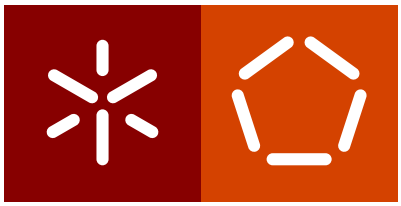
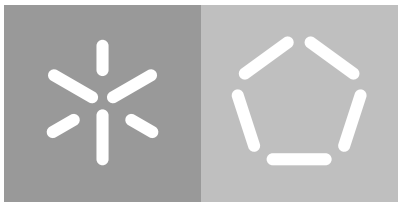


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Tiago Machado Braga

Otimização na alocação de recursos de *cloud computing* num serviço de autenticação de produtos

Fevereiro 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Tiago Machado Braga

Otimização na alocação de recursos de *cloud computing* num serviço de autenticação de produtos

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Dissertação supervisionada por
Professor Doutor José Manuel Ferreira Machado
Doutora Ana Eduarda de Sá e Silva

Fevereiro 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico.

Eu confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

AGRADECIMENTOS

Ao Professor Doutor José Manuel Ferreira Machado pela orientação, paciência, disponibilidade e por partilhar um pouco dos seus vastos conhecimentos comigo.

A toda a equipa da UN1Qnx, que me recebeu da melhor forma e que sempre se mostrou disponível para me ajudar, tanto nas tarefas que me foram atribuídas, como para a realização da presente dissertação. Agradecimento especial à minha coorientadora, Doutora Ana Eduarda de Sá e Silva, que mesmo à distância foi incansável, para que todas as minhas tarefas fossem realizadas e bem sucedidas, prestando apoio esclarecedor. Foi um privilégio sentir-me parte desta equipa e contribuir para o desenvolvimento do seu novo projeto.

À minha família, que sempre me apoiou além dos seus limites. Especialmente ao meus pais, por uma vida de trabalho, que permitiu que hoje esteja aqui a escrever esta dissertação. Sei que não foi fácil, mas espero que hoje estejam orgulhosos.

Aos meus amigos, que me acompanharam nesta jornada, pelo apoio e partilha de conhecimentos. Convosco criei memórias que levarei para a vida.

À minha namorada pelo apoio incondicional. Obrigado por estares sempre ao meu lado.

Por último, à Universidade do Minho, que me acolheu e me formou nestes últimos cinco anos. Foi para mim um privilégio.

RESUMO

A UN1Qnx, S.A., soluções de autenticidade ciber-físicas, é uma empresa sediada em Braga, que desenvolve e comercializa sistemas físicos, eletrónicos e cibernéticos de validação e autenticação de produtos, sendo o objetivo a proteção da marca e o combate à contrafação. Neste momento, a empresa possui um serviço de autenticação de produtos localizado numa máquina virtual na *cloud*, mais especificamente na *Microsoft Azure*. Contudo, a utilização deste serviço é intermitente e passa por períodos de inatividade. Porém, quando utilizado, cada execução do serviço é computacionalmente custosa, o que obriga à utilização de uma máquina virtual que tem em conta o caso de máxima utilização. Assim, nos intervalos entre utilizações os custos acumulam-se sem aproveitar os recursos alocados. Deste modo, esta tese passa por otimizar a utilização dos recursos na *cloud*, tendo em vista tirar proveito da escalabilidade e elasticidade das tecnologias de computação na nuvem, bem como melhorar a latência dos pedidos.

A otimização dos recursos passa por comparar diferentes serviços de diferentes fornecedores e selecionar o que se apresenta como a melhor opção. A fim de realizar estas comparações, fez-se antes uma investigação baseada na metodologia *Design Science Research*.

Primeiramente, explorou-se o ambiente da solução (computação na nuvem) e o ambiente do problema, isto é, qual a situação atual da empresa no que diz respeito ao funcionamento do serviço de validação e dos recursos afetos ao mesmo.

Em segundo lugar, fez-se uma averiguação sobre o estado da arte das tecnologias usadas, das tecnologias que poderiam vir a ser usadas e de outras empresas da mesma área, sobre quais os seus produtos e o seu modo de funcionamento. Por último, investigaram-se métodos de seleção e comparação entre várias opções.

Em terceiro lugar, realizou-se a parte mais trabalhosa e demorada: o desenvolvimento prático. Nesta fase realizaram-se testes de *performance*, a colocação do serviço num *docker container* e a utilização de *kubernetes*. Ainda nesta última parte, houve vária experimentação com diversas arquiteturas. Por fim, o sistema estabilizou numa arquitetura assíncrona, que fez reduzir os custos e, permitiu com que o serviço se adequasse melhor à quantidade de trabalho a processar.

Palavras-chave: Computação na Nuvem, *Cloud*, Otimização, Design Science Research, Escalabilidade, Elasticidade.

ABSTRACT

UN1Qnx, SA, cyber-physical authenticity solutions, is a company headquartered in Braga, which develops and markets physical, electronic and cyber systems for validating and authenticating products, with the aim of protecting the brand and combating counterfeiting . At this moment, the company has a product authentication service located in a virtual machine in the *cloud*, more specifically in *Microsoft Azure*. However, the use of this service is intermittent and goes through periods of inactivity. However, when used, each execution of the service is computationally expensive, which requires the use of a virtual machine that takes into account the case of maximum use. Thus, in the intervals between uses, costs accumulate without taking advantage of the allocated resources. Thus, this thesis involves optimizing the use of resources in *cloud*, with a view to taking advantage of the scalability and elasticity of cloud computing technologies, as well as improving the latency of requests.

Resource optimization involves comparing different services from different providers and selecting the best option. In order to make these comparisons, an investigation based on the *Design Science Research* methodology was carried out.

First, the solution environment (cloud computing) and the problem environment were explored, that is, the current situation of the company with regard to the functioning of the validation service and the resources allocated to it.

Secondly, an inquiry was made about the state of the art of the technologies used, the technologies that could be used and other companies in the same area, about their products and how they work. Finally, selection and comparison methods between various options were investigated.

Thirdly, the most laborious and time-consuming part was carried out: practical development. In this phase, *performance* tests were carried out, the service was placed in a *docker container* and *kubernetes* was started to being used. Also in this last part, there was a lot of experimentation with different architectures. Finally, the system stabilized in an asynchronous architecture that reduced costs and allowed the service to be better suited to the amount of work to be processed.

Keywords: Cloud Computing, Cloud, Optimization, Service selection, Design Science Research, Scalability, Elasticity.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

AHP Analytic Hierarchy Process. 1, 35, 59
AKS Azure Kubernetes Service. 1, 38, 45
API Application Programming Interface. 1, 22, 33, 61
AWS Amazon Web Service. 1, 28

CCU Cloud Harmony Compute Unit. 1, 36
CPU Central Processing Units. 1, 19, 36, 39, 46, 49, 50

DSR Design Science Research. 1, 13, 15

ELECTRE Elimination and Choice Expressing Reality. 1, 36

GB Gigabyte. 1, 36
GCP Google Cloud Platform. 1, 28

HPA Horizontal Pod Autoscaler. 1, 46, 49, 50, 53

IaaS Infrastructure as a Service. 1, 30, 31
IEEE Institute of Electrical and Electronics Engineers. 1, 26
IOP Input/Output Operations. 1, 36

KEDA Kubernetes-based Event Driven Autoscaler. 1, 53, 54

MCDA Multiple Criteria Decision Analysis. 1, 35, 36
ML Machine Learning. 1, 11, 12, 22, 30
MV Máquina Virtual. 1, 8, 9, 12, 24, 38–41, 44, 59, 60

NIST National Institute of Standards and Technology. 1, 26

PaaS Platform as a Service. 1, 31
PROMETHEE Preference Ranking Organization METHod of Enrichment Evaluations. 1, 36

RBAC Role-based access control. 1, 57

SaaS Software as a Service. 1, 28, 31, 32
SO Sistema Operativo. 1, 24, 40

TOPSIS Technique for Order of Preferences by Similarity to Ideal Solution. 1, 36, 37, 59

URL Uniform Resource Locator. 1, 48

vCPUs Virtual Central Processing Units. 1, 24

1

INTRODUÇÃO

Neste capítulo, explicita-se qual o enquadramento e os fundamentos da realização da dissertação, assim como, também, as metas e resultados esperados do trabalho que irá ser levado a cabo.

Impreterivelmente, uma dissertação final de mestrado não poderia ser realizada sem método científico. Portanto, neste capítulo também será especificado a metodologia aplicada.

Por fim, expõe-se a organização do documento.

1.1 CONTEXTO E MOTIVAÇÃO

A UN1Qnx, S.A., soluções de autenticidade ciber-físicas, é uma empresa sediada em Braga e desenvolve sistemas de validação e autenticação de produtos que visam combater a contrafação, protegendo assim as marcas. O produto da empresa consiste numa etiqueta 3D ou 2D gerada aleatoriamente, com o intuito de nem a própria empresa conseguir produzir duas exatamente iguais. No caso 3D, a replicação de uma etiqueta é mais complexa devido a sobreposição de elementos, o que fornece uma segurança extra.

A utilização da etiqueta é bastante simples. Um produto (ex., garrafa de vinho) possui uma etiqueta, que é colocada, por exemplo, na tampa. E a autenticidade do produto é provada através da captura de uma imagem da etiqueta, que é comparada com uma outra imagem tirada após a produção da etiqueta. Um serviço de validação recebe a imagem vinda do utilizador e compara imagens.

Todo o processo descrito implica a existência de vários mecanismos informáticos, e a UN1Qnx recorre à *cloud* pública como meio de implementação. O serviço de validação destaca-se pelo seu custo computacional de execução, adicionando-se a isto limitações no que diz respeito ao tempo de resposta, que terá de ser curto. Este programa realiza comparações entre imagens de etiquetas e analisa o grau de similaridade entre as etiquetas. Facilmente se pode presumir que ambas as capturas de imagem irão ser feitas em ambientes diferentes de iluminação, ângulo, etc. Estas dificuldades são combatidas com o uso de um programa de Machine Learning (ML).

Neste momento, o programa encontra-se a ser executado numa Máquina Virtual (MV) na plataforma *Microsoft Azure*. A MV, não só necessita de estar sempre ligada de forma a garantir o serviço, como também é forçado que seja uma máquina com grande capacidade de processamento, de maneira a realizar os cálculos necessários num período de tempo útil, para todos os pedidos que lhe são feitos. Isto resulta numa máquina ajustada para o número máximo de utilizadores. Acontece que o serviço passa por períodos de inatividade, o que resulta em recursos alocados não utilizados nos quais são feitos investimentos. No futuro, prevê-se que a requisição do serviço aumente, o que resultaria na alocação de uma máquina maior em termos de capacidade de processamento, o que acentua a inviabilidade do estado atual. Assim, é necessário que se façam otimizações, no sentido de reduzir os custos e tirar proveito dos benefícios que as tecnologias *cloud* têm para oferecer como, por exemplo, a escalabilidade e a elasticidade. Escalabilidade pode ser definido como a capacidade de um sistema para acomodar um número de crescente de elementos, de processar uma quantidade crescente de trabalho, ou ser capaz de aumentar a sua capacidade (Bondi, 2000, p.195). Por outro lado, a elasticidade é o grau com que um sistema consegue aumentar e diminuir automaticamente os recursos alocados, de forma a que em qualquer altura esses recursos alocados estejam ajustados da melhor maneira possível às exigências do momento (Herbst et al., 2013, p.24).

Poder-se-ão otimizar outros recursos, para além dos que ao serviço de ML dizem respeito, tendo em conta sete modos principais de otimizar os custos (Azure, 2020b) :

1. Encerrar recursos não utilizados;
2. Redimensionar recursos subutilizados;
3. Reservar instâncias;
4. Tirar partido do Benefício Híbrido do *Azure*;
5. Configurar o dimensionamento automático;
6. Configurar orçamentos e alocar custos a projetos e equipas;
7. Escolher o serviço de computação *Azure* certo.

A otimização dos recursos da empresa poderá passar por vários destes tópicos.
Deste modo, é com base neste contexto que se insere a presente dissertação.

1.2 OBJETIVOS E RESULTADOS ESPERADOS

Por consequência do estado atual da infraestrutura da empresa e *performace* do serviço de validação haverá dois obejctivos principais.

Em primeiro, os recursos alocados terão de ser revistos de forma a que sirvam as necessidades da empresa dinamicamente. Isto é, tirar proveito das tecnologias de escalabilidade e elasticidade mais recentes na área de computação na nuvem pública. Em adição, e por consequência, o custo deve ser adequado à utilização do serviço. Ou seja, deve aumentar consoante a carga de trabalho, mas deve baixar rapidamente caso a carga de trabalho também baixe.

Em segundo lugar, os tempos de processamento também devem baixar, de forma a que um utilizador tenha de esperar menos tempo pelo resultado da sua validação. Esta descida no tempo de validação também teria um impacto significativo e positivo na quantidade de pedidos que o sistema seria capaz de processar por unidade de tempo

Assim, todo o trabalho realizado nesta dissertação passa por otimizar a utilização de recursos *cloud*, por parte do serviço de validação, com o objetivo de tornar simultaneamente a alocação de recursos elástica e o processamento de pedidos mais rápido.

1.3 MÉTODOS E TÉCNICAS

1.3.1 *Design Science Research*

É de suma importância que um projeto como uma dissertação de mestrado siga determinados procedimentos, pois a validade e a confiabilidade de um estudo dependem dos objetivos e de uma metodologia confiável e repetitiva (Garg, 2016, p.640), de forma a que seja construída com rigor e que seja feito um artefacto/s útil para a área em que o problema reside.

Para o efeito, foi selecionada a metodologia de investigação Design Science Research (DSR), representada na Figura 1.

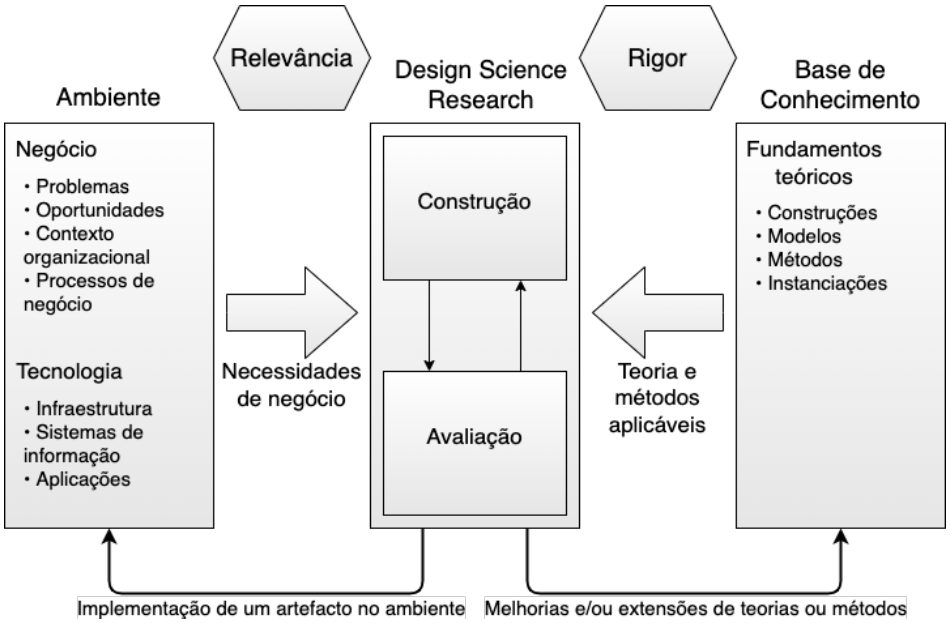


Figura 1: *Design Science Research Framework.* Adaptada de Hevner et al. (2004).

Ambiente

O ambiente pode ser descrito como a área ou espaço do problema e nele residem os problemas, oportunidades, contexto da organização e os processos do negócio (Hevner et al., 2004, p.79). No caso a ser tratado, nesta dissertação, o ambiente do problema, é apresentado no "Contexto e Motivação" (1.1) e é descrito com ainda mais pormenor no Capítulo 2. Através de todos os pontos explicitados na componente ambiente na Figura 1, extraem-se não só requisitos para a investigação que vai ser levada a cabo, mas também critérios para avaliar os resultados da investigação (Hevner, Alan and Chatterjee, Samir, 2010, p.17).

Base de Conhecimento

A base de conhecimento fornece todos os materiais com os quais se vai construir o artefacto e, através dos quais, é possível avaliar esse mesmo artefacto. Os materiais que compõem a base de conhecimento são fundações teóricas, metodologias de investigação, construções modelos, métodos e instanciações (Hevner et al., 2004, p.82), que tipicamente num documento desta espécie se encontram no "Estado da Arte".

Para selecionar esta base de conhecimento foi desenvolvida uma estratégia explicitada na Subsecção 1.3.2.

Design Science Research

No segmento central da Figura 1 surge Design Science Research (DSR). O que a distingue de outras metodologias, nomeadamente, de *routine design* é a procura por encontrar soluções novas e não a aplicação das "melhores práticas". Daí, eleva-se a importância da avaliação (Hevner et al., 2004, p.83) do artefacto, que vai sendo iterativamente construído e moldado.

Segundo Hevner et al. (2004), recolher-se-ão todos meios através dos quais se poderá resolver o problema, determinar a sua utilidade e restrições e caracterizar as constantes de custo e benefício. Por exemplo, para construir uma casa resistente a condições climatéricas adversas e energicamente eficiente seriam enumerados e representados todos os materiais ou combinações destes, definida a sua utilidade e restrições e especificar qual o seu custo. Sem grande análise, revela-se como uma tarefa hercúlea e demasiado exaustiva. De forma análoga, para investigar uma solução para o problema em mão ter-se-ia de elencar toda a infraestrutura capaz de albergar um serviço *web*. De forma a eliminar este problema, a melhor solução passará pela identificação daquela que está mais próxima da solução ótima.

A avaliação visa verificar se o artefacto elaborado vai de encontro com os requisitos levantados, o que permite por sua vez refinar o artefacto.

Ciclo de Relevância

O ciclo de relevância providencia à DSR um contexto aplicacional, onde não estão apenas presentes requisitos para a investigação, mas também critérios de aceitação para a avaliação do artefacto. As iterações neste ciclo irão sistematicamente testar o artefacto no ambiente do problema, o que se pode chamar de teste de campo, e indicar quais os ajustes ou qual o caminho a seguir relativamente à construção do artefacto (Hevner, Alan and Chatterjee, Samir, 2010, p.17).

