Guilherme Rodrigues Alcobia Santos

Licenciado em Ciência e Engenharia Informática

Monitorização Autonómica de Contentores Docker e a sua Aplicação a Serviços da Saúde

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Doutor Hervé Miguel Cordeiro Paulino, Professor Auxiliar,
FCT UNL

Júri

Presidente: Prof. Doutor Miguel Pessoa Monteiro
Vogais: Prof. Doutor Hervé Miguel Cordeiro Paulino
Prof. Doutor Francisco Cipriano da Cunha Martins

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Monitorização Autonómica de Contentores Docker e a sua Aplicação a Serviços da Saúde

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Aos meus pais, ao meu irmão, à Raquel e aos meus amigos que tornaram esta tese possível.

Agradecimentos

Em primeiro lugar gostaria de expressar a minha profunda e sincera gratidão para com o professor Hervé Paulino, pelo seu apoio, dedicação e disponibilidade constante desde o início. A sua orientação foi fundamental para a conclusão da presente Dissertação. Queria também agradecer a todos os professores do Departamento de Informática da FCT pela qualidade e rigor com que exercem a sua função, pois sem dúvida foi um factor preponderante para o sucesso desta Dissertação.

Sou eternamente agradecido aos meus pais por todo o apoio, paciência e sacrifícios ao longo de toda a minha educação, sem eles e os valores por eles transmitidos a elaboração desta dissertação não seria possível. Igualmente agradecido ao meu irmão por ser um modelo para mim e por tudo o que me ensinou.

À Raquel Piteira por todo o amor, carinho e apoio prestado ajudando-me sempre a ultrapassar as dificuldades.

Aos meus amigos de sempre.

A todos os amigos que tive o privilégio de fazer durante o curso. Em especial ao Paulo Faria que partilhou comigo a maioria das adversidades e conquistas durante o curso. Também ao Nuno Mendonça, André Pereira e Paulo Figueira por tudo o que passámos e aprendemos juntos e também ao Pedro Sanches, Rui Sanches, Rui Pinguinhas e Paulo Conceição.

Quero ainda agradecer aos amigos que ganhei durante o Erasmus, as suas diferentes experiências e realidades enriqueceram o modo como encaro a vida.

Por último mas não menos importante queria agradecer a toda a minha equipa da SPMS. Em especial ao Tomé Vardasca pela confiança e pela oportunidade de trabalhar com inúmeras tecnologias disruptivas e pela experiência partilhada, o seu fascínio pela área é uma inspiração. À Bianca Nóbrega pelo apoio e concelhos oferecidos durante o desenvolvimento, a sua experiência contribuiu para melhorar a qualidade da solução. À Inês Armada Brás pela disponibilidade constante em desbloquear situações e facultar informação necessária. Não esquecendo ainda o Pavlo Figol, Michael Fidelis, Tiago Bragança, Matheus Franco, Gustavo Teles, Rui Pereira, Cristiano Marques e Arlete Monteiro, por tudo o que me ensinaram.

Resumo

Ao longo dos últimos anos tem-se assistido a uma evolução tecnológica sem precedentes, onde a maioria se não todos os sectores da sociedade sofreram alterações notáveis relacionadas directa ou indirectamente com a referida evolução. Um exemplo concreto e transversal a todos os sectores da sociedade são as milhares de aplicações móveis existentes que possibilitam a realização de todo o tipo de operações desde transferências bancárias, compra de produtos, consulta meteorológica, marcação de restaurantes entre tantas outras. Contudo o sector da Saúde numa perspectiva nacional, não acompanhou ao mesmo ritmo a oferta de soluções digitais de tantos outros sectores como a Banca ou a Restauração, havendo por isso poucas aplicações que permitam aos utilizadores portugueses interagir digitalmente com a sua saúde e instituições responsáveis.

Os Serviços Partilhados do Ministério da Saúde (SPMS) é uma empresa que nos últimos anos se tem dedicado a responder a esta lacuna através do desenvolvimento de aplicações móveis para a área da saúde e prevê não só aumentar a oferta de soluções como também o número de utilizadores das mesmas. Este aumento no entanto significará uma exigência acrescida sobre os serviços que suportam as aplicações e a infraestructura onde os serviços se encontram alojados. É objectivo da SPMS melhorar a sua infraestructura por forma a suportar eficazmente a adição de novos serviços e o aumento de carga sobre o sistema, tendo sempre como principal objectivo a preservação da qualidade de serviço prestada aos utilizadores. Concretamente a infraestructura terá que ser dinâmica na recuperação de falhas e na alocação de recursos dependendo o mínimo possível da intervenção humana.

Neste sentido a presente dissertação consistiu no desenvolvimento de um sistema autonómico capaz de adaptar dinamicamente os recursos alocados a um conjunto de serviços da saúde tendo em conta um conjunto de métricas alto nível definidas para cada um dos serviços. A adaptação realizada maximiza a qualidade final de cada um dos serviços, minimizando ao mesmo tempo o custo associado. Através dos testes realizados comprovou-se o correcto funcionamento do sistema desenvolvido.

Palavras-chave: Computação Autonómica, Saúde, Qualidade de Serviço, Cliente final, Computação Móvel em Cloud

Abstract

Through the years we have watch a technology evolution without precedents, where the majority of society sectors have gone through modifications, related directly and indirectly with the previous mentioned evolution. A concrete example and common to all sectors of society are the millions of mobile applications nowadays, that enable us to accomplish all types of operations, ranging from money transfer, online shopping, weather forecast to restaurant reservation and others. However, the Healthcare sector from a national perspective, didn't increase at the same pace the offering of digital solutions, in contrast with so many other sectors, therefore there is a gap on mobile applications that enable Portuguese people to interact digitally with their health and associated entities.

The Shared Services of Health Ministry (SPMS) is a company that in the last years has committed to fix the aforementioned gap by developing digital solutions such as mobile applications to the Healthcare sector, predicting an increase on the number of digital solutions but also in the number of users that interact with them. This growth however will be followed by an equal increase on the load of the services that support the mobile applications, and therefore on the infrastructure where the services are deployed. SPMS desires to improve its infrastructure to support the new services and the increase of the system load, always preserving the quality of service of the end user. The reformulated infrastructure must be dynamic meaning that it must be capable of recovering from errors and allocating resources on its own with the minimum human intervention possible.

Given the aforementioned context during this thesis was implemented an autonomic system that is capable of dynamically adapt the resources of a set of healthcare webservices using high level metrics defined for each service. The adaptation maximizes the service quality of each service, minimizing the costs whenever is possible.

The simulation results shows that the system indeed can adapt the services maximizing the service quality.

Keywords: Autonomic Computing, Health Care, Service Quality, End user, Mobile Computing on Cloud

Índice

Lista de Figuras

Siglas

ACI	Azure Container Instances.
API	Application Programming Interface.
CA	Computação Autonómica.
ECA	Event Condition Action.
FHIR	Fast Healthcare Interoperability Resources.
FL	Fuzzy Logic.
IA	Inteligência Artificial.
MAPE-K	Monitor Analyse Plan Execute Knowledge.
MVs	Máquinas Virtuais.
QoS	Quality of Service.
SLA	Service Level Agreement.
SLAs	Service Level Agreements.
SNS	Serviço Nacional de Saúde.
SPA	Sense Plan Act.
SPMS	Serviços Partilhados do Ministério da Saúde.

WSDL Web Services Description Language.

Capítulo 1

Introdução

1.1 Contexto

Os últimos 40 anos [16] foram pautados por uma evolução notória na área da tecnologia de informação, existindo hoje uma panóplia de sistemas altamente complexos para responder a todo o tipo de necessidades. Esta evolução desencadeou uma série de alterações na maioria dos sectores da sociedade, que aproveitaram o progresso para se desenvolverem e oferecerem soluções actuais. No entanto nem todos os sectores fizeram parte da referida evolução, nomeadamente no caso da indústria da saúde. Embora tenham existido progressos consideráveis de um ponto de vista técnico relativamente às ferramentas e métodos utilizados nos procedimentos clínicos, no que diz respeito à gestão de informação médica e oferta de soluções modernas para o paciente poder acompanhar o estado da sua saúde, o desenvolvimento não foi o mesmo.

Devido à importância, complexidade e escala da indústria da saúde nas sociedades actuais tornou-se imperativo o desenvolvimento de soluções que viabilizem um acompanhamento digital da saúde dos pacientes bem como a consulta de várias outras informações úteis relativas à saúde e instituições relacionadas.

Vários exemplos de soluções maturas e modernas existem no panorama internacional como é o caso do Practo[1], uma plataforma acedida por website ou aplicação móvel, que permite a marcação de consultas físicas e online, a consulta de receitas, a compra online de medicamentos associados às receitas, gestão dos horários de medicação, entre outras. Existem ainda associações como a Personal Connected Health Alliance[2] que tem como objectivo a integração da saúde e bem estar no quotidiano das pessoas através de protocolos e regras para interligação com os dispositivos pessoais.

[1] https://www.practo.com/
[2] http://www.pchalliance.org/

Relativamente à situação nacional poucas soluções concretas existem actualmente, embora o número de iniciativas e projectos tenham aumentado. Não obstante existem dois casos concretos, associados a instituições específicas, que reflectem o movimento de modernização e criação de soluções digitais por forma a tornar o controlo da saúde algo ubíquo no quotidiano da sociedade. Concretamente as aplicações MyCuf[3] e Hospital da Luz[4] possibilitam a marcação de consultas e exames, consulta de análises, consulta de receitas entre outras.

Tendo em conta o contexto apresentado e a necessidade mencionada o Ministério da Saúde criou os Serviços Partilhados do Ministério da Saúde (SPMS). A SPMS, fundada em 2010, é uma empresa pública que tem como missão a prestação de serviços partilhados – nas áreas de compras e logística, serviços financeiros, recursos humanos e sistemas e tecnologias de informação e comunicação – às entidades com atividade específica na área da saúde.

A referida empresa iniciou recentemente um processo de reformulação interno com o objectivo de oferecer serviços digitais de saúde ao utente, com o intuito de acompanhar os avanços tecnológicos actuais e ter um papel preponderante no que diz respeito à aproximação do utente e instituições à tecnologia na área da saúde. Exemplos concretos dessa missão são as três aplicações móveis disponibilizadas atualmente.

MySNS - Aplicação móvel que permite consultar notícias do Serviço Nacional de Saúde (SNS), consultar e avaliar instituições de saúde e aceder à área do cidadão.

MySNS Tempos - Aplicação móvel que permite a consulta do tempo médio de espera nas instituições hospitalares do SNS.

MySNS Carteira - Aplicação móvel que permite ao cidadão associar cartões digitais específicos por componentes informativos do seu interesse. O cartão de vacinas e de actividade física são dois exemplos concretos de cartões disponíveis na aplicação. O primeiro cartão permite a consulta das vacinas administradas e as vacinas por administrar, o segundo cartão demonstra o nível de actividade física do utente.

Para que as anteriores aplicações possam funcionar são necessários vários serviços de suporte. Todos os serviços são distintos, o que significa que têm exigências específicas e condicionantes muito próprias. Actualmente estão alojados numa Cloud, e é esta que fornece os recursos aos serviços.

No entanto devido à rápida expansão dos serviços oferecidos, a alocação e o controlo fino dos mesmos não foi uma prioridade, o que significa que não existem quaisquer tipos de garantias sobre a fiabilidade da infraestructura que suporta os referidos serviços. Outro factor a considerar na infraestrutura actual é a impossibilidade de adaptar os recursos e desempenho dos serviços consoante métricas definidas pelo administrador,

[3] https://play.google.com/store/apps/details?id=pt.saudecuf.myCUF&hl=en
[4] https://play.google.com/store/apps/details?id=pt.hospitaldaluz&hl=en

a única adaptação existente é providenciada pela Cloud, adaptação essa francamente insuficiente.

Tendo em conta as fraquezas definidas é objectivo da SPMS desenvoler um sistema que consiga controlar dinamicamente a qualidade de serviço dos serviços alojados, através de métricas personalizadas. O referido sistema tem ainda que ser capaz de recuperar de anomalias internas, de eventos externos que coloquem em causa o correcto funcionamento dos serviços e oferecer a mesma qualidade de serviço aos utilizadores independentemente da carga, minimizando ao mesmo tempo o número de recursos utilizados.

A presente dissertação será desenvolvida em contexto empresarial no âmbito de uma parceria entre o Departamento de Informática da Faculdade de Ciências e Tecnologia da UNL e a Altran, sendo que o projecto será desenvolvido na SPMS, cliente da Altran.

1.2 Formulação do Problema

Com o previsto crescimento do acesso aos serviços por parte dos utentes e com a adição de novos serviços, antevê-se um aumento da carga sobre o sistema, aumento esse que resultará na diminuição da qualidade de serviço caso não haja uma alocação inteligente de recursos. Não menosprezando o aumento do número de serviços por si só, é importante referir que estes têm requisitos e cargas heterogéneas o que contribui para o aumento da complexidade do problema a resolver, pois não é possível utilizar um modelo comum que satisfaça todos os serviços. Por fim, outra característica necessária a considerar tem que ver com os picos de carga, pois são períodos críticos na manutenção do nível de serviço desejado e implicam por isso uma adaptação rápida da alocação de recursos.

Actualmente todos os serviços são executados dentro de contentores Docker[5]. Docker é uma empresa fundada em 2013 que revolucionou o conceito de *Containerization* [7] e potenciou a sua utilização. Um contentor é uma abstração ao nível aplicacional que consiste num ambiente que contém todas as dependências necessárias para a execução de um pedaço de software ou aplicação. A grande vantagem dos contentores é o facto de serem leves, poderem ser iniciados rapidamente e isolarem o software que contêm, permitindo que este corra em qualquer plataforma e sistema operativo de forma segura.

A figura 1.1 representa a arquitectura actual dos serviços que alimentam as aplicações móveis da SPMS. Todos os serviços são executados dentro de contentores Docker, estes por sua vez estão alojados em Máquinas Virtuais (MVs) da Cloud Azure[6].

Com a presente organização, o controlo das Máquinas Virtuais é manual e a auto-escalabilidade dos serviços é apenas assegurada ao nível dos contentores, o que significa que está limitada aos recursos disponibilizados pelas MVs que alojam os contentores. O Azure oferece mecanismos de auto-escalabilidade para as Máquinas Virtuais, de uma forma manual ou automática tendo em conta os valores de utilização de CPU ou até mesmo com base num grupo de regras. No entanto, estas regras são potencialmente não

[5]Docker: https://www.docker.com
[6]Azure: https://azure.microsoft.com

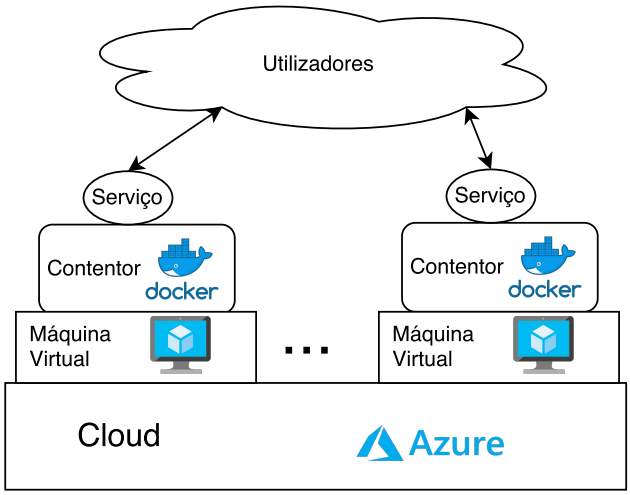

Figura 1.1: Arquitectura geral da secção de aplicações da SPMS

escaláveis, pois por cada novo serviço novas regras terão de ser definidas. Outro problema associado à definição manual de regras, tem que ver com os conflitos que poderão surgir entre as mesmas e o facto destes conflitos só durante a execução poderem ser detectados. Por fim, por se tratarem de regras definidas na plataforma Azure tornam-se totalmente dependentes da mesma para terem validade.

No que se refere à tolerância a falhas, o Docker assegura que os contentores onde os serviços correm são restaurados, uma vez que todos os serviços não têm estado, o mecanismo de recuperação oferecido pelo Docker é suficiente para assegurar a recuperação dos mesmos. A recuperação de MVs por outro lado é gerida pelo Azure tendo por base a "saúde" de uma Máquina Virtual. No entanto, a execução automática do contentor bem como a diferença de endereços de uma máquina virtual ao ser reiniciada são pontos a considerar.

A próxima lista enumera os principais problemas que se pretendem resolver nesta dissertação:

1. O modelo actual não possui nenhum método que considere métricas definidas pelo administrador no processo de alocação de recursos. O que significa que a qualidade de serviço não é tida em consideração.

2. O modelo actual é inteiramente dependente do provedor de recursos no que diz respeito à gestão das máquinas virtuais, não existindo nenhum componente intermédio que adapte os serviços independentemente do provedor.

3. A monitorização dos serviços não é suficientemente detalhada pois não são consideradas métricas específicas sobre o funcionamento de cada serviço, o que por sua vez torna o controlo do sistema vago.

4. A gestão dos serviços é feita de forma isolada, tendo em conta apenas a carga do serviço, menosprezando outros factores e não tendo em consideração os restantes serviços e as ligações entre eles.

5. O modelo actual é consideravelmente dependente da intervenção humana no que diz respeito à gestão das entidades (máquinas virtuais) que suportam os contentores.

Resumidamente o intuito deste projecto consiste no desenvolvimento de um sistema autonómico capaz de adaptar dinamicamente os recursos alocados aos serviços dos quais é responsável, e com isso potenciar a qualidade de serviço dos mesmos. Pretende-se ainda que a adaptação do sistema seja feita considerando os factores convencionais como a utilização do CPU, ou a taxa de processamento de pedidos, mas sobretudo considerando métricas alto nível definidas pelo administrador, para que o sistema consiga obter a melhor qualidade de serviço possível tendo em conta os objectivos definidos. O sistema será totalmente desenhado e desenvolvido em torno dos conceitos contentor e Pod.

1.3 Solução Proposta

À luz dos anteriores desafios e contexto, onde o projecto será desenvolvido, é agora apresentada a solução inicial.

Um dos primeiros pontos que pretendemos endereçar é a dependência do sistema actual em MVs, pois a mesma implica um grande esforço de configuração para iniciar os contentores onde são executados os serviços, o que por sua vez faz com que o poder modular das imagens Docker não seja totalmente aproveitado, tornando o processo de automatização e gestão dos serviços bastante mais complexo.

Neste sentido pretende-se eliminar a referida dependência e fazer com que a execução dos contentores esteja apenas relacionada com a qualidade e configuração das imagens Docker, removendo deste modo toda a etapa de configuração associada às MVs.

Para excluir as Máquinas Virtuais (MVs) do sistema utilizar-se-ão Pods, entidades que irão assegurar a execução dos contentores. O termo *Pod* é utilizado regularmente pela comunidade *Docker* para designar uma entidade que é composta por um ou múltiplos contentores, que comunicam entre si por forma a garantirem o correcto funcionamento da entidade que é o *Pod*. Outro modo de encarar um Pod é pensando no mesmo, como um sistema composto pelas relações dos vários contentores que contém. Através da utilização de *Pods*, todo o ambiente necessário para a execução dos contentores é fornecido à partida, removendo deste modo a etapa de configuração de Máquinas Virtuais, tal como desejado. A utilização de *Pods* como ambiente de suporte aos contentores é a principal diferença estrutural a ser introduzida.

Na figura 1.2 é demonstrada uma representação alto nível do sistema a implementar. A arquitectura proposta vai de encontro aos princípios de um sistema autonómico. Sucintamente um sistema autonómico é um sistema capaz de auto adaptar-se com base em

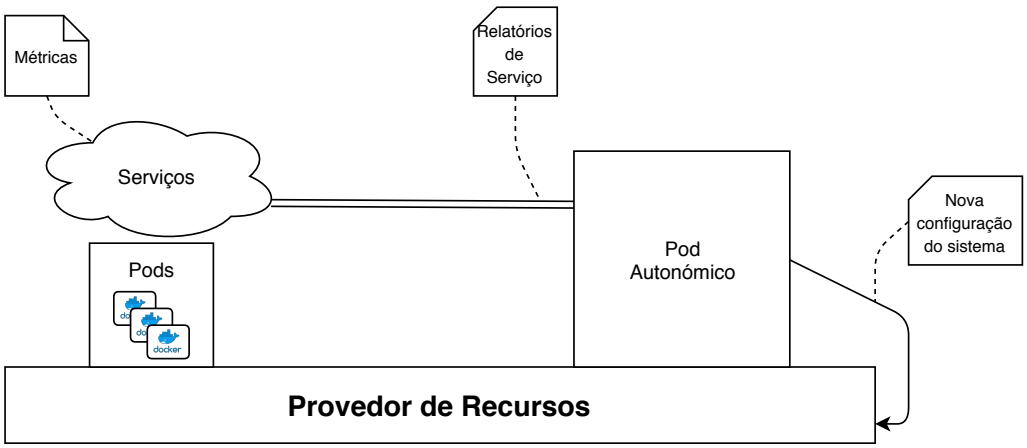

Figura 1.2: Solução proposta (simplificada)

objectivos previamente estabelecidos e estímulos externos, de forma independente e com o mínimo de intervenção humana possível.

A relevância de um sistema autonómico para o caso concreto desta dissertação prende-se com a ambição de desenvolver um sistema capaz de adaptar a quantidade de recursos atribuídos aos serviços de forma dinâmica e autónoma.

Desta forma é objectivo desta dissertação implementar um sistema baseado em Pods e em Contentores, que através da monitorização dos serviços, alojados em Pods, consiga analisar o estado global do sistema e averiguar se o mesmo necessita de ser alterado. Se assim for necessário, deve ainda ser capaz de calcular um estado ideal, tendo em conta o meio onde está inserido e produzir um plano de alteração, que é posteriormente enviado para o provedor de recursos para que este proceda a alteração do estado do sistema. É importante referir que todo este processo tem como objectivo principal maximizar a qualidade de serviço dos utilizadores finais, tentando sempre que possível minimizar o custo associado à execução dos serviços.

1.4 Contribuições

As contribuições do trabalho a desenvolver no contexto desta dissertação são:

- Concepção e implementação de um sistema autonómico capaz de reconfigurar dinamicamente Pods, onde estão alojados os serviços que controla. A reconfiguração basear-se-á na análise dos valores das métricas alto nível definidas para cada serviço que indicam o estado corrente de cada um.

- A avaliação do desempenho do sistema autonómico, relativamente à sua capacidade de optimização da qualidade de serviço e minimização de recursos quando submetido a diferentes cargas.

