim, é possível efetuar a criação de *branches hotfix*. Estes *branches* têm como objetivo corrigir erros que existam no *branch master*, sendo o único *branch* baseado no *branch master* (Atlassian 2021). Uma vez corrigidos os erros, as alterações devem surtir efeito no *branch master* e *develop* (Atlassian 2021). Na Figura 5.3 está representado o esquema da relação entre os *branches* com a utilização deste fluxo de trabalho.

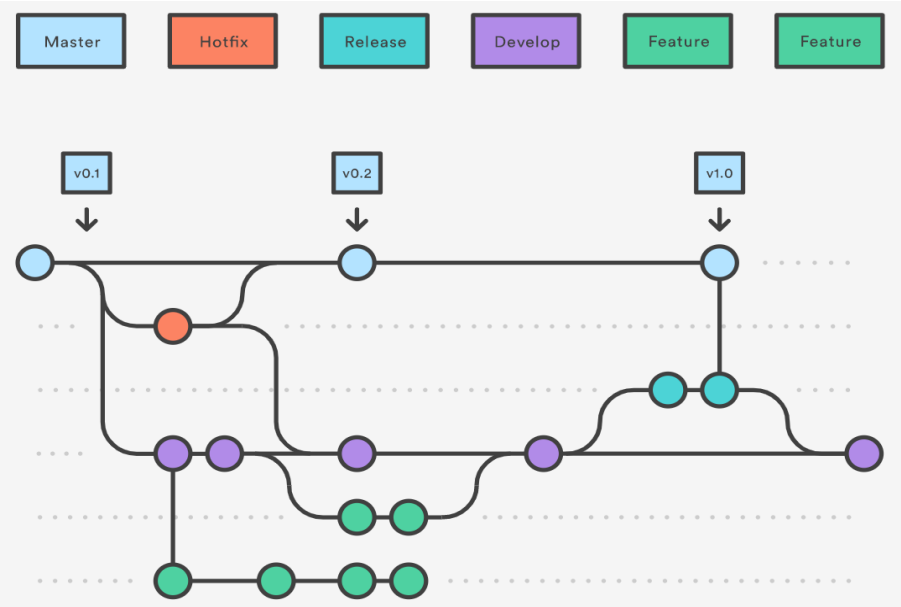


Figura 5.3: Relação entre branches utilizando Gitflow. Fonte: (Atlassian 2021)

5.2 Pré-requisitos

Para ser realizada a construção da solução com êxito, é fundamental executar, previamente, alguns passos manuais. O primeiro e mais simples é a instalação das ferramentas que irão ser utilizadas, neste caso Terraform e Ansible, no computador local.

O segundo passo, é a instalação de uma ferramenta baseada em Python, para uso interno do BMW Group. Esta gera e persiste credenciais da AWS temporários, com duração de 12h, via *Security Token Service*.

O terceiro e último passo, resume-se em declarar o perfil da AWS onde se pretende efetuar a criação da infraestrutura. Esta definição pode também ser gerada através da linha de comandos.

5.3 Infraestrutura base

Tal como referido na secção 4.4, para a criação da infraestrutura base foi utilizada a ferramenta Terraform.

A infraestrutura base, neste contexto, é referente às *subnets*, NAT Gateways, *route tables* e NLBs. A VPC não está descrita como código, uma vez que as politicas impostas pelo BMW Group requerem que esta seja criada manualmente por pessoas autorizadas, sempre que necessário.

Para o efeito, foram criadas, através de recursos Terraform, as *subnets* necessárias para ambos os tipos de *cluster*: 7 *subnets*, 3 privadas e 4 públicas, sendo que para cada *subnet*, existe uma cópia numa AZ diferente. Das três *subnets* privadas, uma é onde se encontram os *worker nodes* dos *clusters* de operações, outra serve para efetuar conexão com o carro, caso necessário, sendo que a última é onde está inserido o BH de operações. Por outro lado, nas *subnets* públicas existe uma onde estão inseridas as bases de dados, outra onde são inseridos os *worker nodes* e o BH de aplicações (devido à necessidade de acesso à *internet*). Existe uma também para hospedar as NAT Gateways e por fim uma para permitir a ligação ao carro. O número máximo de endereços de cada *subnet* está dependente do valor configurado na criação da VPC, pelo que pode variar de ambiente para ambiente, dependendo da necessidade.

Por fim, dentro da *subnet* pública atribuída, foi criada uma NAT Gateway com as devidas *route tables* associadas, viabilizando assim o acesso à *internet*.

Depois da criação da infraestrutura base, o estado atual da solução pode ser observado no esquema da Figura 5.4.

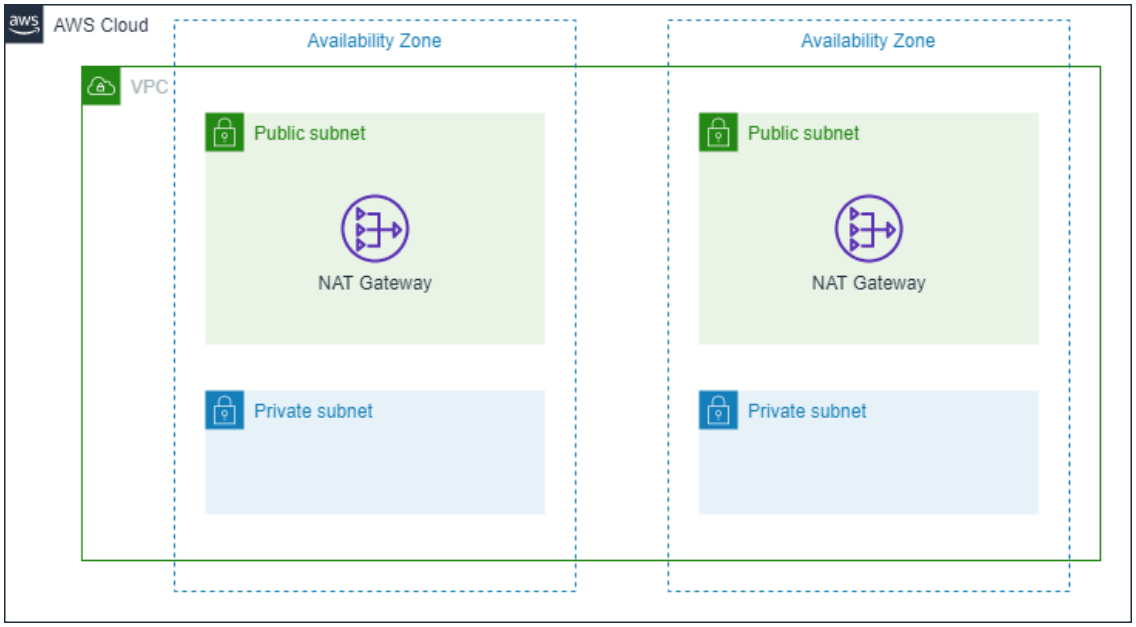


Figura 5.4: Infraestrutura existente após a criação da infraestrutura base.

5.4 Bastion Host

A construção do BH foi dividida em duas partes: a primeira onde é realizado o aprovisionamento da infraestrutura necessária e a segunda onde efetuada a configuração do mesmo.

5.4.1 Aprovisionamento da infraestrutura

Como referido na secção de 4.4, para efetuar o aprovisionamento da infraestrutura necessária para o BH, foi utilizada a ferramenta Terraform.

Para começar a codificar os ficheiros Terraform, é necessário definir previamente os módulos fundamentais para gerar a infraestrutura pretendida, os *providers*. Neste caso, como é apenas pretendida a criação de infraestrutura simples, foi apenas utilizado o *provider* AWS. É também selecionado um *remote backend* onde é guardado o ficheiro de estado da infraestrutura, sendo neste contexto um S3 Bucket.

Tendo sido concluídos os passos anteriores, é necessário selecionar e codificar a infraestrutura que é requerida para o BH. Nesta circunstância, como a máquina a criar é bastante simples, foi utilizado um módulo existente no Terraform Registry. Este módulo é disponibilizado pela AWS e permite facilmente criar uma instância EC2, apenas inserindo vários parâmetros como o seu nome, a *subnet* onde é inserida, os seus SGs, entre outros. É neste passo que é inserida a imagem que é utilizada para configurar a máquina: uma imagem Linux, neste contexto.

A esta infraestrutura está também associado um cargo (*role*), cargo esse que vai conceder as devidas permissões à máquina para esta fazer o pretendido e não mais que isso. Neste caso em específico, é dado, por exemplo, autorização para a máquina efetuar todo o tipo de ações sob o serviço EKS, visto ser o objetivo primordial da máquina. Para além das permissões referidas, a instância possui também o acesso necessário para desempenhar as restantes funções.

Para cada ambiente e região, são utilizados ficheiros com variáveis diferentes, de modo a possibilitar o uso de valores distintos dependendo do objetivo final.

Por fim, foi também necessário, após a construção da infraestrutura, elaborar um método de forma a originar *outputs* que possam ser utilizados na gestão de configurações. Para tal, foi utilizado o recurso Terraform Output, que permite devolver informação de um determinado recurso. Neste caso em particular, foi necessário devolver o IP privado da instância EC2 criada, para ser utilizado no processo de gestão de configurações na subsecção 5.4.2.

Após o aprovisionamento do BH, o estado atual da solução está descrito na Figura 5.5.

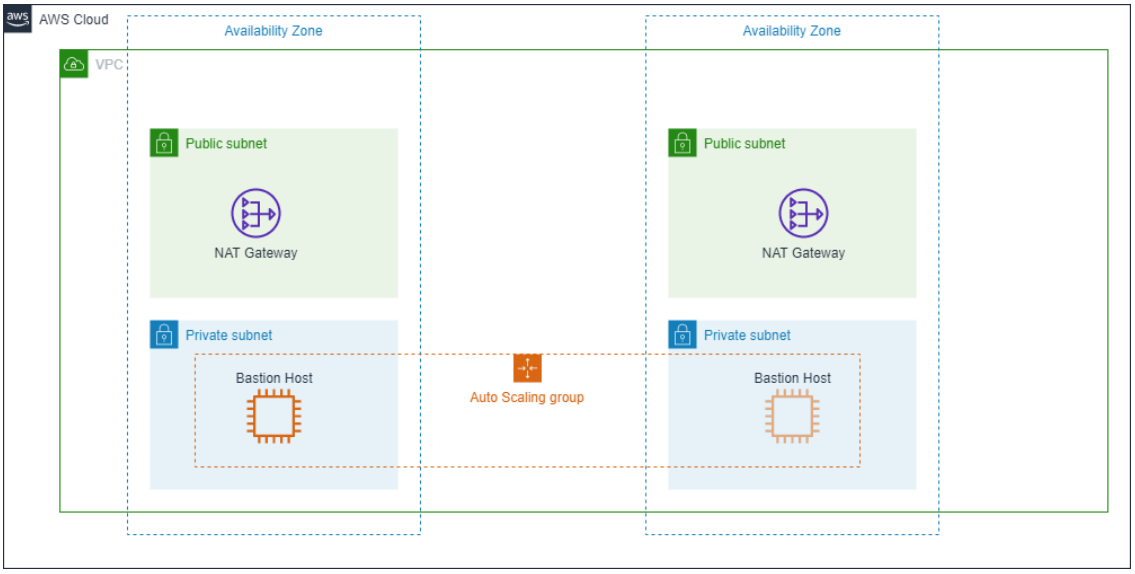


Figura 5.5: Infraestrutura existente após a criação do BH.