Ciclo de Rigor

O ciclo de rigor fornece conhecimento pretérito basilar para que a metodologia seja aplicada rigorosamente, o que garantirá ao artefacto gerado a sua inovação. Neste ciclo, procurar-se-á conhecimento prévio, que serve de ponto de partida para a construção do artefacto, mas também métodos para avaliar o mesmo, em função dos requisitos e restrições capturados no ciclo de relevância. Contudo, não se trata apenas de explorar conhecimento e aplicá-lo à DSR: os resultados da DSR também servirão de adição ou extensão às teorias existentes (Hevner, Alan and Chatterjee, Samir, 2010, p.18).

1.3.2 *Pesquisa Bibliográfica*

De forma a cumprir o ciclo de rigor definido em DSR, foram traçadas estratégias.

Previamente ao início da pesquisa por fontes bibliográficas, foi necessário escolher quais as fontes de informação. As fontes de informação selecionadas foram: *Google Scholar*, *IEEE Xplore*, *Science Direct* e *ResearchGate*.

A próxima etapa passaria por organizar a coleta de informação e os dados resultantes dessa coleta. Para tal, elaborou-se uma estratégia, representada na Figura 2.

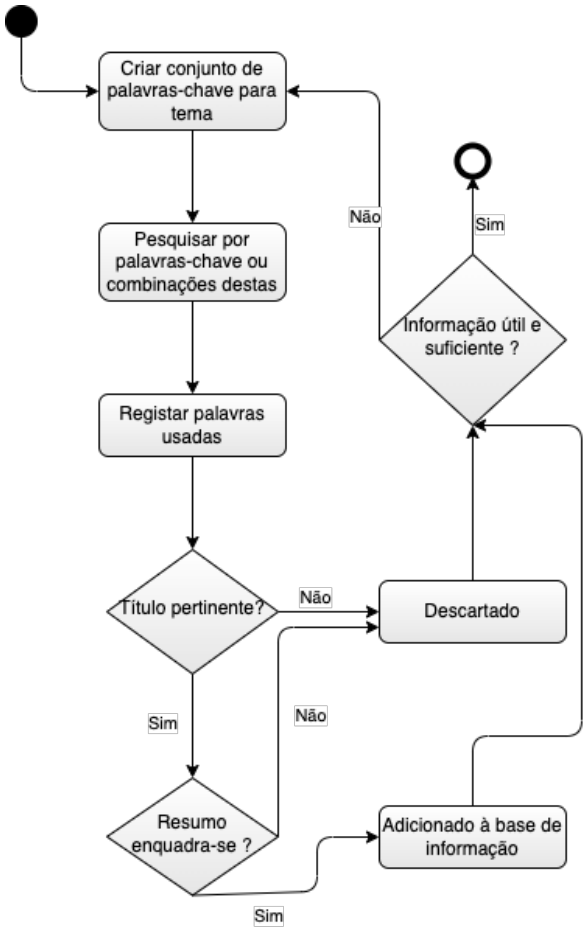


Figura 2: Estratégia de pesquisa bibliográfica definida.

A estratégia definida consiste em manter registo das palavras-chave criadas, utilizadas, e criar uma coleção de fontes bibliográficas variadas, selecionando os resultados que aparentam ser mais pertinentes nas pesquisas realizadas. O objetivo seria, não só criar uma coleção de dados, como também manter registo do que já foi explorado, acrescentando rigor e método ao trabalho desempenhado.

1.3.3 *Testes de Performance*

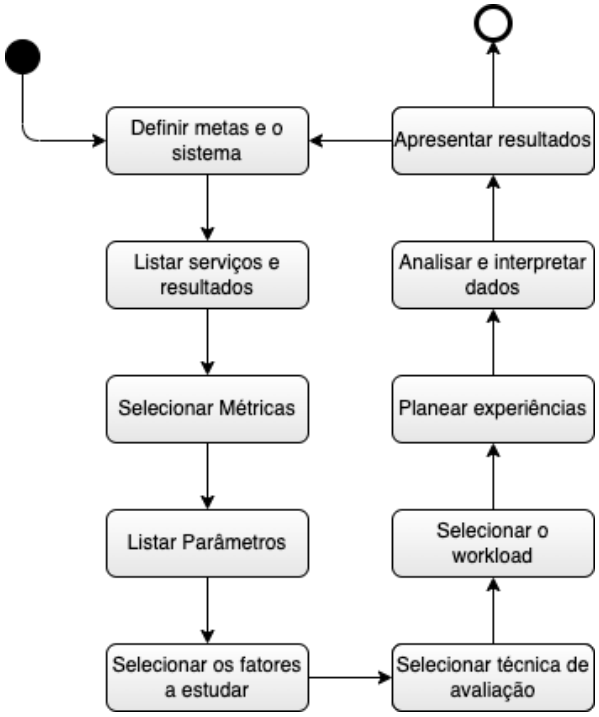


Figura 3: Metodologia de elaboração de testes de *performance*.

A figura 4 representa a metodologia delineada segundo Jain, Raj (1991). Segundo o autor, a metodologia consiste em dez passos cruciais, não só para o planeamento e realização de testes de *performance*, como também para a análise e extração de conclusões dos mesmos.

A metodologia pode ser apresentada em duas fases distintas: pré-teste e pós-teste.

A primeira fase, e a mais longa, consiste em toda a preparação dos testes. Encontram-se elencados de seguida os passos e a sua descrição:

1. **Definir metas e o sistema:** consiste em definir os objetivos do estudo e determinar o que constitui o sistema, através da estipulação de quais são as fronteiras do sistema (Jain, Raj, 1991, p.49).

2. **Listar serviços e resultados:** este passo consiste em registar os serviços que o sistema em análise fornece e quais os seus resultados. Por exemplo, o sistema de validação em análise responde a pedidos *HTTP* e a resposta poderá ser desejável ou não. O serviço poderá responder corretamente (etiqueta válida/inválida), incorretamente, devido a um erro no servidor ou algo semelhante, ou poderá não responder de todo, pois o servidor poderá estar indisponível. Uma falha de rede poderá acontecer ou qualquer problema que impossibilite o pedido de chegar ao destinatário ou a resposta de alcançar o remetente (Jain, Raj, 1991, p.49).

3. **Selecionar métricas:** este passo consiste em discriminar critérios com a finalidade de comparar o desempenho e a estes critérios dá-se o nome de métricas (Jain, Raj, 1991, p.50). Se um serviço funcionar corretamente, o seu desempenho é avaliado pelo tempo que demora a executar tarefas, pelo ritmo com que as executa e pelos recursos consumidos aquando dessa execução. A estas três métricas dá-se o nome de responsividade, produtividade e utilização, respetivamente. O sistema também poderá funcionar incorretamente e, nesse caso, diz-se que ocorreu um erro. Neste caso, é importante classificar os erros e determinar as probabilidades de cada classe de erros. O sistema também poderá não dar resultado nenhum, o que se traduz numa falha do mesmo. Mais uma vez, é importante classificar os casos de falha e determinar as probabilidades com que tal acontece. A estes três casos gerais denominam-se por rapidez, confiabilidade e disponibilidade como se pode ver na figura 4 (Jain, Raj, 1991, p.63).

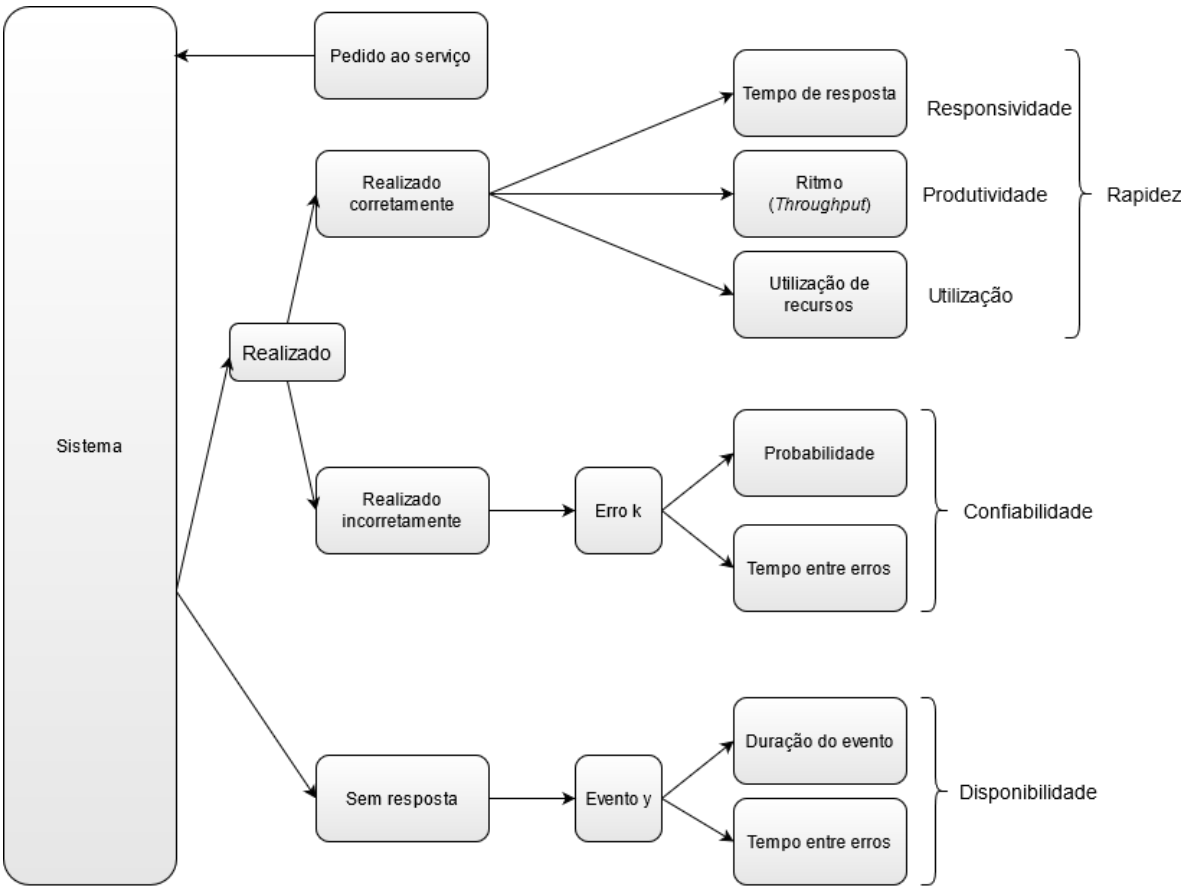


Figura 4: Tipos de resultado de um serviço. Adaptada de Jain, Raj (1991).

4. **Elencar parâmetros:** a listagem de parâmetros consiste em anotar todos os aspetos que poderão influenciar a *performance* dos serviços do sistema. Nesta listagem não estão

apenas incluídas particularidades do sistema em análise, mas também características de *hardware* e *software* do utilizador do serviço. Por exemplo, rapidez do CPU local e rapidez do CPU remoto (Jain, Raj, 1991, p.50).

5. **Selecionar fatores a estudar:** da lista de parâmetros elaborada no passo anterior, pode-se fazer a distinção entre parâmetros que vão variar durante a avaliação e os que não se variam. Aqueles que irão variar chamam-se de factores, e os seus valores denominam-se níveis. A título de exemplo, ao avaliar um servidor *web* pode-se variar o número de utilizadores a realizar pedidos (fator). Para o efeito, podem-se criar *workloads* com dois níveis de utilizadores: pequeno e grande (Jain, Raj, 1991, p.50).

6. **Selecionar técnicas de avaliação:** existem três técnicas gerais de avaliação de desempenho: modelação analítica, simulação e avaliação de um sistema real. A modelação analítica passa por criar modelos matemáticos do sistema e prever os resultados através da resolução de equações. A simulação é uma aproximação mais prática do problema. Nesta situação, a simulação poderá ser concretizada através do uso de programas de simulação ou da elaboração de um sistema protótipo, que, por exemplo, poderá simular apenas tempos de reposta. Por último, temos a avaliação num sistema real, que, como o nome indica, consiste em testar um sistema já desenvolvido completamente (Jain, Raj, 1991, p.52). A consideração chave que se deve ter na escolha da técnica de avaliação é a etapa do ciclo de vida em que sistema se encontra.

7. **Selecionar o *workload*:** o *workload* ou carga de trabalho consiste num conjunto de pedidos ao serviço do sistema. O *workload* depende em grande parte da técnica de avaliação escolhida. Exemplificando, para uma modelação analítica a carga de trabalho poderá ser a probabilidade da receção de pedidos, enquanto que numa avaliação a um sistema real geralmente são elaborados *scripts*, que serão executados no sistema alvo ou numa máquina externa, onde se realizará pedidos ao sistema. A carga de trabalho deverá ser semelhante à carga que o sistema teria numa situação real (Jain, Raj, 1991, p.52).

8. **Planear experiências:** assim que estão reunidas a lista de fatores e os seus níveis, é necessário programar qual a sequência de experiências, de forma a extrair o máximo de informação com o mínimo de esforço. É comum numa primeira instância realizar a experiência com um elevado número de fatores e um número reduzido de níveis e depois, se necessário, refazer a experiência com os fatores mais impactantes e uma quantidade superior de níveis (Jain, Raj, 1991, p.52).

A segunda fase, com menor número de passos, mas, não obstante, menos trabalhosa, é a fase de interpretação dos dados e apresentação de conclusões.

1. **Analisar e interpretar os dados:** um dos passos mais importantes é precisamente a compreensão dos dados resultantes da experiência. No entanto, é necessário ter em conta que os resultados podem variar de experiência para experiência. Portanto, é necessário aplicar alguma metodologia estatística, de forma a retirar conclusões acertadas (Jain, Raj, 1991, p.52).

2. **Apresentar resultados:** por fim, a último etapa é referente à apresentação dos dados, ou seja, são comunicados os resultados ao resto da equipa, mais especificamente aos membros que estão encarregues da tomada de decisões (Jain, Raj, 1991, p.53).

Contudo, se for necessário extrair mais resultados e preparar novos testes deve-se voltar a rever todo o planeamento. Daí a metodologia ser cíclica.

1.4 Organização do Documento

A presente dissertação encontra-se dividida em 4 capítulos. No Capítulo 1, apresenta-se o contexto e os motivos que levaram a que o tema fosse criado e proposto, os objectivos e resultados esperados no final da dissertação; apresentam-se métodos e técnicas desenvolvidas para organizar quer a pesquisa bibliográfica quer a comparação de soluções.

No segundo capítulo, apresentar-se a situação atual dos recursos da empresa. Especificam-se os recursos utilizados, quer de hardware e software, e o funcionamento do serviço de similaridade de imagens.

No terceiro capítulo, encontra-se o estado da arte de várias áreas de interesse. Cada área possui a sua secção e a primeira é *Cloud Computing* (3.1), pois é o ambiente onde reside a solução para a problemática em questão. Nesta secção, é explicitado o conceito e clarifica-se, não só sobre os modelos de implementação, como também sobre os modelos de serviço. Na segunda secção, são explorados outros serviços/empresas, no domínio das tecnologias de autenticação e o seu funcionamento. Numa terceira secção, são discutidos métodos de decisão entre vários serviços.

No quarto capítulo, descreve-se todo o trabalho prático levado a cabo durante a escrita desta dissertação. Neste capítulo são explicados os testes que foram levados a cabo, a colocação do serviço de validação num *docker container* e a utilização de *kuberntes*, assim como as diversas arquiteturas utilizadas.

No quinto e último capítulo, faz-se uma reflexão sobre o trabalho realizado e pondera-se sobre o trabalho a realizar futuramente.

2

ENQUADRAMENTO DA UN1QNX

O estudo do ambiente do problema demarca-se como um passo crucial na resolução desse mesmo problema. Um planeamento efetivo requer um conhecimento, tanto do domínio da aplicação (ambiente do problema) e do domínio da solução (Hevner et al., 2004, p.85).

Na Secção "Contexto e Motivação"(1.1) da introdução o problema já foi exposto, todavia, neste capítulo, irão ser explorados mais detalhadamente alguns aspetos do domínio do problema.

2.1 FUNCIONAMENTO DO SERVIÇO DE VALIDAÇÃO

As etiquetas produzidas não servem apenas para provar a autenticidade do produto associado ao consumidor final. Também são utilizadas para provar a genuinidade do produto ao longo do seu ciclo de vida, por exemplo, antes e depois do seu armazenamento, o produto é suscetível de ser validado, a fim de averiguar a ocorrência de fraude. Na Figura 5 é possível visualizar esta possível utilização.

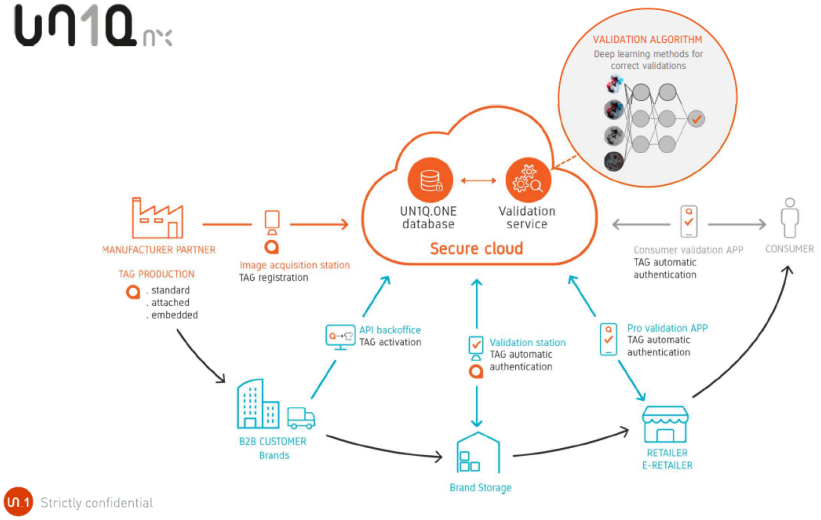


Figura 5: Funcionamento do serviço de autenticação. Fornecida pela UN1Qnx.

Todos os pedidos de validação realizados são enviados a um serviço disponibilizado através de uma Application Programming Interface (API) (*Validation service* na Figura 5). Um *Windows Server* atende os pedidos e utiliza um executável *Python*, que procede ao pré-processamento da imagem e, de seguida, à utilização do modelo de ML, que devolve o grau de similaridade com a imagem tirada à etiqueta com o mesmo código de barras da imagem capturada pelo utilizador. Este processo pode ser observado na seguinte Figura (6).