1.5 Estrutura do Documento

O presente documento está estruturado da seguinte forma:

- No Capítulo 1 é feita uma introdução ao projecto, onde é explicada a motivação por detrás do mesmo, bem como os desafios inerentes. É ainda apresentada a arquitetura geral da solução proposta.
- No Capítulo 2 é analisado o que é a computação autonómica, quais os principais componentes, que modelos são utilizados e quais são os diferentes níveis de autonomia existentes. É analisado o que é a qualidade de serviço, qual a sua importância, que standards existem para a definição formal de métricas e quais as métricas relevantes. São também discutidos algoritmos para a optimização da alocação de recursos.
- No Capítulo 3 é apresentada a solução proposta. São identificados os componentes essenciais da arquitectura bem como a função e responsabilidade de cada componente. São ainda discutidos os detalhes de implementação e as tecnologias utilizadas.
- No Capítulo 4 é feita a avaliação do sistema desenvolvido por forma a perceber se a solução é vantajosa tendo em conta o contexto onde se insere.
- No Capítulo 5 são realizadas as conclusões finais do sistema e são detalhadas as melhorias e adições futuras mais relevantes.

Capítulo 2

Estado da Arte

O presente capítulo aborda as principais temáticas subjacentes ao projecto a desenvolver. Na secção 2.1 é analisado o tópico da Computação Autonómica (CA), na secção 2.2 é explicada como se pode fazer a definição e controlo de métricas através de Service Level Agreements (SLAs). Seguidamente na secção 2.3 é abordada a problemática da monitorização. A avaliação da utilidade do estado do sistema é analisada na secção 2.4. Por fim na secção 2.5 são analisados métodos para a adaptação dinâmica do sistema e na secção 2.6 é proposta uma metodologia para a resolução do problema de optimização associado à alocação de recursos.

2.1 Computação Autonómica

Desde o início da evolução humana e posterior evolução tecnológica que o ser humano baseia-se no que o rodeia para melhorar e criar novas tecnologias. A Inteligência Artificial (IA) é um exemplo moderno, onde as redes neuronais amplamente utilizadas, baseiam-se no funcionamento das células nervosas do cérebro humano.

A IA é mais um passo na direção da criação de sistemas completamente autonómicos, que não necessitam de um ser humano para controlar o seu funcionamento. Sistemas que à semelhança dos organismos vivos, reagem a estímulos do ambiente que os rodeia e com isso tomam decisões e adaptam-se.

O campo da Computação Autonómica (CA) surge no final dos anos 1990 e início dos anos 2000 e é oficializado numa conferência na Universidade de Harvard pelo então vice presidente de investigação da IBM Paul Horn [8]. A grande motivação por detrás do referido paradigma tinha que ver com o aumento exponencial da complexidade dos sistemas e com a incapacidade humana para os conseguir controlar. Desde então o movimento tem ganho cada vez mais aderência e hoje em dia já são muitos os exemplos onde a CA está

presente [2, 3, 9, 10, 27].

2.1.1 Pilares da CA

Um sistema autonómico [8, 17, 22] assenta em quatro pilares:

Auto Configuração - O sistema deve ser capaz de configurar-se a si próprio, sem qualquer tipo de intervenção humana, de acordo com objectivos definidos pelo administrador. De modo a que, quando um componente seja introduzido, consiga integrar-se eficazmente e o resto do sistema seja também capaz de adaptar-se pacificamente à presença do novo componente.

Auto Optimização - O sistema deve ser capaz de continuamente melhorar o seu funcionamento, através da procura e identificação de oportunidades por forma a aumentar a sua eficiência, e com isso melhorar o seu custo e o seu desempenho. Para que isso seja possível, o sistema deverá aperfeiçoar as suas métricas e execução, conforme os eventos ocorridos, até encontrar um balanço perfeito.

Auto Regeneração - O sistema deve ser capaz de detetar, diagnosticar e reparar erros sejam estes bugs ou falhas, no software ou hardware. Para atingir o referido anteriormente, o sistema, poderá recorrer a informação acerca da sua configuração, e através de um componente de diagnóstico de problemas, analisar logs e outros dados, e com isso agir em conformidade.

Auto Proteção - O sistema deve ser capaz de proteger-se de ataques externos, intencionais ou não, e de falhas que possam ocorrer internamente. Esta característica tem uma relação próxima com a capacidade de auto-regeneração. Os objectivos principais deste componente são: antecipar, detectar, identificar e proteger o sistema contra ataques e falhas.

As propriedades anteriormente citadas permitem a obtenção dos princípios, enunciados em [17], que definem um sistema autonómico.

2.1.2 Graus de Autonomia

Segundo a classificação proposta por Huebscher e McCann em [18], um sistema pode ser avaliado em relação a cinco níveis diferentes de autonomia:

Suporte - Nível mais baixo de autonomia, onde o sistema foca-se apenas em determinado aspecto ou componente da arquitectura, para melhorar o desempenho global autonomamente.

Núcleo - Neste patamar a função de gestão está inteiramente responsável por comandar o sistema. No entanto as decisões tomadas não têm em consideração os objectivos alto nível definidos por humanos.

Autónomo - Nível caracterizado por um sistema que se auto-adapta ao meio que o rodeia, por forma a ultrapassar os desafios e eventuais falhas, mas que não mede o seu desempenho e não se preocupa em melhorar a forma como se adapta de maneira a optimizar possíveis objectivos.

Autonómico - Um sistema dotado do nível de autonomia característico deste patamar, para além de englobar todas as anteriores capacidades, tem como foco os objectivos alto nível definidos por humanos, como objectivos de negócio ou SLAs. Neste nível o sistema avalia o seu próprio desempenho e adapta-se em conformidade.

Ciclo fechado - O quinto e último nível, diz respeito a sistemas que para além de autonómicos, recorrem à inteligência para guiar e melhorar a auto-gestão. Um sistema de ciclo fechado engloba sistemas que evoluem a lógica que os guia, dependendo de quão bem sucedida a "lógica anterior" tiver sido. O conceito referido é muito semelhante ao que tem sido estudado no campo da Inteligência Artificial.

A própria IBM propôs primeiramente um conjunto de níveis em [19], mas que contrariamente aos níveis analisados, que se demonstram abrangentes e generalistas, a classificação em causa era demasiado limitativa e de pouca utilidade para a caracterização de sistemas autonómicos no geral.

2.1.3 Modelos

As duas subsecções anteriores explicitam as principais características de um sistema autonómico e como classificar determinado sistema em relação ao seu nível de autonomia. São noções importantes para perceber conceptualmente o que é um sistema autonómico e em que medida diferem de sistemas convencionais. O próximo passo será a materialização destas noções conceptuais num modelo concreto.

Na sua génese os sistemas autónomicos podem ser considerados agentes inteligentes [20, 34]. Um agente inteligente de acordo com a definição mais geral é um termo utilizado para definir um componente hardware ou software capaz de funcionar, sem a intervenção directa do ser-humano ou terceiros possuindo algum tipo de controlo sobre as suas ações e estado interno. É capaz de interagir com outros agentes através de algum tipo de linguagem de comunicação de agentes. É capaz de percepcionar o meio onde está inserido e responder às alterações que nele ocorrem e não se limita a agir conforme estímulos do meio ambiente, é também capaz de agir face a objectivos definidos.

As primeiras arquitecturas propostas para agentes autónomos [23], correspondem quase que na íntegra com as primeiras arquitecturas Sense Plan Act (SPA), utilizadas inicialmente em robots.

Este modelo é caracterizado, como se pode constatar na figura 2.1, por três etapas principais. A primeira fase, intitulada Percepcionar (*Sense*), consiste na aquisição de informação bruta e posterior tradução da mesma, para um formato perceptível pelo sistema. Na segunda fase, Planear (*Plan*), através dos dados recolhidos anteriormente e com base

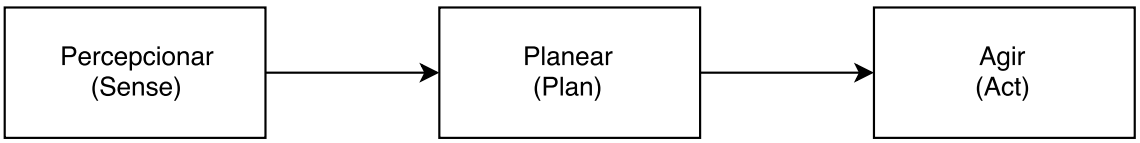

Figura 2.1: Modelo Sense Plan Act

num objectivo, é elaborado um plano para atingir o objectivo. Por fim a terceira e última fase designada Agir (*Act*), consiste na execução das acções definidas no plano.

O referido modelo é sequencial e unidirecional, ou seja os dados passam dos sensores, para o modelo, do modelo para o plano e por fim para os actuadores e nunca no sentido contrário. Grande parte da lógica concentra-se no componente de planeamento.

Um exemplo concreto de um sistema baseado no modelo em questão é o caso da framework proposta por Garlan em [11] que consiste num elemento de monitorização que recolhe dados relativos à execução de um sistema. Os dados recolhidos são abstraídos e comparados relativamente às propriedades de um modelo arquitectural. Propriedades diferentes são avaliadas. Caso infrinjam com as restrições estabelecidas, são tratadas por um mecanismo de reparação que adapta a arquitectura. As alterações efectuadas são propostas ao sistema que é monitorizado.

O SPA é um modelo bastante antigo, que entretanto sofreu algumas iterações [12] para responder às fraquezas que lhe eram características, como por exemplo, o facto de concentrar toda a inteligência num só componente e os restantes elementos não terem grande complexidade (tornaria-se impraticável em termos de escalabilidade, concentrar toda a lógica num só componente). Mas a principal limitação, tinha que ver com o facto de o modelo não lidar bem com a incerteza e imprevisibilidade do ambiente, uma vez que o feedback da execução do plano não era tido em consideração (ciclo aberto) e o fluxo de informação ser irreversivelmente unidirecional.

2.1.3.1 MAPE-K

Um modelo amplamente utilizado na problemática da Computação Autonómica que também se baseia no SPA, embora mais refinado, designa-se Monitor Analyse Plan Execute Knowledge (MAPE-K). Foi proposto pela IBM em 2005 [19], com o objectivo de ser um modelo de referência nos ciclos de controlo para sistemas autonómicos.

No ciclo fechado MAPE-K é possível destacar dois elementos principais, o Gestor Autonómico (*Autonomic Manager*) e o Elemento Gerido (*Managed Element*). O último pode ser qualquer recurso software ou hardware, como por exemplo um servidor Web, uma base de dados, um componente de software específico de uma aplicação, um conjunto de máquinas, etc.

Os sensores recolhem informação sobre o Elemento Gerido. A informação recolhida depende do elemento, se se tratar de um servidor Web, por exemplo, os dados recolhidos podem ser o tempo de resposta aos clientes, a utilização do disco, do CPU e da memória.

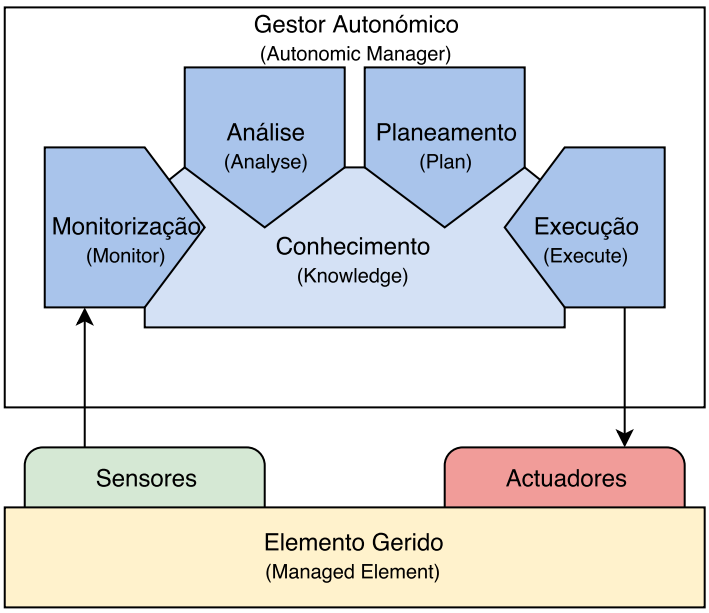

Figura 2.2: Modelo MAPE-K

Os atuadores realizam alterações ao Elemento Gerido. No caso da *Cloud*, um exemplo seria a remoção ou adição de máquinas virtuais.

O Gestor Autonómico é o componente que recebe a informação dos sensores e através dos atuadores altera o Elemento Gerido. Idealmente este componente deve ser configurável por administradores, através da utilização de objetivos alto-nível. É com base na informação recolhida pelos sensores e do conhecimento armazenado que o elemento planeia e executa as acções baixo-nível necessárias para atingir os objectivos anteriores.

A fase de monitorização (*Monitor*) envolve a captura de propriedades do ambiente, e como visto anteriormente é feita através de componentes hardware ou software designados sensores. O tipo de propriedades consideradas na monitorização bem como os sensores utilizados são na maioria dos casos específicos ao Elemento Gerido.

Existem dois tipos de monitorização em sistemas autonómicos [18]:

- **Monitorização Passiva** - Compreende a recolha de informação, sem a alteração do código do Elemento Gerido, através de ferramentas já acessíveis. Um caso comum é utilização de funções do sistema operativo Linux para a obtenção de informação, como é o caso do comando top, para a analisar a utilização do CPU ou mesmo o vmstat.

- **Monitorização Activa** - Este tipo de monitorização requer na maioria dos casos a alteração directa do código para o efeito de monitorização, por forma a ser possível capturar dados específicos dos serviços.

A informação recolhida é depois fornecida ao componente de Análise (*Analyse*), que tem como função observar e analisar a informação, por forma a validar se a execução

do Elemento Gerido está de acordo com as restrições estabelecidas. Caso haja alguma situação anómala e o sistema esteja numa situação que permita a resolução da anomalia, o componente elabora um ”relatório” onde ficam explícitas as alterações a efectuar.

Com base nas modificações necessárias realizar o componente de Planeamento (*Plan*) elabora um plano, onde ficam descriminadas as ações baixo nível a executar.

Por fim o componente de Execução (*Execute*) através da interface dos atuadores executa as ações estabelecidas no plano.

O componente de Conhecimento (*Knowledge*) é uma base de dados ou um registo onde é armazenada informação. A informação pode conter desde: sintomas, regras, pedidos de alteração de regras, planos gerados anteriormente, entre outros. Os dados armazenados neste componente servem para estender o conhecimento do Gestor Autonómico para que este consiga tomar decisões mais criteriosas e mais contextualizadas.

Um aspecto importante a analisar é o facto do componente de Conhecimento estar em contacto com todos os outros. Se considerarmos o componente de Análise por exemplo, pode dar-se o caso de em determinada situação a informação proveniente da Monitorização não ser suficiente para averiguar se se trata de uma anomalia e ser necessário mais dados. Recorrendo ao conhecimento já acumulado podem ser retiradas pistas importantes.

O próprio componente de Execução enquanto realiza as acções definidas no plano, pode actualizar o elemento de Conhecimento em relação ao plano que está a executar, às acções realizadas e eventuais erros.

2.1.4 Aplicação no Trabalho

Esta secção explorou as características principais da Computação Autonómica, o que é, como classificar e como implementar, através do modelo MAPE-K. A pertinência deste ramo da computação em relação ao sistema a desenvolver prende-se com a natureza dinâmica a que o último estará sujeito e com o facto da adaptação ter que ser feita com a menor intervenção humana possível.

O MAPE-K servirá de base para o sistema a desenvolver, facto que será notório na arquitectura da solução apresentada na secção 3.1. Concretamente existirá uma fase de monitorização onde serão recolhidos dados relativos aos serviços em execução por forma a perceber em que estado se encontram. Uma fase de análise/controlo em que utilizando a informação recolhida e processada na fase anterior e de acordo com as restrições definidas pelo administrador é analisado se a qualidade de cada serviço é adequada. Caso não seja é ainda o componente em questão responsável por calcular uma nova configuração para o sistema por forma a melhorar a qualidade de serviço.

A nova configuração acarreta acções, acções essas que serão executadas pelo componente de execução. No caso concreto do sistema a desenvolver existirá um componente específico por provedor de recursos, que traduz as ações gerais em acções específicas da plataforma, para que as alterações possam ser efectuadas. Por fim existirá também um

componente de conhecimento, implementado através de uma base de dados, onde serão guardados vários tipos de informação sobre a execução do sistema bem como os diferentes requisitos estipulados. Relativamente ao nível de autonomia do sistema a desenvolver será Autonómico pois não será capaz de melhorar a sua lógica interna.

2.2 Definição de Qualidade de Serviço

Por forma a que um sistema autonómico consiga analisar e perceber se está ou não num estado favorável necessita de objectivos/métricas que consiga avaliar e com isso inferir um nível de utilidade global do sistema. Nesta secção é abordada a problemática associada à caracterização e levantamento de métricas pertinentes para o contexto concreto do projecto, e ainda a necessidade da utilização de Service Level Agreements (SLAs) para a definição de métricas num formato facilmente perceptível pelo sistema.

Primeiramente é importante definir o que é a qualidade de serviço comummente designada Quality of Service (QoS). A qualidade de serviço representa o desempenho geral de um serviço seja este de que tipo for. Para calcular a referida medida de desempenho, são necessárias métricas que serão posteriormente avaliadas e comparadas, em relação a um valor estipulado.

O primeiro passo na definição do QoS é o levantamento de métricas direcionadas ao sistema.

2.2.1 Métricas QoS

São vários os exemplos de sistemas [4, 9, 10, 13–15, 27], que consideram a qualidade de serviço nas suas soluções, seja num contexto de serviços Web, Sistemas Distribuídos ou Cloud. Na esmagadora maioria dos casos existem três métricas de referência:

Tempo de execução - Representa o tempo que determinado serviço está em execução desde o momento em que foi iniciado.

Disponibilidade - Representa a fração de tempo em que determinado serviço está pronto a receber e processar pedidos.

Taxa de processamento - Representa o ritmo a que os pedidos são eficazmente processados.

Embora as métricas anteriores sejam importantes, não são de todo suficientes se se pretender obter uma granularidade fina em relação ao controlo da qualidade dos serviços e consequentemente do sistema. Para que haja uma adaptação fina do sistema, com base na qualidade de serviço, é necessário agrupar as métricas atómicas por áreas relevantes para o cálculo do QoS global, ou seja a qualidade de serviço é o somatório dos níveis de serviço de várias categorias, como a segurança ou o desempenho, cuja o cálculo dos referidos níveis de serviço é baseado nas métricas pertencentes a cada categoria.

Numa tentativa de categorizar as métricas relevantes para o QoS, B. Sabata propõe em [28], três grandes grupos: desempenho, segurança e importância relativa.

O desempenho diz respeito aos parâmetros relacionados com o desempenho dos serviços como por exemplo: o tempo total para concluir uma tarefa, o volume de computações realizadas e a taxa de erros. Por sua vez a importância relativa diz respeito ao preço que o utilizador está disposto a pagar pela qualidade de determinado serviço. A segurança diz respeito aos níveis de segurança que são fornecidos aos serviços.

É uma abordagem generalista e que por isso não é ideal para representar a natureza espicífica do sistema a desenvolver. Nesse sentido a classificação proposta em [29] enumera sete grandes áreas de requisitos QoS que se aproximam mais das necessidade concretas de um sistema autonómico sendo elas:

1. **Disponibilidade** - Capacidade do serviço estar disponível para ser utilizado.
2. **Acessibilidade** - Capacidade do serviço responder a um pedido.
3. **Integridade** - Capacidade do serviço executar correctamente o que lhe é pedido.
4. **Desempenho** - Qualidade do serviço relativamente ao tempo e eficácia com que realiza as tarefas.
5. **Fiabilidade** - Capacidade do serviço em manter a qualidade e viabilidade do mesmo.
6. **Regulação** - Qualidade de um serviço relativamente aos standards que este utiliza, bem como o respeito para com os requisitos de qualidade acordados.
7. **Segurança** - Capacidade do serviço assegurar os três princípios da tríade de segurança: confidencialidade, integridade e disponibilidade.

A divisão anterior segmenta eficazmente os requisitos QoS a avaliar nos serviços que compõe o sistema. Contudo dada a natureza dinâmica, característica da Computação Autonómica, foi necessário definir dois grupos adicionais para avaliar com maior detalhe a qualidade do sistema em relação à sua adaptabilidade.

1. **Adaptabilidade**. Capacidade do sistema em adaptar os seus constituintes à configuração pretendida.
2. **Sensibilidade**. Capacidade do sistema relativamente à deteção de eventos, que justifiquem a alteração do sistema.

Para a avaliação do nível de serviço das áreas anteriormente definidas são necessárias métricas, características atómicas possíveis de serem medidas e posteriormente avaliadas, por forma a inferir um nível de qualidade. Cada métrica a utilizar irá corresponder a uma das nove categorias. Na tabela 2.1 estão descriminadas as métricas que serão consideradas e qual o seu significado. A propriedade de degradação que aparece na coluna três da

tabela é relativa à deterioração do valor de utilidade da métrica face a reduções no valor concreto da métrica, esta propriedade será importante para a distinção de duas funções utilitárias apresentadas na secção 2.4.1.

Categoria	Métrica	Degradação	Descrição
Disponibilidade	Disponibilidade	Inelástico	Tempo em que um serviço está disponível para receber pedidos ou desempenhar a sua função.
Fiabilidade	Tempo médio entre falhas(crash)	Inelástico	Tempo médio entre as falhas de um serviço.
	Tempo médio para reparar(iniciar)	Inelástico	Tempo médio para o serviço iniciar após uma falha.
Integridade	Taxa de transações falhadas	Elástico	Número de métodos não concluídos por unidade de tempo.
	Taxa de transações concluídas	Elástico	Número de métodos concluídos por unidade de tempo.
Desempenho	Taxa de processamento de pedidos	Elástico	Número de pedidos eficazmente processados por unidade de tempo.
	Latência	Elástico	Tempo médio para processar e responder a um pedido.
Regulação	Taxa de violação de restrições	Elástico	Número de restrições violadas por unidade de tempo.
Adaptabilidade	Tempo médio de adaptação	Elástico	Tempo que o elemento gestor demora a produzir uma nova configuração.

Tabela 2.1: Métricas possíveis.

2.2.2 Representação do QoS através de SLAs

A última etapa da definição do QoS é a sua representação. Para que o sistema consiga interpretar e armazenar as métricas estabelecidas em relação aos diferentes serviços, é necessário criar documentos formais, numa linguagem facilmente interpretável pelo Elemento Gestor, onde estará estipulado o nível de serviço desejado bem como as métricas exigidas para a avaliação da qualidade do serviço. Uma possibilidade é a utilização de SLAs.

Um SLA é um acordo do nível de serviço entre duas entidades, normalmente um cliente e um provedor, onde estão descritas características técnicas e não técnicas de um serviço, incluído os requisitos QoS e as métricas associadas. Existem várias ferramentas e linguagens de especificação de SLAs alguns exemplos importantes são o WSLA, SLAng e o WSOL.

O WSLA [21] criado pela IBM é uma framework para especificar e monitorizar SLAs de serviços Web. A linguagem WSLA define as partes envolvidas no acordo, a descrição do serviço que o provedor oferece ao consumidor e as obrigações do acordo, onde estão definidas as garantias e as restrições do SLA. A referida linguagem é especificada em XML e é compatível com a linguagem de definição de serviços Web (Web Services Description

Language (WSDL)). Embora seja uma solução apelativa e muito completa não é ideal para o sistema que pretendemos construir pois trata-se de uma ferramenta muito madura e com princípios bem definidos onde é dada muita importância às obrigações e penalizações entre as entidades envolvidas, característica que não é relevante para o caso concreto do nosso sistema.