Depois de criada a infraestrutura, é possível aceder ao BH de duas formas: através de SSH ou do serviço AWS Systems Manager. A conexão por meio de SSH é possível graças à criação de um SG que está associado ao BH e que possibilita todo o tráfego a partir de máquinas da rede interna do BMW Group com a chave secreta gerada anteriormente. Por outro lado, também é possível interagir com o BH recorrendo ao serviço AWS Systems Manager sendo para isso necessário entrar na AWS Management Console e criar uma sessão na máquina que se pretende aceder. Este serviço proporciona uma segurança extra relativamente à conexão através de SSH porque possibilita a criação de *logs* das ações efetuadas por cada utilizador, permitindo rastrear de modo geral as operações executadas.

Depois de deter toda a infraestrutura como código, o processo de criação ou alteração de infraestrutura torna-se completamente diferente daquilo que era inicialmente. Com a solução criada, para efetuar este tipo de operações basta modificar os ficheiros de configuração Terraform e aplicá-los à infraestrutura corrente.

5.4.2 Gestão de configurações

Como referido na secção 4.4, para a gestão de configurações do BH foi utilizada a ferramenta Ansible.

Inicialmente, existe a necessidade de criar o ficheiro de configurações da ferramenta, que contém: a localização do ficheiro *inventory*, o tipo de conexão à máquina e as configurações

para possibilitar a execução de comandos como administrador. O ficheiro *inventory* tem como função armazenar a informação relativa a todas as máquinas onde podem ser efetuadas alterações e este pode ser organizado de diversas formas: pela divisão das instâncias segundo o seu tipo, aplicação ou microsserviço, pela divisão das instâncias tendo em conta a sua região ou pela divisão das instâncias considerando o ambiente onde estão inseridas (teste, produção, etc) (Ansible 2021b).

Assim sendo, pensou-se inicialmente em organizar o ficheiro *inventory* efetuando a divisão das máquinas segundo a sua região ou ambiente. Se fosse efetuada a divisão por região, todas as máquinas dessa mesma região seriam afetadas, tanto as máquinas de teste como de produção. Por outro lado, se fosse efetuada a divisão por ambiente, todas as máquinas do mesmo ambiente seriam alteradas, independentemente da sua região. Apesar das possíveis vantagens, no contexto deste projeto não existem casos práticos onde estas abordagens possam ser aplicadas, sendo que as alterações são geralmente aplicadas numa região e ambiente em específico. Tendo isto em consideração, optou-se por aplicar uma combinação entre a divisão das instâncias por ambiente e por região, visto que, ao fazer a divisão de forma isolada eram originados grupos demasiado abrangentes, o que levava ao não aproveitamento da solução. Este tipo de divisão permite assim efetuar uma separação de contextos mais apropriada tendo em conta o panorama atual da solução.

No Bloco de código 5.1 pode ser visualizado um exemplo de divisão do ficheiro *inventory* para dois tipos de ambiente em duas regiões distintas, segundo a abordagem selecionada.

```
1  [eu-central-1-test] -> regiao/ambiente
2  eu-central-1-ops-test -> tipo do BH
3  eu-central-1-app-test
4
5  [eu-central-1-prod]
6  eu-central-1-ops-prod
7  eu-central-1-app-prod
8
9  [cn-north-1-test]
10 cn-north-1-ops-test
11 cn-north-1-app-test
12
13 [cn-north-1-prod]
14 cn-north-1-ops-prod
15 cn-north-1-app-prod
```

Bloco de código 5.1: Exemplo de ficheiro *inventory* com dois ambientes em duas regiões.

Depois de definidas as instâncias, é necessário estabelecer qual o melhor método de acesso às mesmas, de forma a efetuar sua gestão de configurações. Neste caso, o método escolhido para aceder às máquinas foi através de SSH, sendo que a sua definição foi feita num ficheiro com a chave e o IP privado, ambos gerados no aprovisionamento da infraestrutura.

Depois de definidas todas as configurações, resta efetuar a criação do Playbook, um ficheiro onde são inseridas as ferramentas e configurações que se pretendem que sejam geridas. O Playbook é constituído por várias *tasks*, sendo estas responsáveis pela instalação das diversas ferramentas que são utilizadas no BH. Foram inicialmente instaladas múltiplas ferramentas, entre as quais:

- Git - Ferramenta para gerir e transferir código versionado;
- Jq - Ferramenta para processar documentos em formato JSON;

- Tar - Ferramenta para descompactar ficheiros;
- Kubectl - Ferramenta para gerir recursos de Kubernetes;
- Terraform - Ferramenta de aprovisionamento de infraestrutura;
- Helm - Ferramenta para gerir aplicações de Kubernetes.

Para criar as *tasks*, foram utilizados essencialmente três módulos já existentes no Ansible, de modo a facilitar todo o processo: o módulo yum, o módulo unarchive e o módulo shell. O módulo yum, à semelhança da funcionalidade do *package manager* existente na linha de comandos, serve para facilitar a gestão de ferramentas. No Bloco de código 5.2, encontra-se um exemplo de uma *task* que utiliza este módulo para efetuar a instalação de várias ferramentas.

```
- name: Instala uma lista de ferramentas
  yum:
    name: "{{ packages }}"
    state: present
  vars:
    packages:
      - "@Development tools"
      - git
      - jq
```

Bloco de código 5.2: Tarefa que usufrui do módulo yum.

Por outro lado, sentiu-se a necessidade de utilizar o módulo unarchive uma vez que nem todas as ferramentas podem ser instaladas através do módulo yum, entre as quais a ferramenta Terraform e Helm. Este módulo funciona como extrator de ficheiros, de modo a descompactá-los num destino escolhido previamente. Assim, é possível, apenas com uma *task*, efetuar a instalação de ferramentas que, caso contrário, necessitariam de vários comandos para serem instaladas. No Bloco de código 5.3, é possível visualizar um exemplo de como o módulo unarchive é declarado. Neste caso, é efetuada a transferência da ferramenta Terraform que é descompactada na pasta "/usr/local/bin", ficando assim automaticamente instalada.

```
- name: Install terraform {{ terraform_version }}
  unarchive:
    src: "https://releases.hashicorp.com/terraform/{{ terraform_version
    }}/terraform_{{ terraform_version }}_linux_amd64.zip"
    dest: /usr/local/bin
    remote_src: True
```

Bloco de código 5.3: Tarefa que usufrui do módulo unarchive.

Como se pode observar no Bloco de código 5.4, a versão da ferramenta que se pretende instalar estava num formato de variável. Isto resulta essencialmente da separação de contextos entre as várias instâncias nos vários ambientes e regiões. Por razões de segurança e compatibilidade, é vantajoso permitir a atualização da versão de uma ferramenta específica numa instância de teste, de modo a verificar a existência possíveis conflitos, fazendo com que não seja necessário efetuar, ao mesmo tempo, essa atualização num ambiente de produção.

```
terraform_version: 1.0.0
tar_version: 1.34
helm_version: v3.6.1
kubectl_version: v1.21.0
```

Bloco de código 5.4: Exemplo do ficheiro de definição das versões das ferramentas para um ambiente.

O módulo shell é apenas utilizado quando é necessário executar comandos diretamente na máquina, por exemplo para modificar as permissões de um ficheiro necessário para instalar uma ferramenta corretamente ou para mover ficheiros, como é exemplificado no Bloco de código 5.5.

```
- name: Install helm {{ helm_version }}
  unarchive:
    src: "https://get.helm.sh/helm-{{ helm_version }}-linux-amd64.tar.gz
    "
    dest: /tmp
    remote_src: True
- shell: chmod +x /tmp/linux-amd64/helm ; mv /tmp/linux-amd64/helm /usr/
    local/bin
```

Bloco de código 5.5: Exemplo de utilização do módulo shell juntamente com o módulo unarchive.

Assim, com a utilização da ferramenta Ansible, é facilitada a gestão de configurações de uma máquina em específico, bastando fazer a alteração necessária no ficheiro Playbook e executar os comandos necessários.

5.4.3 Automatização da criação do BH

Previamente à criação da infraestrutura, foi criado um bloco de código que garante a criação de um S3 Bucket, caso não exista, para persistir a informação do ficheiro de estado que é gerado após a criação da infraestrutura. O nome deste *bucket* estará diretamente relacionado com o ambiente no qual a infraestrutura está a ser criada e com a região em questão.

Para além da criação do S3 Bucket, é também gerado um par chave, constituído por uma chave pública e outra privada. O algoritmo utilizado para a criação da chave foi o RSA, um algoritmo que se baseia na dificuldade de decifrar grandes números, sendo que o tamanho utilizado para gerar a chave foi de 4096 bits (SSH Academy 2021). Depois de criada, a chave privada foi inserida no Vault, uma ferramenta de gestão de segredos e de proteção de informação sensível, para que possa ser acedida sempre que necessário, por pessoas autorizadas.

A próxima operação passa por inicializar a ferramenta Terraform enviando também as configurações do *remote backend* onde vai ser guardado o ficheiro de estado, como representado no Bloco de código 5.6.

```
terraform init \
  -reconfigure \
  -backend-config="bucket=$S3_BUCKET_NAME" \
  -backend-config="region=$AWS_REGION" \
  -backend-config="key=$AWS_REGION-bastion-host-$AWS_ENVIRONMENT.tfstate
    "
```

Bloco de código 5.6: Inicialização das configurações de Terraform e configuração do *remote backend*.

Depois de executados os comandos de Terraform e já estar criado o BH, é necessário recorrer ao *output* gerado e substituir os valores necessários no ficheiro de configuração de SSH do Ansible para se conseguir conectar com sucesso ao BH.

Por fim, é executado o comando que aplica os ficheiros de configuração do Ansible no BH, limitando a execução apenas ao BH especificado, como pode ser visualizado no Bloco de código 5.7.

```
1 ansible-playbook ansible/playbook.yml --limit $AWS_REGION-
    $AWS_ENVIRONMENT
```

Bloco de código 5.7: Comando de execução do Playbook.

5.5 Cluster de Kubernetes

Para a construção do *cluster* de Kubernetes, foi utilizada a ferramenta Terraform. Ao contrário do BH, para esta infraestrutura não é necessário criar um S3 Bucket para guardar a informação do ficheiro de estado, uma vez que este já foi gerado durante a criação do BH.