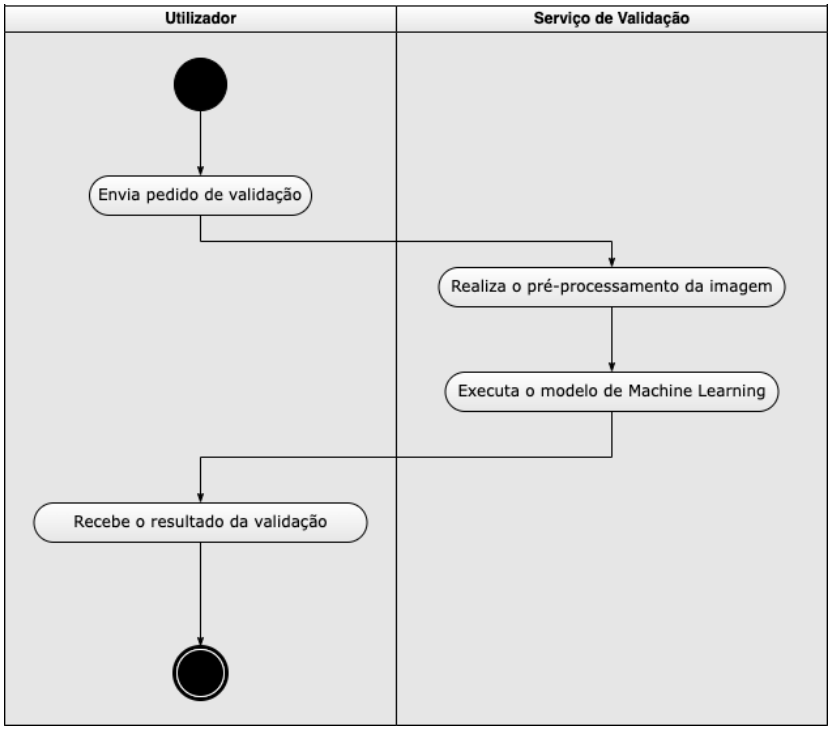


Figura 6: Serviço de validação.

2.1.1 *Ciclo de Vida de uma Etiqueta*

O ciclo de vida destas etiquetas segue, geralmente, os seguintes contornos. Tudo começa com a produção das mesmas, seguindo-se o seu registo através da captura de imagens da etiqueta, com recurso a um sistema ótico dedicado, como se pode ver na Figura 7. A etiqueta poderá ser 2D ou 3D, que serão associadas a produtos de gamas de preços mais baixos ou mais altos, respetivamente.

Figura 7: Etiqueta 3D produzida pela *UN1Qnx*. Fornecida pela *UN1Qnx*.

De seguida, a etiqueta é associada a um determinado produto, sendo que, enquanto a etiqueta permanecer afiliada ao produto, é suscetível de ser validada (através da captura de uma imagem com recurso a um *smartphone*), verificando-se assim a autenticidade do produto. Posteriormente, no ato de venda ou noutro evento, a etiqueta poderá ser desativada, tornando-se assim inválida. Na Figura 8 pode ver-se uma representação gráfica das várias etapas.

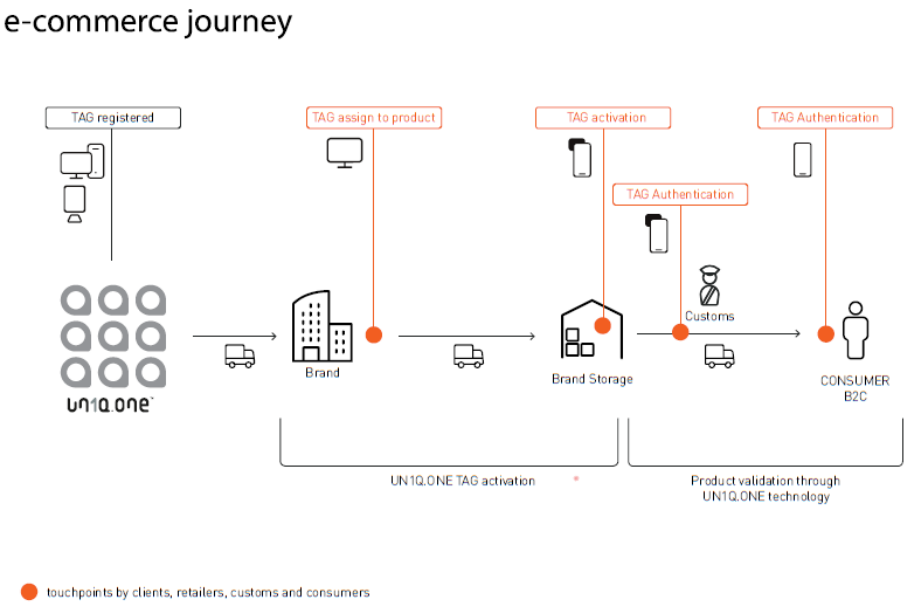


Figura 8: Ciclo de vida de uma etiqueta. Fornecida pela UN1Qnx.

2.1.2 *Recursos*

A empresa escolheu a *Microsoft Azure* para providenciar serviços de computação na nuvem. Seguidamente, apresentam-se o resultado do levantamento dos recursos e tecnologias usadas no serviço de validação.

Hardware

Os recursos fisícos encontram-se num *datacenter* da *Microsoft Azure* na região "*West Europe*". O serviço de validação encontra-se numa MV com as especificações que se encontram na Tabela 1, o que resulta também em recursos virtuais, como vCPUs. Neste caso, apesar de serem usados vCPUs, estes são baseados em processadores *Intel Xeon® Platinum 8168 (SkyLake)* de 2,7 GHz (Azure, 2020a).

Tabela 1: Dados e Especificações da MV atual.

Tamanho	*Standard F4s_v2*
SO	*Windows (Windows Server 2019 Datacenter)*
Região	*West Europe*
vCPUs	4
RAM (GB)	8
MAX IOPS	6400
Preço (€/mês)	227.43

Software

Um pedido que chega é atendido por um servidor do tipo *Windows Server* e invoca um executável, gerado a partir de um *script* escrito em *Python*. Este executável é também ele um servidor que realiza o pré-processamento da imagem e indica qual o grau de similaridade.

2.2 PROBLEMA

O problema reside em dois pontos:

- Consumir recursos quando não utilizado;
- Não suportar cargas crescentes de trabalho;

Segundo Hevner et al. (2004) (p.89), a procura de uma representação eficiente para o problema é crucial para encontrar também uma solução eficiente. Para o efeito, na Figura 9

encontramos uma representação gráfica do que é o problema e na Figura 10 a representação daquilo que seria a solução ótima.

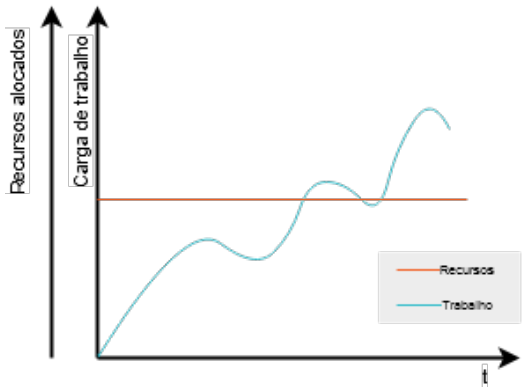


Figura 9: Gráfico representativo do problema.

A solução seria uma que se adaptasse os recursos alocados à carga de trabalho do momento. O que resultaria também em custos dinâmicos e proporcionalmente diretos à quantidade de recursos alocada, em vez de custos constantes.

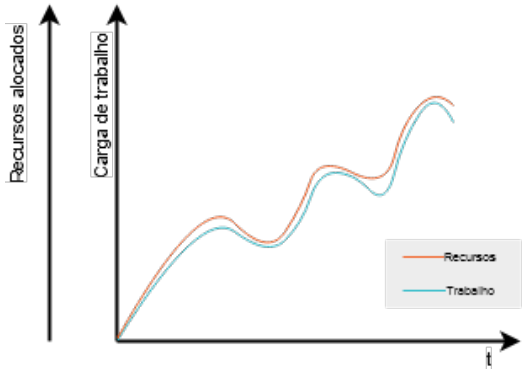


Figura 10: Gráfico representativo da solução ideal.

3

ESTADO DA ARTE

3.1 *Cloud Computing*

Devido ao facto de a empresa adotar a estratégia de implementar todos os seus serviços em recursos na *cloud* pública, esta surge como sendo o domínio da solução, isto é, o espaço onde será construído o artefacto que resolve ou otimiza o serviço já existente. Como tal, foi feita uma exploração desta área.

3.1.1 *O que é Cloud Computing ?*

Há várias definições formais do que é a computação na nuvem:

Tabela 2: Definições de *cloud computing*

Nome	Definição
NIST	"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."(Mell and Grance, 2012, p.100)
IEEE	"It is a paradigm in which information is permanently stored in servers on the internet and cached temporarily on clients that include desktops, entertainment centers, table computers, notebooks, wall computers, handhelds, etc."(Abbasov, 2014, p.1)

Google	"In cloud computing, the capital investment in building and maintaining data centers is replaced by consuming IT resources as an elastic, utility-like service from a cloud “provider” (including storage, computing, networking, data processing and analytics, application development, machine learning, and even fully managed services). Whereas in the past cloud computing was considered the province of startups and aggressively visionary enterprise users, today, it is part of the enterprise computing mainstream across every industry, for organizations of any type and size."(Google, 2020)
Amazon	"Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS)."(Amazon, 2020)
Microsoft Azure	"Simply put, cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping you lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change"(Microsoft, 2020)
Eric Schmidt (CEO of Google Inc)	"It starts with premise that the data services and architecture should be on the servers. We call it Cloud Computing- they should be in a “CLOUD” somewhere."(Abbasov, 2014, p.1)

A computação na nuvem demarca-se como sistemas de informação, capazes de processar informação, armazená-la ou ainda analisá-la. Disponíveis para os membros de uma organização ou ao público geral através de uma rede, por exemplo, da Internet, e que são servidos de três formas distintas, consoante o grau de responsabilidade e controlo.

Breve História

É possível rastrear o conceito de "Computação na Nuvem"até 1961 por John McCarthy no MIT:

> "Se os computadores do tipo que eu tenho defendido se tornarem os computadores do futuro, então a computação talvez um dia se organizará como uma utilidade pública tal como o sistema telefónico é uma utilidade pública... A utilidade de computação poderá tornar-se na base de uma nova e importante indústria." (Surbiryala and Rong, 2019, pp.1-2)

A evolução de computação na nuvem tem as suas raízes em sistemas mais antigos e que foram utilizados muito antes de computação na nuvem existir (Surbiryala and Rong, 2019, p.1). Dois desses sistemas são:

- ***Utility Computing*:** Por volta de 1960, como os preços de processamentos eram elevados, portanto decidiram partilhar os recursos computacionais. Desta forma, os utilizadores teriam acesso a recursos e apenas pagavam os recursos pelo tempo em que os utilizaram (Surbiryala and Rong, 2019, p.2). *Utility Computing* pode ser definido como entrega, conforme a necessidade, de infraestrutura, aplicações e processos de negócio de uma forma segura, partilhada, escalável e baseada em ambientes computacionais baseados em padrões, através da *Internet*, pagando um taxa (Smith et al., 2006, p.14).

- ***Grid Computing*:** com a origem do conceito no princípio dos anos 90 (L. Bote-Lorenzo et al., 2004, p.291) , *grid computing* consiste em resolver problemas computacionalmente pesados, dividindo o problema em vários problemas menores e distribuindo estes por máquinas de baixa *performance* (Surbiryala and Rong, 2019, p.2).

No que diz respeito à indústria, a Salesforce, uma empresa que produz software de gestão de relações com o cliente (Salesforce, 2020) foi das primeiras empresas a trabalhar com o conceito de computação na nuvem no final dos anos 90, oferecendo o seu produto segundo o modelo de *software* como uma plataforma (SaaS) (Surbiryala and Rong, 2019, p.2).

Atualmente, a situação de computação evoluiu consideravelmente, sobretudo no que diz respeito a computação na *cloud* pública. São várias as empresas que providenciam serviços, sendo que as três principais são a Amazon Web Service (AWS), a *Microsoft Azure* e a Google Cloud Platform (GCP).

3.1.2 *Modelos de Implementação*

A computação na nuvem manifesta-se, essencialmente, em quatro tipos de implementação. Os modelos de implementação variam consoante o grau de disponibilidade ao público, podendo fazer-se uso de mais do que um modelo.

- ***Private Cloud*:** neste modelo de implementação a infraestrutura da *cloud* pertence apenas a uma organização, podendo ser gerida pela mesmo ou por terceiros. As

razões para utilizar este tipo de modelo são várias. Pode ser utilizado este modelo apenas para maximizar a utilização de recursos existentes na organização. Também pode ser utilizado para ir ao encontro de requisitos de segurança. Por exemplo, se a organização lida com dados sensíveis, não é desejável que estes sejam processados em outros sistemas informáticos (Abbasov, 2014, p.2). A redução de custos a longo prazo também será uma das razões pelo qual uma organização poderá utilizar este modelo (Surbiryala and Rong, 2019, p.4).

- ***Public Cloud*:** neste modelo os serviços são providenciados por fornecedores *cloud*. Os seus serviços estão publicamente disponíveis e podem ser acedidos através da *internet*, desde através da forma mais conhecida de navegar na Internet, isto é, pelo *browser*, até à linha de comandos. Geralmente o modelo de pagamento é *pay-per-use*, isto é, o utilizador paga pelo que usa, pelo tempo que usa.

- ***Hybrid Cloud*:** chama-se *hybrid cloud*, quando se utiliza dois ou mais modelos de implementação. É utilizado quando, por exemplo, uma organização possui informação que não é crítica ou de menor importância, pelo que poderá então ser processada na *cloud* pública, deixando assim a *cloud* privada disponível para tarefas mais críticas.

- ***Community Cloud*:** por vezes organizações com objetivos semelhantes podem partilhar recursos, dividindo assim custos. É possível comparar a *community cloud* com a *cloud* privada, uma vez que é apenas utilizada por um círculo pequeno e privado de organizações (Surbiryala and Rong, 2019, p.4).

Em suma, se os recursos estiverem disponíveis apenas aos colaboradores de uma empresa, trata-se de uma *cloud* privada. Caso contrário, se os serviços estiverem disponíveis ao público em geral, trata-se de uma *cloud* pública. No entanto, por exemplo, uma empresa pode utilizar os seus recursos locais (*cloud* privada) e exportar algumas das suas atividades para a *cloud* pública, resultando numa combinação das duas, isto é, numa *cloud* híbrida. Por último, como forma de redução de custos, empresas podem criar uma comunidade, montar um sistema de informação, partilhado nessa comunidade (*community cloud*).

3.1.3 *Cloud Pública: Exemplos na indústria*

Atualmente, as empresas podem seguir dois caminhos no que toca na adoção de tecnologias: esperar para que as tecnologias melhorem ou arriscar e cometer erros (Silva et al., 2018). Contudo, as empresas enveredam em maior escala pela última opção, pois apenas uma pequena minoria de empresas não utiliza qualquer tipo de tecnologia na nuvem. Uma vez que, para obter dimensão e alcance, por exemplo, possuir um *website* é imprescindível. E as evidências deste crescimento e utilização massiva de tecnologias podem ser vistas, por

exemplo, na evolução de legislação e direito do mundo virtual (Andrade et al., 2005) e até na possibilidade de tratar dispositivos eletrónicos como pessoas jurídicas (Andrade et al., 2007).

São várias as aplicações das tecnologias na *cloud* e as empresas que as utilizam. Tendo em conta que um dos objetivos do presente relatório é a exploração de tecnologias *cloud*, segundo um modelo de implementação público, expõem-se de seguida alguns exemplos da utilização do mesmo na indústria.

O Millenium BCP (Banco Comercial Português) é um banco português e utiliza o *Microsoft Azure* com a finalidade de realizar os seus *workloads* e para hospedar a sua aplicação. Faz uso de um *cloud* híbrida, pois, sendo um banco, processa informação sensível, que é tratada na nuvem da organização. Contudo, informação que não é confidencial é tratada na nuvem pública. Também é relevante referir que a aplicação móvel, que faz amplo uso da *cloud*, deste banco foi premiada pela *Global Finance* com o prémio de "*Best Digital Bank Award* 2020"(Cebrian et al., 2020).

A Farfetch, uma plataforma que opera no mercado da moda de luxo, também emprega cloud pública (*Microsoft Azure*). Sob a superfície do seu *website*, a Farfetch faz um uso muitissímo variado de ML, para que o utilizador tenha uma experiência personalizada e ao nível dos produtos que compra na plataforma. A título de exemplo: preferências têm impacto nos resultados de pesquisa e no que é sugerido ao utilizador; se um utilizador pesquisar por "vestido vermelho", existe um processo que irá sugerir produtos, que do resultado da análise das suas imagens sejam extraídas tais palavras-chave; a plataforma vende peças de vestuário com *stock* bastante limitado e que são vendidas, tanto na plataforma como na boutique física, e, para que um utilizador *online* não compre um produto que já está esgotado, também existe um modelo que previne estas situações. Estas são apenas algumas utilizações de ML na Farfetch (Batista et al., 2020).

3.1.4 *Modelos de Serviço*

Generalizando, todas os modelos de implementação de computação na nuvem servem para serem servir os seus utilizadores. Contudo, a forma como são utilizados e se apresentam ao utilizador variam. Existe três modelos de serviço e encontram-se elencados de seguida.

- **IaaS:** O utilizador tem acesso a recursos computacionais diversos, geralmente a recursos de processamento, armazenamento e *networks*, entre outros (Miyachi, 2018, p.7). A virtualização é usada frequentemente, de forma a possibilitar o aumento ou diminuição, de uma maneira *ad-hoc*, do uso de recursos físicos, a fim de ir de encontro com as necessidades (Abbasov, 2014, p.2). No entanto, o consumidor não realiza a gestão de nenhuma infraestrutura, mas tem o controlo, por exemplo, sobre sistemas operativos, armazenamento e sobre certos aspetos de *networking* (Miyachi, 2018, p.7). Na Figura

11, poder-se-á ver com mais detalhe as responsabilidades do utilizador/*cloud provider*, no que diz respeito a IaaS.