Por sua vez o WSOL [31] é uma proposta mais focada na medição e definição de restrições QoS, à semelhança do SLAng. É baseada em XML e é também compatível com o standard WSDL. No entanto é uma abordagem muito direcionada à avaliação do QoS através de uma definição rigorosa e exaustiva das características e restrições a considerar em cada método de cada serviço.

A solução mais favorável face às necessidades do sistema a desenvolver é a linguagem SLAng [25] abordada seguidamente.

2.2.2.1 SLAng

O SLAng é uma linguagem para definir acordos do nível de serviço, que oferece um formato para a negociação das propriedades QoS, bem como os meios para a definição e captura das propriedades de forma clara e formal. No entanto, a principal vantagem face às soluções anteriores é a possibilidade de especificar contractos entre entidades de níveis de abstração diferentes, tornando-a assim uma linguagem apropriada para alimentar o raciocínio autonómico por parte de sistemas dinâmicos sensíveis à qualidade do serviço.

Para suportar a especificação das interações entre entidades de níveis hierárquicos diferentes são definidos dois tipos fundamentais de SLAs: SLAs horizontais e SLAs verticais. Os primeiros dizem respeito às interações entre dois pares com o mesmo tipo de serviço, enquanto que os últimos dizem respeito às interações entre pares onde existe uma relação de subordinação, mais concretamente os acordos verticais regulam o suporte que determinada entidade recebe da infraestrutura/entidade onde assenta.

O SLAng é definido através de um esquema XML. O conteúdo de um SLA elaborado através de SLAng varia conforme o serviço, e incorpora os elementos e atributos necessários para uma negociação em particular, de forma geral inclui:

- Uma descrição sumária das entidades interessadas.
- Informação contractual (datas, tempo de validade do acordo).
- Especificação técnica do QoS e das métricas associadas.

Para além das propriedades anteriormente citadas, existem 7 tipos diferentes de SLAs, segundo o SLAng. Quatro dizem respeito a SLAs verticais e três a SLAs horizontais. Não querendo detalhar os sete níveis é contudo relevante mencionar três níveis, que possivelmente serão utilizados:

- Aplicação (Vertical). Entre aplicações/serviços Web e componentes.