Foi requerido um maior número de *providers* para a criação do *cluster* de Kubernetes em comparação ao que foi requerido para o BH, já que o primeiro apresenta uma maior complexidade. Assim sendo, para a criação do *cluster* foram utilizados os seguintes *providers*:

- AWS - Módulo para gerir os recursos existentes no *Cloud Service Provider*;
- Kubernetes - Módulo para gerir recursos de Kubernetes;
- Helm - Módulo para gerir aplicações de Kubernetes;
- Local - Módulo para gerir recursos locais;
- Null - Módulo que não cria nenhum recurso intencionalmente, geralmente usada para fazer algum tipo de arranjo;
- Random - Módulo para dar suporte a operações de aleatoriedade;
- Template - Módulo utilizado para carregar ficheiros em variáveis de Terraform, de modo a serem utilizadas na execução de comandos.

Tirando os módulos para os quais não se efetuou qualquer tipo de configuração, o módulo mais simples de configurar é o AWS, já que para este é apenas necessário indicar qual a região que se pretende que o módulo use. Para além deste, é também essencial efetuar a configuração do módulo Kubernetes e do módulo Helm. Para o módulo de Kubernetes, é preciso inserir o *endpoint* do *cluster* e o *token* associado ao mesmo. Para o módulo Helm, para além da informação anterior, é necessário declarar o caminho para as configurações do Kubernetes.

Inicialmente, para criação do *cluster* é utilizado um módulo publicado pela AWS no Terraform Registry, de nome EKS. À semelhança do módulo utilizado para a criação do BH, este facilita a criação de infraestrutura para o serviço EKS. O *cluster*, definido para utilizar a versão 1.19 de Kubernetes (versão já utilizada anteriormente), é criado juntamente uma *policy*, de modo a conceder aos *worker nodes* a funcionalidade de *auto scaling*. Para além destes, é também criado o SG que estará associado a cada *worker node*, de modo a permitir a conexão de máquinas dentro do VPC ao *cluster* e a conexão aos *node ports* do *cluster*. Relativamente aos *worker nodes*, os limites mínimos e máximos de de instâncias vão depender do ambiente

em que estas se encontram, podendo atingir um máximo de vinte instâncias em ambientes de produção. O tipo de instância utilizado nos *worker nodes* é m5.xlarge.

Caso o *cluster* seja destinado a operações, é essencial criar um ConfigMap, de modo a definir as variáveis de *proxy* necessárias. Este passo é possível através da criação de um *template* com toda a informação necessária para o ConfigMap, como exemplificado no Bloco de código 5.8.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: proxy-environment-variables
  namespace: kube-system
data:
  HTTP_PROXY: http://${http_proxy}
  HTTPS_PROXY: http://${https_proxy}
  NO_PROXY: ${no_proxy}
  http_proxy: http://${http_proxy}
  https_proxy: http://${https_proxy}
  no_proxy: ${no_proxy}
```

Bloco de código 5.8: *Template* criado para ser utilizado.

Este ficheiro foi, então, carregado para um recurso de Terraform através do módulo *template*, tal como se pode observar no Bloco de código 5.9, para que este possa ser utilizado.

```
data "template_file" "kubernetes_proxy_env_variables" {
  template = file("${path.module}/templates/
    kubernetes_proxy_environment_variables.yaml.tpl")

  vars = {
    http_proxy   = var.http_proxy
    https_proxy   = var.http_proxy
    no_proxy     = local.no_proxy_merged
  }
}
```

Bloco de código 5.9: *Template* a ser carregado para um recurso de Terraform.

O *template* referido foi depois aplicado no *cluster* através do módulo Null. Primeiramente, é verificado se o *cluster* que está a ser criado é de operações ou de aplicações. Se este for de operações, espera para que o *cluster* seja totalmente criado e aplica, posteriormente, o comando referido no Bloco de código 5.10.

```
resource "null_resource" "create_cm_with_proxy_environment_variables" {
  count = var.aws_account_type == "ops" ? 1 : 0
  depends_on = [module.eks-cluster]

  provisioner "local-exec" {
    command = "echo \"${data.template_file.
    kubernetes_proxy_env_variables.rendered}\" | kubectl apply -f - "
  }
}
```

Bloco de código 5.10: Recurso Terraform que aplica o *template* carregado anteriormente.

Depois deste processo estar concluído e de já existir o ConfigMap, são aplicadas atualizações a *daemonsets* como o aws-node e kube-proxy, cujo processo é similar ao descrito anteriormente.

Para os *clusters* de aplicações, é necessário criar uma StorageClass, de forma a possibilitar a utilização do sistema de arquivos efs, o qual já é utilizado em alguns microsserviços.

De seguida, e com recurso ao módulo Helm, são criadas três *releases*: uma que instala o *driver*, tornado possível a utilização do efs, outra para instalar o metrics-server, permitindo que hajam métricas nos recursos de Kubernetes e uma última que instala o cluster-autoscaler, dando permissão ao Kubernetes para escalar horizontalmente *worker nodes* sempre que necessário. É importante referir que a instalação destes recursos está dependente da instalação das variáveis ambiente do Kubernetes, aplicadas anteriormente. No Bloco de código 5.11 é exemplificada a utilização do módulo Helm para instalar o *driver* efs.

```
resource "helm_release" "aws_efs_csi_driver" {
  name       = "aws-efs-csi-driver"
  repository = "https://kubernetes-sigs.github.io/aws-efs-csi-driver/"
  chart      = "aws-efs-csi-driver"
  namespace  = local.k8s_default_namespace
  timeout    = 900