- **PaaS:** É a capacidade dada ao consumidor de poder implementar e lançar para produção aplicações criadas pelo mesmo. É de notar que consumidor apenas pode utilizar linguagens de programação, livrarias, serviços e ferramentas suportadas pelo fornecedor de *cloud*. Neste modelo de serviço, o consumidor não tem controlo sobre sistemas operativos, servidores e armazenamento, como no IaaS. Contudo, pode ter controlo sobre algumas configurações do ambiente em que está hospedada a aplicação (Miyachi, 2018, p.7).
- **SaaS:** Ao consumidor é dada a capacidade de utilizar aplicações que estão em funcionamento numa infraestrutura de *cloud*. O consumidor acede a estas aplicações quer através do navegador *web* ou através de programas. Neste contexto, o consumidor realiza apenas a gestão de algumas configurações específicas para o mesmo (Miyachi, 2018, p.7).

As principais diferenças entre modelos de serviço devem-se, essencialmente, ao grau de gestão feita quer pelo utilizador, quer pelo fornecedor de recursos, como se pode visualizar na Figura 11.

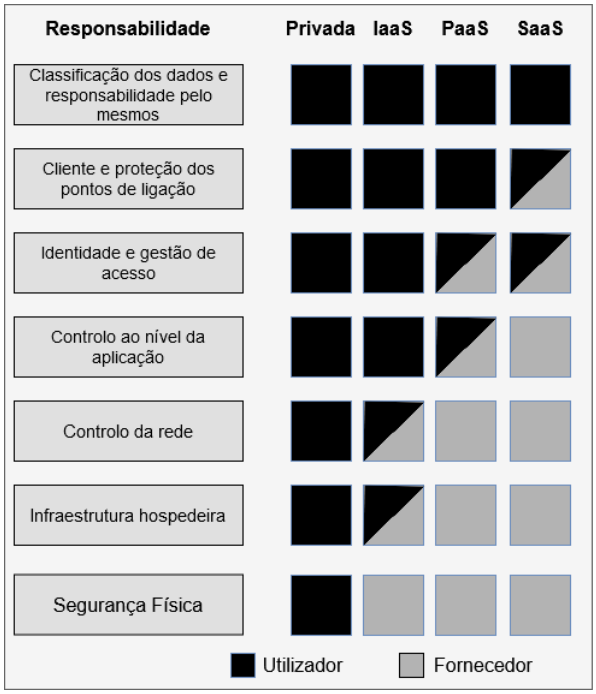


Figura 11: Divisão de responsabilidades nos modelos de serviço. Adaptada de Simorjay (2017).

O modelo IaaS segue o modelo de responsabilidade partilhada, isto é, o consumidor gere o software e o *cloud provider* gere o *hardware*. O modelo PaaS permite que o consumidor se

foque na solução que está a desenvolver, pois responsabilidades como sistemas operativos ou instalação de bibliotecas não são do encargo deste. No modelo SaaS, o consumidor utiliza a solução de outrem e apenas configura alguns aspetos de uso pessoal e os dados que são inseridos. Por exemplo, no *Office* 365 é possível configurar o tipo de letra com que se pretende escrever (configurar) e elaborar (inserir) o texto (dados).

Na Tabela 3 é possível visualizar alguns exemplos de cada um dos tipos de modelo de serviço.

Tabela 3: Exemplos dos modelos de serviço

Modelo de Serviço	Exemplos
IaaS	Windows Azure Virtual Machines, AWS EC2, Google Compute Engine
PaaS	Azure Kubernetes Service, Azure SQL Databases, AWS Lambda
SaaS	Dropbox, Microsoft Office 365, Gmail

3.2 CONCORRÊNCIA

Nesta secção são apresentadas empresas na mesma área de negócio da *UN1Qnx*. A investigação aqui apresentadas foi realizada com o intuito de examinar o que as outras empresas oferecem, principalmente em termos de rapidez e qualidade do serviço.

3.2.1 *Authentic Vision*

A *Authentic Vision* é uma empresa austríaca que fornece tecnologias de anti-contrafação e de autenticação (Vison, 2021). Opera nos mercados de lubrificantes, marcas e licenciamento de tecnologia, vinhos e bebidas espirituais, agroquímicos, peças automóveis, produtos industriais e produtos de saúde (Vision, 2021a).

A forma como a prova de autenticidade funciona é bastante semelhante à da *UN1Qnx*. As etiqueta da *Authentic Vision* é composta por duas componentes distintas: um *Quick Response Code (QR Code)* e uma componente gerada aleatorimente, cuja a própria empresa afirma não conseguir duplicar (Vision, 2021c), que se traduz num holograma de cores semelhante ao da Figura 12 (como o encontrado nas bandas holográficas das notas de euro).

Figura 12: Etiqueta holográfica. Retirada de Vision (2021c).

A validação e verificação da autenticidade de um produto é realizada através de uma aplicação para *smartphone* e segue dois passos simples:

- Capturar uma imagem do *QR Code*;
- Se o *QR Code* for válido são necessárias capturas do holograma de vários ângulos e com boa luminosidade;

Após os cumprir os passos listados anteriormente, a aplicação dá a sua reposta e indica se o produto é contrafeito ou não, demorando apenas segundos (Vision, 2021b, p.10).

3.2.2 *Visua*

A *Visua* é uma empresa fundada em 2012 e tem a sua sede em Dublin na Irlanda. Não opera em nenhum mercado específico, mas a aplicabilidade do seu conceito é consideravelmente ampla. Disponibiliza serviços de validação e autenticação de produtos, mas não inclui nenhuma componente física (e.g., etiqueta, selos, etc), fazendo uso apenas de inteligência artificial (Visua, 2021a).

O serviço é disponibilizado através de uma API e apenas é necessário submeter imagens de um produto. Mais tarde, quando se pretender verificar a sua autenticidade, basta capturar imagens do mesmo e uma resposta de afirmativo/negativo será devolvida consoante a similaridade com as imagens submetidas.

No que diz respeito a desempenho, declaram que o seu serviço é altamente escalável (Visua, 2021b), preciso e imediato (Visua, 2021a).

3.2.3 *Zortag*

A *Zortag* é uma empresa com presença em vários países tais como Estados Unidos da América, México, China e Índia. De todas as empresas concorrentes analisadas esta é a que possuí o produto mais idêntico ao da *UN1Qnx*.

A sua etiqueta é constituída por um código QR e por partículas 3D distribuídas aleatoriamente, tal como na Figura 13. Contudo, a semelhança com a *UN1Qnx* não se fica apenas pela etiqueta, mas também pelo sistema de validação, que é feito através da captura de uma imagem, utilizando a aplicação da *Zortag*.

Figura 13: Etiqueta 3D. Retirada de Zortag (2021).

3.2.4 *Certilogo*

A *Certilogo* é uma empresa fundada em 2006 e o seu produto, à semelhança das restantes empresas, consiste numa etiqueta que é colocada no produto.

A etiqueta produzida por esta empresa poderá ser constituída por três componentes distintas, que se traduzem em três formas de validar o produto. Essas componentes são um código QR, um código numérico denominado código CLG e uma componente que permite validação por NFC como se pode ver na Figura 14.

Figura 14: Etiqueta da *Certilogo*. Retirada de Certilogo (2021).

Neste caso, visto que existem múltiplas formas de validar um produto, o processo é distinto. Se a validação for através de QR, tem de ser capturada uma fotografia, se for pelo código CLG, poderá ser tirada uma imagem ou poderá ser inserido manualmente e, por último, se a validação for feita por NFC, quem valida terá de ter a aplicação instalada num dispositivo compatível com a tecnologia NFC.

3.3 SELEÇÃO DE SERVIÇOS DE *CLOUD COMPUTING*

3.3.1 *Analytic Hierarchy Process*

O Analytic Hierarchy Process (AHP), usado como um método de Multiple Criteria Decision Analysis (MCDA), é bastante conhecido e amplamente usado (Whaiduzzaman et al., 2014, p.459377), quer em MCDA, quer em planeamento, alocação de recursos e resolução de conflitos (Saaty, 1987, p.161).

O seu criador define-o como "uma teoria de medição relativa de critérios intangíveis"(Saaty, 2014). Trata-se de uma medição relativa, pois, um dos passos principais logo após definir os critérios é comparar par-a-par a prioridade entre os mesmos, o que resulta numa matriz de comparação.

Sun et al. (2013) realiza uma seleção de serviços médicos na nuvem centrada em preferências do utilizador, o que demonstra a versatilidade e utilidade e comparação de soluções entre vários pontos de vista. Foram selecionados como seus critérios o desempenho, a segurança e o custo. Destes fez uma subdivisão em subcritérios mais específicos, como se pode observar na Tabela 4.

Tabela 4: Critérios usados por Sun. Adaptada de Sun et al. (2013).

Critério	Subcritério	Descrição
Performance	Tempo de Resposta	Tempo medido entre envio do pedido e receção da resposta.
	Throughput	Quantidade de pedidos concluídos por unidade de tempo.
Segurança	Disponibilidade	Se o serviço existe e está disponível instantaneamente.
	Confiabilidade	Grau de certza do serviço.
Custo	Custo	O preço em que se incorre ao utilizar o serviço.

Após a realização de todos os cálculos, a solução escolhida foi também a que era preferida em termos de confiabilidade, o que não é de estranhar, visto que se trata de um serviço médico.

Numa outra investigação, Rehman et al. (2012) levou a cabo uma comparação entre máquinas virtuais e utilizou vários métodos. Identificou como critérios memória (GB), custo por hora, CPU (CCU)[1], IOP e memória (CCU). Os seus resultados foram inconclusivos, uma vez que métodos diferentes identificaram máquinas diferentes como melhor solução. Não obstante, apontou TOPSIS, ELECTRE e PROMETHEE como melhores opções para este estudo. TOPSIS por ser simples computacionalmente quando o número de opções é alargado e ELECTRE e PROMETHEE por serem melhores quando o número de serviços é menor, mas a quantidade de critérios é vasto (Rehman et al., 2012, p.251).

3.3.2 *Technique for Order of Preferences by Similarity to Ideal Solution*

A Technique for Order of Preferences by Similarity to Ideal Solution (TOPSIS) é um método de MCDA e manisfesta preferência pela solução ideal (Whaiduzzaman et al., 2014, p.459378).

Foi proposto em 1981 por Hwang e por Yoon e teve os seus últimos desenvolvimentos em 1993 por Hwang, Lai e Liu. Atua enveredando sempre pela melhor opção, tendo como meta a solução ideal (Tzeng, Gwo-Hshiung and Huang, Jih-Jeng, 2011, p.69). Nos últimos trabalhos que foram feitos é definida como:

> "The proposed TOPSIS for MODM algorithm is developed for solving multiple objective decision-making problems by considering two reference points of the

1 Esta medida é um agregado de vários benchmarks de desempenho diferentes. Mais detalhes em http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html.

> positive ideal solution and the negative ideal solution simultaneously. The basic principle of compromise of TOPSIS for MODM is that the chosen solution should be as close to the positive ideal solution as possible and as far away from the negative ideal solution as possible."(Hwang et al., 1993, p.889)

Resumidamente, o método consiste em definir a melhor e pior soluções e um conjunto de opções. De seguida, de entre todos as opções possíveis, é escolhida aquela que se encontra mais próxima da ótima e mais afastada da mais desvantajosa.

Em Lo et al. (2010) é utilizado uma variante de TOPSIS denominada *Fuzzy* TOPSIS e também foram tidas em conta as opiniões de vários consumidores. Este método tem como base critérios de escolha difusos (*fuzzy*), por exemplo, um serviço é qualificado pelos consumidores numa escala de "Muito pobre", "Pobre", "Mediano", "Bom"e "Muito Bom"em vez de uma escala numérica. Estas opiniões são depois transformadas em *Fuzzy Numbers* e, uma vez convertida, a escala linguística numa escala numérica, é aplicado o método TOPSIS, combinando as diferentes opiniões.

Exemplos mais práticos da utilização do método *Fuzzy* TOPSIS são dados em Boran et al. (2009) e Cheng et al. (2011). Em Boran et al. (2009) são selecionados cinco fornecedores de peças automóveis, com base em quatro critérios tendo em conta a opinião de três indivíduos. Os critérios são:

- Qualidade do produto;
- Proximidade de relações;
- Rapidez da entrega;
- Preço.

E são categorizados segundo a escala ordinal mencionada anteriormente. Neste exemplo, a opinião de cada indivíduo também não é igual e tem pesos distintos na escolha da solução.

Em Cheng et al. (2011) selecionou-se a configuração de serviços de um escritório com base nos critérios "Luminosidade", "Temperatura"e "Humidade". Foram identificados três serviços de iluminação, dois serviços de temperatura e um de desumidificação. Foram reunidas opiniões, segundo a mesma escala ordinal usada nos exemplos anteriores, de vários utilizadores, cuja opinião tem o mesmo peso.

4

DESENVOLVIMENTO

Neste capítulo, descrever-se-á o desenrolar do trabalho prático, efetuado durante a realização da presente dissertação.

4.1 EXPLORAÇÃO DE SERVIÇOS CLOUD

O desenvolvimento da presente dissertação iniciou-se com a exploração de tecnologias que pudessem satisfazer as necessidades e objetivos impostos pela empresa. Portanto, com finalidade de averiguar quais as tecnologias mais recentes na *Microsoft Azure* que poderiam servir essas necessidades, foi levada a cabo uma investigação. Investigação essa que resultou na descoberta das seguintes tecnologias:

- *Azure Machine Learning*
- *AKS*
- *Azure Functions*
- *Azure Container Instances*
- *Azure App Service*
- *Azure Virtual Machine Scale Sets*

Numa primeira instância, destas tecnologias, a que se revelou mais pertinente foram os *Azure Virtual Machine Scale Sets*.

Virtual Machines Scale Sets permitem que um utilizador crie e faça a gestão de conjuntos de máquinas virtuais, sendo que o número de máquinas virtuais poderá variar consoante a demanda. Esta solução demonstra-se a mais vantajosa, pois a solução atual da empresa já se encontra numa MV. E, deste modo, os processos de desenvolvimento da empresa não seriam alterados.

Assim, tendo em vista utilizar *Virtual Machine Scale Sets* partiu-se para os testes de *performance*.

4.2 TESTES DE *performance*

Uma vez escolhida uma possível solução para avançar, foram planeados testes de *performance* que seguem a metodologia definida na Subsecção 1.3.3.

4.2.1 *Definição do Sistema*

O objetivo do caso de estudo é comparar o desempenho de diferentes MV, que albergam o mesmo sistema de validação de etiquetas. O sistema é composto pelo cliente, pelo canal e pelo servidor. O servidor é composto por uma máquina que alberga o sistema em questão. A máquina onde está a ser executado o serviço é a componente de estudo. O estudo será feito de forma a que o impacto de componentes externas ao sistema seja minimizado.

4.2.2 *Serviços*

No serviço de validação em questão, existem dois tipos de pedido e dizem respeito a um serviço: validação de etiquetas. Para realizar uma validação, é necessário enviar, consecutivamente, os dois tipos de pedido: um pedido do tipo *Top* e um do tipo *Angle*. Para qualquer um dos tipos de pedido enviado, os campos são os mesmos e apenas mudam certas configurações, pelo que o tamanho do pedido não é considerado impactante no desempenho do serviço.

4.2.3 *Métricas*

As métricas selecionadas para a experiência que irá ser levada a cabo serão o tempo por pedido, *throughput* do sistema e utilização de recursos durante a experiência.

4.2.4 *Parâmetros*

A coleta de parâmetros é uma parte importante da metodologia de testes de desempenho, pois, não só permite ao analista verificar o impacto das características do sistema, como também estabelecer que dados é que deverão ser recolhidos. Os parâmetros reunidos são:

- Rapidez da rede de *internet*;
- Rapidez do CPU local;
- Rapidez do CPU remoto;
- *Overhead* do sistema operativo ao lidar com a rede

As categorias selecionadas foram *General Purpose*, *GPU*, *Optimized Memory* e *Compute Optimized*.

4.2.5 *Factores*

Os fatores selecionados nesta experiência são MV da *Microsoft Azure*. Tendo em conta a hierarquia na organização de máquinas, como se pode ver na figura 15, apontaram-se as diferentes categorias como sendo os níveis do fator selecionado.

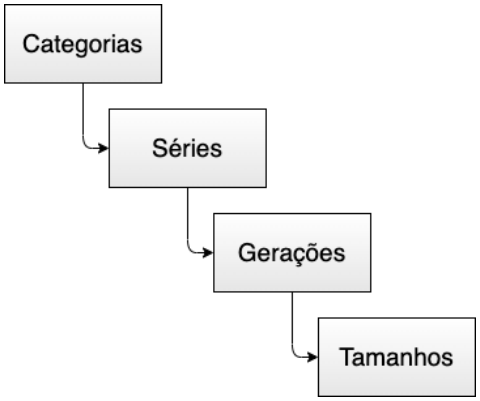


Figura 15: Níveis de MV *Azure*.

A quantidade categorias selecionadas foi quatro e as categorias foram:

Tabela 5: Especificações da MVs a testar.

Categoria	Tamanho	vCPUs	RAM	Preço (€/mês)
GPU	Standard_NV4as_v4	4	14	292,49
General Purpose	Standard_D4as_v4	4	16	254,86
Memory Optimised	Standard_E4as_v4	4	32	300,42
Compute Optimised	Standard F4s_v2	4	8	227.43

As MVs acima representadas foram escolhidas para teste porque se encontram na mesma gama e porque os seus custos mensais vão de 227€ (preço da MV atual) e 300€. Sendo que esta gama de preços foi escolhida pela UN1Qnx. Todos os SO utilizados pelas máquinas são *Ubuntu* 20.04 e encontram-se na região *West Europe*.

4.2.6 *Técnica de Avaliação*

Visto que o sistema já se encontra numa etapa avançada do ciclo de vida, isto é, o sistema já foi desenvolvido e já se encontra nos primeiros projetos piloto com outras empresas, a técnica de avaliação pela qual se optou foi a avaliação de um sistema real.

4.2.7 *Workload*

Tendo em conta que a técnica de avaliação selecionada foi avaliação de um sistema real, o *workload* consiste na execução de um *script* executado localmente, que envia pedidos para a máquina alvo.

SCRIPTS DE TESTE Os *scripts*, presentes no Apêndice D, consistem num conjunto de executáveis *powershell* que fazem uso do *software Jmeter* para realizar os pedidos ao servidor.

A estrutura dos *scripts* é a seguinte:

1. Questionar qual a réplica do teste onde pretende começar e a réplica onde pretende acabar. Isto permite com que se façam, por exemplo, as duas primeiras réplicas de teste primeiro e as outras duas mais posteriormente.
2. Para cada uma das máquinas selecionadas troca-se a categoria da MV para o tamanho com que se vai testar.
3. Liga-se a máquina.
4. Liga-se os servidores de validação.
5. Configuram-se os nomes dos ficheiros onde iram ser gravados os testes.
6. Iniciam-se os testes.
7. Interrompe-se o funcionamento da MV.
8. Volta-se ao passo 1.