Listagem 2.1: Exemplo de um SLA feito em SLAng.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <SLAng xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:noNamespaceSchemaLocation="dave/TAPAS/SLAng0_5/SLAng0_5.xsd">
4    <Vertical>
5      <Hosting>
6        <Id sls_id="49258" service_id="Replication"/>
7        <Client>
8        </Client>
9        <Server>
10         <Performance response_time="2.6" peak_time_latency="4.7"
               successful_transactions="98%" processin_speed="843"/>
11       </Server>
12       <Mutual>
13         <Service_schedule start="2002-12-13" end="2003-12-13"/>
14         <Failure_clauses compensation="(100%-availability)*4.6"
               exclusion_clauses="Client caused outages"/>
15       </Mutual>
16     </Hosting>
17   </Vertical>
18 </SLAng>
```

- Hosting (Vertical). Entre containers e provedores de componentes.
- Serviço (Horizontal). Entre componentes e provedores de serviços Web

Na figura 2.1 é possível observar o exemplo de um SLA especificado através do formato SLAng. Trata-se de um SLA vertical to tipo "Hosting" entre um provedor de componentes e um provedor de aplicações. Dentro do elemento "Hosting" na linha 5 do SLA estão definidas as identificações do cliente e do servidor da linha 7-13 e da linha 14-25 respectivamente. Na secção do servidor para além da sua identificação, que é omitida para reduzir o documento apenas às métricas, estão especificadas as suas responsabilidades para com o cliente, por forma a que este consiga atingir o nível QoS desejado. É possível constatar por exemplo que no que diz respeito ao desempenho do servidor, na linha 20, este tem de garantir no máximo um tempo de resposta de 2,6 segundos e a percentagem de transações bem sucedidas tem de ser 98%. Por fim é registada a data do contracto bem como a sua duração na linha 27 e as cláusulas associadas à violação das condições acordadas na linha 28.

2.3 Monitorização

Tendo uma maneira formal de representar métricas e requisitos QoS, é agora importante perceber como podem ser recolhidos os dados relativos às métricas definidas.

Na secção 2.1.3.1 foram apresentadas duas formas de monitorização: passiva e activa. Enquanto que a primeira recorre a funções disponibilizadas por omissão nos sistemas onde os serviços são alojados, a segunda forma é mais complexa, uma vez que os dados que pretendem ser recolhidos são específicos ao funcionamento interno de determinado serviço. O facto da monitorização activa ser tão direcionada, faz com que seja necessário

adicionar lógica específica para a captura de informação e posterior partilha com um serviço externo.

No caso concreto do sistema a desenvolver optou-se por utilizar a monitorização activa uma vez que as métricas a avaliar estão muito associadas ao funcionamento interno dos serviços, funcionamento esse que para ser monitorizado requer que sejam feitas alterações directas ao código de cada serviço. Esta opção vai também de encontro ao objectivo da SPMS que pretende ter uma monitorização capaz de produzir relatórios muito mais detalhados do que aqueles que actualmente produz.

Face aos requisitos anteriores e devido à importância da monitorização, num sistema como aquele que se pretende desenvolver, tendo ainda em conta a complexidade que a produção de tal ferramenta acarretaria, optou-se por utilizar uma solução já existente. A solução designa-se Prometheus e é analisada de seguida.

O Prometheus é um software open-source, utilizado como ferramenta de monitorização e alerta de sistemas. Foi originalmente desenvolvido pela SoundCloud em 2012.

Entre as várias características que compõe o Prometheus, destacam-se as seguintes:

- Modelo de dados multidimensional, com séries temporais de dados identificadas pelo tipo de métrica e pares chave-valor, com o nome da métrica e o valor associado.
- Linguagem poderosa e flexível para a obtenção de dados.
- Recolha de métricas através de pedidos HTTP.
- Os serviços alvo são descobertos através de um mecanismo de descoberta de serviços ou por configuração estática.

Através da figura 2.3 é possível perceber o funcionamento do Prometheus que consiste na recolha de métricas de um grupo de serviços de forma directa ou indirecta, no último caso a recolha é feita através de um intermediário. A informação recolhida é armazenada localmente, onde é posteriormente agregada por forma a gerar novos dados, caso seja possível. Nesta etapa é ainda avaliada a presença de valores anómalos. Se forem detectadas anomalias o sistema envia alertas.

O Prometheus define 4 tipos de métricas: Contadores, Indicadores, Histogramas e Sumários.

Contadores - Métricas cumulativas que apenas podem subir de valor. Estas métricas são ideais para representar o número de pedidos, o número de processos activos e semelhantes.

Indicadores - Utilizados para representar métricas que tanto podem subir de valor como descer são ideais para representar a utilização de memória, temperatura e semelhantes.

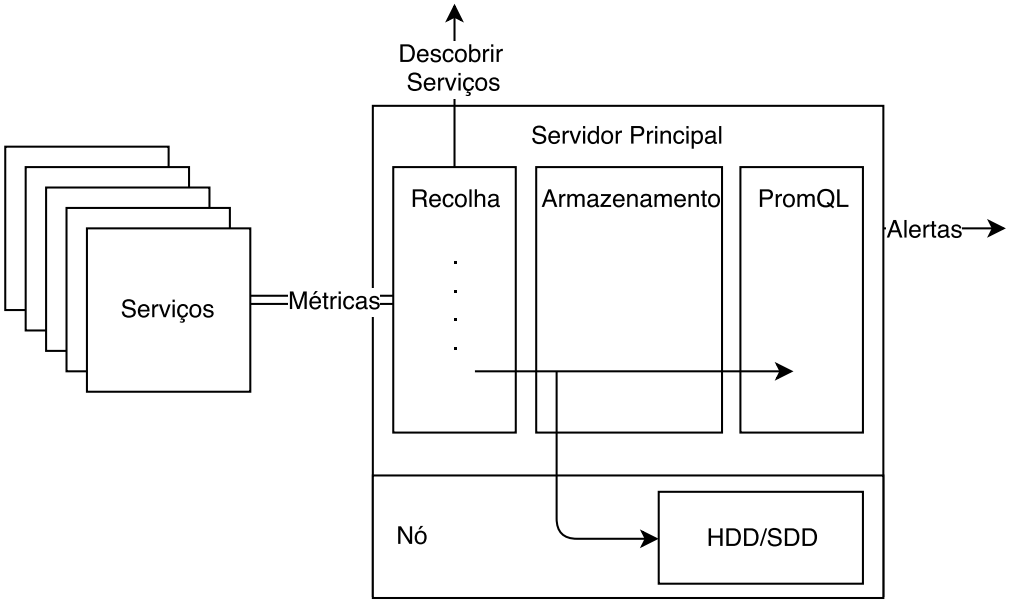

Figura 2.3: Servidor principal do Prometheus, retirado de prometheus.io[a]

[a] https://prometheus.io/docs/introduction/overview

Histogramas - Agrupam as observações que recolhem em classes de acordo com o valor que apresentam. Um histograma no Prometheus cria outras métricas compostas como a soma do valor de todos os dados recolhidos, o número de eventos observados e as classes referidas anteriormente.

Sumários - São semelhantes aos Histogramas calculando ainda quantis configuráveis ao longo de uma janela temporal.

A escolha de uma ferramenta como o Prometheus, tem que ver em primeiro lugar com a necessidade de gerir esforços no que diz respeito ao desenvolvimento dos componentes a integrar no sistema. Através da referida ferramenta é possível usufruir de uma solução altamente testada, fiável e acreditada pela indústria poupando ainda o tempo que demoraria a desenvolver um software semelhante, permitindo assim o foco noutras tarefas importantes do sistema.

O segundo ponto tem que ver inteiramente com as vantagens oferecidas por esta solução em relação às restantes. No website do Prometheus existe uma página dedicada[1] à comparação rigorosa entre as alternativas, pelo que não serão aqui explicitadas. Mas de uma forma sucinta as principais razões são as seguintes:

- Suporte avançado relativamente a métricas definidas pelo administrador.
- Linguagem utilizada para obter informação da base de dados.
- Elevada disponibilidade do servidor responsável pelo levantamento das métricas.

[1] https://prometheus.io/docs/introduction/comparison/

- Integração total com o Docker.

2.4 Avaliação do desempenho

Uma característica essencial de um sistema autonómico é a sua capacidade de optimizar-se. Mais concretamente um sistema autonómico tem de ser capaz de optimizar as suas acções de acordo com os objectivos definidos. Mas de que maneira podem então ser definidos os objectivos e que mecanismos deve o sistema utilizar para traduzir os objectivos em ações baixo nível que atinjam o estado de optimização desejado? Esta pergunta vai ser respondida nas próximas secções, começando por esta. Nesta secção é abordada a problemática associada a avaliação do nível QoS do sistema. Para atingir o objectivo proposto utilizar-se-ão funções utilitárias, abordadas seguidamente.

2.4.1 Funções utilitárias

No contexto da Computação Autonómica, as funções utilitárias [30, 33] são utilizadas para inferir o grau de utilidade dos vários estados do sistema. Tipicamente as referidas funções são associadas às metodologias que recorrem à procura heurística (ver secção 2.5.2.3).

No caso concreto do sistema a desenvolver vão existir dois tipos de funções utilitárias: de atributos e globais.

2.4.1.1 Funções Utilitárias de Atributos

As funções utilitárias de atributos representam a utilidade de disponibilizar certa funcionalidade, ou a utilidade de uma métrica QoS atingir um determinado valor.

Dada a natureza heterogénea de cada métrica é possível dividi-las em dois grandes grupos: elásticas e inelásticas. Uma métrica é considerada elástica quando a deterioração da sua utilidade é sempre marginal face a possíveis reduções no valor da métrica. Para representar a natureza destes dois tipos de métricas existem duas equações de refêrencia que podem ser utilizadas [1, 5].

Tipicamente para o caso de métricas elásticas é utilizada uma função logarítmica:

$$U_i(v) = a_i log(v + b_i) \tag{2.1}$$

Para métricas inelásticas a função de referência produz uma sigmoide:

$$U_i(v) = \frac{1}{1 + e^{a_i(ref_i - v)}} \tag{2.2}$$

As anteriores equações representam bem o impacto da degradação da utilidade quando as métricas associadas pioram. No caso da função logarítmica quando o valor da métrica

reduz, o impacto na utilidade é relativamente insignificante ao passo que a mesma variação na função sigmoide implica uma redução drástica. Os parâmetros a_i e b_i servem para ajustar as funções, o ref_i representa o valor QoS desejado.

2.4.1.2 Funções Utilitárias Globais

As funções utilitárias globais servem para combinar valores de utilidade de atributos num único valor escalar. Uma abordagem possível é a média ponderada dos vários valores.

$$U_g = \frac{\sum_{i=1}^{N} U_i \times w_i}{\sum_{i=1}^{N} w_i} \tag{2.3}$$

Onde U_g representa a utilidade global, N o número de atributos, U_i a utilidade de um atributo i e w_i o peso do atributo i. Contudo esta abordagem pode não ser indicada para todas as soluções uma vez que podem produzir resultados onde a solução ideal maximiza apenas um grupo de atributos, negligenciando os restantes. Preferencialmente seria ideal utilizar uma solução que maximizasse todos os atributos tendo em conta o seu peso. Um possível método é a média geométrica.

$$U_g = (\prod_{i=1}^{N} U_i^{w_i})^{1/\sum_{i=1}^{N} w_i} \tag{2.4}$$

Soluções híbridas com os dois métodos anteriores também são válidas.

2.4.2 Aplicação no Trabalho

As funções utilitárias são um componente essencial na adaptação dinâmica de um sistema autonómico e nesta secção foram apresentados dois tipos, as funções de atributos e as funções globais. Como o nome indica as primeiras serão utilizadas para calcular a utilidade de uma métrica isoladamente, depois de o valor de utilidade de todas as métricas de um serviço ter sido calculado utilizar-se-ão funções globais, que receberão como argumento os valores de utilidade de cada métrica, para produzir um valor geral de utilidade para cada categoria. A composição dos valores de utilidade de cada categoria produz o valor de utilidade de cada serviço face aos atributos atribuídos.

Este é o princípio genérico por trás da utilização das funções utilitárias no sistema a desenvolver. As funções aqui apresentadas não são de nenhum modo finais prevê-se que durante o desenvolvimento do sistema e devido à natureza heterogénea de cada métrica possam ser realizadas alterações.

2.5 Planeamento e execução

As Funções Utilitárias não são a única forma possível para a toma de decisões em sistemas autonómicos, existem outros mecanismos, mecanismos esses analisados nesta secção, por forma a perceber as vantagens e desvantagens de uns em relação aos outros. Outro

tópico abordado e de extrema importância para o sistema a desenvolver e os sistemas autonómicos no geral, são as metodologias responsáveis pela adaptação dinâmica dos sistemas, com base nos mecanismos apresentados. As referidas metodologias representam a lógica que guia os sistemas no seu processo de reconfiguração, optimização e resolução de problemas. No final da secção é explicado qual a metodologia utilizada e o porquê.

2.5.1 Definição de objectivos

Para que um sistema possa planear e auto adaptar-se tem de ter um conjunto de objectivos alto nível e regras para atingir os anteriores objectivos. Usualmente estes são expressos [18] através de três métodos: Event Condition Action (ECA), Políticas de Objectivo ou até mesmo através de Funções Utilitárias.

ECA - Os ECAs são regras que estipulam a execução de ações quando determinados eventos ocorrem e certas condições são verificadas. Um exemplo pode ser a seguinte situação, onde em determinado sistema quando o tempo de resposta de 95% dos servidores Web excede os 2s e existem recursos suficientes, o sistema aumenta o número de servidores activos.

Políticas objectivo - Este mecanismo é mais alto nível no sentido em que são especificados critérios que caracterizam estados desejáveis, mas o processo e trabalho de descobrir como atingir o estado, fica à responsabilidade do sistema. Por exemplo poderia-se especificar que o tempo de resposta de um servidor Web deveria ser abaixo dos 2 segundos. Através de regras e conhecimento prévio, armazenado no componente de conhecimento, o sistema seria capaz de adicionar ou remover recursos à medida que for necessário para atingir o estado desejável.

Funções Utilitárias - As funções utilitárias contrariamente às outras abordagens são capazes de definir um nível quantificável para cada estado. Uma função utilitária tem como argumento um número variável de atributos e retorna o nível de utilidade do estado definido pelos atributos passados como argumento. Um exemplo seria uma função utilitária que ao receber como input o tempo de resposta de um servidor Web e de um servidor aplicacional retornaria a utilidade da combinação entre os tempos de resposta do servidor Web e aplicacional. Desta forma quando não estão disponíveis recursos suficientes para alocar, a melhor partição de recursos disponíveis entre o servidor Web e o servidor aplicacional pode ser encontrada.

Os mecanismos referidos contudo têm algumas contrapartidas. No caso dos ECAs, quando um número de políticas são especificadas, conflitos entre as mesmas podem ocorrer e só durante a execução serem descobertos. Nos referidos casos não é claro como é que o sistema deve reagir, e um componente adicional de resolução de conflitos é necessário. É ainda importante considerar a escalabilidade de tal solução, pois não é viável definir regras para todos os possíveis estados de um sistema.

No caso das políticas de objectivo, contrariamente aos ECAs, é necessário que o elemento gestor planei um conjunto de ações para responder a determinada situação, sendo por isso mais exigentes computacionalmente. Sofrem ainda de um problema, que é o facto dos estados serem classificados como desejáveis ou indesejáveis, o que significa que quando não é possível atingir o estado desejado, o sistema não sabe qual dos estados indesejados escolher. Por fim as próprias funções utilitárias podem (dependendo do que se pretende avaliar) tornar-se extremamente difíceis de definir, pois todos os aspectos que influenciam a decisão da função necessitarem de ser quantificados de alguma forma, onde potencialmente as funções definidas na secção 2.4.1.1 não sejam adequadas.

2.5.2 Abordagens ao planeamento e execução

Existem três abordagens principais no que diz respeito à definição e posterior execução de acções, visa à adaptação do sistema. As abordagens em questão são: Teoria de Controlo, Inteligência Artificial ou a Procura Heurística.

2.5.2.1 Teoria de controlo

A teoria de controlo é uma disciplina da engenharia preocupada com o controlo de sistemas dinâmicos em continua operação. Tipicamente associado a esta temática estão os ciclos de feedback, onde os resultados obtidos são passados como argumento para o início do novo ciclo e assim continuamente.

Esta área é amplamente utilizada em diferentes sectores, tomando como exemplo o controlo de máquinas, uma situação ilustrativa é o caso da utilização de um termómetro que capta a temperatura ambiente e consoante o valor registado altera a temperatura definida no ar-condicionado.

Em termos práticos a teoria de controlo funciona bem quando existe uma propriedade passível de ser controlada que possa ser continuamente ajustada. Um exemplo, no caso concreto de um sistema autonómico, seria por exemplo o controlo do número médio de pedidos a um serviço para permitir quando necessário, disponibilizar mais recursos. No entanto existem fragilidades neste método quando utilizado em sistemas de dimensão considerável. Exemplos dessas fragilidades são as interações entre as diferentes regras de controlo e os possíveis conflitos e riscos não detetáveis, o facto de se tratar de um método muito específico à implementação, pondo assim em causa a evolução e a optimização do sistema. A implementação do método em análise é levada acabo por ECAs, definidas anteriormente.

2.5.2.2 Inteligência Artificial

Muitas das tecnologias utilizadas na Inteligência Artificial podem ser aplicadas em sistemas autonómicos. Algumas das técnicas mais relevantes são apresentadas de seguida.

Redes Bayesianas, utilizadas em Machine Learning, possibilitam a computação de interações complexas através de condicionantes probabilísticas. Utilizando as referidas

redes as observações empíricas do sistema e ambiente são analisadas e convertidas em conhecimento. O conhecimento é posteriormente utilizado pelos componentes de controlo para suportar o planeamento e a toma de decisões [26].

Outra área relevante é designada Fuzzy Logic (FL), que consiste num método de raciocínio parecido ao humano. A abordagem FL imita o processo de toma de decisões humanas que envolve todas as possibilidades intermédias entre os valores sim e não. Concretamente no caso na Computação Autonómica pode ser utilizado para aprender relações entre parâmetros e métricas nos sistemas autonómicos [35].

A última metodologia considerada é designada, Reinforcement Learning, e é inspirada pela psicologia comportamental. Este método preocupa-se em perceber como é que os agentes podem adaptar o seu comportamento dentro do ambiente onde estão inseridos, por forma a maximizar o seu desempenho. Um simples incentivo é o necessário para um componente aperfeiçoar o seu comportamento, esta estratégia é conhecida como Reinforcement Signal. Um exemplo concreto seria a situação onde gestores autonómicos vão experimentado e aprendendo as consequências de várias ações. Uma possível desvantagem deste método, é o eventual tempo que os agentes demoram para atingirem comportamentos relativamente optimizados.

2.5.2.3 Procura Heurística

Muitos modelos autonómicos [10] são baseados em soluções que recorrem à procura de configurações ideais no espaço de configurações do modelo do sistema. Os referidos modelos oferecem a capacidade potencial de prever o desempenho de qualquer configuração do sistema.

Para atingir esse fim são utilizados algoritmos de procura responsáveis por explorar o espaço de configuração do sistema com o intuito de encontrar a configuração mais indicada, que na maioria dos casos é aquela que maximiza a utilidade do sistema (ver secção 2.4). Os algoritmos de procura utilizados vão desde simples algoritmos de força bruta que vasculham todo o espaço de configuração (potencialmente proibitivo), a heurísticas complexas que reduzem o espaço de procura. Recentemente o método em análise tem-se revelado uma escolha popular na alocação de recursos [5, 6, 36].

2.5.3 Aplicação no Trabalho

Nesta secção foram discutidos os principais métodos de adaptação utilizados na Computação Autonómica, foram apresentados os pontos fortes e os pontos fracos de cada um e para cada método foram ainda apresentados exemplos concretos da sua utilização.

Relativamente ao sistema a desenvolver optou-se por utilizar a Procura Heurística recorrendo a Funções Utilitárias, uma vez que a utilização de métodos assentes na Teoria de Controlo seriam rapidamente ultrapassados pelo aumento do número de serviços do sistema, não sendo por isso uma solução escalável, factor preponderante devido ao rápido crescimento da SPMS.

A solução baseada na Inteligência Artificial não será utilizada maioritariamente por duas razões, a primeira tem que ver com o principal problema a resolver neste projecto que é a alocação de recursos mantendo o QoS e minimizando os recursos necessários, os métodos baseados na Inteligência Artificial são na sua maioria direcionados para o ajuste inteligente de parâmetros internos com o objectivo de dotar o sistema de inteligência por forma a que este consiga melhorar as suas configurações internas e ser completamente autonómico. Esta problemática não é de todo o objectivo principal (embora possa ser algo a acrescentar no futuro), e as abordagens assentes na IA são muito focadas nesse aspecto. O segundo factor tem que ver com a relação resultados/esforço, para o problema a resolver a procura heurística proporciona óptimos resultados com um esforço menor. Assim sendo a Procura Heurística foi a abordagem escolhida porque permite não só a obtenção de bons resultados para a resolução do problema e é escalável.

2.6 Problema de Optimização

Nas duas secções anteriores foram apresentados os principais mecanismos utilizados na adaptação de um sistema autonómico para a optimização do seu estado relativamente aos objectivos estipulados. Um componente essencial para traduzir a utilidade de um estado são as funções utilitárias. Dada uma função utilitária o sistema ou componente deve utilizar uma técnica de optimização conjuntamente com o modelo do sistema, para determinar o melhor estado possível.

Tipicamente os meios para atingir determinado estado consistem no ajuste de parâmetros do sistema ou na alocação de recursos. Sendo que as condicionantes a que determinado sistema está sujeito variam e que falhas podem ocorrer, o processo de optimização é recorrente.

Foram analisados vários métodos de optimização no contexto da Computação Autonómica e da Computação em Rede [5, 6, 15, 32, 36]. Muitas das soluções apresentadas modelam o sistema como uma composição de tarefas atómicas que compõe uma tarefa maior, em que o objectivo é a distribuição das tarefas atómicas por vários serviços por forma a optimizar os custos e o desempenho do sistema. Esta modelação não vai de encontro ao caso específico do projecto a desenvolver onde o que é pretendido é optimizar a qualidade de serviço minimizando ao mesmo tempo o número de recursos atribuídos a cada serviço. Contudo a modelação proposta em [32] assemelha-se muito ao caso concreto do sistema a desenvolver e por isso mesmo será utilizada extensivamente. Por outro lado a abordagem proposta em [6] embora direcionada para o ambiente de Computação na Rede, oferece uma boa perspectiva na utilização de heurísticas variadas e será também importante na definição desse componente.

De seguida é então descrita a modelação do sistema a desenvolver.

2.6.1 Modelação do Sistema

Para que seja possível optimizar o sistema é necessário modelá-lo. O sistema a desenvolver é composto por serviços, serviços esses que correm dentro de contentores Docker que por sua vez estão alojados em máquinas virtuais disponibilizadas pelo provedor de recursos (neste caso o Azure). Como referido ao longo do documento pretende-se optimizar a qualidade de serviço minimizando ao mesmo tempo o número de recursos necessários.

No que diz respeito aos recursos a granularidade de controlo será ao nível das máquinas virtuais isto é, não será possível especificar precisamente os recursos a atribuir a determinado serviço, em vez disso existirão tipos de máquinas virtuais, cada tipo com diferentes recursos. Esta decisão foi tomada não só para simplificar o problema de optimização mas também porque a utilização de máquinas virtuais como controlo de recursos é algo comum entre os provedores de recursos, característica que potencia a interoperabilidade entre provedores. Os contentores e os serviços serão considerados como uma só unidade dado que os primeiros têm como única função garantir um ambiente de execução para os serviços.

Assim sendo o sistema é composto por um conjunto de serviços $S = (s_1, s_2, ..., s_i, ..., s_m)$ e por um conjunto de classes de máquinas virtuais $V = (v_1, v_2, ..., v_k, ..., v_c)$ onde $v_k = (v_k^{cpu}, v_k^{ram})$ especifica a capacidade do CPU e a memória da máquina virtual, expressa em MHz e megabytes respectivamente.

Cada serviço terá associado um módulo responsável por medir o nível de serviço obtido com determinados recursos em relação à carga da aplicação. Este módulo tem associado duas funções utilitárias, uma mapeia o nível de serviço para um valor de utilidade e a segunda mapeia os recursos para um valor de utilidade.

A função utilitária de recursos u_i do serviço s_i é definida como $u_i = f_i(N_i)$ onde N_i é o vector de máquinas virtuais atribuídas a s_i. $N_i = (n_{i1}, n_{i2}, ..., n_{ik}, ..., n_{ic})$ em que n_{ik} é o número de máquinas virtuais da classe v_k atribuídas ao serviço s_i.

Por forma a restringir o espaço de soluções possíveis são definidos limites máximos no número de máquinas virtuais por cada classe ($N_i^{max} = n_{i1}^{max}, n_{i2}^{max}, ..., n_{ik}^{max}, ..., n_{im}^{max}$) e o número total de máquinas virtuais (T_i^{max}) para cada serviço. As restrições podem ser representadas da seguinte forma:

$$n_{ik} \leq n_{ik}^{max} \quad 1 \leq i \leq m \quad \text{and} \quad 1 \leq k \leq c \tag{2.5}$$

$$\sum_{k=1}^{c} n_{ik} \leq T_i^{max} \quad 1 \leq i \leq m \tag{2.6}$$

A modelação do sistema está concluída de seguida é endereçado o problema de optimização.

2.6.2 Optimização do Sistema

O objectivo da optimização já foi enunciado várias vezes o que não foi dito é que o objectivo traduz-se na procura de vectores sub-óptimos N_i para cada serviço s_i, maximizando ao mesmo tempo um valor de utilidade global U_{global}.

Em casos onde os recursos sejam assegurados localmente através de máquinas físicas limitadas, para além das restrições definidas na subsecção anterior é importante definir outras nomeadamente:

$$\sum_{i=1}^{m}\sum_{k=1}^{c} n_{ik}.s_k^{cpu} \leq \sum_{j=1}^{q} C_j^{cpu} \tag{2.7}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{c} n_{ik}.s_k^{ram} \leq \sum_{j=1}^{q} C_j^{ram} \tag{2.8}$$

Por fim os vectores N_i necessitam de maximizar um função de utilidade global expressa como a soma pesada do valor de utilidade das funções utilitárias das aplicações e um custo de operação:

$$U_{global} = maximizar \sum_{i=1}^{m} (\alpha_i \times u_i - e.custo(N_i)) \tag{2.9}$$