  depends_on = [null_resource.configure_kube_proxy_environment_variables
    ]
}
```

Bloco de código 5.11: Exemplo de instalação do *driver* do efs através do módulo Helm.

De modo a automatizar a criação do *cluster* e à semelhança do BH, foi criado um *script* para agilizar o processo. Neste caso, como é apenas utilizada uma ferramenta, a operação acaba por ser facilitada, sendo apenas necessário executar o Bloco de código 5.6 e de seguida aplicar os ficheiros de Terraform criados, com os parâmetros também similares aos do *script* criado para o BH. Este *script* é executado dentro do BH.

Concluída a construção do *cluster*, é obtida a infraestrutura representada na Figura 5.6.

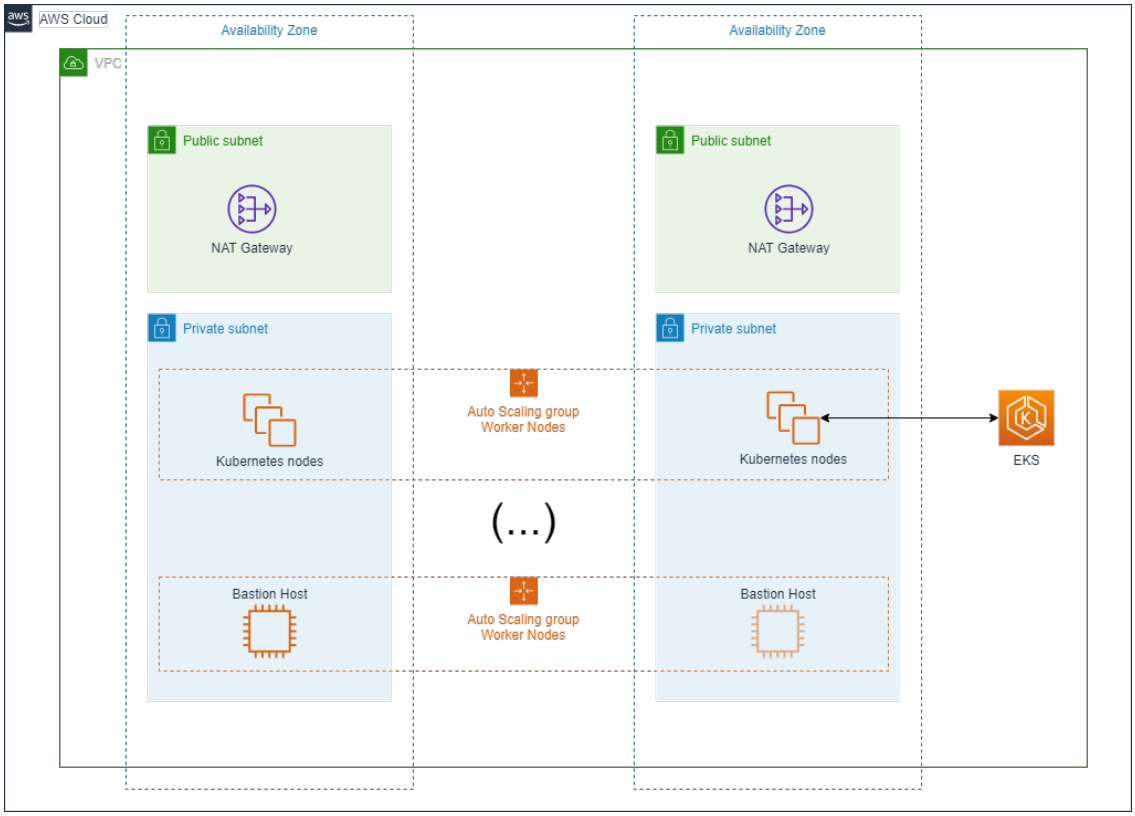


Figura 5.6: Solução final.

5.5.1 Melhorias no cluster

Foram efetuadas algumas melhorias ao *cluster* de Kubernetes durante a recriação do mesmo através de IaC. As melhorias foram aplicadas aos *worker nodes*, porque anteriormente à construção da nova solução não havia gestão dos mesmos por parte do EKS. Anteriormente, os nós eram geridos através de um ASG na AWS criado e gerido manualmente, de modo a serem criados nós, caso necessário.

Este processo não requeria muita manutenção por parte dos colaboradores, mas sempre que era necessário efetuar atualizações no *cluster*, o cenário complicava-se, derivado da necessidade de cumprir os seguintes passos:

- Atualizar o *Control Plane* para a nova versão;
- Criar um novo ASG com as novas versões;
- Adicionar os *target groups* necessários ao novo ASG;
- Drenar os nós um a um.

Para além destes processos terem de ser efetuados de forma bastante minuciosa, é também essencial que estes sejam feitos na ordem correta, caso contrário pode facilmente ser originado *downtime* nas aplicações presentes no *cluster*. Nesse sentido, e tendo em conta os aspetos negativos referidos anteriormente, foi efetuada uma melhoria de modo a colmatar o elevado número de passos manuais que era necessário efetuar, sendo os *worker nodes* atualmente geridos pelo EKS. É importante referir que esta delegação de gestão não apresenta nenhum custo associado, sendo que só é pago os recursos criados (AWS 2021h).

Esta melhoria foi possível com a adição de um *node group* ao *cluster*. A configuração do *node group* é uma operação simples de se realizar, sendo apenas necessário indicar o número máximo e mínimo de *worker nodes* a serem criados no *cluster*, o *launch template* que é utilizado sempre que é necessário criar mais um nó e a restante informação sobre as máquinas, como o tipo da instância, a imagem utilizada para criar a instância, entre outros. Depois de inserida toda a informação, o próprio EKS gera um ASG e a partir desse momento o *cluster* passa a ser totalmente gerido pela AWS.

Para efetuar uma atualização já com o *cluster* gerido pela AWS, é apenas necessário atualizar o *Control Plane*, atualizar também a imagem utilizada no *launch template* dos *worker nodes* para igualar a versão do *Control Plane* e modificar o *node group* para utilizar essa nova versão do *launch template*. Com essas 3 ações o *cluster* é atualizado automaticamente sem existir a necessidade de drenar cada nó manualmente.

5.5.2 Conexão entre o BH e o cluster

Derivado do uso da ferramenta Terraform, aquando da criação do *cluster* existe uma propriedade que faz com que a definição do contexto do Kubernetes seja realizada automaticamente: write_kubeconfig. Caso esta propriedade seja ativada e seja definida uma localização para inserção do ficheiro de configuração do Kubernetes na variável config_output_path, o Terraform encarrega-se de fazer essa modificação aquando da criação do *cluster*, permitindo assim a conexão automática entre o BH e o *cluster*.

5.6 Instalação de uma ferramenta no cluster

Como forma de comprovar o bom funcionamento do *cluster*, foi efetuada a implantação de uma ferramenta de CI/CD no mesmo, tendo sido selecionado o Jenkins. Foi escolhida esta ferramenta devido à pré existência de ficheiros de configuração Terraform para a sua criação, permitindo assim a integração do *cluster* com as soluções já existentes.

A primeira etapa para efetuar a instalação da ferramenta foi a criação de um novo *namespace*, onde a mesma é inserida. Depois de criado o *namespace*, são, por fim, executados no BH os *scripts* de configuração Terraform que permitem implantar o Jenkins no *cluster*, disponibilizando a ferramenta para os colaboradores.

Capítulo 6

Avaliação da solução

Neste capítulo é realizada a avaliação da solução produzida. Esta avaliação conta com uma fase onde são definidas as várias hipóteses que possibilitam a validação da solução, declaração dos vários indicadores para avaliar a solução e avaliação da solução considerando esses indicadores.

6.1 Definição de hipóteses a avaliar

A elaboração das hipóteses que são utilizadas para avaliar a solução criada foi baseada nos objetivos deste projeto. Foram, assim, desenvolvidas duas hipóteses mutuamente exclusivas.

Hipótese nula (H_0) - A solução não permite automatizar nem agilizar a criação de *clusters*; q < 70%

Hipótese alternativa (H_1) - A solução permite automatizar e agilizar a criação de *clusters*. q $\geq$ 70%

O q apresentado nas duas hipóteses reflete a qualidade geral da solução quando aplicado o método apresentado na Secção 6.3. Este método utiliza os indicadores de maior importância identificados para avaliar o projeto.

6.2 Indicadores

Antes de realizar a avaliação da solução, é importante identificar os critérios utilizados neste processo. Para este projeto, o indicador utilizado é o grau de conformidade entre a solução criada e os requisitos definidos na Secção 4.2, considerando as seguintes indicadores:

- Tempo de criação da infraestrutura;
- Custo da utilização da solução;
- Possibilidade de escalar a aplicação;
- Percentagem de disponibilidade do sistema;
- Consistência da solução;
- Tempo de resposta das aplicações inseridas na solução;
- Manutibilidade da solução;
- Modularidade da solução.

Para além destes indicadores, também são considerados os valores de um questionário realizado a vários colaboradores.

6.3 Metodologia de avaliação

A metodologia utilizada para a avaliação da solução é o *Quantitative Evaluation Framework* (QEF) devido à sua modularidade ao contexto pretendido e à possibilidade de quantificar o peso de cada grandeza. Nesta Secção é feita uma introdução ao modelo, seguido da sua aplicação.

6.3.1 Modelo QEF

O modelo de avaliação QEF permite analisar a qualidade de um projeto de *software* através da decomposição de fatores direta ou indiretamente mensuráveis, em métricas mensuráveis (Escudeiro e Bidarra 2008). Este modelo adequa-se ao projeto desenvolvido devido a ser um projeto de *software* e considerando que a qualidade da solução está relacionada com o número de requisitos cumpridos. A qualidade das soluções é avaliada segundo um espaço tridimensional, onde cada espaço contém um número de fatores que se pretende avaliar.

O modelo é constituído por vários conceitos, entre eles (Escudeiro e Bidarra 2008):

- Dimensão - Corresponde às caraterísticas mais importantes e indispensáveis da solução. É necessária a criação de várias dimensões independentes para a aplicação do modelo e deve ser este o primeiro passo a executar;
- Fator - Corresponde a um agregado de requisitos e pertence a uma dimensão;
- Requisitos - Corresponde à unidade mais pequena deste modelo, de modo a representar uma métrica mensurável.

Depois de definidas as dimensões, os fatores de cada dimensão e os requisitos para cada fator, é necessário efetuar a classificação e o grau de conclusão (0 - 100%) de cada requisito. A classificação é efetuada tendo em conta uma escala de 0-10, onde 10 é um requisito fundamental e 0 é um requisito irrelevante (Escudeiro e Bidarra 2008). De seguida, é necessário definir os pesos relativos de cada fator dentro de uma dimensão, valor que pode oscilar entre 0 e 1. Depois de definidos os pesos relativos, é necessário calcular a contribuição de cada fator na respetiva dimensão, recorrendo à Equação 6.1, onde m é o número de requisitos no fator, pr_m o peso de cada requisito e pc_m a percentagem de concretização do mesmo (Escudeiro e Bidarra 2008).

$$fator = \frac{1}{\sum_m pr_m} \times \sum_m (pr_m \times pc_m) \tag{6.1}$$

Para calcular a contribuição de cada dimensão no cálculo da qualidade da solução, pode ser utilizada a Equação 6.2, onde p_n corresponde ao peso relativo definido para cada fator e o $fator_n$ corresponde ao valor calculado no passo anterior (Escudeiro e Bidarra 2008).

$$dimensao = \sum_n (p_n \times fator_n), \sum_n (p_n = 1) \wedge p_n \in [0, 1] \tag{6.2}$$

A penúltima parte, passa por calcular o desvio global do sistema relativamente ao sistema ideal com a utilização dos valores das dimensões calculados na etapa anterior, como representado na Equação 6.3.

$$D = \sqrt{\sum_{j}(1 - \frac{Dim_j}{100})^2} \tag{6.3}$$

Por fim, é possível calcular a qualidade da solução em percentagem, através da Equação 6.4.

$$q = (1 - \frac{D}{\sqrt{n}}) \times 100, q \in [0, 100] \tag{6.4}$$

6.3.2 Execução do modelo

Nesta secção é feita a aplicação do modelo QEF, considerando os passos seguidos na Secção 6.3.1. Numa fase inicial são definidas as dimensões a utilizar, os fatores de cada dimensão e os requisitos, considerando os objetivos e requisitos estabelecidos nas Secções 1.3 e 4.2.

Tabela 6.1: Resultado da fase de definição das várias grandezas do modelo QEF.

Dimensão	Fator	Requisito