Na Figura 16 encontra-se a estrutura do plano de testes no *Jmeter*. O plano de teste consiste num controlador aleatório que escolheria uma etiqueta aleatória, por exemplo, "1 120010001016". No interior desta etiqueta encontram-se outras duas directorias com um controlador aleatório para pedidos *top* e outro para pedidos *angle*. Que, por sua vez, também escolherá primeiro um pedido *top* aleatório e, de seguida, um pedido *angle*. Este processo irá repetir-se enquanto a *thread* estiver ativa.

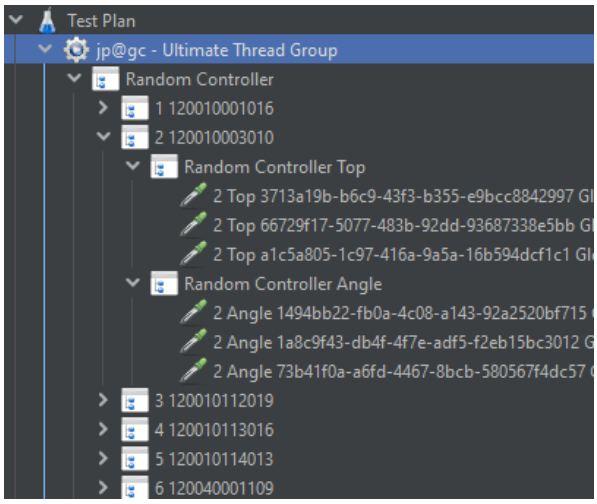


Figura 16: Estrutura de testes do *Jmeter*.

PREPARAÇÃO DOS DADOS DE TESTE De forma a ter os pedidos mais realistas, foram pedidos dados sobre etiquetas à empresa, com o intuito de os transformar em pedidos para realizar ao serviço.

SCRIPTS DE TRADUÇÃO DE DADOS A UN1Qnx forneceu uma pasta com imagens e ficheiros de texto com dados sobre cada uma das imagens. Dados esses suficientes para realizar pedidos de validação.

Após analisar a estrutura da pasta fornecida e o conteúdo dos ficheiros foi elaborado um *script* em *pyhton* que percorria a pasta, daí gerava os pedidos e colocava-os numa base de dados.

Após este tratamentos dos dados escreveu-se ainda um outro *script pyhton* que retirava os pedidos da base de dados e gerava o ficheiro *jmx* de configuração do teste utilizado pelo *Jmeter* para realizar os pedidos durante os testes.

Na Figura 17 encontra-se o esquema da tabela de *MySQL* onde os dados relativos aos pedidos são guardados. Explicando a tabela, temos o campo *mode* que representa o tipo de pedido *top* ou *angle*. O campo *barcode* representa o código de barras da etiqueta. *Filename* guarda o nome da imagem no banco de imagens. De seguida, pode-se ver os campos *xi*, *yi*, *xf* e *yf*, que são as coordenadas entre a quais a etiqueta se encontra na imagem com o nome guardado em *filename* e por fim temos o campo *globalindex*, que é um campo onde se encontra um identificador da imagem sequencial gerado pela base de dados no momento de inserção, pois o identificador da imagem pode ser calculado concatenando o *mode* com o *filename*.

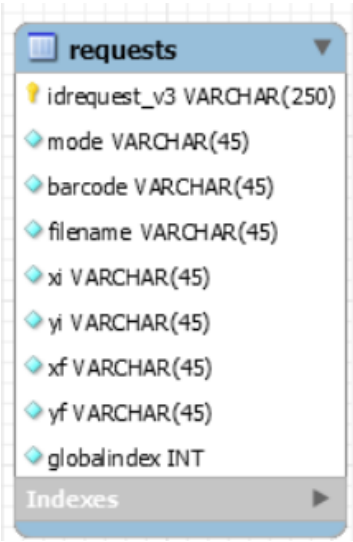


Figura 17: Esquema da base de dados onde são guardados os pedidos.

4.2.8 *Planeamento da experiência*

Os testes que iriam ser executados consistiriam numa bateria de pedidos que vai aumentando gradualmente a quantidade de pedidos feitos por minuto. O objectivo seria ter numa situação de carga máxima de 10 utilizadores a fazer pedidos sem parar, mas sequencialmente. Demoraria cerca de 60 segundos a atingir esse número de utilizadores. Mantém-se nesse número de utilizadores máximo durante 240 segundos e, de seguida, vai decrescendo o número de pedidos até zero. O período de decréscimo de utilizadores duraria 20 segundos. O teste duraria ao todo 20 minutos, pois seria repetido 4 vezes. Na Figura 18 pode ver-se como estaria planeada a experiência.

Figura 18: Plano da experiência.

4.2.9 *Análise de Dados*

Realizados os testes devem ser analisados os dados a fim de tomar decisões.

APRESENTAÇÃO DOS DADOS A síntese dos resultados dos testes realizados encontra-se na Tabela 6.

Tabela 6: Resumo dos resultados dos testes realizados.

Série da Máquina	Número Total de Pedidos	Número de pedidos corretos	Número de erros	Throughput	Média de tempo (ms)
N	202	171	31	6.3	74273
D	202	180	22	6.6	69249
E	198	168	30	6.2	77453
F	185	159	26	6.36	77806

Dos resultados obtidos conclui-se que escolher uma outra MV não seria o caminho a seguir, pois a os tempos não baixaram o suficiente e, por consequência, não havia ganho custo/*performance* suficiente.

4.3 COLOCAÇÃO DO SERVIÇO NUM CONTAINER

Após a realização dos testes, constatou-se que a utilização de MV não será a solução mais adequada, tanto porque a *performance* entre máquinas não era significativa, e se a empresa desejar trocar o serviço de máquina terá de ser feita a instalação de todas as dependências do serviço. Foram verificadas quais são, efetivamente, as dependências e, para tal, recorreu-se ao comando *pip freeze*, o que resultou na lista presente no apêndice A. Visto que a listagem conta com sessenta e sete dependências *python*, tirando as dependências do sistema operativo, foi proposta a utilização de *Docker containers*. A colocação do serviço num *container* iria trazer diversas vantagens imediatas, tais como:

- **Mobilidade:** pois a máquina onde o serviço irá correr apenas necessita de ter o *Docker* instalado para tudo funcionar;
- **Automação do *Deployment*:** visto que apenas é necessário iniciar o *container* o *deployment* poderá ser automatizado;
- **Rapidez do *Deployment*:** porque de cada vez que se troca o serviço de máquina hospedeira não é necessário instalar todas as dependências.

A passagem do serviço para um *container* foi direta. Para não alterar o fluxo de trabalho da empresa, fez-se uso do software *wine*, que permite correr ficheiros executáveis gerados em *Windows* (ambiente utilizado pela empresa) num *container Linux*. O que pouparia em custos de licenciamento de *software Windows*, pois imagens *Docker Windows* apenas podem ser executadas em máquinas hospedeiras *Windows*, que por sua vez necessitam de licenças.

Uma vez num *container*, o serviço poderia arrancar numa máquina com qualquer tipo de sistema operativo. Contudo, o serviço não seria elástico quando apenas colocado num *container*, para isso teria de se utilizar outro tipo de tecnologia.

4.4 Colocação do serviço em Kubernetes

Após a colocação do serviço de validação num *Docker container*, devido às suas dependências, recorreu-se às opções elencadas em 4.1. Destas determinou-se que o serviço de *Kubernetes* do *Azure* seria melhor opção. Esta foi a decisão tomada pois o *Kubernetes* possui as seguintes vantagens:

- É possível escalar horizontalmente dinamicamente, segundo métricas internas ou externas ao *cluster*;
- Permite a utilização de *Docker containers*;
- A infraestrutura é autogerida ao nível dos nós e dos *pods*, isto é, é possível definir rotinas de manutenção que verificam o estado dos *pods* e que, se necessário, reiniciam *pods* que não estão a funcionar devidamente;
- Toda a infraestrutura é definida através da definição de ficheiros *yaml* de configuração. O que permite que a infraestrutura seja altamente configurável, fácil de assimilar e esteja gravada em ficheiros que não se alteram e não dependentes de uma *interface* do *cloud provider*;
- Fácil de alterar e configurar em qualquer momento futuro, pois rapidamente se altera e modifica a arquitetura sem alterar o funcionamento do serviço de validação.
- Aumento da produtividade, porque devido à abstração e facilidade de configuração o tempo consumido em alterações é reduzido relativamente a outras opções.

Devido a estas razões, optou-se por hospedar o serviço de validação no AKS. Contudo, a arquitetura utilizada foi modificada ao longo do tempo, quer devido a descoberta de erros, quer devido à mudança de estratégia de serviço síncrono/assíncrono por parte da empresa.

4.4.1 *Primeira Arquitetura*

Na Figura 19, apresenta-se o diagrama da primeira e mais simples de todas as arquiteturas elaboradas. Esta arquitetura consiste no serviço de validação da *UN1Qnx S.A.* colocado num *docker container* por *pod*, podendo um *deployment* conter vários *pods*. Este *deployment* é exposto ao exterior através de um *service* do tipo *load balancer*. Já o número de *pods* é controlado consoante a percentagem de CPU utilizado, sendo que este aspeto é monitorizado pelo HPA.

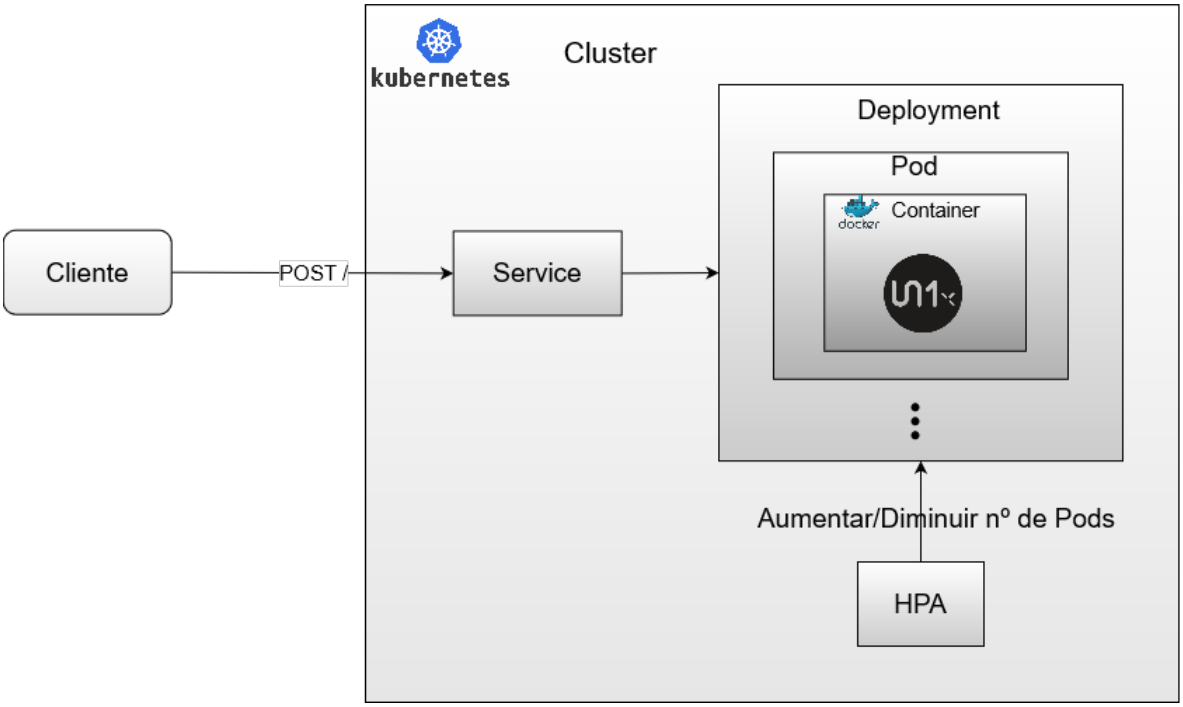


Figura 19: Diagrama da primeira arquitetura.

Contudo, após a utilização desta primeira arquitetura o número de pedidos que resultavam em erro, chegou aos 50%. Foi verificada toda a instalação e colocação do serviço de validação em *docker* e nenhum erro foi detetado.

Após várias revisões foi decidido que se faria uso de *Spectrum-based Fault Localization*. Esta técnica de teste consiste em considerar componentes ou condições e realizar vários testes onde estas condições variam. Os resultados devem gerar uma matriz semelhante à presente da Tabela 7, onde também se deverá guardar se o teste falhou ou não. Após o registo dos resultados, poder-se-á proceder ao cálculo dos coeficientes de *Jaccard*. Este coeficiente indicará qual a probabilidade de cada uma das condições ser a responsável pelo erro conforme a grandeza do número, sendo quanto maior mais probabilidade de aquela ser a causa do erro (Abreu et al., 2007). O coeficiente é calculado da seguinte forma:

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Na qual as letras significam o seguinte:

- M_{11}**:** Número de vezes em que a componente estava presente e o teste falhou;
- M_{01}**:** Número de vezes em que a componente não estava presente e o teste falhou;
- M_{10}**:** Número de vezes em que a componente estava presente e o teste não falhou.

Após as revisões feitas ao serviço, devido ao elevado número de erros, foram elaboradas componentes para avançar com a *Spectrum-based Fault Localization*.

- **Componente 1 (C.1):** reinício do serviço após o teste anterior;
- **Componente 2:** imposição de limites de recursos no ficheiro de configuração do *deployment kubernetes*;
- **Componente 3:** pedidos *top* e *angle*;
- **Componente 4:** apenas pedidos *top*;
- **Componente 5:** apenas pedidos *angle*;
- **Componente 6:** serviços separados para cada tipo de pedido.

Em seguida, elaboraram-se os testes dos quais resultou a Tabela 7.

Tabela 7: Matriz de semelhança de *Jaccard*, segundo os testes realizados.

	C. 1	C. 2	C. 3	C. 4	C. 5	C. 6	Fail
teste 1	1	1	1	0	0	0	1
teste 2	0	1	0	1	0	0	0
teste 3	0	1	0	0	1	0	1
teste 4	1	0	0	0	1	0	0
teste 5	1	1	0	0	1	0	0
teste 6	1	0	1	0	0	0	1
teste 7	1	0	0	1	0	0	0
teste 8	1	0	0	1	0	0	0
teste 9	1	0	0	0	1	0	0
teste 10	1	0	1	0	0	1	0
teste 11	1	0	1	0	0	0	1
teste 12	1	0	1	0	0	0	1
teste 13	1	0	1	0	0	1	0
teste 14	1	0	1	0	0	1	0
teste 15	1	1	1	0	0	1	0
Coeficiente de Jacard	0.36	0.15	0.36	0	0.07	0	0

Discorrendo sobre a Tabela 7, pode-se observar que a causa mais provável de erro é realizar pedidos *top* e *angle* para um serviço único. Adotou-se a utilização de serviços distintos para cada tipo de pedido e daí surge a segunda arquitetura.

4.4.2 *Segunda Arquitetura*

Da inadequação para atender pedidos de ambos os tipos, nasce a segunda arquitetura, repsentada na Figura 20. É de notar que esta arquitetura é bastante semelhante à primeira, com a simples diferença de os pedidos serem encaminhados para serviços dedicados para *top* e *angle*. Para tal, acrescentou-se um *ingress NGINX* que realiza essa triagem consoante o URL do pedido.

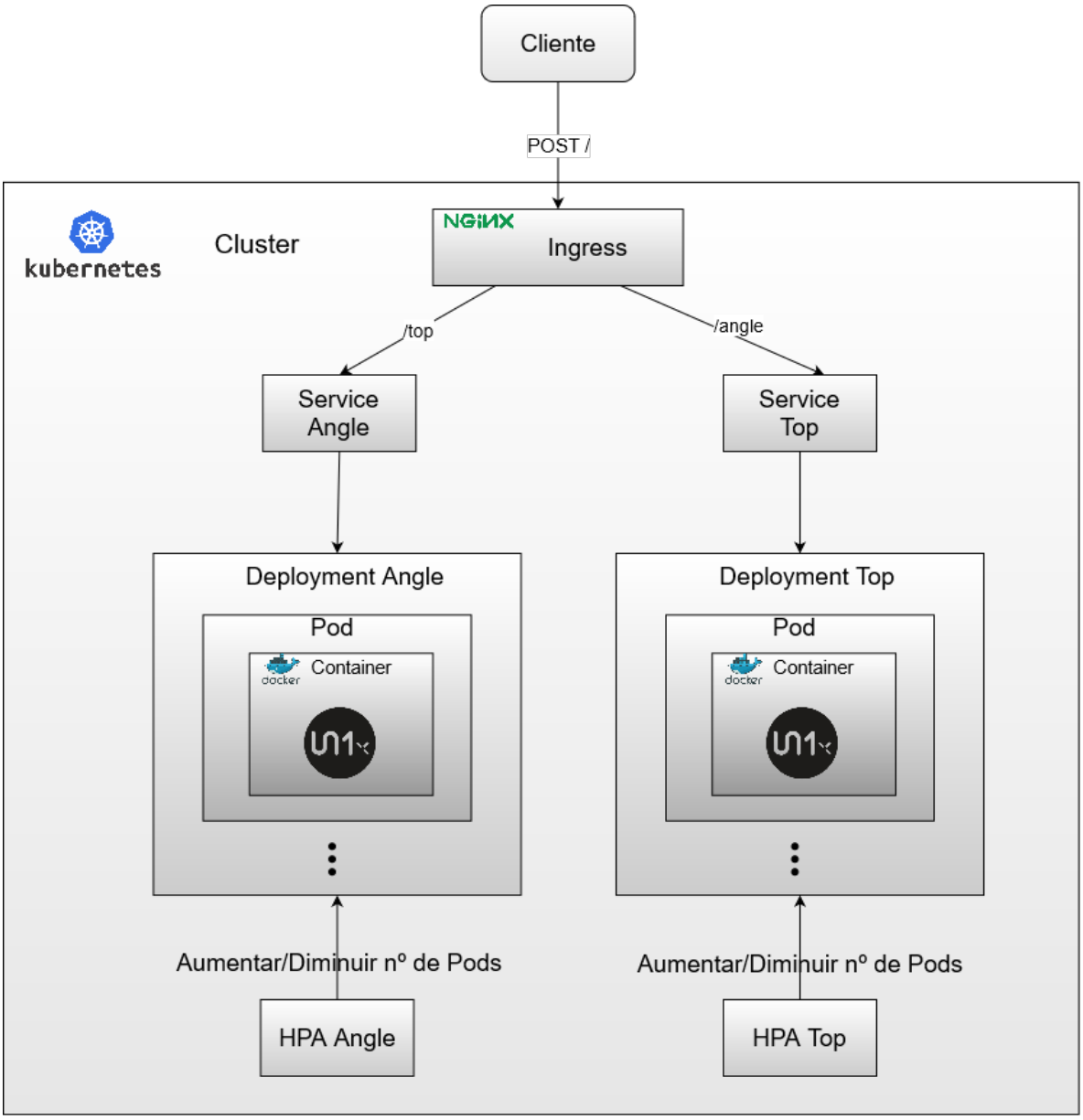


Figura 20: Diagrama da segunda arquitetura.