onde $0 < \alpha_i < i$ e $\sum_{i=1}^{m} \alpha_i = 1$. e é um coeficiente que permite ao administrador controlar a importância do custo do sistema em relação ao desempenho. A função $custo(N_i)$ é sobre o vector de alocação de máquinas virtuais e tem de ter a mesma escala que as funções de utilidade do serviço. O resultado da optimização é um conjunto de vectores N_i que satisfazem as restrições apresentadas nas equações 2.5 e 2.6 (e as restrições 2.7 e 2.8 em caso de soluções locais onde existem recursos finitos) e que maximizam a função U_{global}. Através da comparação dos vectores de alocação produzidos com os calculados anteriormente é possível perceber quais as máquinas virtuais que devem ser destruídas, criadas ou redimensionadas.

2.7 Conclusões

As anteriores secções estão ordenadas com base nos componentes essenciais do modelo MAPE-K discutido na secção 2.1.3.1. Começou-se por fazer o levantamento das métricas relevantes para a definição de objectivos alto nível, essenciais para a adaptação dinâmica do sistema. Por forma a existir um formato claro na definição e interpretação destes objectivos, são utilizados SLAs, contractos onde estão estipuladas as métricas a analisar em cada serviço e os valores aceitáveis para cada uma dessas métricas. Existem variadas linguagens para definir SLAs, foram apresentadas três dessas linguagens mas a escolhida foi o SLAng pelas razões referidas na secção 2.2.2.

Devido à especificidade das métricas a avaliar em cada serviço o sistema realizará uma monitorização activa. Os valores obtidos durante a monitorização são depois analisados para que o sistema se consiga adaptar. Um componente essencial da análise são as funções utilitárias que permitem a tradução do valor das métricas para valores de utilidade, utilizados posteriormente para avaliar o estado global do sistema face à sua configuração. Desta análise percebe-se se os objectivos definidos previamente estão a ser cumpridos. Se não estiverem são necessários métodos para a optimização da configuração do sistema. A Procura Heurística é um desses métodos e é o que será utilizado. Para que se possa recorrer ao referido método é necessário modelar o sistema por forma a mapear a utilidade global das possíveis configurações do sistema e com isso perceber qual a melhor solução.

Os dois parágrafos anteriores resumem as etapas e factores a considerar no desenvolvimento de um sistema autonómico. É com base nessas etapas que a solução proposta no capítulo 3 foi desenhada.

Capítulo 3

Sistema de Monitorização Autonómico de Contentores

No presente capítulo é apresentado o sistema desenvolvido, começando por uma descrição geral na secção 3.1 da arquitectura do sistema e dos componentes essenciais que o compõem. A secção 3.2 aborda o elemento responsável pela gestão de SLAs. Na secção 3.3 é analisado o *Pod* de Serviço, componente responsável pela monitorização e gestão de um serviço. Na secção seguinte 3.4 é analisado o *Pod* Autonómico o elemento responsável pela coordenação da adaptação do sistema. Por fim na secção 3.5 são feitas as considerações finais sobre o sistema desenvolvido.

3.1 Descrição Geral

O objetivo desta dissertação é conceber e implementar um Sistema Autonómico capaz de adaptar dinamicamente os recursos alocados a um conjunto de serviços, por forma a optimizar a qualidade de serviço global do sistema e minimizar os custos associados, melhorando deste modo a experiência dos utilizadores finais que interagem indirectamente com os referidos serviços através de aplicações móveis.

Os serviços são os elementos monitorizados, dos quais se infere um valor de utilidade através da comparação do valor das métricas expostas pelos mesmos, com os valores limite acordados nos SLAs de cada serviço. Um SLA é um documento definido através do formato SLAng (ver secção 2.2.2.1) que contém informação sobre o acordo realizado entre o serviço e determinada aplicação, sendo a informação mais importante os valores máximos definidos para cada métrica do serviço a que o SLA se refere. Para cada serviço a ser monitorizado é definido um SLA que é adicionado à Base de Dados de SLAs.

No que diz respeito às métricas expostas, para o sistema desenvolvido foram utilizadas as seguintes métricas: **taxa de transações falhadas**, **indisponibilidade**, **latência** e **taxa**

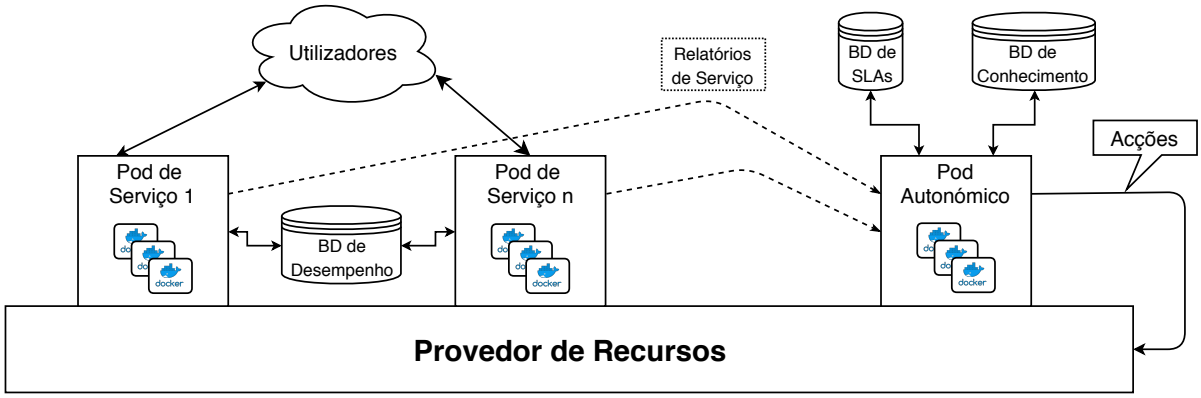

Figura 3.1: Arquitectura do sistema desenvolvido

média de pedidos por instância. Uma explicação mais detalhada sobre a escolha das anteriores métricas e da sua necessidade é dada na secção 3.3.2.

Todos os serviços são executados dentro de contentores *Docker*, podendo existir uma ou mais instâncias de um serviço. Cada uma das instâncias correrá num contentor, isolada das restantes. Cada contentor terá um número de CPUs virtuais (vCPUs) e de GigaBytes (GB) de memória. Ao longo do documento será utilizada a expressão "configuração do serviço", que se refere precisamente ao número de vCPUs e GB de memória alocados a determinado serviço.

O sistema desenvolvido consegue ajustar o seu valor de utilidade, actualizando as configurações dos vários serviços, ou seja, alocando mais ou menos recursos a cada serviço conforme as suas necessidades. As referidas reconfigurações são consequência de uma análise periódica que averigua a utilidade do sistema. Se o valor de utilidade for inferior ao valor aceitável é realizada uma reconfiguração de todos os serviços, entrando o sistema num novo estado, considerado ideal para as exigências correntes do sistema. O novo estado é definido por forma a maximizar a utilidade dos serviços, minimizando ao mesmo tempo o custo dos mesmos. O espaço de configurações possíveis é calculado tendo em conta o desempenho de configurações anteriores. Esta informação é recolhida da Base de dados de Desempenho.

Detalhamos agora a arquitetura inicialmente proposta na figura 1.2. Como é possível constatar pela figura 3.1 existem dois elementos essenciais o ***Pod* de Serviço** e o ***Pod* Autonómico**. Tendo por base o modelo MAPE-K (consultar secção 2.1.3.1) o *Pod* de Serviço é o Elemento Gerido enquanto que o *Pod* Autonómico é o Gestor Autonómico, responsável pela gestão e modificação do primeiro. Concretamente o *Pod* de Serviço contém todos os componentes necessários para o controlo e monitorização de um serviço enquanto que o *Pod* Autonómico contém todos os componentes essenciais para a gestão e modificação dos elementos geridos tendo em conta o fluxo do modelo MAPE-K.

Falta apenas detalhar a função e utilidade das três Bases de Dados que fazem parte do sistema:

Base de Dados de SLAs armazena todos os SLAs dos serviços activos, dos serviços por validar e dos serviços inactivos. É através desta base de dados que se torna possível controlar os serviços a monitorizar.

Base de Dados de Conhecimento não tem grande relevância para o sistema actual, a sua inclusão tem como objectivo eventuais melhorias futuras ao sistema, por forma a atingir o último nível de autonomia (ver secção 2.1.2). Contudo a sua função consiste no armazenamento de informação importante, sobre a execução de cada componente do *Pod* Autonómico, por forma a melhorar a gestão interna do *Pod*.

Base de Dados de Desempenho responsável por armazenar todos os dados relativos ao desempenho dos vários serviços e possibilitar que o *Pod* Autonómico consiga calcular a configuração ideal, no caso do sistema necessitar de ser alterado.

Através da figura 3.1 é possível perceber o fluxo principal e a arquitectura do sistema desenvolvido. O sistema é composto por *n Pod*s de Serviço, sendo *n* o número de serviços existentes. O *Pod* Autonómico recolhe relatórios de serviço dos vários *Pod*s de Serviço e com essa informação analisa o estado do sistema. Se a qualidade de serviço global não estiver de acordo com o estabelecido nos SLAs, são executadas acções através da API do provedor de recursos por forma a alterar a configuração dos diferentes serviços e com isso melhorar a utilidade do sistema.

3.1.1 Tecnologias Escolhidas

Todos os componentes dos Pods expõem uma API REST onde estão definidas as operações suportadas. Optou-se pela utilização do protocolo REST, por ser o protocolo mais utilizado internamente pela equipa de aplicações da SPMS no que diz respeito à comunicação de serviços web, devido às vantagens intrínsecas do protocolo, nomeadamente: o desempenho, o tamanho das mensagens, e pelo suporte nativo do Nodejs ao referido protocolo.

Os *Pod*s são um conceito comum na comunidade Docker, no entanto cada provedor de recursos, que suporte *Pod*s, tem a sua forma concreta de implementar este mecanismo. No caso do Azure estas entidades podem ser criadas através do serviço Azure Container Instances (ACI)[1] em que em vez de *Pod* o grupo de contentores designa-se Grupo.

A figura 3.2 demonstra a arquitectura apresentada anteriormente na figura 3.1, mas especificando agora a camada adicional acrescentada pelo mecanismo de suporte a *Pod*s do Azure, designado Instâncias de Contentores.

Embora as especificidades de um Grupo sejam semelhantes às de um *Pod*, o primeiro tem algumas restrições que são importantes referir, pois as mesmas tiveram um impacto significativo no desenho e implementação da solução.

[1] Azure Container Instances: https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview

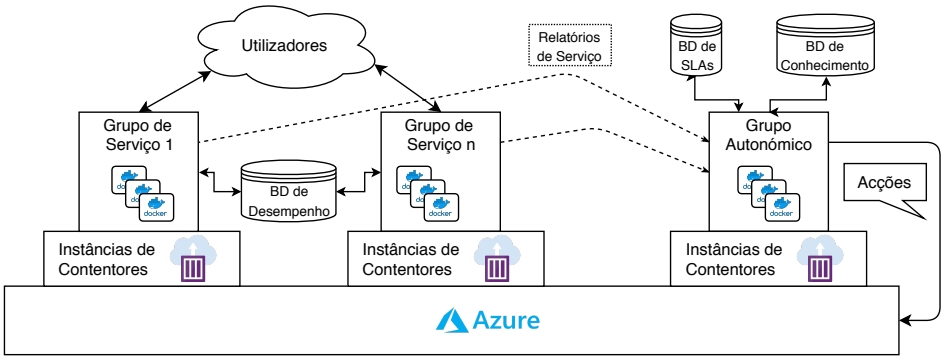

Figura 3.2: Arquitectura do sistema desenvolvido em Azure

Todos os contentores de um Grupo partilham o mesmo espaço de portas, logo não é possível existirem contentores diferentes com as mesmas portas expostas. Todos os Grupos têm um número máximo de recursos disponíveis, mais concretamente no máximo podem ser disponibilizados 4 vCPU (virtual CPUs) e 13 GB de memória para cada Grupo. Assim sendo, todos os contentores alocados a um Grupo terão que respeitar os recursos máximos do mesmo. Um exemplo de uma configuração válida é demonstrada na figura 3.3.

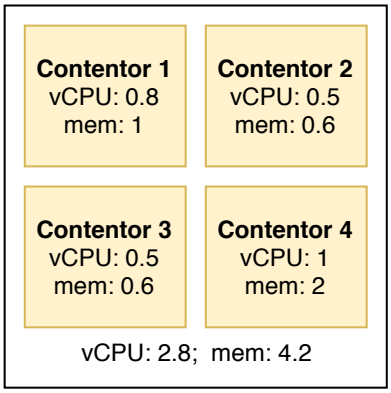

Figura 3.3: Exemplo de um Grupo do ACI

Através do serviço ACI é possível utilizar toda a conveniência das imagens Docker, possibilitando a construção da infraestructura que suporta os serviços em torno do conceito "Contentor," o que reduz drasticamente a complexidade da implementação e permite uma solução mais modular, pois o sistema na prática funciona apenas à base de imagens Docker que dão origem a contentores e que por isso mesmo são praticamente independentes do provedor de recursos.

3.2 Gestão e Interpretação de SLAs

A primeira etapa para que possa ocorrer a monitorização e adaptação dinâmica do sistema, consiste no registo dos SLAs de cada serviço a monitorizar. O registo é realizado através dos endpoints expostos pelo Gestor de SLAs.

O fluxo principal de submissão de SLAs está ilustrado na figura 3.4. O primeiro passo compreende a submissão de um SLA em formato SLAng através do endpoint demonstrado na figura. Se o SLA submetido estiver de acordo com o formato estipulado e passar as validações realizadas pelo Gestor de SLAs, é armazenado na Base de Dados de SLAs com o estado válido, caso contrário é descartado e uma mensagem a informar que o SLA submetido é inválido é devolvida.

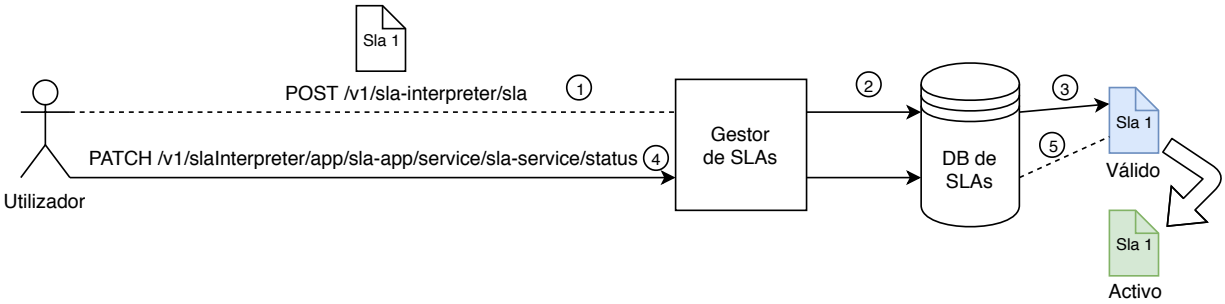

Figura 3.4: Fluxo do registo e activação de SLAs

O estado válido indica que o documento submetido é valido e está pronto a ser activado. A activação de um SLA tem como objectivo despoletar a monitorização e análise do serviço a que esse SLA diz respeito. Se o utilizador responsável pela a activação de SLAs entender que o SLA deve transitar para o estado activo, solicita essa alteração e o SLA a partir desse momento entra em vigor, significando que o serviço associado a esse mesmo SLA é agora monitorizado e o seu estado é tido em conta no processo de adaptação autonómico.

É ainda possível cancelar um SLA. Esta operação tem como objectivo cancelar a monitorização de um serviço sujeito ao SLA a ser cancelado.

Para além do controlo e interpretação de SLAs o Gestor tem ainda um papel importante para com os restantes elementos do sistema, pois disponibiliza endpoints que permitem a consulta de SLAs segundo certos critérios, como o nome do SLA, a aplicação a que estão associados, o nome do serviço a que estão associados, o estado do SLA.

3.3 Pod de Serviço

Nesta secção será apresentado o *Pod* de Serviço, começando primeiramente pela sua arquitectura, abordando os elementos principais e quais as suas funções. De seguida é descrita a monitorização do serviço, como ocorre e no que consiste. No final da secção são explicados os detalhes da implementação e as tecnologias utilizadas.

3.3.1 Arquitectura Interna

A cada serviço a ser monitorizado é atribuído um *Pod*, que é composto pelas instâncias do serviço e por mais três elementos: o **Proxy**, o **Gestor** e o **Monitorizador**. A figura 3.5 representa a arquitectura do *Pod* em análise.

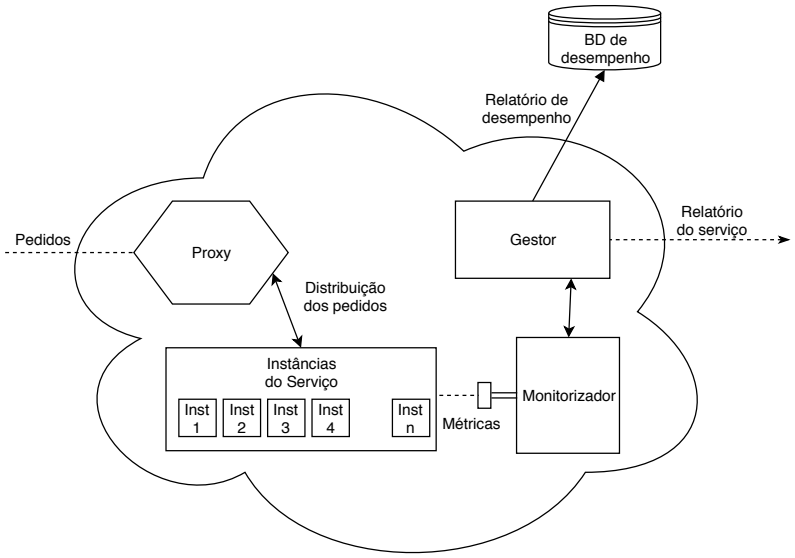

Figura 3.5: Arquitectura do Pod de Serviço

O **Serviço** é o elemento que compreende o serviço web responsável por oferecer determinada funcionalidade ao utilizador final. É representado por um conjunto de instâncias do serviço, a única restrição é que a soma dos recursos alocados a cada instância não ultrapasse o máximo de recursos disponíveis para o *Pod* (tendo em conta os recursos já alocados para o Monitorizador, o Proxy e o Gestor). Todas as instâncias são independentes umas das outras, tendo cada uma a mesma função. O número de instâncias é variável consoante a configuração proposta para o serviço e a sua função. O objectivo de ter várias instâncias é permitir que a carga não se concentre num só ponto, possibilitando a resposta a pedidos em simultâneo.

O **Proxy** tem como função distribuir os pedidos efectuados ao serviço, pelas várias instâncias do mesmo. A sua importância prende-se com o controlo dos pedidos e a distribuição dos mesmos por cada instância, por forma a distribuir a carga equitativamente.

O terceiro elemento é o **Monitorizador** que tem como função, a recolha de todas as métricas expostas pelas instâncias do serviço.

O quarto e último elemento é deveras importante para a adaptação do *Pod* de Serviço, o **Gestor**, tem duas funções essenciais. A primeira, consiste na recolha periódica do estado do *Pod* e do seu armazenamento na Base de Dados de Desempenho. A segunda função, consiste em reportar ao *Pod* Autonómico o estado actual do *Pod* de Serviço a que está associado, para que o primeiro consiga proceder à análise do sistema. O Gestor para além de guardar dados essenciais sobre o desempenho do serviço pertencente ao *Pod* onde

está inserido, também remove grande parte da complexidade que ficaria agregada ao *Pod* Autonómico, uma vez que assume o processo de agregação de dados e de elaboração do relatório de cada serviço.

3.3.2 Monitorização

A monitorização em qualquer sistema autonómico é um componente essencial, pois é através deste processo que são recolhidos os dados que servirão de guia à adaptação do sistema.

A monitorização compreende a recolha das métricas expostas pelas instâncias do serviço do *Pod*. Cada *Pod* de Serviço tem um Servidor Prometheus responsável por monitorizar as instâncias do serviço, do *Pod* onde está inserido.

No caso concreto do sistema desenvolvido optou-se por descentralizar o componente de monitorização por todos os *Pods* de Serviço, querendo isto dizer que todos os *Pods* de Serviço contêm um componente de monitorização, responsável por recolher as métricas expostas pelas instâncias do serviço do *Pod* a que pertencem. A razão para esta decisão tem que ver com a escalabilidade da solução. Tendo um só elemento de monitorização, significaria ter um único ponto responsável pela monitorização de todos os serviços e um único ponto de falha, consequentemente. Se o peso de monitorizar *n* serviços implicasse a falha ou a degradação do serviço de monitorização, toda a solução ficaria comprometida, pois nenhum dado conseguiria ser recolhido e a solução tornar-se-ia inútil. Desta forma a carga associada à monitorização e ao processamento das métricas dos vários serviços é distribuída localmente pelos mesmos. Desta forma o *Pod* Autonómico tem um componente de Monitorização menos sobrecarregado que tem como função recolher os relatórios com a informação já processada dos vários Pods de Serviço.

Tal como foi enunciado na secção 2.3 por forma a obter uma monitorização fina e mais personalizada optou-se por realizar uma monitorização activa, querendo isto dizer que é necessário numa primeira fase definir quais as métricas que devem ser expostas em cada serviço, e numa segunda fase programar o modo como estas devem ser produzidas.

A tabela 3.1 mostra as métricas atómicas, de acordo com os tipos permitidos pelo Prometheus (ver secção 2.3), utilizadas para controlar os serviços. Como é possível constatar existem seis métricas do tipo contador e uma do tipo Histograma, a sua função encontra-se descrita na tabela.

Os contadores definidos anteriormente são incrementados sempre que ocorrer um evento relevante. No caso da métrica *methods* toda a vez que um método é invocado a métrica é incrementada, por outro lado no caso da métrica *unsuccessMethods* apenas quando um método lança uma exceção é que a métrica é incrementada. A métrica *requestDuration* tem um funcionamento ligeiramente diferente pois é um histograma, esta métrica é iniciada quando um pedido é recebido e finaliza quando o mesmo termina, registando deste modo a duração da execução do pedido.

As métricas previamente mencionadas possibilitam a monitorização detalhada de

Identificador	Tipo	Função
methods	Contador	Conta o número de métodos invocados
unsuccessMethods	Contador	Conta o número de métodos que não concluíram com sucesso
successMethods	Contador	Conta o número de métodos que concluíram com sucesso
requests	Contador	Conta o número de vezes que os endpoints expostos foram invocados
unsuccessRequests	Contador	Conta o número de pedidos que não concluíram com sucesso
successRequests	Contador	Conta o número de pedidos que concluíram com sucesso
requestDuration	Histograma	Mede a duração de um pedido

Tabela 3.1: Métricas atómicas utilizadas.

particularidades acerca dos serviços. Contudo o principal motivo associado à definição das métricas anteriormente listadas, é a possibilidade de as combinar por forma a se poderem computar métricas mais complexas, como aquelas que estão definidas na tabela 2.1, sendo essas as métricas utilizadas para a adaptação do sistema.

Das métricas referidas na tabela 2.1 foram implementadas as seguintes: **Taxa de transações falhadas** (ttf), **Indisponibilidade** (ind) e **Latência** (lt). Foi ainda implementada uma métrica adicional, designada **Taxa média de pedidos por instância** ($tmpi$). Estas métricas foram escolhidas por possibilitarem uma boa avaliação e adaptação do sistema, tendo em conta valores limite definidos em SLAs. De uma forma mais pormenorizada, a ttf permite averiguar se há um aumento do ritmo de operações falhadas num serviço, o que permite inferir o estado da "saúde" do mesmo.

A ind diz respeito à indisponibilidade, optou-se pela indisponibilidade em detrimento da disponibilidade por ser mais prático analisar tal valor em comparação com o estipulado no SLA. A métrica é referente à percentagem de tempo que determinado serviço esteve indisponível. Como é apresentado na equação 3.2 o cálculo da métrica em questão não tem em conta o estado das instâncias, esta decisão teve que ver com o facto de raramente se dar o caso de todas as instâncias que compõem o serviço estarem inactivas, dando assim a ideia que a disponibilidade do serviço é 100% o que é erróneo assumir, pois é frequente que um serviço devido à sobrecarga de pedidos mesmo tendo instâncias activas, não consiga processar todos os pedidos que recebe. Devido a esta razão calcula-se a indisponibilidade, tendo em conta a percentagem do ritmo de transações falhadas ao longo de um intervalo de tempo, que no caso, é 10 minutos.

A lt possibilita monitorizar o tempo que determinado serviço demora a processar os pedidos que recebe. Esta informação não só é importante para perceber o estado do serviço, como para garantir que o tempo que o utilizador final tem de aguardar para ter o seu pedido concluído não ultrapassa um valor limite. A métrica ttf é importante pois

permite que um serviço nunca tenha uma média de pedidos por segundo muito elevada, beneficiando as configurações com mais instâncias.

As restantes métricas da tabela 2.1 não foram implementadas pelos seguintes motivos: a taxa de processamento de pedidos é uma boa métrica visual, mas no que diz respeito ao controlo do sistema não é relevante pois um valor baixo não é necessariamente um mau indicador, pode apenas significar que poucos pedidos estão a ser realizados, e por isso torna-se difícil definir um valor limite no SLA; a taxa de transações concluídas é redundante uma vez que já é analisada a taxa de transações falhadas; o tempo médio entre falhas não foi implementado, devido à dificuldade de calcular o seu valor, tendo em conta as várias instâncias por serviço; a taxa de violação de restrições é mais interessante num contexto de penalização monetária aquando da violação dos SLAs; o tempo médio de adaptação não foi implementado por não ser utilizado no cálculo da reconfiguração óptima, uma vez que se relaciona directamente com os componentes do *Pod* Autonómico. No futuro se for realizada a adaptação do *Pod* Autonómico será fundamental incluir esta métrica.

O valor de cada uma das métricas anteriormente referidas (ttf, ind, lt, $tmpi$) é calculado seguindo as fórmulas 3.1 a 3.4, respectivamente.

$$ttf = \sum_{i=1}^{n} taxa\_mf_i[10m] \tag{3.1}$$

$$ind = \frac{\sum_{i=1}^{n} taxa\_pf_i[10m]}{\sum_{i=1}^{n} taxa\_pt_i[10m]} \tag{3.2}$$

$$lt = \frac{\sum_{i=1}^{n} taxa\_dp_i[10m]}{\sum_{i=1}^{n} taxa\_pt_i[10m]} \tag{3.3}$$

$$tmpi = \frac{\sum_{i=1}^{n} taxa\_pt_i[10m]}{\sum_{i=1}^{n} percent\_ta_i[10m]} \tag{3.4}$$

onde a variável n se refere ao número total de instâncias de um serviço, a $taxa\_mf$ representa o ritmo de métodos falhados, a $taxa\_pf$ representa o ritmo de pedidos falhados, a $taxa\_pt$ representa o ritmo de pedidos recebidos, a $taxa\_dp$ representa a duração média dos pedidos em segundos e a $percent\_ta$ representa a percentagem de tempo que uma instância esteve activa. Todas as variáveis anteriores dizem respeito a um período de 10 minutos, altura em que as mesmas reiniciam e dão início a um novo período de 10 minutos, por forma a evitar que o bom desempenho durante um período não ofusque um eventual desempenho deficiente.

É na configuração das métricas anteriores que a monitorização do *Pod* se baseia, onde periodicamente as mesmas são recolhidas pelo Servidor Prometheus e guardadas internamente. O processo de Monitorização compreende ainda a recolha e processamento efectuado pelo Gestor, das métricas armazenadas no servidor Prometheus, que resulta num relatório que é posteriormente guardado na Base de dados de Desempenho, para que o desempenho da configuração, tendo em conta a carga vigente, possa ser considerado

mais tarde no processo de adaptação dinâmico. O fluxo de monitorização termina com a recolha por parte do Monitorizador do *Pod* Autonómico, dos relatórios produzidos pelo Gestor de cada *Pod* de Serviço, que agrega a informação num só relatório, que representa o estado do sistema.

3.3.3 Detalhes de Implementação

Ao longo da implementação dos conceitos e componentes abordados anteriormente, muitas escolhas e detalhes tiveram de ser considerados. Esta subsecção analisa os mais importantes, começando pelo componente Proxy, explicando as suas características e importância. Posteriormente é analisado o servidor Prometheus, utilizado como elemento Monitorizador. A subsecção termina com a explicação das tecnologias escolhidas.

3.3.3.1 Proxy

Outro motivo pelo qual foi necessário incluir um Proxy no *Pod* de Serviço, tem que ver com o facto de um Grupo no ACI não poder expor mais do que 5 portas para o exterior. Se existirem mais do que 5 instâncias de um serviço deixa de ser possível aceder directamente a essas instâncias. Devido a esta restrição apenas as portas 8080, 80 e 403 são expostas. A primeira permite aceder a uma página com informação sobre as instâncias activas, as portas 80 e 403 estão ligadas ao Proxy que consoante a validade do pedido redireciona-o para a porta interna da instância do serviço adequada.

Para reduzir a complexidade da implementação e utilizar um Proxy com garantias utilizou-se uma solução já existente, designada Traefik[2]. Traefik é um proxy reverso implementado em Golang que é facilmente configurável e customizável, outro benefício deste proxy em relação a outras soluções como o Nginx[3], é a sua integração com os contentores Docker.

Um dos requisitos para a viabilidade do sistema desenvolvido é a capacidade de adaptação dinâmica do mesmo. Este requisito é inimigo de ficheiros de configuração estáticos, esta necessidade foi o principal motivador da escolha do Proxy Traefik, que suporta reconfiguração dinâmica. Não aprofundando em demasia este tópico, os contentores Docker são executados sobre um daemon Docker que está instalado num computador ou numa máquina virtual. O daemon Docker utiliza uma socket UNIX designada docker.sock como ponto de entrada para a API Docker. Todas as operações sobre contentores e outras entidades do Docker passam por esta socket. O Traefik escuta esta mesma porta para detectar automaticamente os contentores criados.

No entanto, como os contentores de um *Pod* estão isolados e é impossível ter acesso a esta socket, esta característica importante do Traefik é impossível de utilizar. Assim sendo, devido a este condicionamento, estipulou-se que no máximo poderão existir seis instâncias de um serviço, com as portas pré definidas num ficheiro de configuração. Esta

[2]Traefik: https://traefik.io/
[3]Nginx: https://www.nginx.com/

Listagem 3.1: Ficheiro de configuração de um Servidor Prometheus