Funcionalidade	Caraterísticas fundamentais	F01 - Automatização da criação de infraestrutura
		F02 - Gestão de clusters através de um BH
	Proteção contra erros humanos	F03 - Documentação de infraestrutura
		F04 - Mecanismo de autorização
Não funcional	Fiabilidade	NF01 - Criação de infraestrutura por qualquer colaborador
		NF05 - Solução não deve deteriorar o tempo de resposta das aplicações
		NF07- Criação de testes funcionais
	Adaptabilidade	Escalamento automático do número de nós do *cluster*
		NF10 - Solução utilizada em várias regiões e ambientes
	Manutibilidade	NF08 - Facilidade de atualização das versões de Kubernetes
		NF09 - Facilidade de atualização das versões das ferramentas do BH
		NF02 - Solução deve ter 99.9% de disponibilidade
		NF03 - Redução em pelo menos 50% do tempo de recuperação do cluster em caso de desastre
		NF04 - Construção consistente de infraestrutura

continua na próxima página

Tabela 6.1 (continuação)

Dimensão	Fator	Requisito
		NF11 - Redução dos custos de manutenção
Restrições	Ferramentas	NF12 - Infraestrutura na AWS
		NF13 - Utilização da ferramenta EKS
	Implementação	NF14 - Definição de IaC

Depois de definidos os três tipos de conceitos do modelo, são estabelecidas as classificações de cada requisito e o respetivo grau de conclusão. Para a realização deste processo, os requisitos são verificados isoladamente tendo em conta a escala representada na Tabela 6.2.

Tabela 6.2: Referências para classificação dos requisitos.

0	2	4	6	8	10
Irrelevante	Opcional	Necessário	Importante	Muito Importante	Fundamental

De modo a determinar o grau de conclusão de cada requisito, é efetuada uma análise individual de modo a clarificar o ponto de situação de cada um.

6.3.2.1 F01 - Automatização do processo de criação infraestrutura

Este requisito foi avaliado segundo o número de passos manuais necessários para construir a solução: caso o número seja entre 1 e 3 assume-se 100% conclusão, de 4 a 6 assume-se 50% e mais de 6 assume-se 0%.

Neste contexto, são necessários 3 passos manuais para a construção de toda a solução: executar o *script* para criar a infraestrutura base, executar o *script* para o BH e finalmente para o *cluster*. Portanto, é assumida uma conclusão de 100%.

6.3.2.2 F02 - Gestão de clusters através de um BH

Este requisito é avaliado segundo a existência ou não de um BH para gerir o *cluster*. Se existir, assume-se 100% de conclusão, senão é assumida 0% de conclusão do requisito.

Neste caso foi criado um BH para efetuar essa gestão, portanto é definido como 100% de conclusão.

6.3.2.3 F03 - Documentação de infraestrutura

Este requisito é avaliado segundo a existência de documentação na infraestrutura. Caso exista, este requisito considera-se 100% concluído, senão é assumida 0% de conclusão do requisito.

Neste contexto foi criada documentação da infraestrutura, considerando assim o requisito foi totalmente concluído.

6.3.2.4 F04 - Mecanismo de autorização

Este requisito é avaliado segundo a criação ou não de um mecanismo de autorização. À semelhança de requisitos anteriores, caso haja a criação do mecanismo é considerada 100% de conclusão, caso contrário 0%.

Ao longo deste projeto não foi possível efetuar a criação do mecanismo de autorização, portanto é assumido um grau de conclusão de 0%.

6.3.2.5 NF01 - Criação de infraestrutura por qualquer colaborador

O método de avaliação deste requisito foi conduzido através de um questionário realizado a várias pessoas (incluindo membros da equipa).

Para efetuar uma tentativa de avaliação quantitativa deste requisito, foram utilizadas as médias entre a questão apresentada na Figura A.3 e na Figura A.5, dando um valor de 85%.

6.3.2.6 NF02 - Solução deve ter 99.9% de disponibilidade

Este requisito é avaliado considerando a documentação das ferramentas utilizadas.

Segundo os dados apresentados pela AWS, cada instância Amazon EC2 possui 99,99% de disponibilidade, o serviço EKS garante uma disponibilidade de 99,95% e o serviço de armazenamento S3 Bucket permite obter 99,99% de disponibilidade, podendo ser assumido, assim, o cumprimento do requisito (AWS 2020a, 2021c,l).

6.3.2.7 NF03 - Redução em pelo menos 50% do tempo de recuperação do cluster em caso de desastre

Para avaliar se este requisito é cumprido ou não, foi comparado o tempo de criação da infraestrutura manualmente com o tempo de criação da mesma recorrendo à solução criada.

Tabela 6.3: Tempos de criação dos vários componentes.

	Com IaC	Manualmente
Tempo de criação da infraestrutura base	*	*
Tempo de criação do BH	1m:20s	7m:45s
Tempo de instalação de ferramentas no BH	3m:35s	20m
Tempo de criação do *cluster*	17m:45s	16h

Por questões de logística, não possível ser efetuado o teste de criação da infraestrutura base devido a decisões superiores, portanto este não contará para a avaliação da solução. Relativamente às execuções realizadas, todas elas foram conduzidas por um colaborador com 2 anos de experiência em desenvolvimento Java e 1 ano de DevOps.

Considerando os resultados obtidos, pode-se comprovar que o requisito foi cumprido a 100%.

6.3.2.8 NF04 - Construção consistente de infraestrutura

Este requisito é avaliado segundo a possibilidade de executar os comandos e obter sempre o mesmo resultado.

Na seleção das ferramentas, foram selecionadas ferramentas que garantiam idempotência, permitindo obter o mesmo resultado mesmo ao executar os *scripts* múltiplas vezes sobre o mesmo recurso. Também com recurso a IaC é garantido a consistência de construção da infraestrutura.

6.3.2.9 NF05 - Solução não deve deteriorar o tempo de resposta das aplicações

Este requisito é avaliado segundo as caraterísticas da infraestrutura criada.

No contexto do projeto, o *cluster* foi criado com *worker nodes* com capacidade computacional superior à existente previamente, garantindo, com as mesmas configurações, tempos de resposta iguais ou até melhores.

6.3.2.10 NF06 - Escalamento automático do número de nós do cluster

Este requisito é avaliado segundo a possibilidade ou não de efetuar o escalamentos dos nós de forma automática.

Na solução criada, os nós do *cluster* são geridos pela AWS, permitindo o seu escalamento automático dentro dos limites de número de nós estabelecidos.

6.3.2.11 NF07- Criação de testes funcionais

Este requisito é avaliado segundo a existência ou não de testes funcionais.

Durante o decurso deste projeto não foram efetuados testes funcionais à solução.

6.3.2.12 NF08 - Facilidade de atualização das versões de Kubernetes

Este requisito é avaliado segundo o questionário realizado.

Ponderando a média das respostas à questão apresentada na Figura A.1, é aferida uma percentagem de conclusão de 75% deste requisito.

6.3.2.13 NF09 - Facilidade de atualização das versões das ferramentas do BH

Este requisito é avaliado segundo o questionário realizado.

Ponderando a média das respostas à questão da Figura A.2, a percentagem de conclusão correspondente é de 80%.

6.3.2.14 NF10 - Solução utilizada em várias regiões e ambientes

Este requisito é classificado consoante a possibilidade de reutilizar o código criado para criar a infraestrutura para vários ambientes e regiões.

Como apresentado no Capítulo 5, a solução permite criar a infraestrutura em múltiplas regiões, ambientes e também para os diferentes tipos de *cluster*. Para tal, são utilizados ficheiros de variáveis diferentes para cada construção, de modo a permitir alterar as propriedades necessárias.

6.3.2.15 NF11 - Redução dos custos de manutenção

Este requisito é analisado considerando o valor gasto previamente para manter a infraestrutura e o valor atual.

Neste requisito não existem valores concretos, no entanto é possível observar que com a utilização de IaC, a probabilidade de criação de erros na infraestrutura é menor, reduzindo as perdas relativas a *downtime*, não existe necessidade de existir uma equipa especializada para criação e manutenção de infraestrutura, reduzindo assim o custo de mão de obra e como esta solução reduz o tempo de criação de infraestrutura, o custo de mão de obra por hora acaba por ser mais eficiente.

6.3.2.16 NF12 - Infraestrutura na AWS

Este requisito é classificado segundo a utilização do *Cloud Service Provider* AWS para a criação da solução.

No contexto deste projeto, toda a infraestrutura foi criada dentro do ecossistema AWS.

6.3.2.17 NF13 - Utilização da ferramenta EKS

Este requisito é classificado segundo a utilização do serviço EKS.

Neste projeto, para facilitar a criação de um *cluster* de Kubernetes foi utilizado o serviço EKS que garante a gestão de múltiplos componentes pela AWS.

6.3.2.18 NF14 - Definição de IaC

Este requisito é classificado segundo a definição de infraestrutura como código.

No decorrer deste projeto, a infraestrutura criada foi definida na totalidade como código, recorrendo a ferramentas como Terraform e Ansible.

6.3.2.19 Classificação dos requisitos

Considerando as análises efetuadas a cada requisito, como resultado foi obtida a Tabela 6.4, onde é exibido a percentagem de conclusão de cada requisito e a sua classificação.

Tabela 6.4: Resultados finais.

Requisito	Classificação	% conclusão
F01 - Automatização da criação de infraestrutura	10	100
F02 - Gestão de clusters através de um BH	10	100
F03 - Documentação de infraestrutura	8	100
F04 - Mecanismo de autorização	2	0
NF01 - Criação de infraestrutura por qualquer colaborador	8	85
NF02 - Solução deve ter 99.9% de disponibilidade	10	100
NF03 - Redução em pelo menos 50% do tempo de recuperação do cluster em caso de desastre	6	100