Contudo, após alguma utilização desta arquitetura, observou-se que a utilização de HPA baseados na utilização de CPU não seria a melhor. Esta métrica revelou-se inapta, uma vez que, por decisão da empresa, o serviço apenas processa um pedido de cada vez, devido à grande utilização de recursos por processamento de um pedido. O que resultaria em níveis de CPU constantes, que consequentemente não espoletariam nenhuma adição de recursos.

4.4.3 *Terceira Arquitetura*

Apesar das melhorias que a segunda arquitetura trouxe, ainda haviam dois problemas por resolver que consistiam: a não persistência dos pedidos, isto é, se um pedido falhasse o mesmo não era feito novamente e o usos indevido da utilização média de CPU como métrica de expansão e diminuição do serviço. Para o efeito, desenvolveu-se uma terceira arquitetura, que introduziu um servidor desenvolvido em *NodeJs e Express* a atender os pedidos vindos do exterior. Esse servidor ficaria responsável pela persistência dos pedidos e do mesmo se criariam rotas que permitiriam extrair métricas mais adequadas, para dinamizar o serviço como o número de pedidos *Top* e *Angle*.

O HPA, considerado inadequado, continuaria a ser utilizado, mas de uma forma diferente. Desta vez, quando um determinado pedido atingisse o servidor *NodeJs* um contador interno iria ser incrementado. De seguida, estas métricas seriam recolhidas pelo *Prometheus* e novamente recolhidas pelo HPA, que aumentariam ou diminuíram o número de *pods* consoante o valor de pedidos feitos nos últimos minutos.

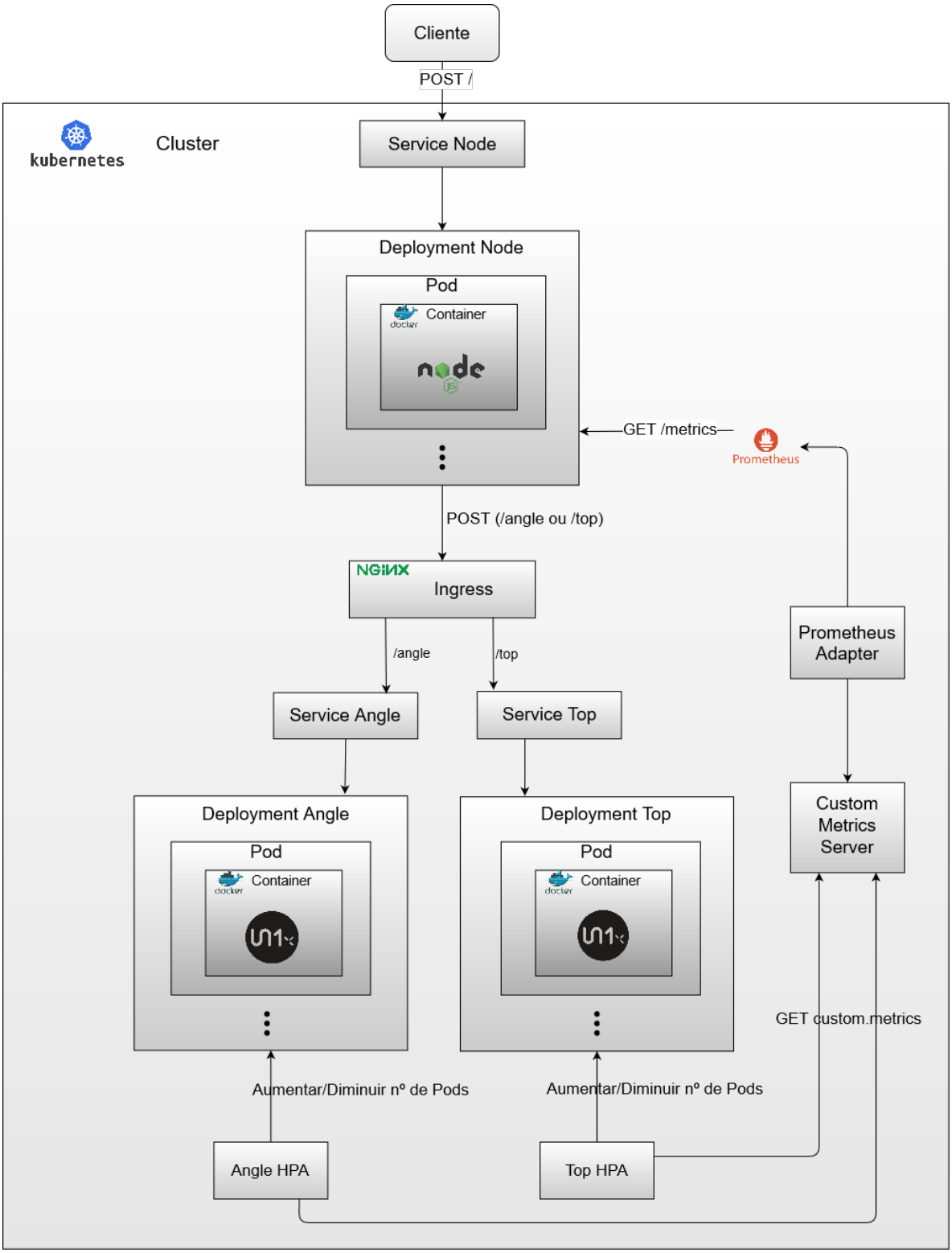


Figura 21: Diagrama da terceira arquitetura.

No entanto, a arquitetura apresentada na Figura 21 deixou de ser utilizada, pois a empresa decidiu adotar uma estratégia que passaria por utilizar pedidos assíncronos.

4.4.4 *Quarta Arquitetura*

A quarta arquitetura foi criada, devido ao elevado tempo de resposta a cada pedido, tempo esse que se deterioraria aquando da chegada de múltiplos pedidos. Portanto, a empresa decidiu alterar a sua estratégia, enquanto que uma otimização do serviço de validação não é levada a cabo. A estratégia passa tornar a validação de uma etiqueta num processo assíncrono. Ou seja, anteriormente a esta arquitetura um utilizador realiza um pedido e esse pedido necessita de receber uma resposta.

Na arquitetura presente na Figura 22 o cliente realiza o pedido e recebe uma resposta, mas desta vez com a informação que esse mesmo pedido foi registado com sucesso para posterior processamento. O pedido é guardado na fila correspondente, conforme seja *top* ou *angle* e permanece nessa fila até que o serviço de validação o retire da mesma. Esta mudança de *design* permite também que se simplifique a arquitetura e se elimine componentes, tais como o *ingress* e *service*, pois, desta vez, o serviço não recebe pedidos, mas vai antes retirar pedidos da fila.

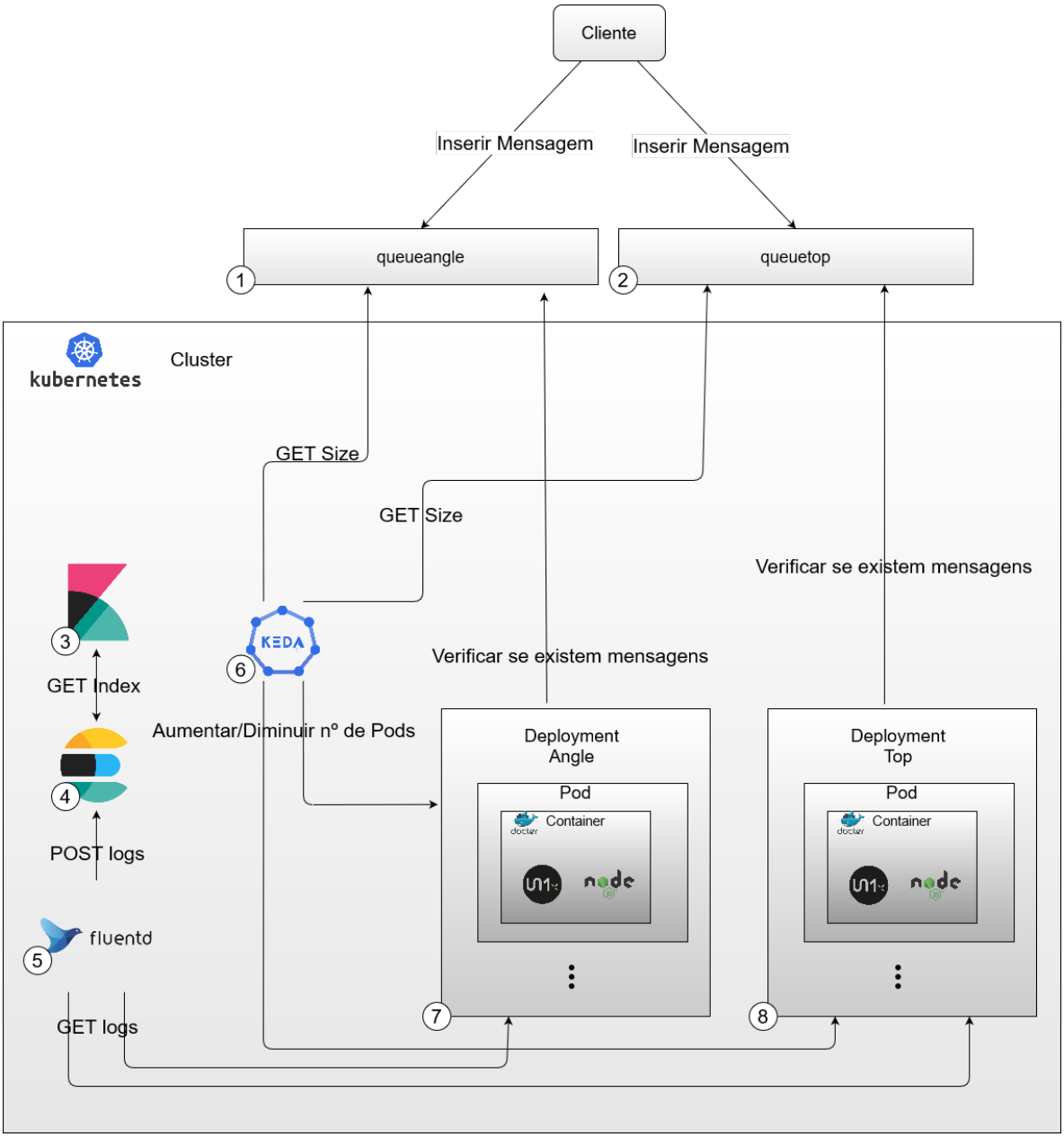


Figura 22: Diagrama da quarta arquitetura.

Contudo, a eliminação de *ingress* e *services* não foi a única mudança. Também se eliminou o HPA, pois a utilização de filas permitiu que se utilizasse outros métodos para mudar dinamicamente o número de *pods*, nomeadamente o número de pedidos em fila. Para tal, fez-se uso de KEDA. KEDA significa *Kubernetes Event-driven Autoscaling* e, como o nome indica, permite que se faça escalonamento automático em *Kubernetes*. Esta componente pode ser usada e em simultâneo com HPA e até permite que a funcionalidade seja estendida sem interferir com a já existente (KEDA, 2021).

Visto que o serviço se encontra agora alojado em *Kubenetes*, a visualização de impressões para o ecrã dificulta-se, pois esta é umas consequências da containerização, o que complica o *debug* e a manutençaõ do *software*. A fim de mitigar este problema fez-se uso da *stack* de *logging* EFK , isto é, *Elasticsearch*, *Fluentd* e *Kibana*. O *Fluentd* é um coletor de dados que permite com que haja um camada de *logging* unificada (Fluentd, 2021) . Cada nó irá ter um agente *Fluentd* e, por sua vez, cada agente *Fluentd* irá enviar enviar os *logs* recolhidos para o *Elasticsearch*. O *Elasticsearch* agrupa os dados e disponibiliza ferramentas de pesquisa e análise sobre os mesmos (Elasticsearch, 2021). Por último *Kibana* fornece uma *interface* gráfica que permite visualizar, pesquisar e analisar os dados recolhidos (Kibana, 2021).

De uma forma resumida estas são as componentes da arquitetura em vigor:

1. **queueangle:** fila utilizada para guardar pedidos do tipo *angle*;
2. **queuetop:** fila utilizada para guardar pedidos do tipo *top*;
3. **Kibana:** permite visualizar os registos capturados
4. **Elasticsearch:** permite analisar e pesquisar os registos;
5. **Fluentd:** captura impressões no ecrã feitas por aplicações dentro do *Kubenetes*;
6. **KEDA:** monitoriza o tamanho das filas de forma a controlar o escalonamento de *pods* do serviço de validação;
7. **Serviço de Validação Angle:** valida pedidos do tipo *angle*;
8. **Serviço de Validação Top:** valida pedidos do tipo *top*;

Estrutura do código

O código gerado foi organizado em duas diretorias principais.

O *backends* diz respeito à configuração da imagem *docker*, na qual está o serviço de validação de etiquetas.

No *infrastructure/kubernetes*, encontra-se um conjunto de ficheiros *yaml* que contêm toda a configuração da infraestrutura *kubernetes*.

```
1         backends..................... 1.
2         infrastructure/kubernetes..... 2.
```

Listagem 4.1: Estrutura do repositório

No diz respeito à pasta *backends* podemos ver o seu conteúdo na listagem a seguir.

```
1         healthServer.................. 1.
2               config.....................   1.1
3                     conf.js.................   1.2
```

```
4               healthServer.js.............   1.3
5               package-lock.json...........   1.4
6         validateApp..................... 2.
7               config......................   2.1
8                     vapp-conf-angle.js......   2.2
9                     vapp-conf-top.js........   2.3
10                    vapp-conf.js............   2.4
11              validateApp.js..............   2.5
12              package-lock.json...........   2.6
13        dockerfile-angle................ 3.
14        dockerfile-top.................. 4.
15        .dockerignore................... 5.
16        entrypoint.sh................... 6.
```

Listagem 4.2: Estrutura da pasta *backends*

1. Nesta diretoria encontra-se o servidor pelo qual se pode averiguar o estado de saúde do programa.
   1.1. Diretoria onde se encontram ficheiros de configuração do servidor.
   1.2. Ficheiro de configuração que contém váriáveis globais do programa. Exemplo: nome de ficheiros relativos à saúde global do serviço.
   1.3. Ficheiro *javascript* onde se encontra o código que define o servidor de saúde do serviço.
   1.4. Ficheiro onde se encontram definidas as dependências do servidor.
2. Diretoria do serviço que alimenta o validador com pedido de validação caso existam pedidos em espera na respetiva fila.
   2.1. Diretoria onde se encontram ficheiros de configuração do servidor.
   2.2. Ficheiro de configuração que contém váriáveis globais da *validateApp* para pedidos do tipo *Angle*. A diferença entre este ficheiro e o seguinte na listagem é na variável *queuename*. Nenhum destes ficheiros é utilizado em *runtime*. Contudo, em *buildtime* um destes ficheiros substitui o ficheiro "*vapp-conf.js*", que é o utilizado pelo serviço em si em "*validateApp.js*", consoante se esteja a fazer *build* de uma imagem *top* ou *angle*.
   2.3. Ficheiro de configuração que contém váriáveis globais da *validateApp* para pedidos do tipo *Top*.
   2.4. Ficheiro de configuração que contém váriáveis globais da *validateApp*. Este ficheiro é o que é realmente utilizado em *runtime* (ver 2.2.).
   2.5. Ficheiro *javascript* onde se encontra o código que define o serviço *node*.
   2.6. Ficheiro onde se encontram definidas as dependências do servidor.

3. *Dockerfile* para a construção da imagem *Angle*.
4. *Dockerfile* para a construção da imagem *Top*.
5. Ficheiro onde se define o que não copiar para a imagem aquando do *build* da imagem *docker*.
6. Aquando da inicialização de um *container* este *script* é utilizado

Na próxima listagem encontra-se o conteúdo da pasta *infrastructure/kubernetes*.

```
1       auth-triggers..... 1.
2             auth-queue-trigger.yaml.....   1.1
3       configmaps..... 2.
4             fluentd-configmap.yaml.....   2.1
5       daemonsets..... 3.
6             fluentd-daemonset.yaml.....   3.1
7       deployments..... 4.
8             elastic-index.conf.....   4.1
9             elasticsearch-deployment.yaml.....   4.2
10            kibanadeployment.yaml.....   4.3
11            validation-angle-deployment.yaml.....   4.4
12            validation-top-deployment.yaml.....   4.5
13      keda-scale-objects..... 5.
14            angle-keda-scale-object.yaml.....   5.1
15            top-keda-scale-object.yaml.....   5.2
16      rbac..... 6.
17            fluentd-rbac.yaml.....   6.1
18      secrets..... 7.
19            queue-secret.yaml.....   7.1
20            storage-secret.yaml.....   7.2
21      services..... 8.
22            elasticsearch-service.yaml.....   8.1
23            kibana-service.yaml.....   8.2
24      base64key.ps1..... 9.
```

Listagem 4.3: Estrutura da pasta *infrastructure/kubernetes*

1. Nesta directoria encontram-se todos os *triggers* de autenticação. Um *trigger* de autenticação é útil, pois permite definir o método de autenticação e também que esse método de autenticação seja reutilizado por outros objetos.
   1.1. Método de autenticação para a *storage queue*.
2. Aqui encontram-se todos os ficheiros de configuração do tipo *configmap*.
   2.1. Configuração do ficheiro "*fluentd.conf*"que define as fontes de dados (*source*) e para onde os enviar (*match*). O *Fluentd* lê o ficheiro "*fluentd.conf*", contudo estão

definidos mais ficheiros que o "*fluentd.conf*"utiliza, o que permite uma melhor organização.