```
global:
  scrape_interval:     5s
  evaluation_interval:   5s

scrape_configs:
   - job_name: 'spms-service'
     scrape_interval: 5s
     file_sd_configs:
      - files:
        - service-targets.json
```

abordagem embora não seja ideal, uma vez que limita o número de instâncias de um serviço num *Pod*, é atenuada pelo facto de periodicamente o Proxy averiguar a saúde das instâncias, não podendo acontecer o caso de redirecionamento de pedidos para instâncias não activas. Um pedido realizado a uma instância que entretanto ficou inactiva e que o Proxy não foi capaz de detectar, é perdido. Dependendo das exigências do sistema o tempo entre validações do estado das instâncias pode ser reduzido por forma a diminuir a probabilidade de perda de pedidos.

3.3.3.2 Servidor Prometheus

Tal como no caso do Proxy a monitorização tem de ser o mais dinâmica possível para que a escalabilidade e qualidade geral da solução não seja comprometida. Por esta razão é necessário evitar implementações restritas a casos específicos, que necessitem de ser refeitas de caso a caso.

O servidor Prometheus para funcionar correctamente requer um ficheiro YAML[4] onde estão definidas, entre outras características, a frequência com que o servidor recolhe as métricas expostas pelos serviços, o nome do serviço que deve monitorizar, os endereços dos mesmos e quais as métricas compostas que devem ser produzidas.

YAML é uma linguagem com um formato legível por humanos que é normalmente utilizada para ficheiros de configuração. Um exemplo de um ficheiro de configuração do Prometheus é mostrado na listagem 3.1. As propriedades *scrape_interval* e *evaluation_interval* dizem respeito ao ritmo de recolha de dados e avaliação de métricas compostas, respectivamente. A propriedade *job_name* identifica o nome do serviço a monitorizar, a propriedade *scrape_interval* define o ritmo de recolha de dados associados a um serviço em específico e dentro da propriedade *file_sd_configs* estão os nomes dos ficheiros que contêm a definição das métricas compostas.

Na configuração do servidor Prometheus é possível definir mecanismos capazes de descobrir dinamicamente as instâncias de um determinado serviço, alguns dos mecanismos disponíveis são: registos DNS, integração com serviços externos como Consul[5]. No

[4]YAML: http://yaml.org/
[5]Consul: https://www.consul.io/

entanto nesta fase do sistema optou-se por uma configuração estática, pois devido à limitação do número de portas por instâncias de serviço, imposta pelo Proxy do *Pod* de Serviço, a não utilização de um serviço de descoberta dinâmico não terá um impacto significativo. Concretamente a configuração das instâncias a monitorizar é feita através da declaração das portas expostas num serviço para o efeito da monitorização.

O Prometheus disponibiliza duas formas para a obtenção de métricas compostas: através da sua definição num ficheiro de configuração ou através de perguntas via REST, ambas as formas no formato PromQL[6], uma linguagem funcional disponibilizada pelo Prometheus que permite ao utilizador agregar e selecionar dados.

No caso concreto do sistema desenvolvido são utilizados os dois modos de obtenção de métricas. O ficheiro de configuração é utilizado para possibilitar a definição de todas as métricas complexas possíveis de analisar num serviço e para permitir a visualização da evolução do valor das métricas ao longo do tempo. As perguntas REST são utilizadas para que o Gestor consiga recolher apenas as métricas definidas no SLA de cada serviço, aquando do processo de adaptação dinâmica.

Uma vez que todos os serviços partilham um servidor Prometheus com a mesma configuração é possível reutilizá-la para todos os eventuais serviços, tornando-se deste modo acessível a configuração da monitorização de um novo serviço. Esta solução é viável devido ao facto de todos os serviços se encontrarem isolados uns dos outros, desta forma é possível partilhar as mesmas palavras-chave que o servidor prometheus utiliza para avaliar as métricas, sem existirem conflitos. Concretamente como é possível visualizar na listagem 3.1 a propriedade *job-name* indica o nome do serviço a monitorizar, utilizando o mesmo nome genérico na implementação das métricas básicas em cada serviço a monitorizar, torna-se possível utilizar o mesmo ficheiro de configuração.

Desta forma todas as métricas possíveis de analisar no sistema são partilhadas por todos o serviços. Tal não vai contra as métricas definidas no SLA para cada serviço por forma a avaliar a sua utilidade, pois através do gestor é possível perguntar ao servidor Prometheus sobre as métricas relevantes para cada serviço. Daí a necessidade de utilizar as perguntas. Resumidamente o ficheiro de configuração define todas as métricas que podem ser avaliadas permitindo deste modo consultar os gráficos de evolução das mesmas para todos os serviços, e as perguntas ao Servidor Prometheus são utilizadas pelo Gestor para obter apenas informação sobre as métricas relevantes para a avaliação autonómica de determinado serviço.

3.3.3.3 Tecnologias Escolhidas

Todos os componentes produzidos foram implementados com a linguagem Typescript[7] utilizando o NodeJs[8]. O Typescript é um superconjunto do Javascript cuja a principal

[6] PromQL: https://prometheus.io/docs/prometheus/latest/querying/basics/
[7] Typescript: https://www.typescriptlang.org/
[8] Nodejs: https://nodejs.org/en/

vantagem são os tipos e a maior orientação para objectos sendo possível definir interfaces e classes facilmente. O Nodejs é um runtime de Javascript sobre o qual os ficheiros Javascript produzidos pela compilação dos ficheiros Typescript, são executados. Optou-se por utilizar as referidas tecnologias de maneira a facilitar a integração do trabalho desenvolvido com o conjunto de tecnologias já utilizadas na SPMS.

A Base de Dados de Desempenho é não relacional, mais concretamente é uma Base de Dados de Documentos. A tecnologia utilizada foi o MongoDB[9] devido à fácil integração com o Nodejs e por forma a mais uma vez facilitar a adoção da solução no ecossistema da SPMS.

3.4 Pod Autonómico

O *Pod* Autonómico é o elemento mais complexo do sistema desenvolvido. Como foi proferido na secção 3.1, o presente *Pod* tem o papel de um Gestor Autonómico, sendo responsável pela análise e modificação dos Elementos Geridos (os *Pods* de Serviço). O cálculo do novo estado do sistema resulta da resolução de um problema de restrições que tem como objectivo optimizar a utilidade global dos serviços e minimizar os custos associados.

Nesta subsecção será analisada a arquitectura do *Pod*, bem como a função dos vários intervenientes, que compõem o fluxo de adaptação Autonómico.

3.4.1 Arquitectura Interna

O *Pod* Autonómico contém todos os elementos necessários de um Gestor Autonómico: o Monitorizador, o Analisador, o Planeador e o Executor. Contém adicionalmente um Proxy para controlar eficazmente os pedidos efectuados ao *Pod*. O *Pod* Autonómico comunica ainda com as Bases de Dados de Desempenho e Conhecimento e com a Base de Dados de SLAs através do Gestor de SLAs, como é possível visualizar na figura 3.6.

O **Controlador** é o componente responsável pela monitorização dos vários Pods de Serviço existentes, e pelo controlo do fluxo de adaptação autonómico. De uma forma sumária, a sua função consiste na recolha dos relatórios de execução de todos os *Pod*s de Serviço monitorizados, e no processamento da informação que alimenta os restantes componentes do *Pod*.

O **Analisador** é o componente responsável pelo cálculo do novo estado ideal, tendo por base a análise do estado actual. Para a obtenção do estado ideal necessita de resolver um problema de restrições que no caso do problema a solucionar, consiste na alocação de recursos por forma a maximizar a utilidade e minimizar o custo associado às configurações dos vários serviços. No final produz uma configuração ideal de acordo com as exigências correntes.

O **Planeador** tem como função desenvolver um plano de acções, que ao ser executado permita ao sistema aproximar-se do estado ideal. O plano gerado considera a configuração

[9]MongoDB: https://www.mongodb.com/what-is-mongodb

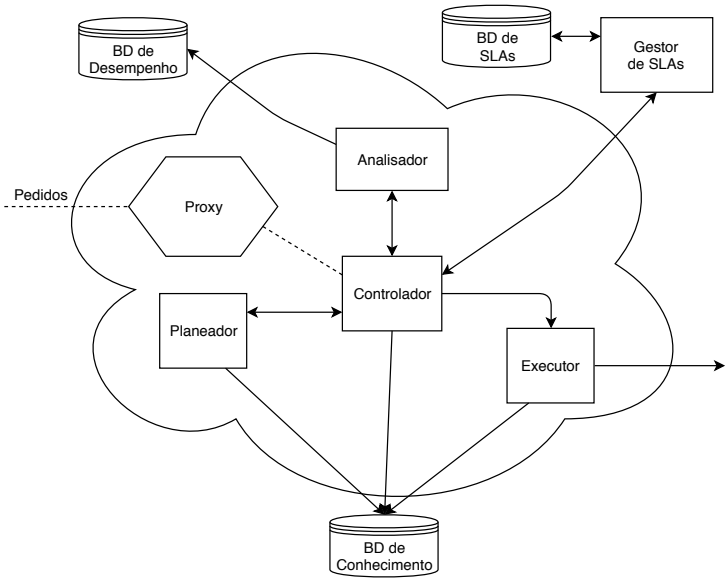

Figura 3.6: Arquitectura do *Pod* Autonómico

actual do sistema por forma a minimizar o impacto das alterações necessárias, evitando sempre que possível acções desnecessárias.

O **Executor** é responsável por executar o plano criado pelo Planeador. Este elemento comunica directamente com o provedor de recursos, daí ser necessário uma implementação por cada provedor a ser suportado.

Todos os componentes guardam informação útil da sua execução na Base de Dados de Conhecimento que poderá servir para a automatização completa do sistema. Tanto a Base de Dados de Conhecimento, de Desempenho e de SLAs não fazem parte do *Pod* Autonómico, embora sejam importantes para o funcionamento dos seus elementos.

3.4.2 Controlador

Por forma a oferecer uma visão geral sobre o funcionamento do sistema esta secção aborda o papel do Controlador, pois é este, o elemento responsável por despoletar todos os restantes elementos. Numa primeira fase procede ao levantamento de informação sobre o estado do sistema, posteriormente é responsável por processar e enviar os dados entre os restantes elementos do *Pod*, para que a adaptação autonómica possa ocorrer.

O referido elemento disponibiliza apenas um endpoint, responsável pela alteração do sistema. No entanto ao chamar esta função ocorrem várias subtarefas, como o levantamento dos serviços activos, a recolha dos estados dos serviços activos, a elaboração do relatório a reportar ao Analisador, elaboração do relatório de actualização do sistema e produção do plano de actualização do sistema.

O processo de alteração do sistema é ilustrado na figura 3.7, alguns elementos foram omitidos por forma a simplificar o fluxo. De seguida serão detalhadas cada uma das etapas que compõem a alteração do sistema.

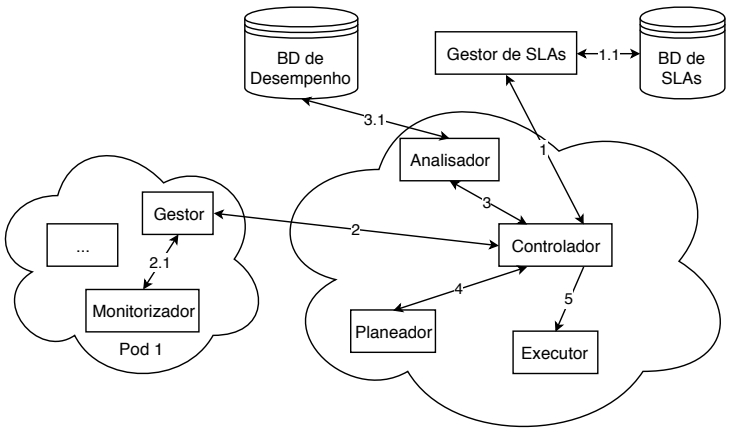

Figura 3.7: Análise do sistema

A primeira etapa do processo de alteração, consiste na recolha da informação relacionada com os serviços activos. Para obter esta informação o Controlador comunica com o Gestor de SLAs de modo a que este lhe faculte a lista de serviços com SLAs activos. Obtendo esta informação o Controlador comunica com os Gestores dos Pods de serviço activos, por forma a obter o relatório do estado de cada serviço. O pedido efectuado por parte do Controlador faz com que cada Gestor comunique com o Monitorizador do respectivo *Pod*, por forma a obter informação sobre o serviço.

À medida que os Gestores vão terminando os relatórios, os mesmos vão sendo entregues ao Controlador que entretanto ficou à espera de n relatórios, sendo n igual ao número de serviços activos. Assim que todos os relatórios tiverem sido retornados o fluxo continua.

A próxima fase compreende o processamento dos dados enviados pelos vários Gestores de modo a criar um relatório do sistema, que é posteriormente enviado para o Analisador para que este possa retornar o estado ideal. Terminado o relatório do sistema o mesmo é enviado para o Analisador, que efectua o procedimento explicado na secção 3.4.3.

Após a resposta do Analisador é feito o processamento dos dados recebidos, por forma a criar um relatório de actualização de sistema que será posteriormente enviado ao Planeador. O relatório mencionado é em tudo semelhante à resposta devolvida pelo Analisador, exceptuando a presença de informação sobre o estado actual do sistema. A adição desta informação é necessária para que o Planeador consiga comparar o estado proposto, com o estado antigo e assim criar um plano de alteração. Após a elaboração do relatório de actualização o mesmo é enviado ao Planeador que por sua vez devolve um plano com as acções a realizar por forma a atingir o estado objectivo.

O processo de alteração termina com o envio ao Executor do plano produzido pelo Planeador. O primeiro fica encarregue de realizar as operações descritas, recorrendo à API do provedor de recursos e assim alterar o estado do sistema.

3.4.3 Analisador

O Analisador é o componente responsável pela resolução do problema de optimização da utilidade global do sistema, utilizando como fonte de conhecimento os dados armazenados na Base de Dados de Desempenho. É também responsável pelo cálculo do novo estado ideal do sistema.

O componente em análise é composto por três fases principais, sendo elas: recolha e processamento dos dados da Base de dados de Desempenho, resolução do problema de alocação de recursos para os diferentes serviços e resolução do problema de distribuição dos recursos por instâncias em cada serviço. A primeira tem como função recolher e processar toda a informação relevante e necessária para a resolução do problema de optimização, a segunda tem como objectivo principal, definir o total de recursos para cada serviço, tendo em conta as exigências sobre os mesmos, e tentado sempre minimizar o custo e maximizar a utilidade global do sistema. A última fase designada fase de distribuição de recursos, consiste na definição do número de instâncias por serviço e dos recursos a alocar a cada uma.

As próximas três subsecções analisam em detalhe as três fases previamente mencionadas, detalhando a lógica presente em cada uma.

3.4.3.1 Recolha e processamento de dados

A fase de recolha e processamento de dados é crítica para a correcta análise do sistema, e posterior cálculo da configuração ideal. É através da informação recolhida nesta fase que o Analisador conseguirá inferir um nível de utilidade para cada configuração calculada, mesmo para cenários desconhecidos, pois recorrendo ao desempenho dos serviços em situações anteriores com cargas semelhantes, poderá calcular o valor de utilidade de uma configuração próximo da realidade, e com isso encontrar a melhor configuração para o estado corrente do sistema.

Nesta fase um aspecto crítico a ter em conta é o volume de dados a processar, pois sendo os recursos atribuídos a cada componente do *Pod* Autonómico limitados é imperativo reduzir ao máximo todo o tipo de informação e computação desnecessária. Desta forma os dados devolvidos pelos pedidos efectuados à Base de Dados, têm de conter apenas a informação estritamente necessária. Devido a esta condicionante apenas os dados relativos a um determinado número de pedidos por segundo são devolvidos.

Concretamente, estando a analisar o serviço a e sabendo que o mesmo está actualmente sujeito a x pedidos por segundo, o Analisador efectua uma pergunta à base de dados por forma a obter todos os dados relativos ao serviço a, com o número de pedidos igual a x

A figura 3.8 demonstra a informação registada na base de dados relativamente à métrica Latência do serviço a. Como é possível constatar o eixo dos Ys refere-se ao valor da métrica, e o eixo dos Xs refere-se ao número de pedidos por segundo. Para cada ponto está associada ainda uma configuração, como no caso de 1,5 pedidos por segundo, onde

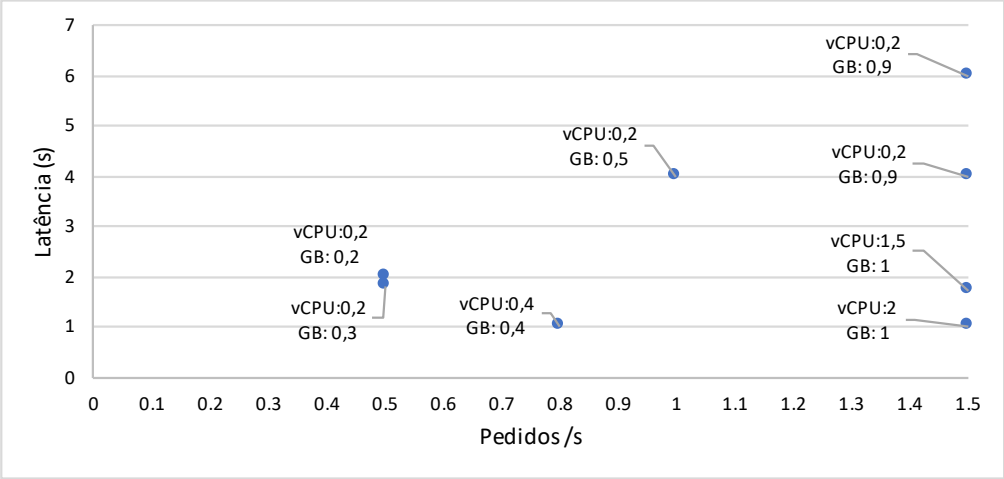

Figura 3.8: Valores exemplo da métrica latência do serviço *a*

existem 4 valores registados e todos com configurações diferentes. É importante destacar que quanto mais recursos estiverem alocados, melhores valores da métrica serão obtidos. No caso da Latência quanto menor for o valor da métrica melhor.

Como a Base de Dados ao início não contém muita informação é preciso tornar a procura menos precisa (caso não hajam entradas com o número de pedidos igual ao questionado), ou seja, considerar pedidos por segundo próximos do objectivo.

Tendo em conta a latência do serviço *a*, e supondo que é realizada uma pergunta por forma a obter os dados referentes a uma carga de 1,3 pedidos por segundo, é possível constatar que não existem dados associados a essa carga, no entanto existem dados associados a cargas próximas do objectivo, que podem ser utilizadas.

Neste caso o Analisador procuraria o maior valor de número de pedidos por segundo existente na base de dados imediatamente antes do número de pedidos alvo, e o menor valor de número de pedidos por segundo existente na base de dados imediatamente a seguir ao número de pedidos alvo. No caso concreto em análise seriam retornados os dados com o número de pedidos por segundo igual a 1 e a 1,5.

Por outro lado se fosse feito um pedido tendo em conta 1,5 pedidos por segundo, seriam apenas retornadas as 4 entradas ilustradas na figura 3.8.

Este comportamento possibilita o retorno dos dados necessários não sobrecarregando a memória com informação irrelevante, permitindo desta forma que mesmo sem o número de pedidos exacto, possa ser inferido um nível de utilidade com base em configurações próximas.

Feita a primeira filtragem de dados o Analisador agrega os dados que recebeu. Se existirem vários dados com configurações iguais e pedidos por segundo diferentes (como acontece quando não existe o número exacto de pedidos por segundo exigido) os valores com as mesmas configurações são agrupados e é feita a média do valor da métrica associado a essas configurações. No final deste processo é produzida uma lista com todas as configurações existentes para o número de pedidos exigido, contendo ainda o nome e

valor da métrica bem como o nome do serviço associado.

Devido à importância das configurações antigas no cálculo das novas configurações, é relevante referir que quanto mais populada estiver a Base de Dados, mais precisos serão os dados devolvidos e por conseguinte as configurações propostas.

$$util_m = limite_m - val_m \tag{3.5}$$

$$util_s = \sum_{i=1}^{m} util_i \tag{3.6}$$

Por fim é calculado o valor de utilidade das diferentes configurações. Este cálculo é realizado recorrendo ao valor limite estipulado para cada métrica no SLA de cada serviço. A utilidade de uma métrica *m* denotada por *util_m* é o seu valor limite menos o valor corrente da métrica (ver equação 3.5). Para calcular a utilidade do serviço *s*, denotada por *util_s*, e da sua configuração é calculado o somatório das utilidades de todas as métricas desse serviço (ver equação 3.6). Para além da utilidade é ainda calculado o valor de cada configuração, a necessidade deste valor é explicada na secção seguinte.

3.4.3.2 Fase de Alocação

A fase de Alocação consiste na definição e resolução de um problema de restrições que é apresentado de seguida. A definição das restrições foi inspirada em parte no sistema proposto em [32] e com base nos sucessivos testes realizados ao sistema. Para cada serviço activo no sistema é definido um conjunto de restrições. Primeiramente definem-se as restrições associadas ao limite de recursos disponíveis para cada *Pod* de Serviço. Sendo as mesmas agora apresentadas:

$$\forall x; \quad x \in S_a \quad Pod_{min\_cpu} \leq cpu_x \leq Pod_{max\_cpu} \tag{3.7}$$

$$\forall x; \quad x \in S_a \quad Pod_{min\_mem} \leq mem_x \leq Pod_{max\_mem} \tag{3.8}$$

Através das restrições anteriormente definidas restringe-se a memória e o CPU de todos os serviços activos, ao mínimo e máximo permitido num *Pod*. O segundo grupo de restrições define as restrições associadas ao custo, dos recursos alocados. O custo total da configuração de um serviço é igual à soma do custo do CPU proposto mais o custo da memória proposta (eq 3.10). Por sua vez o custo do CPU e da memória é calculado multiplicando o custo de um vCPU e de um GB por segundo pelo o total dos respectivos recursos (eq 3.9).

$$cost_{cpu} = cost_{vcpu} \times cpu \quad e \quad cost_{mem} = cost_{gb} \times mem \tag{3.9}$$

$$cost_s = cost\_cpu_s + cost\_mem_s \tag{3.10}$$

O terceiro grupo de restrições é extenso, pois é onde são definidas as restrições associadas ao cálculo da utilidade das configurações propostas. Nesta etapa são utilizadas as configurações devolvidas na fase de recolha e processamento de dados, dependendo destas podem ocorrer dois cenários:

1. Das configurações recolhidas é possível extrair uma configuração máxima e uma mínima, ambas diferentes.

2. Das configurações recolhidas é possível extrair uma configuração máxima e uma mínima, ambas iguais e por isso existe apenas um valor que é o máximo e o mínimo ao mesmo tempo.

A configuração máxima é a configuração que apresenta a melhor combinação de recursos (na maioria dos casos a mais dispendiosa também), a mínima é exactamente o oposto. Para comparar as configurações anteriores, é calculado um valor de configuração para cada uma. O valor é calculado através da seguinte fórmula.

$$val_{config} = cpu_{config} * cpu_{imp} + mem_{config} \tag{3.11}$$

onde val_{config} corresponde ao valor da configuração, o cpu_{config} e a mem_{config} ao CPU e à memória da configuração respectivamente, e o cpu_{imp} à importância dada ao CPU. A configuração com o maior valor é a configuração máxima, a configuração com o menor valor é a configuração mínima.

O valor de configuração e a procura da configuração mínima e máxima é crucial para o cálculo da utilidade, uma vez que o valor de utilidade de todas as configurações propostas é inferido a partir da comparação da configuração proposta com as configurações máxima e mínima. As restrições associadas ao cálculo da utilidade tendo em conta a utilidade das configurações máxima e mínima é agora descrito.

A referida comparação é realizada através da divisão do valor da configuração proposta pelo valor da configuração mínima e máxima, como é especificado na equação 3.12.

$$comp_{min} = \frac{val_{config}}{val_{min\_config}} \quad e \quad comp_{max} = \frac{val_{config}}{val_{max\_config}} \tag{3.12}$$

onde $comp_{min}$ representa o valor da comparação entre o valor da configuração proposta e o valor da configuração mínima, e o $comp_{max}$ o mesmo, só que tendo em conta o valor da configuração máxima. Da referida divisão podem ocorrer três casos se a configuração máxima e mínima forem distintas e dois casos se a configuração máxima e mínima forem iguais, os cinco casos são ilustrados na figura 3.9.

Através da comparação anterior torna-se possível inferir um nível de utilidade plausível. O referido valor é calculado através da comparação da posição do valor de configuração, da configuração proposta em relação à posição do valor da configuração mínima e máxima. A próxima lista demonstra qual o valor de utilidade em cada cenário.

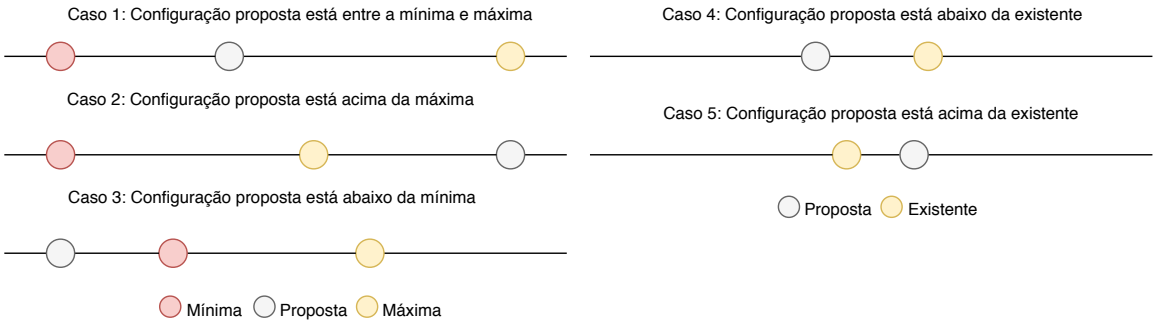

Figura 3.9: Comparação entre valores de configuração

- No caso da configuração estar entre a mínima e a máxima, a utilidade é igual:

$$util_{config} = util_{min\_config} + comp_{min} \times soft \tag{3.13}$$

- No caso da configuração ser maior que a máxima, a utilidade é igual:

$$util_{config} = util_{max\_config} + comp_{max} \times soft \tag{3.14}$$

- No caso da configuração ser menor que a mínima, a utilidade é igual:

$$util_{config} = util_{min\_config} - comp_{min} \times soft \tag{3.15}$$

- No caso da configuração máxima e mínima serem iguais e a proposta ser maior que a existente, a utilidade é igual:

$$util_{config} = util_{exist\_config} + comp_{exist} \times soft \tag{3.16}$$

- No caso da configuração máxima e mínima serem iguais e a proposta ser menor que a existente, a utilidade é igual:

$$util_{config} = util_{exist\_config} - comp_{exist} \times soft \tag{3.17}$$

Nas equações anteriores a variável *comp* representa a divisão entre o valor da configuração proposta e a configuração máxima, mínima e existente. A variável $soft$ é um valor que permite controlar a importância das diferenças entre configurações, para o cálculo da utilidade.

Tendo a utilidade da configuração, o grupo de restrições final tem como objectivo calcular o valor final do serviço, que é obtido através da seguinte fórmula.

$$val_{servico} = util_{servico} - cost_{servico} \times cost_{importance} \tag{3.18}$$

Serviço	vCPU	Memória
s_utente	0.8	0.3
s_medicamento	3	7.8

Tabela 3.2: Exemplo do retorno da fase de alocação

A variável $val_{servico}$ representa o valor do serviço, a variável $util_{servico}$ representa a utilidade da configuração proposta, a variável $cost_{servico}$ representa o custo da configuração proposta e a variável $cost_{importance}$ é a importância dada ao custo da configuração.

Tendo o valor dos vários serviços é calculado o valor do sistema, que resulta da soma ponderada dos valores de todos os serviços, como é demonstrado na equação 3.19.

$$val_{sistema} = \sum_{i=1}^{S} w_i \times val_i \tag{3.19}$$

onde $val_{sistema}$ indica o valor do sistema, S o número de serviços, w_i e val_i o peso e valor do serviço i, respectivamente.

Por fim é feita a pesquisa no espaço de soluções por forma a encontrar a melhor solução que respeita todas as restrições definidas anteriormente e que maximize o valor do sistema. A tabela 3.2 ilustra uma solução proposta na fase de Alocação.

3.4.3.3 Fase de Distribuição

A última fase do Analisador consiste na divisão dos recursos atribuídos aos serviços na fase de alocação, por instâncias. O processo consiste na definição de três restrições para cada serviço:

1. As primeiras restrições definem os intervalos de valores de CPU e de memória que é possível atribuir a uma instância.

2. A segunda restrição define que a soma dos recursos de todas as instâncias de um serviço seja igual ao número de recursos alocados para o serviço.

$$\sum_{i=1}^{c} cpu_i \leq Pod_{cpu} \quad 1 \leq i \leq c \tag{3.20}$$

$$\sum_{i=1}^{c} mem_i \leq Pod_{mem} \quad 1 \leq i \leq c \tag{3.21}$$

3. A terceira restrição define que a diferença entre os recursos atribuídos às instâncias não pode ultrapassar os 30%, por forma a impedir que a maioria dos recursos se concentrem numa só instância.

$$\forall s; s \in S, \quad \nexists i; i \in Inst_s \quad \frac{min(cpu_{si}, cpu_{s\neg i})}{max(cpu_{si}, cpu_{s\neg i})} < 0.7, \tag{3.22}$$

Serviço	Instâncias	vCPU	Memória
s_utente	inst_1	0.8	0.3
s_medicamento	inst_1	0.6	1,5
	inst_2	0.8	2,1
	inst_3	0,8	2,1
	inst_4	0,8	2,1

Tabela 3.3: Exemplo do retorno da fase de distribuição

$$\forall s; s \in S, \quad \nexists i; i \in Inst_s \quad \frac{min(cpu_{si}, cpu_{s\neg i})}{max(cpu_{si}, cpu_{s\neg i})} < 0.7 \tag{3.23}$$

À semelhança da fase de alocação é realizada uma pesquisa em profundidade que no final devolve as soluções possíveis tendo em conta as restrições definidas previamente.

No final desta etapa é produzido um relatório onde estão definidos os recursos de cada serviço bem como a configuração das instâncias por serviço, como é ilustrado na tabela 3.3.

3.4.4 Planeador

Nesta secção é apresentada a fase de planeamento responsável pela elaboração do plano onde estão descriminadas as acções necessárias para atingir-se o estado objectivo, proposto pelo Analisador.

O componente responsável pela fase de planeamento é o Planeador. O plano produzido nesta etapa, consiste num conjunto de acções a efectuar para cada serviço. Existem três acções possíveis:

Adicionar instância(*add_inst*) - Indica que devem ser adicionadas *x* instâncias de uma determinada configuração a um serviço

Remover instância(*rem_inst*) - Indica que devem ser removidas *x* instâncias de uma determinada configuração de um serviço

Não alterar instância(*same_inst*) - Indica que as instâncias de uma determinada configuração devem ser mantidas

Estas operações são as operações básicas que compõem um plano, e são produzidas pelo Planeador através da comparação do estado actual do sistema e do estado proposto.

Nas tabelas 3.4 e 3.5 estão identificadas a configuração corrente e a configuração proposta do serviço_1 respectivamente, tal como é proposto no relatório de actualização. Existem quatro colunas: a primeira referente ao nome do serviço, a segunda às instâncias dos serviços e as duas restantes referentes aos recursos das instâncias. Através da comparação das configurações correntes e das configurações propostas dos vários serviços, o

Serviço	Instâncias	vCPU	Mem
serviço_1	inst_1	1	1
	inst_2	1	1
	inst_3	0,8	1,5
	inst_4	0,4	0,7

Tabela 3.4: Configuração corrente do *serviço_1*, num relatório de actualização

Serviço	Instâncias	vCPU	Mem
serviço_1	inst_1	1,5	2,1
	inst_2	1	1
	inst_3	0,3	0,5

Tabela 3.5: Configuração proposta do *serviço_1*, num relatório de actualização

Planeador é capaz de elaborar um plano. O modo como as configurações são comparadas é agora descrito.

As instâncias da configuração proposta são iteradas e analisadas, se existirem instâncias iguais à configuração actual é criado um grupo, identificado pelas configurações da instância em análise e a operação *same_inst*. Ao referido grupo é ainda adicionado um contador onde está definido o número de instâncias desse tipo. Na mesma iteração são também analisadas as instâncias a adicionar, o processo é o mesmo, neste caso se a instância iterada não tiver configuração semelhante nas instâncias da configuração actual é criado um grupo com as configurações da instância proposta, com a operação *add_inst* e um contador que é incrementado por cada instância com configurações semelhantes.

A última etapa na criação do plano, consiste na iteração das instâncias actuais de modo a identificar quais as instâncias a remover. Para todas as instâncias sem correspondência na configuração proposta é criado um grupo com a configuração da instância, a operação *rem_inst* e um contador que indica o número de instâncias a remover.

Serviço	Operação	Quantidade	vCPU	Mem
serviço_1	add_inst	1	1,5	2,1
		1	0,3	0,5
	rem_inst	1	1	1
		1	0,4	0,7
	same_inst	1	1	1

Tabela 3.6: Plano de actualização do sistema.

No final deste processo é produzido um relatório semelhante ao que é apresentado na tabela 3.6, onde estão descriminadas as alterações que necessitam de ser efectuadas para cada serviço.

3.4.5 Executor

A presente secção analisa a etapa de execução responsável pela realização da actualização do sistema definida no plano. O componente responsável pela execução do plano designa-se Executor.

A sua função consiste na execução do plano produzido pelo Planeador, de acordo com o provedor de recursos onde os serviços a serem alterados estão alojados. Para que esta tarefa seja possível, o Executor deve conter uma implementação por provedor de recursos de acordo com a interface apresentada na listagem 3.2. Tendo uma implementação por provedor de recursos as acções indicadas no plano de alteração, são correspondidas aos métodos da API da listagem anteriormente indicada. Por razões de desempenho a execução das acções descriminadas no plano, não corresponde na íntegra ao mapeamento directo das mesmas com os métodos expostos pelo provedor, a explicação é feita na secção 3.4.6.2.

Listagem 3.2: API do elemento Executor