NF04 - Construção consistente de infraestrutura	10	100
NF05 - Solução não deve deteriorar o tempo de resposta das aplicações	10	100

continua na próxima página

Tabela 6.4 (continuação)

Requisito	Classificação	% conclusão
NF06 - Cluster de fácil escalamento	10	100
NF07- Criação de testes funcionais	4	0
NF08 - Facilidade de atualização das versões de Kubernetes	6	75
NF09 - Facilidade de atualização das versões das ferramentas do BH	6	80
NF10 - Solução utilizada em várias regiões e ambientes	10	100
NF11 - Redução dos custos de manutenção	4	100
NF12 - Infraestrutura na AWS	10	100
NF13 - Utilização da ferramenta EKS	10	100
NF14 - Definição de IaC	10	100

Considerando que o peso de cada fator vai depender do número de requisitos inseridos no mesmo, resta apenas calcular a contribuição de cada fator, seguido do cálculo da contribuição de cada dimensão e só depois seria possível concluir a qualidade da solução.

6.4 Resultados

Depois de efetuados todos os cálculos, a solução foi avaliada com uma qualidade de 95%, possuindo um desvio global do sistema de 0.14. A qualidade da solução revelou ser saliente derivado a vários fatores, incluindo:

- Elevado número de requisitos resolvidos (89%);
- Requisitos não resolvidos possuíam uma classificação baixa, devido à sua prioridade.

Relativamente às várias dimensões definidas, a dimensão com maior qualidade foi a dimensão das restrições, com 100% de qualidade, tendo as restantes uma qualidade de 90%. Isto deve-se ao facto das restantes dimensões possuírem requisitos por concluir. É de salientar que todos os requisitos com classificação 10 (prioridade máxima), foram concluídos a 100%.

No Apêndice sA é apresentado o questionário realizado na totalidade, tendo sido obtidas avaliações bastante satisfatórias por parte dos colaboradores. Destaca-se as respostas para a questão apresentada na Figura A.4 e na Figura A.5, sendo que 100% dos inquiridos utilizariam esta solução para efetuar a criação de um *cluster* e consideram que a solução facilita esse processo.

Capítulo 7

Conclusão

Neste capítulo é efetuada uma pequena conclusão acerca do trabalho realizado começando por sumariar o trabalho efetuado e o estado solução final, seguido das limitações e trabalho a efetuar no futuro. Para finalizar, é efetuada uma apreciação do projeto.

7.1 Sumário

Derivado da existência de demasiados processos manuais durante a implantação de *clusters* de Kubernetes na *cloud*, surgiu a necessidade de investigar uma forma de automatizar a operação, com o objetivo de, para além de automatizar, diminuir o tempo de criação da infraestrutura, entre outros.

De modo a dominar de maneira geral as várias ferramentas existentes que podiam ser relevantes no desenvolvimento do projeto, foi efetuada uma análise, por categoria, das múltiplas possibilidades existentes e das suas funções, permitindo assim uma decisão informada no que toca à escolha das tecnologias.

Seguido do processo de familiarização das ferramentas e dos diversos projetos de interesse para a solução criada, surge um dilema relativo à abordagem a seguir: a utilização de uma ferramenta multifuncional contra a utilização de duas ferramentas dedicadas. A decisão recaiu para a utilização de duas ferramentas dedicadas, principalmente pela qualidade da solução final e pela separação de funções, utilizando assim cada ferramenta para o seu propósito.

Como resultado obtém-se uma solução que através de *scripts* e IaC automatiza todo o processo de criação de componentes de *networking*, um BH e um *cluster* de Kubernetes em menos de 25 minutos. Esta solução para além da velocidade e agilidade que proporciona, permite a sua reutilização do mesmo código para os vários ambientes existentes, para as diversas regiões onde é necessário realizar este tipo de operações e para as duas categorias de *cluster* atualmente existentes nas soluções das equipas.

Na última etapa, foi avaliada a solução construída recorrendo ao modelo QEF e também a um formulário preenchido por vários membros da equipa e outros colaboradores. Através do modelo, a qualidade da solução foi avaliada em 95%, constatando assim a elevada pertinência e qualidade do projeto para a empresa. É importante enfatizar também os resultados positivos obtidos através do questionário realizado aos vários colaboradores da empresa.

7.2 Limitações e trabalho futuro

Para além do trabalho desenvolvido, sobraram ainda alguns requisitos que devido a limitações no tempo não foram possíveis de efetivar, entre eles a criação de um mecanismo de autorização e a construção de testes para a infraestrutura criada. Para além destes, seria também interessante estender esta solução a outro *Cloud Service Provider*, porque apesar das ferramentas utilizadas serem agnósticas a *Cloud Service Provider*, o utilizado foi a AWS. Assim era possível deduzir o tempo para migrar toda a infraestrutura, caso necessário.

Foi, no entanto, efetuada uma rápida pesquisa sobre a existência dos vários serviços utilizados pelo projeto nos principais *Cloud Service Providers*, concluindo-se que seria possível efetuar a migração dos mesmos, sendo para isso necessário alterar o *provider* na ferramenta Terraform e também modificar alguns dos módulos utilizados (Cloud 2021).

7.3 Apreciação final

A realização deste projeto resultou num desenvolvimento tanto a nível pessoal como profissional. A um nível profissional, foi possível cimentar e expandir todo o conhecimento sobre infraestrutura, DevOps e IaC. Permitiu também aumentar familiaridade com várias tecnologias e metodologias das quais não tinha conhecimento. A um nível pessoal, permitiu-me superar várias adversidades que surgiram pelo caminho, melhorar a gestão de tempo e ao mesmo tempo impulsionar a capacidade de decisão.

Assim sendo, na generalidade o trabalho foi concluído com sucesso e espero que impacte positivamente o dia-a-dia dos colaboradores da empresa.

Bibliografia

Accurics (2021). *Terrascan - Accurics*. url: https://www.accurics.com/products/terrascan/ (acedido em 03/10/2021).

Alabonte (mai. de 2019). *Construction of swarm cluster of Docker*. url: https://programmer.help/blogs/construction-of-swarm-cluster-of-docker.html (acedido em 07/02/2021).

Alija, Nexhati (2017). «Justification of Software Maintenance Costs». Em: *International Journal of Advanced Research in Computer Science and Software Engineering* 7.3, pp. 15–23. issn: 22776451. doi: 10.23956/ijarcsse/v7i2/01207.

Amazon Web Services (2020). *AWS DevOps - O que é DevOps? - Amazon Web Services*. url: https://aws.amazon.com/pt/devops/what-is-devops/ (acedido em 01/10/2021).

Andrade Trevizano, Waldir (2005). *Emprego do Método da Análise Hierárquica (A.H.P.) na seleção de processadores*. Rel. téc.

Ansible (fev. de 2021a). *Ansible architecture*. url: https://docs.ansible.com/ansible/latest/dev%7B%5C_%7Dguide/overview%7B%5C_%7Darchitecture.html (acedido em 06/03/2021).

– (set. de 2021b). *How to build your inventory — Ansible Documentation*. url: https://docs.ansible.com/ansible/latest/user%7B%5C_%7Dguide/intro%7B%5C_%7Dinventory.html%7B%5C#%7Did7 (acedido em 08/09/2021).

– (fev. de 2021c). *Intro to playbooks*. url: https://docs.ansible.com/ansible/latest/user%7B%5C_%7Dguide/playbooks%7B%5C_%7Dintro.html (acedido em 06/03/2021).

– (2021d). *Introduction to modules*. url: https://docs.ansible.com/ansible/latest/user%7B%5C_%7Dguide/modules%7B%5C_%7Dintro.html (acedido em 06/03/2021).

Arora, Anushka (mar. de 2020). *AWS EKS vs KOPS - What to choose?* url: https://devtron.ai/blog/aws-eks-vs-kops-what-to-chose/ (acedido em 05/02/2021).

Atlassian (2019). *What is DevOps? | Atlassian*. url: https://www.atlassian.com/devops (acedido em 01/10/2021).

– (2021). *Gitflow Workflow*. url: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (acedido em 15/02/2021).

AVI Networks (2021). *What is Container Orchestration? Definition & Related FAQs | Avi Networks*. url: https://avinetworks.com/glossary/container-orchestration/ (acedido em 02/02/2021).

AWS (2020a). *Amazon EKS Service Level Agreement*. url: https://aws.amazon.com/pt/eks/sla/ (acedido em 10/04/2021).

– (set. de 2020b). *Introduction to AWS CloudFormation*. url: https://app.pluralsight.com/library/courses/introduction-aws-cloudformation/table-of-contents (acedido em 31/01/2021).

– (2021a). *Amazon EC2*. url: https://aws.amazon.com/pt/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime%7B%5C&%7Dec2-whats-new.sort-order=desc (acedido em 06/02/2021).

– (2021b). *Amazon EKS*. url: https://aws.amazon.com/pt/eks/?whats-new-cards.sort-by=item.additionalFields.postDateTime%7B%5C&%7Dwhats-new-cards.

sort-order=desc%7B%5C&%7Deks-blogs.sort-by=item.additionalFields.createdDate%7B%5C&%7Deks-blogs.sort-order=desc (acedido em 06/02/2021).

AWS (2021c). *Amazon Simple Storage Service FAQs*. url: https://www.amazonaws.cn/en/s3/faqs/ (acedido em 10/10/2021).

– (2021d). *Amazon VPC - Virtual Private Cloud*. url: https://aws.amazon.com/pt/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate%7B%5C&%7Dvpc-blogs.sort-order=desc (acedido em 02/03/2021).

– (2021e). *Auto Scaling groups*. url: https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html (acedido em 04/03/2021).

– (2021f). *AWS CloudFormation Designer interface overview*. url: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer-overview.html (acedido em 01/02/2021).

– (2021g). *Definição de preço do Amazon EKS*. url: https://aws.amazon.com/pt/eks/pricing/ (acedido em 06/02/2021).

– (2021h). *Managed node groups*. url: https://docs.aws.amazon.com/eks/latest/userguide/managed-node-groups.html (acedido em 19/09/2021).