3. Directoria onde estão definidos os *daemonsets* do *kubernetes*.
   3.1. Neste ficheiro encontra-se definido o *daemonset* do *Fluentd*, visto que cada máquina necessita de ter um *pod* do *Fluentd* a fim de os *logs* de todos os serviços serem capturados.
4. Nesta directoria estão definidos todos *deployments* necessários à aplicação.
   4.1. Apesar de não ser um *deployment* este ficheiro define a *policy* de índices e um *index template* do *Kibana*. Definições essas que são necessárias para o controlo da acumulação de *logs*.
   4.2. *Deployment* do *elasticsearch*.
   4.3. *Deployment* do *Kibana*.
   4.4. *Deployment* do serviço de validação que consome pedidos *angle*.
   4.5. *Deployment* do serviço de validação que consome pedidos *top*.
5. Diretoria onde se encontram todas definições de objectos que são susceptíveis de serem aumentados/diminuídos através do KEDA.
   5.1. Este ficheiro faz ligação entre o KEDA e o *deployment angle*. Isto é, define o *deployment angle* como alvo de aumento de *pods* caso o número de pedidos na fila *queueangle* aumente e vice-versa.
   5.2. Este ficheiro faz ligação entre o KEDA e o *deployment top*. Isto é, define o *deployment* top como alvo de aumento de *pods* caso o número de pedidos na fila *queuetop* aumente e vice-versa.
6. Diretoria onde se encontram todos RBAC.
   6.1. RBAC do *Fluentd*, pois este necessita de permissão para ler os *logs* de todos os *pods*.
7. Diretoria onde se encontram os segredos utilizados.
   7.1. Segredo de acesso às *queues*. Utilizado pelos "*keda-scale-objects*"para averiguar tamanho das filas.
   7.2. Segredo de acesso à *storage*. Utilizado pelos *deployments top* e *angle* para utilizar o *Azure* file *shares* como armazém de fotografias de etiquetas e outros ficheiros comuns a todos os *pods*.
8. Diretoria onde se definem os serviços.

8.1. Um serviço expõe um *deployment* ao exterior. Neste ficheiro, é definido o serviço que expõe o *deployment* do *elasticsearch* (4.2).

8.2. Ficheiro que define o serviço que expõe o *Kibana* (4.3).

9. *Script powershell* auxiliar que codifica uma string em base64. Este *script* serviu para codificar strings que se encontram no segredos (7.).

Documentação do Repositório

De forma a ter um bom grau de organização foi criada documentação no repositório onde foi colocada a solução construída. A documentação está presente sob a forma de ficheiros *Readme*, que estão presentes nas estruturas de ficheiros das listagens da subsubsecão anterior.

5

CONCLUSÃO

5.1 OBJETIVOS CONCRETIZADOS

O tema desta dissertação surgiu no âmbito da otimização da infraestrutura da *cloud* da UN1Qnx. A empresa detinha o seu serviço de validação a ser executado numa MV única, o que impossibilitava com que o serviço se ajustasse à carga de trabalho, especialmente quando esta era elevada. Um segundo problema seria o tempo que cada pedido levava a ser processado.

Em primeiro lugar, foi explorado o ambiente do problema, o ambiente da solução e foi selecionado conhecimento base para que o problema identificado seja resolvido de um modo eficaz e para que as decisões futuras sejam tomadas de forma resoluta.

A assimilação do problema passou, precisamente, por entender o serviço que presta, o seu modo de funcionamento e o meio onde está implantado e como está implementado. A percepção e identificação do problema é crucial à tarefa de pesquisa bibliográfica, pois só ao perceber o problema é que é possível selecionar a base certa de conhecimento.

A pesquisa bibliográfica e identificação de conhecimento foi executada a seguir, mas será sempre uma tarefa presente devido ao caráter cíclico, assim como o entendimento do problema.

O estudo do estado da arte fez parte do estudo do conhecimento e consistiu em vários pontos. Estudar o ambiente da solução, isto é, *public cloud computing*, que se revelou como um espaço com grande potencial, onde ocorreram grandes evoluções nos últimos dez anos, e com cada vez mais aderência. Tanto pela diversidade de formas em que pode ser implementado, como pelo vasto leque de formatos em que pode ser servido ao utilizador. De forma a aumentar a nível de capacidades e entendimento da questão, também foi tirado um certificado (AZ-900). Foram também investigadas outras empresas na mesma área de negócio, mais especificamente a *Authentic Vision*, *Visua*, *Zortag* e *Certilogo*, a fim de averiguar quais as outras soluções por outras empresas e qual a *performance* que dizem oferecer. Por último, foram investigados métodos de seleção entre múltiplas opções e da variedade de métodos encontrados foram para já explorados dois, o AHP e o TOPSIS.

Em segundo lugar, começou-se o desenvolvimento do trabalho, isto é, a realização de tarefas práticas como a escrita de código, ficheiros de configuração e documentação. O desenvolvimento iniciou-se com a exploração de serviços de *cloud*, inclusive com a obtenção de uma certificação *Azure*. Esta exploração de serviços *cloud* serviria para aprender quais as diferentes opções que a *UN1Qnx* tem à ao seu dispor, para colocar o seu serviço, de forma a poder-se deliberar qual seria a melhor das opções. A averiguação do que existe na *cloud* foi feita de antemão, mas, não obstante, o primeiro caminho a explorar seriam as MV, pois era o que a empresa já utilizava. Numa primeira instância, pensava-se que o problema seria a *mv* a ser utilizada de momento, que poderia não ter sido a escolha mais adequada para a empresa. Portanto, foram realizados testes de *performance* numa seleção de máquinas distintas na sua categoria, mas semelhantes no seu custo mensal. Para a realização destes testes, foi feita uma investigação sobre como realizar testes.

Feita a investigação, foi feito um estudo do que realmente se iria testar, foram solicitados dados à empresa para poder realizar pedidos e foi criado também um plano de testes. O plano de testes foi construído principalmente com duas ferramentas, *Powershell* e *Jmeter*. Foi criado um ficheiro de configuração em *Jmeter* que continha a seguinte configuração: são criados incrementalmente ao longo de um minuto 10 *threads* que simulam 10 utilizadores e que fazem pedidos sem parar, mas sequencialmente. Quando estiverem ativos os 10 utilizadores são mantidos durante 240 segundos. No final deste período de tempo, o número de utilizadores é decrementado durante 30 segundos até zero.

Findos os testes, analisaram-se os dados e concluiu-se que este não seria o caminho a seguir, pois as *performances* não foram muito diferentes entre as máquinas, não eram baixos o suficiente e, consequentemente, a relação entre custo/*performance* não era favorável. Durante esta fase de testes o código do serviço de validação foi manuseado e, durante estas consultas, foi possível tomar consciência do número de dependências necessárias para executar o serviço.

Tendo em conta que o serviço corria de momento numa MV, pensou-se que seria vantajoso fazer uso de *docker containers*, pois iria tornar a aplicação muito mais móvel e iria impedir que se tivesse que instalar todas a dependências de cada vez que se mudasse a aplicação de máquina hospedeira. Futuramente esta mudança também seria vantajosa, por exemplo, para a empresa ser capaz de lançar *releases* da aplicação com mais facilidade e rapidez.

Realizou-se a migração e ponderou-se novamente qual o caminho a seguir, segundo as possíveis opções. Desta vez, considerou-se que *kubernetes* seria o caminho a seguir, pois permitiria ajustar o serviço à carga de trabalho, segundo condições amplamente personalizáveis. Investigou-se mais sobre a tecnologia e, por fim, colocou-se o serviço no *kubernetes*. Os resultados pareciam promissores, contudo, havia um grande número de erros e chegou-se à conclusão, através da utilização de *Spectrum-based Fault Localization*, que o pedidos *top* e *angle* teriam de ser atendidos separadamente. Alterou-se a arquitetura e tudo

funcionava como era suposto. Neste momento, a capacidade de atendimento era igual à atual numa *mv*, mas os custos tinham sido reduzidos de 227€ para 90€. Entretanto, surgiram outros problemas como a persistência de pedidos, pois cada serviço de validação só era capaz de atender um pedido de cada vez, o que não aguentaria com vários utilizadores simultaneamente. Para tal, modificou-se a arquitetura, para atender os pedidos de forma assíncrona. Consequentemente, teve-se de criar um serviço em *node*, que faz o *get* de pedidos e os fornece ao serviço de validação. A estas alterações ainda se adicionaram mecanismos de *logging* com a utilização da EFK *stack*.

Concluiu-se, assim, que vários aspetos do serviço de validação podiam ser otimizados e melhorados. Não foi possível melhorar aspetos como o tempo de resposta a pedidos de validação, mas foi possível não tornar a aplicação elástica e mais adaptável ao número de pedidos que têm de ser processados como também reduzir os custos em 60% para a mesma capacidade de atendimento.

5.2 TRABALHO FUTURO

Uma vez que a aplicação de *containers*, em conjunto com a utilização de *kubernetes*, permitiu resolver a questão de elasticidade da aplicação e, consequentemente, a capacidade de resposta da aplicação e a mudança para uma arquitetura assíncrona veio mitigar a propensão de erro e viabilizar os elevados tempos de resposta. Sugere-se como objetivos futuros a construção de uma *pipeline* de testes, não só aquando da aplicação de mudanças para um ramo do repositório, mas também quando este ramo é fundido com o ramo *master*. Neste caso, poder-se-iam fazer uso de testes unitários e testes da API.

Outras otimizações poderiam ser levadas a cabo ao nível do serviço de validação. Tais como reformulação da sua arquitetura e melhorias no que toca a tempos de resposta, pois o ideal seria chegar a uma arquitetura síncrona que servisse o utilizador imediatamente.

BIBLIOGRAFIA

Babak Abbasov. Cloud Computing: State Of The Art Research Issues. In *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–4. IEEE, oct 2014. ISBN 978-1-4799-4119-3. doi: 10.1109/ICAICT.2014.7035932. URL https://ieeexplore.ieee.org/document/7035932.

Rui Abreu, Peter Zoeteweij, and Arjan J.C. van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*, pages 89–98, 2007. doi: 10.1109/TAIC.PART.2007.13.

Amazon. What is cloud computing, 2020. URL https://aws.amazon.com/what-is-cloud-computing/?nc1=h_ls.

Francisco Andrade, José Neves, Paulo Novais, José Machado, and António Abelha. Legal security and credibility in agent based virtual enterprises. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Angel Ortiz, editors, *Collaborative Networks and Their Breeding Environments*, pages 503–512, Boston, MA, 2005. Springer US. ISBN 978-0-387-29360-8.

Francisco Andrade, Paulo Novais, José Machado, and José Neves. Contracting agents: legal personality and representation. *Artificial Intelligence and Law*, 15(4):357–373, Dec 2007. ISSN 1572-8382. doi: 10.1007/s10506-007-9046-0. URL https://doi.org/10.1007/s10506-007-9046-0.

Microsoft Azure. Séries de máquinas virtuais, 2020a. URL https://azure.microsoft.com/pt-pt/pricing/details/virtual-machines/series/.

Microsoft Azure. Otimizar os custos com o azure, 2020b. URL https://azure.microsoft.com/pt-pt/overview/cost-optimization/.

Alexandre Batista, Bruno Basto, and Moisés Soares. Como construir arquiteturas corporativas escaláveis preparadas para ai?, nov 2020. URL https://bit.ly/3clmYOR.

André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd International Workshop on Software and Performance*, WOSP '00, page 195–203, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 158113195X. doi: 10.1145/350391.350432. URL https://doi.org/10.1145/350391.350432.

Fatih Emre Boran, Serkan Genç, Mustafa Kurt, and Diyar Akay. A multi-criteria intuitionistic fuzzy group decision making for supplier selection with topsis method. *Expert Systems with Applications*, 36(8):11363 – 11368, 2009. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2009.03.039. URL http://www.sciencedirect.com/science/article/pii/S0957417409002772.

Istvan Cebrian, Thiago Born, and Nuno Guedes. Como construir arquiteturas nativas com kubernetes, serverless devops?, dez 2020. URL https://bit.ly/3clmYOR.

Certilogo. Try the certilogo demo experience, 2021. URL https://tryme.certilogo.com/.

Ding-Yuan Cheng, Kuo-Ming Chao, Chi-Chun Lo, and Chen-Fang Tsai. A user centric service-oriented modeling approach. *World Wide Web*, 14(4):431–459, Jul 2011. ISSN 1573-1413. doi: 10.1007/s11280-011-0115-7. URL https://doi.org/10.1007/s11280-011-0115-7.

Elasticsearch. Elasticsearch: The official distributed search analytics engine, 2021. URL https://www.elastic.co/elasticsearch/.

Fluentd. Fluentd | open source data collector | unified logging layer, 2021. URL https://www.fluentd.org/.

Rakesh Garg. Methodology for research I. *Indian Jornal of Anaesthesia*, 60(9): 640–645, sep 2016. ISSN 0019-5049. doi: 10.4103/0019-5049.190619. URL https://www.ijaweb.org/article.asp?issn=0019-5049;year=2016;volume=60;issue=9;spage=640;epage=645;aulast=Garg.

Google. What is cloud computing ? google cloud, 2020. URL https://cloud.google.com/what-is-cloud-computing.

Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, June 2013. USENIX Association. ISBN 978-1-931971-02-7. URL https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, mar 2004. doi: 10.2307/25148625. URL http://www3.cis.gsu.edu/vvaishnavi/9220Sp07/Documents/Hevner%20et%20al.%202004%20MISQ.pdf.

Hevner, Alan and Chatterjee, Samir. *Design Science Research in Information Systems*. 2010. ISBN 978-1-4419-5653-8. URL https://link.springer.com/chapter/10.1007/978-1-4419-5653-8_2#citeas.

Ching-Lai Hwang, Young-Jou Lai, and Ting-Yun Liu. A new approach for multiple objective decision making. *Computers Operations Research*, 20(8):889 – 899, 1993. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(93)90109-V. URL http://www.sciencedirect.com/science/article/pii/030505489390109V.

Jain, Raj. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991. ISBN 978-0-471-50336-1. URL https://www.wiley.com/en-us/exportProduct/pdf/9780471503361.

KEDA. Keda | kubernetes event-driven autoscaling, 2021. URL https://keda.sh/.

Kibana. Kibana: Explore, visualize, discover data | elastic, 2021. URL https://www.elastic.co/kibana/.

Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gómez-Sánchez. Grid Characteristics and Uses: A Grid Definition. In *Grid Computing*, pages 291–298. Springer, Berlin, Heidelberg, feb 2004. ISBN 978-3-540-24689-3. doi: https://doi.org/10.1007/978-3-540-24689-3_36. URL https://link.springer.com/chapter/10.1007%2F978-3-540-24689-3_36.

C. Lo, D. Chen, C. Tsai, and K. Chao. Service selection based on fuzzy topsis method. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pages 367–372. IEEE, apr 2010. ISBN 978-1-4244-6702-0. doi: 10.1109/WAINA.2010.117. URL https://ieeexplore.ieee.org/abstract/document/5480593.

Peter Mell and Timothy Grance. The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology. In *Public Cloud Computing: Security and Privacy Guidelines*, pages 97–101. Nova Science Publishers, Inc., 2012. doi: https://doi.org/10.6028/NIST.SP.800-145. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

Microsoft. What is cloud computing ? a beginner's guide microsoft azure, 2020. URL https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

Christine Miyachi. What is "Cloud"? It is time to update the NIST definition? *IEEE Cloud Computing*, 5(3):6–11, jun 2018. ISSN 2325-6095. doi: 10.1109/MCC.2018.032591611. URL https://ieeexplore.ieee.org/document/8383652.

Z. u. Rehman, O. K. Hussain, and F. K. Hussain. Iaas cloud selection using mcdm methods. In *2012 IEEE Ninth International Conference on e-Business Engineering*, pages 246–251. IEEE, feb 2012. ISBN 978-1-4673-2601-8. doi: 10.1109/ICEBE.2012.47. URL https://ieeexplore.ieee.org/abstract/document/6468246.

R.W. Saaty. The analytic hierarchy process—what it is and how it is used. *Mathematical Modelling*, 9(3):161–176, 1987. ISSN 0270-0255. doi: https://doi.org/10.1016/0270-0255(87)90473-8. URL http://www.sciencedirect.com/science/article/pii/0270025587904738.

Thomas L. Saaty. *Analytic Heirarchy Process*. American Cancer Society, 2014. ISBN 9781118445112. doi: https://doi.org/10.1002/9781118445112.stat05310. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat05310.

Salesforce. Who we are - salesforce emea, 2020. URL https://www.salesforce.com/eu/company/about-us/.

Fábio Silva, Ricardo Martins, Marco Filipe Vieira Gomes, Alexandre Ferreira da Silva, José Manuel Machado, Paulo Novais, and Cesar Analide. Cloud computing environments for simulation of adaptable standardized work and electronic work instructions in industry 4.0. 2018.

Frank Simorjay. Sharedresponsibilitiesfor cloud computing, apr 2017. URL https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE1KVsM.

Matthew Smith, Michael Engel, Thomas Friese, Bernd Freisleben, Gregory A. Koenig, and William Yurcik. Security issues in on-demand grid and cluster computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 14–24. IEEE, may 2006. ISBN 0-7695-2585-7. doi: 10.1109/CCGRID.2006.1630919. URL https://ieeexplore.ieee.org/document/1630919.

M. Sun, T. Zang, X. Xu, and R. Wang. Consumer-centered cloud services selection using ahp. In *2013 International Conference on Service Sciences (ICSS)*, pages 1–6. IEEE, may 2013. ISBN 978-1-4673-6258-0. doi: 10.1109/ICSS.2013.26. URL https://ieeexplore.ieee.org/abstract/document/6519752.

Jayachander Surbiryala and Chunming Rong. Cloud Computing: History and Overview. In *2019 IEEE Cloud Summit*, pages 1–7. IEEE, aug 2019. ISBN 978-1-7281-3101-6. doi: 10.1109/CloudSummit47114.2019.00007. URL https://ieeexplore.ieee.org/document/9045506.

Tzeng, Gwo-Hshiung and Huang, Jih-Jeng. *Multiple Attribute Decision Making: Methods and Applications*. Chapman and Hall/CRC, jul 2011. ISBN 9781439861578. URL https://books.google.pt/books?hl=pt-PT&lr=&id=_C7n1ar4f8IC&oi=fnd&pg=PP1&dq=Multiple+Attribute+Decision+Making:+Methods+and+Applications&ots=5hhow045rN&sig=CGXZMYjt-cbIaFrAd3yxumXweh4&redir_esc=y#v=onepage&q=Multiple%20Attribute%20Decision%20Making%3A%20Methods%20and%20Applications&f=false.