```
1  export interface ICloudProvider {
2    // lista todos os Pods do Provedor de Recursos
3    listContainerPods();
4
5    // obtém o Pod identificado por podId
6    getContainerPod(podId: string);
7
8    // cria um novo Pod identificado por podId com a configuracao requests
9    createContainerPod(podId: string, request: any);
10
11   // remove um Pod ifentificado por podId
12   deleteContainerPod(podId: string);
13
14   // devolve os logs de um contentor identificado por container
15   getContainerInstanceLogs(podId: string, container: string);
16
17   //actualiza os recursos de um contentor identificado por container
18   updateContainerInstanceResources(podId: string, container: string, resources: any);
19
20   // actualiza os recursos de todos os contentores de um Pod identificado por podId
21   updateContainersInstancesResources(podId: string, resources: any);
22
23   // aumenta o numero de contentores de um Pod identificado por podId
24   increaseContainersCount(podId: string, count: number);
25
26   // dimininui o numero de contentores de um Pod identificado por podId
27   decreaseContainersCount(podId: string, count: number);
28 }
```

3.4.6 Detalhes de Implementação

Nesta secção são explicados alguns detalhes que ficaram por explicar relacionados com a configuração e execução dos serviços. A subsecção termina com a discussão das tecnologias utilizadas para desenvolver os componentes do *Pod* Autonómico.

3.4.6.1 Configuração e utilização dos Serviços

Nesta subsecção será explicado como é que os serviços são configurados e iniciados. Todos os serviços são lançados em contentores Docker, para executá-los é necessário um daemon Docker, que é providenciado pelo ACI. No entanto é necessário um elemento ainda mais importante onde está descriminado o que o contentor deve conter. Este elemento é a imagem Docker do serviço e é ilustrado na listagem 3.3

Listagem 3.3: Ficheiro de configuração Docker de um serviço