– (2021i). *NAT gateways*. url: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html (acedido em 04/03/2021).

– (2021j). *O que é a distribuição contínua?* url: https://aws.amazon.com/pt/devops/continuous-delivery/ (acedido em 09/02/2021).

– (2021k). *O que significa integração contínua?* url: https://aws.amazon.com/pt/devops/continuous-integration/ (acedido em 09/02/2021).

– (2021l). *Recursos do Amazon EC2*. url: https://aws.amazon.com/pt/ec2/features/ (acedido em 10/04/2021).

– (2021m). *Recursos do Amazon EKS*. url: https://aws.amazon.com/pt/eks/features/ (acedido em 06/02/2021).

– (2021n). *Regions and Zones - Amazon Elastic Compute Cloud*. url: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html (acedido em 04/03/2021).

– (2021o). *SDKs e toolkits de programação da AWS*. url: https://aws.amazon.com/pt/tools/ (acedido em 01/02/2021).

– (2021p). *Security groups for your VPC*. url: https://docs.aws.amazon.com/vpc/latest/userguide/VPC%7B%5C_%7DSecurityGroups.html (acedido em 12/09/2021).

– (2021q). «VPCs and subnets». Em: *AWS*. url: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html.

Baker, Mark e Rajkumar Buyya (1999). «Cluster Computing at a Glance». Em:

Bannan, James (nov. de 2019). *Getting Started with Chef Fluency*. url: https://app.pluralsight.com/library/courses/getting-started-chef-fluency/table-of-contents (acedido em 13/02/2021).

Bellavance, Ned (out. de 2019). *Terraform - Getting Started*. url: https://app.pluralsight.com/course-player?clipId=4ffc60c2-9f03-40d8-a6df-b7668efe8466 (acedido em 30/01/2021).

Bernardete, Andreia e Ferreira Fernandes (2018). *Determinantes da Intenção de Compra de Produtos Naturais*. Rel. téc. Coimbra, p. 110.

BMW (2020). *BMW*. url: https://www.bmw.pt/pt/topics/fascination-bmw/critical-techworks.html (acedido em 31/12/2020).

Brikman, Yevgeniy (set. de 2016). *Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation | by Yevgeniy Brikman | Gruntwork*. url: https:

//blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c (acedido em 31/01/2021).

Brikman, Yevgenjy (2019). *Terraform: Up & Running, 2nd Edition*. url: https://learning.oreilly.com/library/view/terraform-up/9781492046899/.

Brooks, Mike (dez. de 2018). *Infrastructure as Code: Imperative vs Declarative*. url: https://tech.ovoenergy.com/imperative-vs-declarative/ (acedido em 30/01/2021).

Chamberlain, Doug (mar. de 2018). *Containers vs. Virtual Machines (VMs): What's the Difference?* url: https://blog.netapp.com/blogs/containers-vs-vms/ (acedido em 06/02/2021).

Chan, Mike (2018). *15 Infrastructure as Code Tools to Automate Deployments*. url: https://www.thorntech.com/2018/04/15-infrastructure-as-code-tools/ (acedido em 30/01/2021).

Cloud, Google (2021). *Compare os serviços da AWS e do Azure com o Google Cloud*. url: https://cloud.google.com/free/docs/aws-azure-gcp-service-comparison.

Cloud Native Landscape (2021). *CNCF Cloud Native Interactive Landscape*. url: https://landscape.cncf.io/?zoom=250 (acedido em 08/02/2021).

CloudFlare (2021). *What is a subnet?* url: https://www.cloudflare.com/learning/network-layer/what-is-a-subnet/ (acedido em 04/03/2021).

Co, golibrary (abr. de 2020). *Infrastructure as Code — Orchestration, Provisioning & Configuration Management (Ansible & Terraform)*. url: https://medium.com/@golibraryco/infrastructure-as-code-orchestration-provisioning-configuration-management-ansible-7cd0615e49c5 (acedido em 06/03/2021).

Concourse CI (2021a). *Hello World pipeline*. url: https://concourse-ci.org/hello-world-example.html (acedido em 14/02/2021).

– (2021b). *Jobs*. url: https://concourse-ci.org/jobs.html (acedido em 14/02/2021).

– (2021c). *Pipelines*. url: https://concourse-ci.org/pipelines.html (acedido em 14/02/2021).

– (2021d). *Tasks*. url: https://concourse-ci.org/tasks.html%7B%5C#%7Dschema.task.platform (acedido em 14/02/2021).

– (2021e). *User Roles & Permissions - Concourse CI*. url: https://concourse-ci.org/user-roles.html (acedido em 14/02/2021).

Critical Techworks (2020). *Critical TechWorks*. url: https://www.criticaltechworks.com/ (acedido em 19/12/2020).

Dashora, Saurabh (2019). *Understanding the AWS Security Groups and Best Practices to use them*. url: https://progressivecoder.com/understanding-aws-security-groups-and-best-practices-to-use-them/ (acedido em 10/04/2021).

Docker (2021a). *Swarm mode key concepts*. url: https://docs.docker.com/engine/swarm/key-concepts/ (acedido em 07/02/2021).

– (2021b). *Swarm mode overview*. url: https://docs.docker.com/engine/swarm/ (acedido em 07/02/2021).

Dorneles, Lucas (out. de 2015). *Ahp*. url: https://pt.slideshare.net/lucasD1993/ahp-54446197 (acedido em 28/02/2021).

Eeles, Peter (2001). «Capturing Architectural Requirements». Em: *The Rational Edge*. url: http://www.therationaledge.com/content/nov%7B%5C_%7D01/t%7B%5C_%7DarchitecturalRequirements%7B%5C_%7Dpe.html.

Eeles, Peter e Peter Cripps (2009). *The Process of Software Architecting*. url: https://learning.oreilly.com/library/view/the-process-of/9780321617484/.

Escudeiro, Paula Maria e José Bidarra (2008). «Quantitative Evaluation Framework (QEF)». Em: *Revista Ibérica de Sistemas e Tecnologias de Informação*. url: https://www.researchgate.net/publication/257579051 (acedido em 06/10/2021).

Ethridge, Eric (2021). *AWS CloudFormation vs. Terraform: Which One Should You Choose?* url: https://www.missioncloud.com/blog/aws-cloudformation-vs-terraform-which-one-should-you-choose (acedido em 06/03/2021).

Farrell, Brandon (2021). *GitHub Terrascan*. url: https://github.com/accurics/terrascan (acedido em 03/10/2021).

Freitas, Ruben (jan. de 2019). *Você sabe o que são ferramentas de orquestração de containers?* url: https://vertigo.com.br/ferramentas-de-orquestracao-de-containers/ (acedido em 02/02/2021).

Google Trends (2021a). *App container*. url: https://trends.google.pt/trends/explore?date=2008-12-17%202021-02-19%7B%5C&%7Dq=app%20container (acedido em 19/02/2021).

– (mar. de 2021b). *AWS CloudFormation, Terraform*. url: https://trends.google.pt/trends/explore?date=2014-01-01%202021-03-06%7B%5C&%7Dq=AWS%20CloudFormation,Terraform (acedido em 06/03/2021).

– (mar. de 2021c). *Chef, Ansible, Puppet*. url: https://trends.google.pt/trends/explore?cat=5%7B%5C&%7Ddate=2014-01-01%202021-03-06%7B%5C&%7Dq=Chef,Ansible,Puppet (acedido em 06/03/2021).

– (fev. de 2021d). *DevOps*. url: https://trends.google.pt/trends/explore?date=all%7B%5C&%7Dq=DevOps (acedido em 24/02/2021).

Guerriero, Michele et al. (2019). «Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry». Em: url: https://dibt.unimol.it/staff/fpalomba/documents/C47.pdf.

Hallowell, David L (2021). *QFD: When and How Does It Fit in Software Development?* url: https://www.isixsigma.com/tools-templates/qfd-house-of-quality/qfd-when-and-how-does-it-fit-software-development/ (acedido em 02/03/2021).

Hassan, Asser (fev. de 2016). *Introduction to Chef*. url: https://4sysops.com/archives/introduction-to-chef/ (acedido em 05/03/2021).

HostGator (abr. de 2020). *O que é e como funciona um Cluster Computacional?* url: https://www.hostgator.com.br/blog/cluster-computacional-o-que-e/ (acedido em 19/02/2021).

Jenkins (2021a). *Jenkins download and deployment*. url: https://www.jenkins.io/download/ (acedido em 09/02/2021).

– (2021b). *Pipeline*. url: https://www.jenkins.io/doc/book/pipeline/ (acedido em 09/02/2021).

– (2021c). *Pipeline Syntax*. url: https://www.jenkins.io/doc/book/pipeline/syntax/ (acedido em 10/02/2021).

Jenkins X (2021). *jx*. url: https://jenkins-x.io/commands/jx/ (acedido em 12/02/2021).

Johari, Aayushi (nov. de 2019). *Chef vs Puppet vs Ansible vs Saltstack: Which One to Choose*. url: https://www.edureka.co/blog/chef-vs-puppet-vs-ansible-vs-saltstack/ (acedido em 07/03/2021).

Julian (fev. de 2021). *Joget on Kubernetes*. url: https://dev.joget.org/community/display/DX7/Joget+on+Kubernetes (acedido em 06/02/2021).

Kasireddy, Preethi (mar. de 2016). *A Beginner-Friendly Introduction to Containers, VMs and Docker*. url: https://medium.com/free-code-camp/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b (acedido em 06/02/2021).

Katalon (2021). *Best 14 CI/CD Tools You Must Know | Updated for 2021*. url: https://www.katalon.com/resources-center/blog/ci-cd-tools/ (acedido em 09/02/2021).

Koen, P A et al. (1996). «Fuzzy Front End : Effective Methods, Tools, and Techniques». Em: *Industrial Research* pp, pp. 5–35. url: http://www.stevens.edu/cce/NEW/PDFs/FuzzyFrontEnd%7B%5C_%7DOld.pdfNEW/PDFs/FuzzyFrontEnd%7B%5C_%7DOld.pdf.

Koen, Peter A (2004). «Understanding the Front End : A Common Language and Structured Picture». Em: *Power*.

Koen, Peter A., Heidi M.J. Bertels e Elko Kleinschmidt (2014). «Managing the front end of innovation-part I: Results from a three-year study». Em: *Research Technology Management* 57.2, pp. 34–43. issn: 19300166. doi: 10.5437/08956308X5702145.

Koen, Peter A., Heidi M.J. Bertels e Elko J. Kleinschmidt (2014). «Managing the front end of innovation-part II: Results from a three-year study». Em: *Research Technology Management* 57.3, pp. 25–35. issn: 19300166. doi: 10.5437/08956308X5703199.

Kops (2021). *kOps - Kubernetes Operations*. url: https://kops.sigs.k8s.io/ (acedido em 05/02/2021).

Kubernetes (fev. de 2019). *Standardized Glossary*. url: https://kubernetes.io/docs/reference/glossary/?all=true%7B%5C#%7Dterm-control-plane (acedido em 03/02/2021).

– (ago. de 2020a). *Kubernetes Components*. url: https://kubernetes.io/docs/concepts/overview/components/ (acedido em 03/02/2021).