Authentic Vision. Markets - authentic vison, 2021a. URL https://www.authenticvision.com/markets/.

Authentic Vision. The roi of anti-counterfeiting whitepaper, 2021b. URL https://www.authenticvision.com/wp-content/uploads/2021/01/Whitepaper-ROI-of-Anti-Counterfeiting_final-27.01.2021.pdf.

Authentic Vision. Solutions - authentic vison, 2021c. URL https://www.authenticvision.com/solutions/.

Authentic Vison. About - authentic vison, 2021. URL https://www.authenticvision.com/about/.

Visua. Counterfeit detection powered by best-in-class visual-ai, 2021a. URL https://visua.com/counterfeit-detection-powered-by-visual-ai/.

Visua. Visua - the visual-ai people, 2021b. URL https://visua.com/visua-the-visual-ai-people/.

Md Whaiduzzaman, Abdullah Gani, Nor Badrul Anuar, Muhammad Shiraz, Mohammad Nazmul Haque, and Israat Tanzeena Haque. Cloud service selection using multicriteria decision analysis. *The Scientific World Journal*, 2014:459375–459385, Feb 2014. ISSN 2356-6140. doi: 10.1155/2014/459375. URL https://doi.org/10.1155/2014/459375.

Zortag. Zortag - about us, 2021. URL https://www.zortag.com/AboutUs.

A

DEPENDÊNCIAS DO SERVIÇO DE VALIDAÇÃO

```
absl-py==0.11.0
altgraph==0.17
astor==0.7.1
astroid==2.5.1
cached-property==1.5.2
colorama==0.4.4
cssselect2==0.3.0
cycler==0.10.0
Cython==0.29.16
decorator==4.4.2
future==0.18.2
gast==0.3.3
grpcio==1.35.0
h5py==2.10.0
ImageHash==4.0
imageio==2.9.0
importlib-metadata==3.4.0
isort==5.7.0
Jinja2==2.10.1
Keras==2.2.4
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
lazy-object-proxy==1.5.2
lxml==4.6.2
Markdown==3.3.3
MarkupSafe==1.1.1
matplotlib==3.0.3
mccabe==0.6.1
mock==4.0.3
networkx==2.5
numpy==1.17.3
opencv-python==4.1.2.30
pandas==0.24.2
pefile==2019.4.18
```

```
Pillow==6.0.0
protobuf==3.15.1
PyInstaller==3.6
pylint==2.7.2
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2021.1
PyWavelets==1.0.3
pywin32-ctypes==0.2.0
PyYAML==5.4.1
reportlab==3.5.42
scikit-fuzzy==0.4.1
scikit-image==0.15.0
scikit-learn==0.20.3
scipy==1.4.1
six==1.15.0
svglib==1.0.0
tensorboard==2.2.0
tensorflow==2.2.0
tensorflow-estimator==2.2.0
termcolor==1.1.0
tinycss2==1.0.2
toml==0.10.2
tornado==6.0.2
tqdm==4.32.2
typed-ast==1.4.2
typing-extensions==3.7.4.3
webencodings==0.5.1
Werkzeug==1.0.1
wrapt==1.12.1
XlsxWriter==1.1.8
zipp==3.4.0
```

Listagem A.1: Dependências do Serviço de Validação.

B

FICHEIROS DE CONFIGURAÇÃO DO *KUBERNETES*

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: validation-service-top
spec:
  selector:
    matchLabels:
      app: validation-service-top
  template:
    metadata:
      labels:
        app: validation-service-top
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: type
                operator: In
                values:
                - validation
      terminationGracePeriodSeconds: 60
      containers:
      - name: validation-service-top
        image: un1qnxcr.azurecr.io/validation-service-top:latest
        lifecycle:
```

```
29          preStop:
30            exec:
31              command:
32                - sed -i '/run/ s/true/false/' /app/validateApp/
    config/execution-conf.json
33        imagePullPolicy: Always # IfNotPresent
34        resources:
35          requests:
36            memory: "2Gi" # Gi
37            cpu: "1300m"
38          limits:
39            memory: "3Gi"
40            cpu: "2000m"
41        ports:
42        - name: port-top
43          containerPort: 80
44        volumeMounts:
45        - mountPath: /app/data
46          name: imagesvolume
47
48        startupProbe:
49          httpGet:
50            path: /started
51            port: port-top
52          initialDelaySeconds: 90
53          failureThreshold: 100
54          periodSeconds: 15
55          timeoutSeconds: 5
56
57        livenessProbe:
58          httpGet:
59            path: /health
60            port: port-top
61          initialDelaySeconds: 600
62          periodSeconds: 180
63          timeoutSeconds: 5
64          successThreshold: 1
65          failureThreshold: 2
```

```
66
67        volumes:
68        - name: imagesvolume
69          azureFile:
70            secretName: storage-secret
71            shareName: image-volume/data
72            readOnly: false
```

Listagem B.1: Configuração do Deployment Top.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    namespace: default
5    name: validation-service-angle
6  spec:
7    selector:
8      matchLabels:
9        app: validation-service-angle
10   template:
11     metadata:
12       labels:
13         app: validation-service-angle
14     spec:
15       affinity:
16         nodeAffinity:
17           requiredDuringSchedulingIgnoredDuringExecution:
18             nodeSelectorTerms:
19             - matchExpressions:
20               - key: type
21                 operator: In
22                 values:
23                 - validation
24       terminationGracePeriodSeconds: 60
25       containers:
26       - name: validation-service-angle
27         image: un1qnxcr.azurecr.io/validation-service-angle:latest
28         lifecycle:
29           preStop:
30             exec:
```

```
            command:
              - /bin/sh
              - -c
              - sed -i '/run/ s/true/false/' /app/validateApp/config/execution-conf.json
    imagePullPolicy: Always # IfNotPresent
    resources:
      requests:
        memory: "2Gi" # Gi
        cpu: "1300m"
      limits:
        memory: "3Gi"
        cpu: "2000m"
    ports:
    - name: port-angle
      containerPort: 80
    volumeMounts:
    - mountPath: /app/data
      name: imagesvolume

    startupProbe:
      httpGet:
        path: /started
        port: port-angle
      initialDelaySeconds: 90
      failureThreshold: 100
      periodSeconds: 15
      timeoutSeconds: 5

    livenessProbe:
      httpGet:
       path: /health
       port: port-angle
      initialDelaySeconds: 600
      periodSeconds: 180
      timeoutSeconds: 5
      successThreshold: 1
      failureThreshold: 2
```

```
68
69          volumes:
70          - name: imagesvolume
71            azureFile:
72              secretName: storage-secret
73              shareName: image-volume/data
74              readOnly: false
```

Listagem B.2: Configuração do Deployment Angle.

```
1  apiVersion: keda.sh/v1alpha1
2  kind: ScaledObject
3  metadata:
4    name: azure-queue-scaledobject-top
5    namespace: default
6  spec:
7    cooldownPeriod: 300  # default 300s
8    pollingInterval: 5 # default 30s
9    minReplicaCount: 1  # default 0
10   maxReplicaCount: 4 # default 100
11   scaleTargetRef:
12     name: validation-service-top
13   triggers:
14   - type: azure-queue
15     metadata:
16       queueName: queuetop
17       queueLength: "1" # default 5
18     authenticationRef:
19         name: azure-queue-auth
```

Listagem B.3: Configuração do serviço *KEDA Top*.

```
1  piVersion: keda.sh/v1alpha1
2  kind: ScaledObject
3  metadata:
4    name: azure-queue-scaledobject-angle
5    namespace: default
6  spec:
7    cooldownPeriod: 300  # default 300s
8    pollingInterval: 5 # default 30s
9    minReplicaCount: 1  # default 0
```

```
  maxReplicaCount: 3 # default 100
  scaleTargetRef:
    name: validation-service-angle
  triggers:
  - type: azure-queue
    metadata:
      queueName: queueangle
      queueLength: "1" # default 5
    authenticationRef:
        name: azure-queue-auth
```

Listagem B.4: Configuração do serviço *KEDA Angle*.

```
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: azure-queue-auth
  namespace: default
spec:
  secretTargetRef: # Optional.
  - parameter: connection  # Required. Tem de se chamar connection
    name: keda-queue # Required.
    key: connectionstring # Required.
```

Listagem B.5: Ficheiro de autenticação para o serviço *KEDA*.

```
apiVersion: v1
kind: Secret
metadata:
  name: storage-secret
type: Opaque
data:
  azurestorageaccountname: <nome da conta em base 64>
  azurestorageaccountkey: <chave da storage account em base 64>
```

Listagem B.6: Segredo da *storage account*.

```
apiVersion: v1
kind: Secret
metadata:
  name: keda-queue
data:
```

```
6  azurestorageaccountname: <nome da conta em base 64>
7  connectionstring: <connection string em base 64>
```

Listagem B.7: Segredo da *azure queue storage*.

C

DOCKERFILE DO SERVIÇO DE VALIDAÇÃO

```
FROM ubuntu:20.04

ENV VALIDATOR_VERSION="validator-07-05-2021"
ENV SAS_TOKEN="..."

WORKDIR /app

COPY backends .

# Create data directory for the volume images mount
RUN mkdir data

## Let apt-get know we are running in noninteractive mode
ENV DEBIAN_FRONTEND noninteractive

## Make sure image is up-to-date
RUN apt-get update \
    && apt-get -y upgrade \
    && apt-get -y install wget gnupg \
    && apt -y install curl

RUN apt-get -y install unzip
RUN wget https://un1qone.blob.core.windows.net/validators/$VALIDATOR_VERSION.zip$SAS_TOKEN -k -O
     $VALIDATOR_VERSION.zip
RUN unzip $VALIDATOR_VERSION.zip
RUN rm $VALIDATOR_VERSION.zip


##############
# Wine setup #
##############

## Enable 32 bit architecture for 64 bit systems
RUN dpkg --add-architecture i386

```

```
35 ## Add wine repository
36 RUN wget -nc https://dl.winehq.org/wine-builds/winehq.key
37 RUN apt-key add winehq.key
38 RUN wget -qO- https://dl.winehq.org/wine-builds/Release.key | apt-key add -
39 RUN apt-get -y install software-properties-common \
40     && add-apt-repository 'deb http://dl.winehq.org/wine-builds/ubuntu/ bionic main' \
41     && apt-get update
42
43 ## Install wine and winetricks
44 RUN apt-get -y install --install-recommends winehq-devel cabextract
45
46 ## Install nodejs and npm
47 RUN apt -y install nodejs
48 RUN apt -y install npm
49
50 # Switch to create either a top service or angle service
51 COPY ./backends/validateApp/config/vapp-conf-top.js ./validateApp/config/vapp-conf.js
52
53 RUN npm install --prefix ./validateApp/ ./validateApp/
54
55 RUN npm install --prefix ./healthServer/ ./healthServer/
56
57 EXPOSE 80
58
59 RUN chmod 755 ./entrypoint.sh
60
61 CMD ./entrypoint.sh $VALIDATOR_VERSION
```

Listagem C.1: *Dockerfile* do Serviço de Validação.

D

SCRIPTS DE TESTE

```
$resourceGroup = "..."
$vmName = "..."

#               General Purpose       GPU              Optimized memory     Compute Optimized
$vmsizes = @("Standard_D4as_v4", "Standard_NV4as_v4", "Standard_E4as_v4", "Standard_F4s_v2")

# Start Server Variables
$executableName = "....exe"
$startServersScript = ".\StartUN1QnxServers.ps1"
$port1 = "8090"
$port2 = "8091"

# Jmeter Variables
$exe = 'C:\Users\Luis\Desktop\Dissertacao\Testes\apache-jmeter-5.4\bin\jmeter.bat'
$jmxfile = 'C:\Users\Luis\Desktop\Dissertacao\Testes\jp@gc_Ultimate_Thread_Group_10users_prod.
    jmx'
$resultpath = 'C:\Users\Luis\Desktop\Dissertacao\Testes\PowerShellScripts\Testes\Resultados\'

$beginningrelic = 1
$endingreplic = 1
do {
    $inputbeginvalid = [int]::TryParse((Read-Host 'De (replicas)?'), [ref]$beginningrelic)
    $inputendvalid = [int]::TryParse((Read-Host 'Ate (replica)?'), [ref]$endingreplic)
    if ((-not $inputbeginvalid) -or (-not $inputendvalid)) {
        Write-Host "Nao foram inseridos inteiros..."
    }
} while ((-not $inputbeginvalid) -or (-not $inputendvalid))

For ($i=$beginningrelic; $i -le $endingreplic; $i++) {

    Write-Host (Get-Date -Format "HH-mm-ss") '> Replica ' $i

    foreach($vmsize in $vmsizes) {

        Write-Host '#################################################################################'
```

```
35        Write-Host (Get-Date -Format "HH-mm-ss") '> A preparar para testar a maquina ' $vmName '
     com o tamanho ' $vmsize
36
37        # Atualizar tamanho da maquina
38        while ($true) {
39            try {
40                $vm = Get-AzVM -ResourceGroupName $resourceGroup -VMName $vmName
41                Write-Host (Get-Date -Format "HH-mm-ss") '> A redimensionar a maquina...'
42                $vm.HardwareProfile.VmSize = $vmsize
43                Update-AzVM -VM $vm -ResourceGroupName $resourceGroup
44                Write-Host (Get-Date -Format "HH-mm-ss") '> Maquina redimensionada.'
45                break
46            }
47            catch {
48                Write-Host (Get-Date -Format "HH-mm-ss") '> Erro no redemensionamento da maquina
    .'
49                Start-Sleep -Seconds 10
50            }
51        }
52
53
54
55        # Ligar maquina
56        while ($true) {
57            try {
58                Write-Host (Get-Date -Format "HH-mm-ss") '> A ligar a maquina...'
59                Start-AzVM -ResourceGroupName $resourceGroup -Name $vmName
60                Write-Host (Get-Date -Format "HH-mm-ss") '> Maquina inicializada.'
61                break
62            }
63            catch {
64                Write-Host (Get-Date -Format "HH-mm-ss") '> Erro a ligar a maquina.'
65                Start-Sleep -Seconds 10
66            }
67        }
68
69
70        # Ligar servidores de validacao
71        while ($true) {
72            try {
73                Write-Host (Get-Date -Format "HH-mm-ss") '> A ligar servidores de validacao...'
74                Invoke-AzVMRunCommand -ResourceGroupName $resourceGroup -Name $vmName -CommandId
     'RunPowerShellScript' `
75                    -ScriptPath $startServersScript -Parameter @{"arg1" = $port1;"arg2" = $port2
    ;"arg3"= $executableName}
76                Write-Host (Get-Date -Format "HH-mm-ss") '> Vamos esperar um bocado...'
```

```
77              Start-Sleep -Seconds 900 # esperar 10 minutos que o servidores estejam realmente
    ligados e a maquina repousada
78              Write-Host (Get-Date -Format "HH-mm-ss") '> Servidores ligados.'
79              break
80          }
81          catch {
82              Write-Host (Get-Date -Format "HH-mm-ss") '> Erro a ligar os servidores.'
83              Start-Sleep -Seconds 10
84          }
85      }
86
87
88      # Criar configuracoes iniciais para os testes
89      try {
90          Write-Host (Get-Date -Format "HH-mm-ss") '> A realizar os benchmarks a maquina.'
91          $batchDirectory = (Get-Date -Format "dd-MM-yyyy")
92          $resultSet = 'Replica' + $i + '-' + $vmsize + '-' +(Get-Date -Format "HH-mm-ss")
93          $resultfolder = $resultpath + $batchDirectory + '\' + $resultSet + '\'
94          $resultsfile = $resultfolder + $resultSet + '.csv'
95          New-Item -Path $resultpath -Name $batchDirectory -ItemType "directory" -Force
96          New-Item -Path ($resultpath + $batchDirectory) -Name $resultSet -ItemType "directory
    " -Force
97      }
98      catch {
99          Write-Host (Get-Date -Format "HH-mm-ss") '> Erro nos pedidos. A parar m quina.'
100         Stop-AzVM -ResourceGroupName $resourceGroup -Name $vmName -Force
101     }
102
103
104
105     # Iniciar medicao do tempo
106     $sw = [Diagnostics.Stopwatch]::StartNew()
107
108     # Iniciar Testes
109     $ProcessInfo = Start-Process -FilePath "$exe" -ArgumentList "-n -t $jmxfile -l
    $resultsfile -e -o $resultfolder" -Wait -PassThru
110
111     # Esperar que o teste termine
112     If ($ProcessInfo.ExitCode -eq 0) {
113         Write-Host (Get-Date -Format "HH-mm-ss") '> Teste a maquina ' $vmsize ' terminado'
114     }
115     Else {
116         # Program encountered an error
117         Write-Host 'Erro no teste'
118     }
119     $sw.Stop()
120     $sw.Elapsed
```

```
121
122         # Parar maquina
123         while ($true) {
124             try {
125                 Write-Host (Get-Date -Format "HH-mm-ss") '> A parar a maquina...'
126                 Stop-AzVM -ResourceGroupName $resourceGroup -Name $vmName -Force
127                 Write-Host (Get-Date -Format "HH-mm-ss") '> Maquina parada.'
128                 break
129             }
130             catch {
131                 Write-Host (Get-Date -Format "HH-mm-ss") '> Erro na paragem da maquina.'
132                 Start-Sleep -Seconds 10
133             }
134         }
135
136         Write-Host (Get-Date -Format "HH-mm-ss") '> Testes na maquina ' $vmName ' com o tamanho
    ' $vmsize ' terminados'
137         Write-Host '#############################################################################'
138     }
139 }
140
141 Write-Host (Get-Date -Format "HH-mm-ss") '> Testes terminados'
142
143 Stop-AzVM -ResourceGroupName $resourceGroup -Name $vmName -Force
```

Listagem D.1: TestarTodasMaquinasNVezes.ps1

```
1   param(
2       [string]$arg1,
3       [string]$arg2,
4       [string]$arg3
5   )
6
7    cd C:\Users\un1qone\Desktop\un1qnx\validator\
8
9    Start-Process -Verb runas -FilePath "cmd.exe" -ArgumentList "/C $arg3 $arg1"
10   Start-Process -Verb runas -FilePath "cmd.exe" -ArgumentList "/C $arg3 $arg2"
```

Listagem D.2: StartUN1QnxServers.ps1