– (out. de 2020b). *Namespaces*. url: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ (acedido em 04/02/2021).

– (out. de 2020c). *Understanding Kubernetes Objects*. url: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/ (acedido em 04/02/2021).

– (out. de 2020d). *What is Kubernetes?* url: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (acedido em 03/02/2021).

– (2021). *Kubernetes API health endpoints | Kubernetes*. url: https://kubernetes.io/docs/reference/using-api/health-checks/ (acedido em 13/10/2021).

Lindič, Jaka e Carlos Marques da Silva (2011). «Value proposition as a catalyst for a customer focused innovation». Em: *Management Decision* 49.10, pp. 1694–1708. issn: 00251747. doi: 10.1108/00251741111183834.

Luksa, Marko (2018). *Kubernetes in Action*. Manning Publications, p. 624.

Luu, Kevin (out. de 2020). *Intro to Terraform: Provision EC2 Instance and Install Jenkins*. url: https://levelup.gitconnected.com/intro-to-terraform-provision-ec2-instance-and-install-jenkins-3ccb7e2d60f6 (acedido em 21/02/2021).

Malaval, Nicolas (jul. de 2016). *How to Record SSH Sessions Established Through a Bastion Host*. url: https://aws.amazon.com/pt/blogs/security/how-to-record-ssh-sessions-established-through-a-bastion-host/ (acedido em 02/03/2021).

Martikainen, Atte (nov. de 2017a). *A framework of Front End of Innovation*. url: https://www.researchgate.net/figure/A-framework-of-Front-End-of-Innovation%7B%5C_%7Dfig1%7B%5C_%7D324208591 (acedido em 16/02/2021).

– (nov. de 2017b). *The New Concept Development (NCD)-model. (Koen et al., 2001)*. url: https://www.researchgate.net/figure/The-New-Concept-Development-NCD-model-Koen-et-al-2001%7B%5C_%7Dfig12%7B%5C_%7D324208591 (acedido em 16/02/2021).

Martin, Derek (out. de 2019). *Estilo de arquitetura de microsserviços*. url: https://docs.microsoft.com/pt-pt/azure/architecture/guide/architecture-styles/microservices (acedido em 19/02/2021).

Matsumota, Leonardo (abr. de 2018). *Analytic Hierarchy Process (AHP) – método de apoio a decisões complexas*. url: https://leonardo-matsumota.com/2018/04/20/analytic-hierarchy-process-ahp-metodo-de-apoio-a-decisoes-complexas/ (acedido em 27/02/2021).

Microsoft Azure (2020). *O que é o DevOps? Uma Explicação do DevOps | Microsoft Azure*. url: https://azure.microsoft.com/pt-pt/overview/what-is-devops/ (acedido em 01/10/2021).

Morgan, Andrew (jun. de 2020). *Using Jenkins X for Cloud-native CI/CD*. url: https://app.pluralsight.com/library/courses/jenkins-x-cloud-native-ci-cd/table-of-contents (acedido em 12/02/2021).

Morris, Kief (2020). *Infrastructure as Code, 2nd Edition*. 2ª ed. url: https://learning.oreilly.com/library/view/infrastructure-as-code/9781098114664/ (acedido em 24/09/2021).

Moura Da Silva, Roterdan, Mischel Carmen e Neyra Belderrain (2005). *Considerações sobre métodos de decisão multicritério*. Rel. téc.

N, Dharmalingam (fev. de 2020). *Introduction to Chef*. url: https://www.whizlabs.com/blog/chef-introduction/ (acedido em 05/03/2021).

Neap, Halil Shevket e Tahir Celik (1999). «Value of a Product: A Definition». Em: *International Journal of Value-Based Management* 12.2, pp. 181–191. issn: 0895-8815. doi: 10.1023/A:1007718715162. url: https://link.springer.com/article/10.1023/A:1007718715162.

NetApp (2021). «What are Containers?» Em: *NetApp*. url: https://www.netapp.com/devops-solutions/what-are-containers/.

Noll, Jones (jun. de 2020). *Implementação de infraestrutura como código para provisionamento e deploy de aplicações*. Rel. téc. Lajeado: Universidade do Vale do Taquari.

Oliveira, Beatriz (jul. de 2020). *Containers*. url: https://medium.com/sysadminas/containers-e4adf391de87 (acedido em 19/02/2021).

Puppet (2021a). *Introduction to Puppet*. url: https://puppet.com/docs/puppet/7.4/puppet%7B%5C_%7Doverview.html (acedido em 05/03/2021).

– (2021b). *Resources*. url: https://puppet.com/docs/puppet/7.4/lang%7B%5C_%7Dresources.html (acedido em 05/03/2021).

Purohit, Ketan (2020). «Executing DevOps & CI/CD, Reduce in Manual Dependency». Em: *International Journal of Scientific Development and Research*. issn: 2455-2631. url: www.ijsdr.org.

Red Hat (2021a). *What is container orchestration?* url: https://www.redhat.com/en/topics/containers/what-is-container-orchestration (acedido em 02/02/2021).

– (2021b). *O que são os microsserviços?* url: https://www.redhat.com/pt-br/topics/microservices/what-are-microservices (acedido em 19/02/2021).

– (2021c). *What are cloud service providers?* url: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-providers (acedido em 20/02/2021).

Rich, Nick e Matthias Holweg (2000). «Value Analysis». Em: *INNOREGIO: dissemination of innovation and knowledge management techniques*, pp. 1–31.

Sacolick, Isaac (jan. de 2020). *What is CI/CD? Continuous integration and continuous delivery explained*. url: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html (acedido em 08/02/2021).

Schults, Carlos (2019). *What Is infrastructure as code? how it works, best practices, tutorials*. url: https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/ (acedido em 29/01/2021).

Schwaber, Ken e Jeff Sutherland (jul. de 2013). *Guia do Scrum™ Um guia definitivo para o Scrum: As regras do jogo Julho de 2013 Desenvolvido e mantido por Ken Schwaber e Jeff Sutherland*. Rel. téc. (Acedido em 31/12/2020).

Scrum Portugal (abr. de 2017). *Scrum - framework de desenvolvimento de produtos, abordagem ágil*. url: http://www.scrumportugal.pt/scrum/ (acedido em 28/12/2020).

Seroter, Richard (ago. de 2019). *Getting Started with Concourse*. url: https://app.pluralsight.com/library/courses/concourse-getting-started/table-of-contents (acedido em 13/02/2021).

Sirota, Alex (nov. de 2019). *Getting Started with Travis CI*. url: https://app.pluralsight.com/library/courses/getting-started-travis-ci/table-of-contents (acedido em 13/02/2021).

Sloan, Joseph D (2004). *High Performance Linux Clusters*. O'Reilly Media, Inc, p. 368. (Acedido em 24/09/2021).

Srivastava, Deeksha (fev. de 2021). *Chef vs. Puppet vs. Ansible vs. Saltstack: A Complete Comparison*. url: https://medium.com/successivetech/chef-vs-puppet-vs-ansible-vs-saltstack-a-complete-comparison-9af8f1790c0d (acedido em 07/03/2021).

SSH Academy (2021). *How to use ssh-keygen to generate a new SSH key*. url: https://www.ssh.com/academy/ssh/keygen (acedido em 06/09/2021).

Stars, Django (fev. de 2017). *Continuous Integration. CircleCI vs Travis CI vs Jenkins*. url: https://medium.com/hackernoon/continuous-integration-circleci-vs-travis-ci-vs-jenkins-41a1c2bd95f5 (acedido em 13/02/2021).

Strachan, James (mar. de 2018). *Introducing Jenkins X: a CI/CD solution for modern cloud applications on Kubernetes*. url: https://www.jenkins.io/blog/2018/03/19/introducing-jenkins-x/ (acedido em 12/02/2021).

Tamburri, Damian Andrew et al. (2017). «DevOps: Introducing Infrastructure-as-Code». Em: doi: 10.1109/ICSE-C.2017.162. url: https://www.researchgate.net/profile/Damian-Tamburri/publication/318124847_DevOps_Introducing_Infrastructure-as-Code/links/5b17c6bbaca272021ce9147e/DevOps-Introducing-Infrastructure-as-Code.pdf.

Terraform (2020). *Terraform by HashiCorp*. url: https://www.terraform.io/ (acedido em 30/01/2021).

– (2021a). *Providers*. url: https://www.terraform.io/docs/language/providers/index.html (acedido em 02/10/2021).

– (2021b). *Resource Blocks*. url: https://www.terraform.io/docs/language/resources/syntax.html (acedido em 02/10/2021).

– (2021c). *Terraform vs. CloudFormation, Heat, etc.* url: https://www.terraform.io/intro/vs/cloudformation.html (acedido em 06/03/2021).

Terratest (2021). *Quick start*. url: https://terratest.gruntwork.io/docs/getting-started/quick-start/ (acedido em 03/10/2021).

Thones, Johannes (2015). «Microservices». Em: *IEEE Xplore*. url: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=%7B%5C&%7Darnumber=7030212.

Travis CI (2020). *Travis CI Client (CLI and Ruby library)*. url: https://github.com/travis-ci/travis.rb%7B%5C#%7Dreadme (acedido em 13/02/2021).

– (2021a). *Building a Python Project*. url: https://docs.travis-ci.com/user/languages/python/ (acedido em 13/02/2021).

– (2021b). *Travis CI - Test and Deploy with Confidence*. url: https://travis-ci.com/signin?redirectUrl=https%7B%5C%%7D3A%7B%5C%%7D2F%7B%5C%%7D2Ftravis-ci.com%7B%5C%%7D2F (acedido em 13/02/2021).

Travis CI (2021c). *Travis CI - Test and Deploy Your Code with Confidence*. url: `https://travis-ci.org/` (acedido em 13/02/2021).

Veritis (2021). *Chef Vs Puppet Vs Ansible - Comparison of DevOps Management Tools*. url: `https://www.veritis.com/blog/chef-vs-puppet-vs-ansible-comparison-of-devops-management-tools/` (acedido em 07/03/2021).

Woodall, T (2003). «Conceptualising 'value for the customer': an attributional, structural and dispositional analysis». Em: *Academy of Marketing Science Review* 2003.12. issn: 1526-1794.

Apêndice A

Questionário de avaliação da solução

Para melhor avaliar a solução desenvolvida, foi concebido um formulário para obter o *feedback* de múltiplos colaboradores em relação à mesma, obtendo assim os seguintes resultados. Este questionário foi preenchido no total por 6 colaboradores.

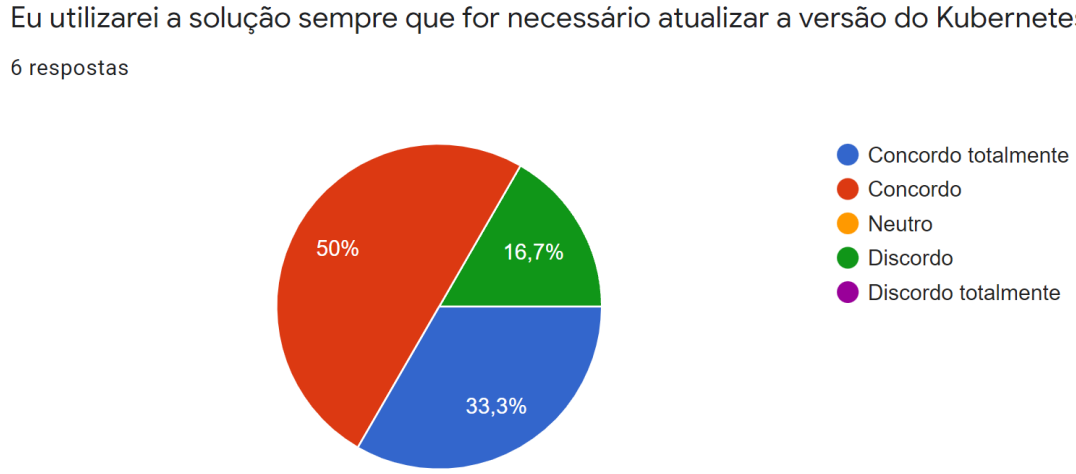


Figura A.1: Primeira questão.

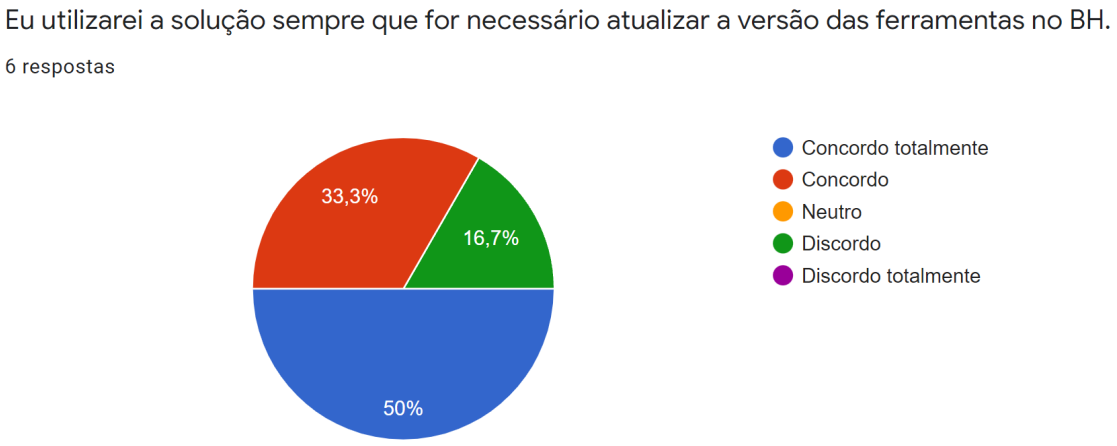


Figura A.2: Segunda questão.

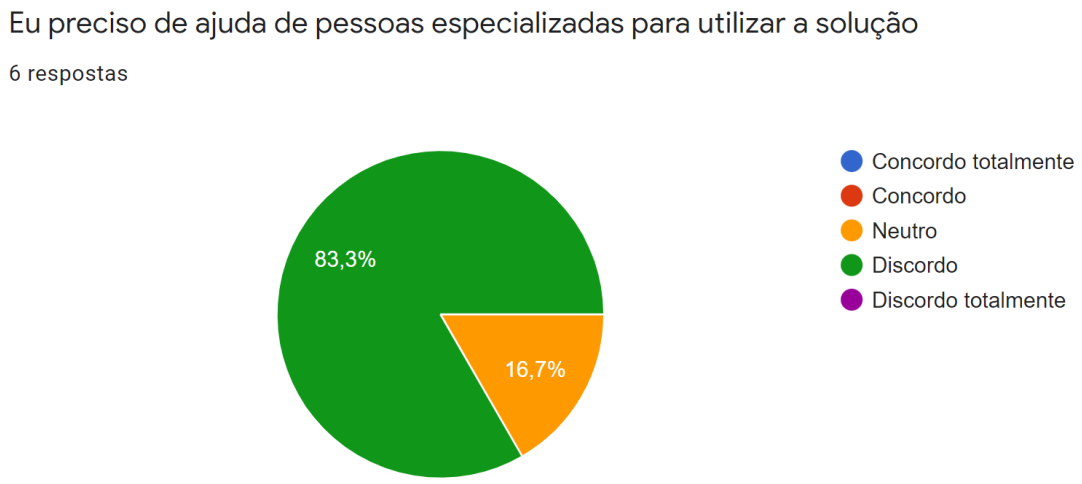


Figura A.3: Terceira questão.

Figura A.4: Quarta questão.

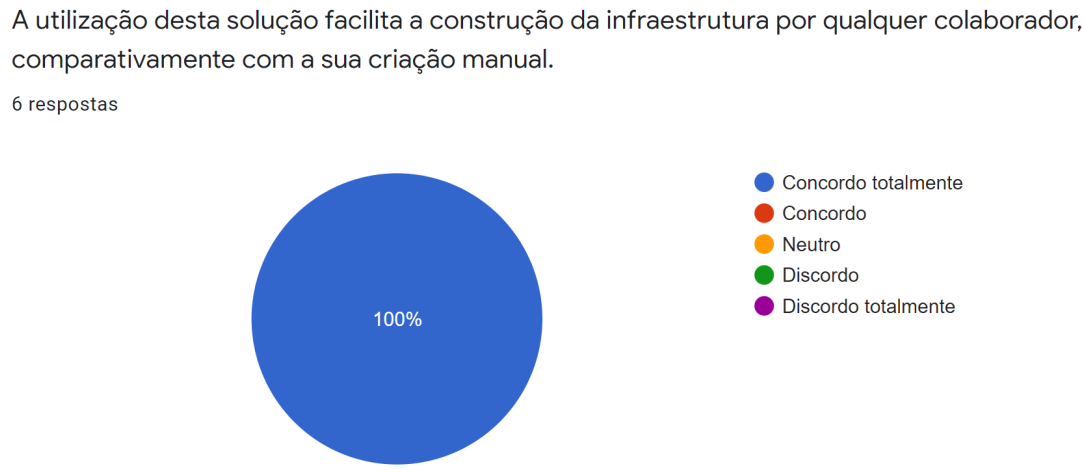


Figura A.5: Quinta questão.