```
1  FROM node:8-alpine
2
3  RUN mkdir -p /usr/src/app
4  WORKDIR /usr/src/app
5
6  COPY package.json /usr/src/app/
7  COPY package-lock.json /usr/src/app/
8  RUN npm install pm2 -g && npm set registry
       http://npm-repo.westeurope.azurecontainer.io:4873 && npm install
9
10 COPY ./dist /usr/src/app/dist
11 COPY process.yml /usr/src/app/
12
13 EXPOSE 10100
14 CMD ["sh", "-c", "pm2-docker process.yml"]
```

A imagem Docker é um ficheiro YAML onde está definida toda a informação necessária para construir e executar o contentor. Os elementos mais importantes são a instrução FROM que indica a imagem base do contentor, as instruções anteriores ao EXPOSE servem para configurar a directoria que vai conter o serviço Nodejs e para adicionar o repositório npm privado onde alguns pacotes npm são descarregados. A directoria EXPOSE define a porta do contentor que deve ser exposta e por fim a instrução CMD indica o comando que deve ser executado para lançar o serviço. Todos os serviços têm uma imagem Docker semelhante, exceptuando o Analisador, que por ter sido desenvolvido em Java tem uma configuração bastante diferente.

Para que as imagens possam ser utilizadas no Azure é necessário efectuar alguns passos, por forma agilizar este processo foram desenvolvidos scripts bash para automatizar o processo. Os scripts são responsáveis pela compilação de todos os serviços, pela construção das images de todos os serviços e publicação das mesmas para o repositório Docker que está alojado no Azure.

3.4.6.2 Executor

Um detalhe importante a referir sobre a execução do plano, é o facto do Executor não realizar as operações básicas como de um plano se tratasse, mas sim utilizar as mesmas para construir um objecto com a configuração do *Pod* esperada após as execuções das operações básicas. O objecto em causa é posteriormente enviado para o Azure, dando origem a um novo *Pod* que substitui o antigo.

O processo anterior tem como objectivo optimizar a criação de *Pods* no Azure. Actualmente o Azure só suporta a criação de *Pods*, sendo impossível alterar um já existente, desta forma para poder respeitar a API do Executor, apresentada na secção 3.4.5, as operações que modifiquem a configuração do *Pod* resultam invariavelmente na destruição do *Pod* e posterior criação, ou seja, por cada acção que altere o estado do *Pod* é necessário apagar o *Pod* e lançar um novo, desta feita de acordo com a acção executada.

Num plano com inúmeras acções, a limitação da API do Azure significaria um custo temporal e económico tremendamente alto. Por todas as razões mencionadas a implementação do Azure no Executor suporta a remoção e a actualização de instâncias com os custos mencionados, mas as mesmas não são utilizadas durante o processo de adaptação dinâmico, sendo que na prática é produzida uma configuração de um novo *Pod* tendo em conta as instâncias a adicionar e as instâncias a manter. Desta forma reduz-se o número de chamadas à API do Azure de n pedidos (sendo n o número de remoções + actualizações) para apenas uma chamada.

3.4.6.3 Tecnologias Escolhidas

À semelhança do *Pod* de Serviço todos os elementos do *Pod* Autonómico foram implementados em Typescript devido às mesmas razões apresentadas na secção 3.3.3.3. O Analisador foi o único elemento que não foi programado através de Typescript, tendo sido desenvolvido em Java. Esta decisão foi tomada devido à falta de bibliotecas suficientemente robustas e funcionais para a resolução de problemas de restrições no ecossistema do Nodejs.

No que diz respeito à biblioteca utilizada para a resolução do problema de optimização optou-se pela biblioteca JaCoP[24], uma vez que esta oferecia integração com a definição de problemas em Minizinc[10], tecnologia utilizada no início da definição e modelação das restrições. Foi também considerada outra biblioteca para a modelação de restrições designada Choco[11], mas uma vez que já tinha sido iniciado o desenvolvimento com a biblioteca JaCoP e ambas as bibliotecas têm sintaxes parecidas optou-se pela primeira.

3.5 Considerações Finais

No presente capítulo foi detalhado o sistema desenvolvido, começando numa primeira fase pela a arquitectura e os seus elementos principais, destacando o *Pod* de Serviço e o *Pod* autonómico, que foram especificamente desenhados para o sistema desenvolvido. Foi também explicado ao detalhe o funcionamento do sistema e as relações existentes entre os componentes. O capítulo termina com a explicação de detalhes de implementação que não forem possíveis incluir ao longo da secção do *Pod* Autonómico mas que são indispensáveis para perceber com maior clareza a solução desenvolvida.

[10] Minizinc: http://www.minizinc.org/
[11] Choco solver: http://www.choco-solver.org/

O próximo capítulo é inteiramente dedicado à avaliação do sistema aqui apresentado. Ao longo do capítulo será possível avaliar o desempenho do sistema desenvolvido e perceber quais as grandes vantagens que a solução aqui proposta trará.

Capítulo 4

O caso de uso da SPMS e a sua avaliação

Por forma a validar a arquitectura e implementação do sistema desenvolvido, no presente capítulo são apresentados os resultados obtidos durante as simulações realizadas, que irão permitir demonstrar o desempenho do novo sistema e o seu comportamento em diferentes cenários. A avaliação realizada tem como objectivo principal perceber o impacto do novo sistema na maximização da utilidade dos vários serviços tendo em conta as várias situações testadas.

4.1 Métricas de Avaliação

Através dos testes demonstrados neste capítulo pretende-se analisar o comportamento do sistema desenvolvido em vários cenários e perceber acima de tudo o impacto da adaptação dinâmica no valor de utilidade dos vários serviços.

Por forma analisar o comportamento foram realizados testes em ambiente simulado. Não foram efectuados testes em ambiente real por duas razões, a primeira devido ao facto de se tratar de um sistema a integrar num contexto real não se poderia testar os serviços utilizados em produção sem primeiro passarem por um conjunto de testes internos da SPMS que atestem a viabilidade do sistema. O segundo motivo pelo qual o teste em ambiente real não foi realizado, tem que ver com o facto da carga actual sobre os serviços não ser significativa para causar um impacto relevante no desempenho dos serviços, correndo o risco de nunca ser necessário a adaptação dos mesmos, o que tornaria impossível avaliar o sistema desenvolvido.

Para a avaliação do desempenho do sistema em ambiente simulado, são consideradas as seguintes métricas:

- a latência dos serviços, sob diferentes cargas

- o custo, associado às configurações dos serviços
- a utilidade dos serviços
- o cumprimento dos SLAs

4.2 Serviços Utilizados

Por forma a testar o sistema, foram modificados três serviços da SPMS, todos diferentes no que diz respeito ao desempenho e funções. O primeiro serviço é responsável por recolher informação sobre utentes e transformar o formato da informação recolhida de acordo com o padrão europeu de saúde FHIR[1], o serviço designa-se ***s_utente***. O referido serviço consiste num *middleware* que comunica com um serviço implementado em Java responsável pela procura de utentes. No que diz respeito às funções testadas utiliza-se a procura por número nacional de saúde, que é bastante eficiente e retorna informação sobre um só utente, sendo ainda utilizada a procura por nome e género que tem uma duração superior e pode retornar informação sobre vários utentes. Em média um pedido demora 1,5 segundos a ser resolvido.

O segundo serviço é responsável pela consulta de medicamentos e transformação dos mesmos no formato FHIR que neste caso provoca um *overhead* significativo. A consulta de medicamentos é realizada tenho em conta o nome comercial ou o composto activo e a dosagem, e é um processo muito exigente comparativamente à procura de utentes. O serviço designa-se ***s_medicamento*** e em média demora entre 7 a 8 segundos a retornar dados.

O último serviço permite a consulta de patologias e despachos associados, os dados recolhidos são também transformados no formato anteriormente mencionado. Este serviço em concreto é muito responsivo na medida em que existem poucos dados na Base de Dados de patologias o que faz com que o poder de processamento do serviço seja maior. O serviço designa-se ***s_patologia***. Em média um pedido realizado a este serviço demora 0,09 segundos a concluir.

Para cada serviço testado foi criado um SLA distinto, que reflecte os objectivos a atingir em cada serviço. Os valores limite de cada métrica dos anteriores serviços podem ser consultados nas tabelas 4.1, 4.2 e 4.3 respectivamente.

4.3 Ambiente simulado

A avaliação do sistema recorrendo ao ambiente simulado foi efectuada através de três volumes de carga distintos:

- **Volume de carga 1** - É um volume estático sem variações acentuadas do número de utilizadores virtuais e consequentemente do número de pedidos

[1] https://www.hl7.org/fhir/

Serviço	Métrica	Limite
s_medicamento	indisponibilidade	10%
	latência	8s
	média de pedidos por instância	2 pedidos /s
	ritmo de transações falhadas	1 pedido /s

Tabela 4.1: Limites das métricas do serviço *s_medicamento*

Serviço	Métrica	Limite
s_utente	indisponibilidade	5%
	latência	1.5s
	média de pedidos por instância	3 pedidos /s
	ritmo de transações falhadas	0.1 pedido /s

Tabela 4.2: Limites das métricas do serviço *s_utente*

Serviço	Métrica	Limite
s_patologia	indisponibilidade	1%
	latência	0.1s
	média de pedidos por instância	100 pedidos /s
	ritmo de transações falhadas	0.01 pedido /s

Tabela 4.3: Limites das métricas do serviço *s_patologia*

- **Volume de carga 2** - Este volume caracteriza-se por um número reduzido de utilizadores no início e por um aumento brusco de utilizadores a meio da execução terminando novamente com um número reduzido de utilizadores

- **Volume de carga 3** - O volume é caracterizado por um aumento constante do número de utilizadores

Serão analisados três cenários, cada um reflecte a evolução de uma ou duas métricas. O primeiro cenário compara a evolução dos serviços tendo em conta a utilidade e o custo associado, o segundo cenário avalia a evolução da latência e o último cenário permite averiguar o cumprimento dos valores limite definidos nos SLAs. Todos os testes realizados têm em conta um período de 60 minutos.

Inicialmente pretendia-se testar o sistema com um número superior de serviços em simultâneo, mas devido aos limites definidos pelo Azure só é possível criar 60 contentores por hora, o que torna impossível os testes com mais de quatro serviços em simultâneo. Também devido às restrições anteriores a partir de dois serviços em simultâneo foi necessário reduzir o número de instâncias máximas por serviço pelo que só são apresentados testes com 1 ou 2 serviços, por forma a não restringir ainda mais o número máximo de recursos disponíveis.

Antes de se iniciar a apresentação dos testes realizados é importante referir que a reconfiguração de um serviço compreende o tempo entre a remoção do *Pod* antigo até ao momento em que o novo *Pod* está pronto a receber pedidos e demora aproximadamente um minuto, este valor pode variar consoante a disponibilidade do provedor de recursos utilizado, neste caso o Azure. Durante o tempo referido nenhum dado sobre o serviço é registado uma vez que o Gestor do Pod, responsável por essa operação, não está operacional por ter sido reiniciado juntamente com o Pod.

Devido a esta condicionante os gráficos apresentados em seguida, referentes aos serviços com adaptação, não contêm todos os pontos como os serviços correspondentes sem adaptação.

4.3.1 Evolução do Custo e da Utilidade

A presente secção analisa a evolução da utilidade dos vários serviços bem como o custo associado às novas configurações propostas pelo Pod Autonómico. Através dos testes aqui realizados pretende-se analisar o impacto da adaptação autonómica no que diz respeito ao valor de utilidade dos serviços, sendo expectável que os serviços sem adaptação tenham um desempenho inferior aos serviços com adaptação. É ainda analisado o impacto monetário associado às reconfigurações dos serviços para assim perceber os custos e ganhos face ao sistema antigo.

4.3.1.1 Testes com 1 Serviço

O primeiro teste é referente ao serviço *s_medicamento*. No gráfico 4.1 é possível visualizar a evolução da utilidade do serviço com e sem adaptação, ao longo de 60 minutos. É notória a diferença de desempenho entre os serviços, sendo que o serviço com adaptação autonómica apresenta um valor de utilidade razoavelmente constante e muito superior ao valor do outro serviço em análise. Neste teste, o Pod Autonómico analisou o sistema a cada 10 minutos e foram realizados em média 2 pedidos por segundo. Ambos os serviços começaram com os mesmos recursos: 1 instância com 0.5 vCPU e 0.5 GB de memória.

Até ao minuto 20 os valores de utilidade são semelhantes pois os dois serviços apresentam a mesma configuração, a partir desse momento e devido ao facto da utilidade ter sofrido uma redução acentuada, o Analisador procede à primeira reconfiguração do serviço (consultar tabela 4.4), sendo possível visualizar nos 10 minutos seguintes uma ligeira subida da utilidade mantendo-se esta consideravelmente acima da utilidade do serviço sem adaptação.

Nos restantes minutos ocorrem mais três análises e três reconfigurações devido ao crescimento constante do número de pedidos por segundo. No minuto 30 o Analisador altera novamente a configuração do serviço, desta feita reduzindo ligeiramente a memória alocada. Após a anterior reconfiguração e devido ao aumento ligeiro da utilidade é realizada uma terceira reconfiguração aos 40 minutos que reduz os vCPUs alocados, por forma a reduzir o custo.

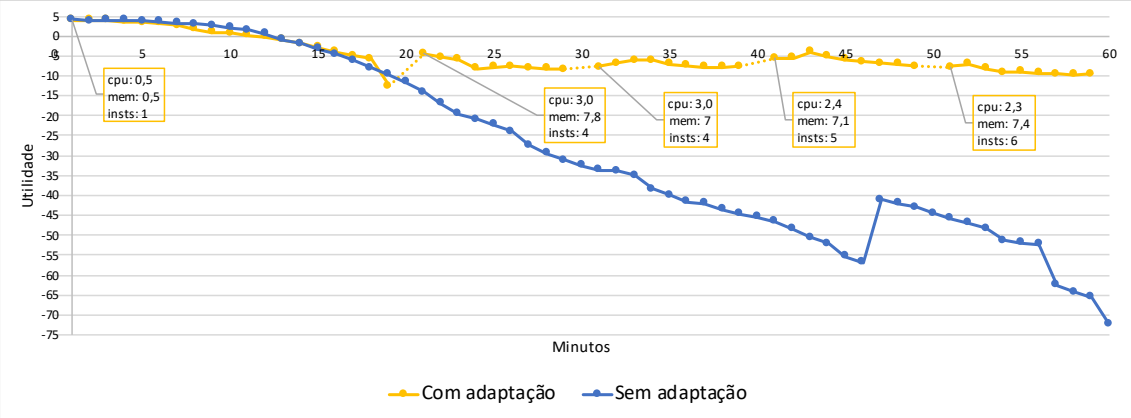

Figura 4.1: Evolução da utilidade do serviço *s_medicamento* sob o volume de carga 3

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	4	3	7.8
2ª reconfiguração	4	3	7
3ª reconfiguração	5	2.4	7.1
4ª reconfiguração	6	2.3	7.4

Tabela 4.4: Reconfigurações do serviço *s_medicamento* sob o volume de carga 3

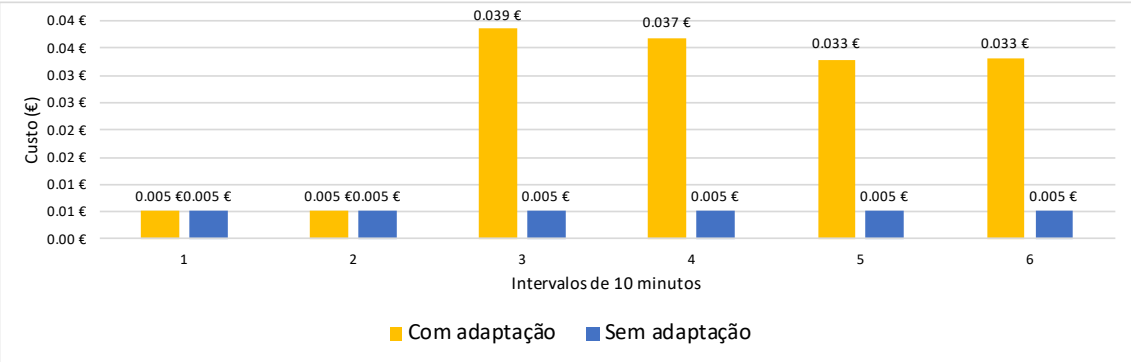

Figura 4.2: Evolução do custo do serviço *s_medicamento* sob o volume de carga 3

Até aos 50 minutos devido à utilidade apresentar um valor constante, o Analisador executa a quarta reconfiguração semelhante à anterior mas com uma redução mínima dos vCPUs e com um acréscimo de 300MB de memória. A partir do minuto 50 a utilidade em ambos os serviços deteriora-se devido ao aproximar do número máximo de pedidos. No fim da execução do teste o serviço com adaptação apresenta uma utilidade 7 vezes superior à do serviço sem adaptação.

Através da figura 4.2 é possível perceber o custo associado às configurações dos serviços. Como era expectável o serviço sem adaptação tem sempre o mesmo custo a cada dez minutos, uma vez que a sua configuração não varia. O serviço com adaptação nos primeiros 20 minutos apresenta o mesmo custo que o serviço anterior, visto que as configurações são idênticas. O ponto 3 do gráfico equivale à reconfiguração mais dispendiosa sendo que

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	3	0.8	1.6
2ª reconfiguração	5	2.8	7
3ª reconfiguração	3	1.9	5.7
4ª reconfiguração	1	0.2	0.3

Tabela 4.5: Reconfigurações do serviço *s_medicamento* sob o volume de carga 2

as seguintes devido à redução de recursos produzem configurações ligeiramente mais económicas custando cada uma cerca 3 cêntimos a cada 10 minutos.

O comportamento do sistema no contexto anterior vai de encontro ao esperado, confirmando a necessidade de se alocar mais recursos por forma a fazer face a um eventual aumento de carga o que implica um aumento do custo.

Embora a utilidade do serviço com adaptação tenha sido consideravelmente superior ao serviço sem adaptação, o valor obtido está longe do ideal, pois foi sempre negativo a partir do minuto 12. Quando o valor de utilidade é negativo significa que a soma de uma ou mais métricas negativas (uma métrica é negativa se a sua utilidade for negativa, resultado do valor corrente da métrica ser superior ao estabelecido no SLA) sobrepõe-se à soma das métricas positivas, querendo isto dizer que uma ou mais métricas estiveram abaixo do que foi definido nos seus SLAs. O que estes valores sugerem é que o máximo de recursos e instâncias possíveis de alocar não são suficientes para fazer face as exigências do serviço *s_medicamento* em concreto.

O próximo teste analisa o comportamento do serviço *s_medicamento* quando ocorre um crescimento brusco da carga a meio da execução. Na figura 4.3 constata-se que o desempenho do serviço com adaptação é superior ao serviço sem adaptação. O pico de carga ocorre do minuto 25 ao minuto 40.

Até ao minuto 30 ambos os serviços têm o mesmo valor de utilidade, a reconfiguração proposta ao minuto 20 pouco ou nenhum impacto teve na melhoria da utilidade do serviço com adaptação. Entre o minuto 20 e o minuto 30 o ritmo de pedidos aumenta o que agrava a utilidade de ambos os serviços. Devido à diminuição da utilidade, o Analisador ao minuto 30 altera novamente a configuração do serviço, alocando mais recursos e instâncias, por forma a que o serviço consiga suportar a carga corrente.

Nos restantes minutos ocorrem mais duas análises e duas reconfigurações. Devido ao elevado número de recursos atribuídos na segunda reconfiguração, o serviço com adaptação foi capaz de recuperar a sua utilidade, razão pela qual ao minuto 40 é realizada uma nova reconfiguração responsável por diminuir o número de recursos alocados. A última reconfiguração ocorre ao minuto 50 e consiste na atribuição dos recursos mínimos ao serviço visto que o número de pedidos diminuiu drasticamente.

É importante mencionar a completa incapacidade do serviço sem adaptação de lidar com o aumento da carga, ficando o mesmo inoperacional ao minuto 40.

A tabela 4.5 ilustra as configurações propostas em cada reconfiguração. O gráfico 4.4

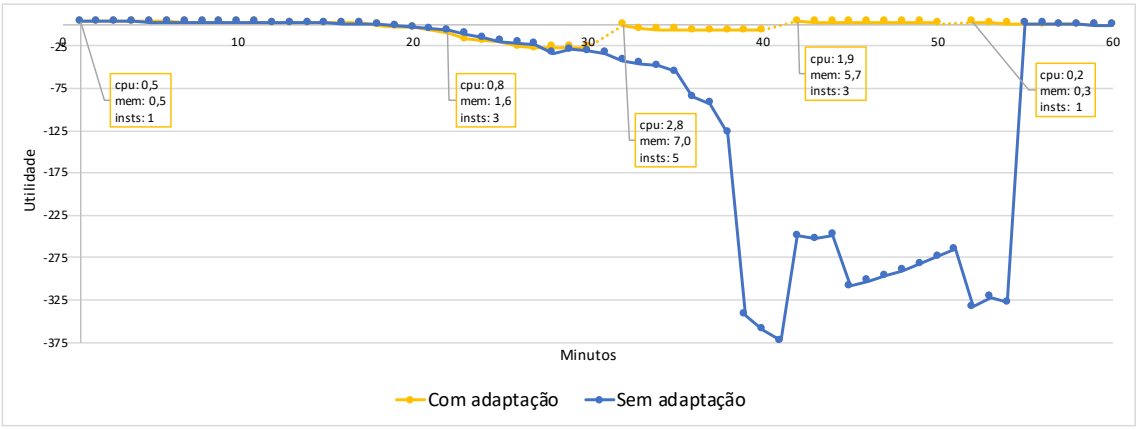

Figura 4.3: Evolução da utilidade do serviço *s_medicamento* sob o volume de carga 2

demonstra o custo associado à execução dos dois serviços analisados.

Como era expectável nos primeiros 20 minutos o custo dos dois serviços é igual, pois a configuração de ambos é idêntica. Nos 20 minutos seguintes o serviço com adaptação é mais dispendioso fruto das adaptações propostas para fazer face ao aumento acentuado do número de pedidos. Nos 20 minutos finais o custo associado ao serviço com adaptação diminui, no ponto 5 o custo por cada dez minutos é reduzido de 3 cêntimos para 2 cêntimos. Na última reconfiguração por terem sido alocados os recursos mínimos ao serviço, o custo é reduzido drasticamente sendo inclusivamente inferior ao custo do serviço sem adaptação, que se manteve constante ao longo dos 60 minutos como seria de esperar.

À semelhança do teste anterior a diferença de desempenho entre o serviço com adaptação e sem adaptação é notória especialmente durante o pico de carga onde a utilidade do serviço sem adaptação desceu para valores que inviabilizam a sua utilização. No entanto, contrariamente ao teste anterior a utilidade do serviço com adaptação foi positiva mais de 50% do tempo de execução. Tal deve-se à diferença de cargas entre os volumes utilizados. Enquanto que o volume de carga 3 é caracterizado por um ritmo de pedidos elevado que coloca constantemente o serviço em sobrecarga o volume 2 é caracterizado a maior parte do tempo por um ritmo de pedidos baixo, período esse em que os recursos disponíveis possibilitam ao serviço *s_medicamento* obter um valor de utilidade positivo.

O próximo caso a analisar tem em conta uma carga de pedidos exigente e sem oscilações consideráveis sob o serviço *s_patologia*. O objectivo deste teste é averiguar se o sistema é capaz de adaptar o serviço para uma configuração mais económica quando a carga corrente assim o permite. Os serviços foram iniciados com recursos diferentes. Ao serviço sem adaptação foi atribuída uma instância com 1 vCPU e 1 GB de memória, enquanto que ao serviço com adaptação foi atribuído uma instância com 0.5 vCPU e 0.5 GB de memória.

Através do gráfico 4.5 é possível visualizar o comportamento esperado. Até ao minuto 10 as utilidades de ambos os serviços são iguais daí em diante o Analisador, tendo em conta o ritmo de pedidos efectuados ao serviço, constata que este não precisa da

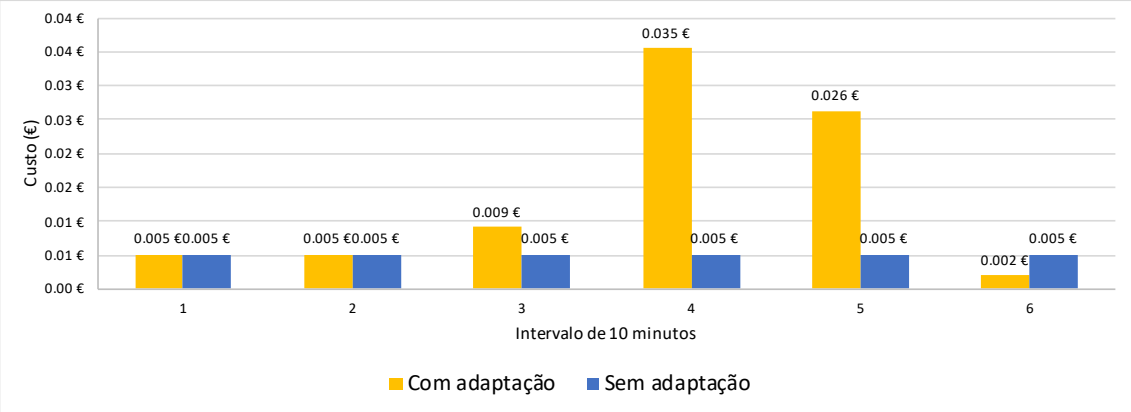

Figura 4.4: Evolução do custo do serviço *s_medicamento* sob o volume de carga 2

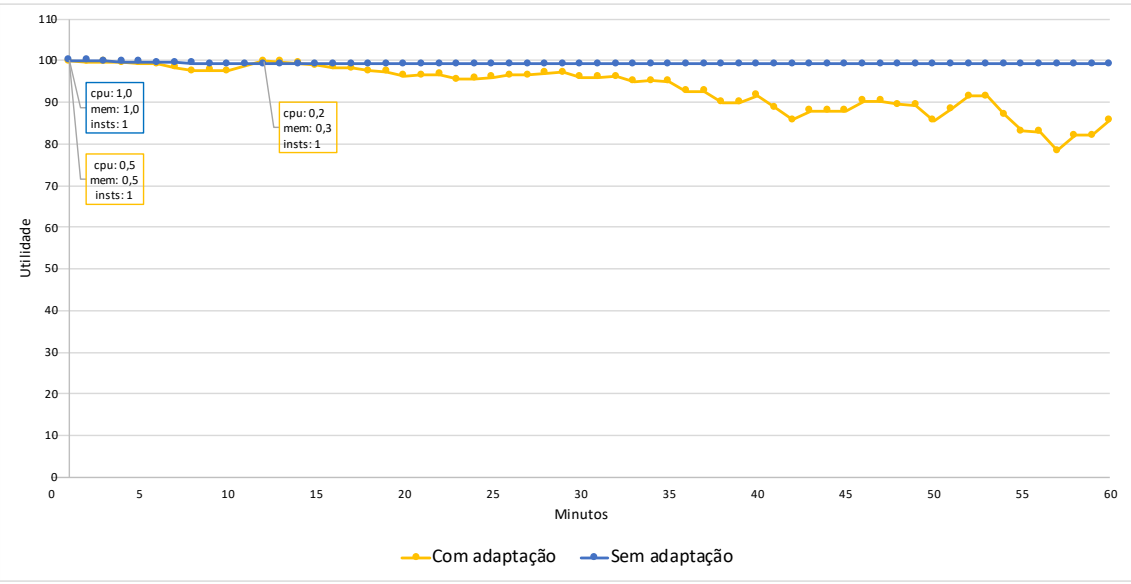

Figura 4.5: Evolução da utilidade do serviço *s_patologia* sob o volume de carga 1

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	1	0.2	0.3

Tabela 4.6: Reconfigurações do serviço *s_patologia* sob o volume de carga 1

totalidade dos recursos que lhe foram atribuídos e minimiza-os. Após a primeira reconfiguração ocorrem mais 4 análises, mas devido ao facto de a utilidade se manter em valores satisfatórios não é realizada nenhuma reconfiguração. É possível visualizar a diminuição da utilidade do serviço com adaptação comparativamente à utilidade do serviço sem adaptação, uma vez que o último possui mais recursos.

O gráfico 4.6 demonstra a evolução do custo do serviço *s_patologia* tendo em conta o volume de carga 1. No presente teste optou-se por alocar mais recursos ao serviço sem adaptação, para evidenciar os ganhos monetários do sistema com adaptação. Tal como

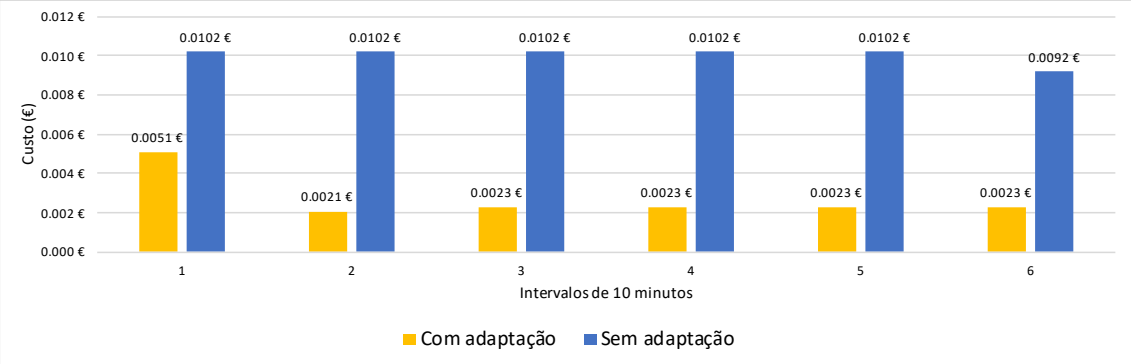

Figura 4.6: Evolução do custo do serviço *s_patologia* sob o volume de carga 1

foi explicado anteriormente existe uma única reconfiguração depois do minuto 10, que diminui o custo da configuração do serviço com adaptação. É possível perceber que o ganho monetário relativamente à diferença de utilidades dos dois serviços é considerável, validando a adaptação correcta do sistema.

Como se trata de um serviço muito eficiente a utilidade permanece elevada ao longo dos 60 minutos pois o mesmo é capaz de processar pedidos muito rapidamente o que possibilita a redução dos custos associados, sem impacto na utilidade.

4.3.1.2 Testes com 2 Serviços

Através dos testes com dois serviços pretende-se averiguar se o sistema continua a conseguir adaptar individualmente os serviços conforme as exigências de cada um e se é capaz de propor configurações distintas para cada serviço. Para os dois exemplos abordados será ainda possível comparar o desempenho dos serviços geridos pelo sistema autonómico e dos mesmos serviços sem adaptação.

O primeiro teste realizado recorre aos serviços *s_medicamento* e *s_utente* ambos com cargas distintas, o primeiro é submetido ao volume de carga 3 e o último ao volume de carga 2. Neste teste todos os serviços foram iniciados com a mesma configuração: uma instância com 0.5 vCPU e 0.5 GB de memória. As análises para este teste foram realizadas de 12 em 12 minutos devido ao limite de instâncias que se podem criar numa hora.

O gráfico 4.7 ilustra o teste referido. Mais uma vez é notória a diferença de desempenho dos serviços com e sem adaptação.

Nos primeiros 12 minutos todos os serviços apresentam valores de utilidade semelhantes, devido às suas configurações serem idênticas e por a carga a que estão sujeitos ser semelhante. Depois dos 12 minutos ocorre a primeira análise e a primeira reconfiguração. Nas tabelas 4.7 e 4.8 estão descriminadas as reconfigurações dos serviços *s_medicamento* e *s_utente* respectivamente.

A primeira reconfiguração é caracterizada pelo aumento dos recursos atribuídos ao serviço *s_utente* devido ao aumento da carga, e à redução dos recursos alocados ao serviço *s_medicamento* devido ao valor de utilidade ser positivo e por forma a reduzir o custo

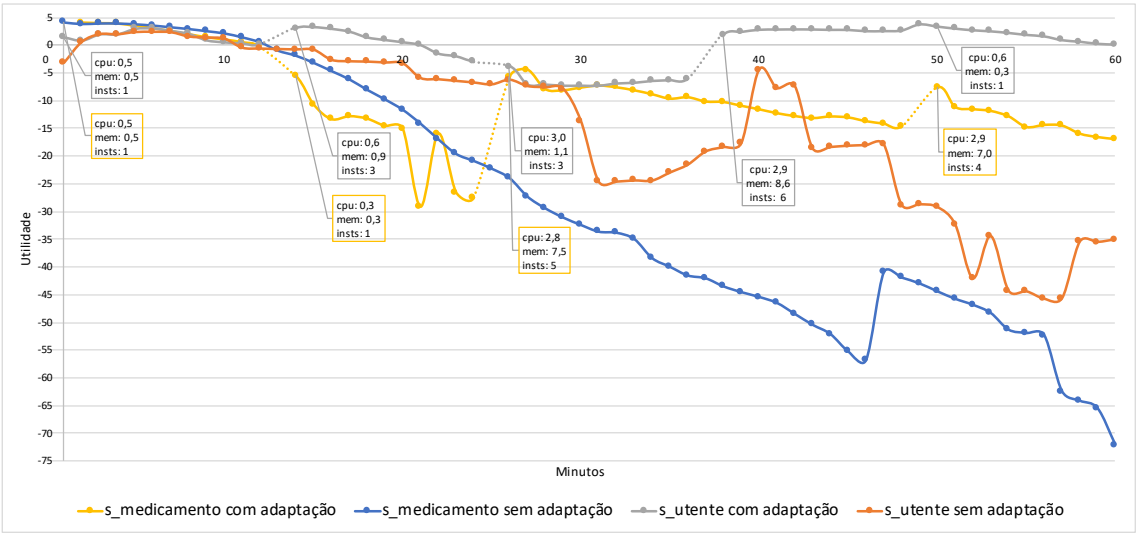

Figura 4.7: Evolução da utilidade dos serviços *s_medicamento* e *s_utente* quando sujeitos a cargas heterogéneas

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	1	0.3	0.3
2ª reconfiguração	5	2.8	7.5
3ª reconfiguração	4	2.9	7.0

Tabela 4.7: Reconfigurações do serviço *s_medicamento* sob o volume de carga 3

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	3	0.6	0.9
2ª reconfiguração	3	3	1.1
3ª reconfiguração	6	2.9	8.6
4ª reconfiguração	1	0.6	0.3

Tabela 4.8: Reconfigurações do serviço *s_utente* sob o volume de carga 2

associado.

Até à segunda análise que ocorre aos 24 minutos percebe-se que a reconfiguração proposta para o serviço *s_medicamento* teve um impacto negativo no seu desempenho, pois os recursos atribuídos não são suficientes para fazer face ao aumento constante da carga, resultando na deterioração considerável da sua utilidade sendo esta inclusive, inferior à do serviço sem adaptação. Por outro lado o serviço *s_utente* tem um desempenho satisfatório, uma vez que os valores de utilidade são positivos e superiores ao serviço sem adaptação.

Durante a segunda reconfiguração são aumentados consideravelmente os recursos atribuídos a ambos os serviços, ao serviço *s_medicamento* por forma a reverter a descida acentuada de utilidade e ao serviço *s_utente* devido ao aumento brusco da carga. Após a referida reconfiguração a utilidade do serviço *s_medicamento* melhora drasticamente

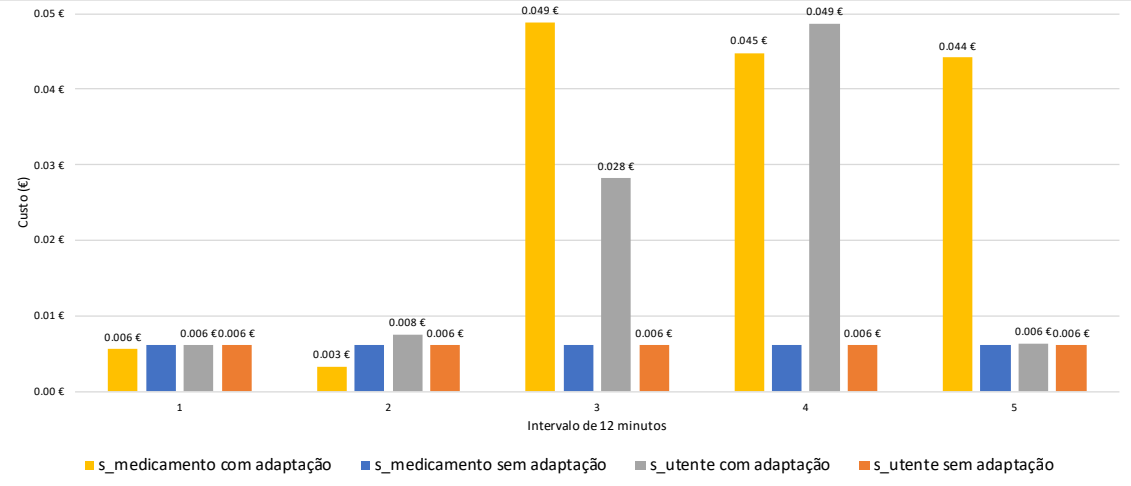

Figura 4.8: Evolução do custo dos serviços *s_medicamento* e *s_utente* sob cargas heterogéneas

e é agora muito superior ao serviço sem adaptação. O serviço *s_utente* sofre uma ligeira diminuição no seu valor de utilidade contudo até à próxima análise a utilidade mantém-se constante não deteriorando muito a qualidade de serviço durante o período de maior carga como pretendido.

Nos restantes minutos ocorrem mais duas análises e mais três reconfigurações no total. A terceira reconfiguração ocorre por volta dos 36 minutos e só os recursos do serviço *s_utente* são alterados devido ao período que antecedeu a reconfiguração caracterizado pelo valor negativo da utilidade.

A última reconfiguração é realizada aos 48 minutos e as configurações de ambos os serviços são alteradas. Ao serviço *s_medicamento* é atribuído mais 0.1 de vCPU e reduzido 0.5 GB de memória por forma a reduzir o seu custo, uma vez que a utilidade associada à nova configuração é semelhante à anterior só que mais económica. O serviço *s_utente* por outro lado sofre uma redução considerável dos seus recursos por estes não se justificarem face a carga corrente.

O gráfico 4.8 demonstra o custo associado à execução do teste em análise. No primeiro intervalo de 12 minutos o custo de todos os serviços é igual. No intervalo seguinte devido à primeira reconfiguração os custos dos serviços com adaptação são diferentes, nomeadamente, devido ao aumento do número de pedidos ao serviço *s_utente* este apresenta o maior custo dos 4 serviços contrariamente ao serviço *s_medicamento* que apresenta o menor custo fruto da redução dos recursos base.

Nos dois intervalos seguintes, devido ao elevado número de recursos atribuídos aos serviços com adaptação o custo dos mesmos é consideravelmente superior aos serviços sem adaptação. No quinto intervalo o custo do serviço *s_utente* é de novo muito baixo consequência do número reduzido de recursos atribuídos, por outro lado o serviço *s_medicamento* continua com um custo elevado embora menor do que no intervalo anterior.

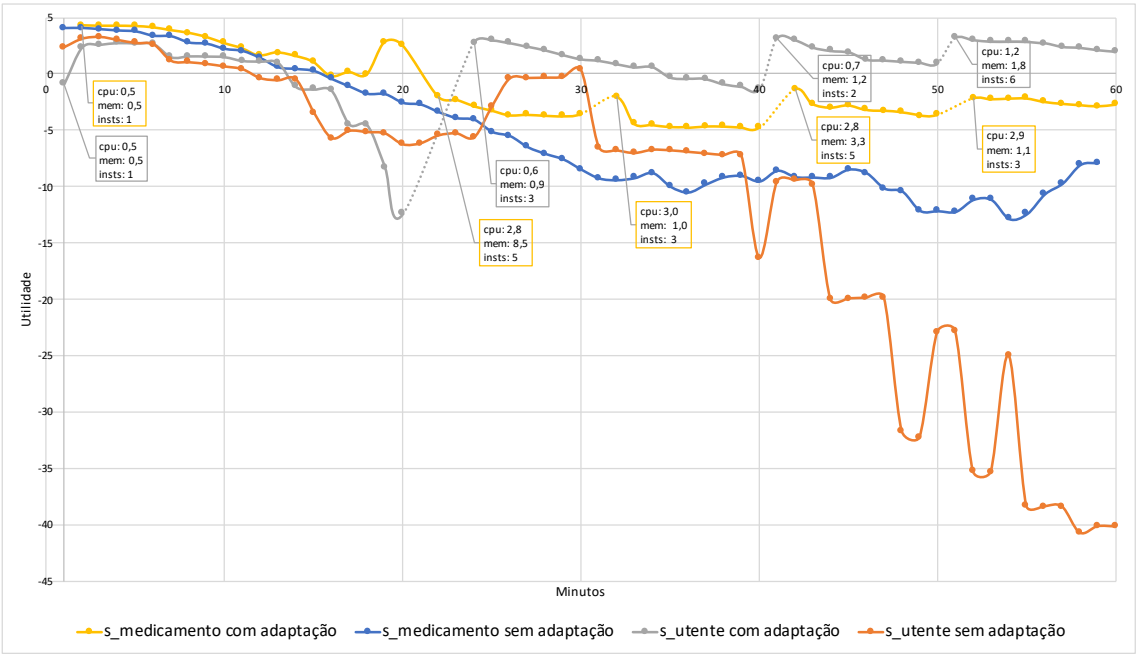

Figura 4.9: Evolução da utilidade dos serviços *s_medicamento* e *s_utente* quando sujeitos a cargas homogéneas

Com este teste evidenciou-se uma vez mais o correcto desempenho do sistema autonómico e os ganhos de utilidade face ao sistema tradicional. Embora o valor de utilidade do serviço *s_medicamento* não tenha sido o desejado, foi o melhor possível com os recursos disponíveis, por outro lado a adaptação do serviço *s_utente* revelou-se eficiente tendo este apresentado valores de utilidade positivos e muito superiores ao serviço sem adaptação. De referir a situação ocorrida no serviço *s_medicamento* antes da segunda adaptação (entre os minutos 18 e 24), onde devido à periodicidade das análises e à redução dos recursos, consequência do bom desempenho da fase anterior, o desempenho da fase seguinte é deficiente pois o serviço não tem recursos suficientes para suportar a carga. Seria interessante incluir uma capacidade de previsão e alterar a análise periódica por uma solução mais fidedigna.

O próximo teste tem como objectivo submeter os dois serviços anteriores a um volume de carga semelhante e perceber que tipo de adaptação ocorre. Neste teste foi utilizado o volume de carga 1 e todos os serviços foram iniciados com os mesmos recursos: uma instância com 0.5 vCPU e 0.5 GB de memória. Por ser um teste onde potencialmente serão criados poucos contentores, devido à carga constante, a análise aos serviços foi efectuada de 10 em 10 minutos contrariamente ao teste anterior, onde o limite de criação de 60 contentores por hora poderia ser facilmente ultrapassado. O resultado do teste é demonstrado no gráfico 4.9.

Até ocorrer a primeira análise a utilidade dos serviços é semelhante devido à semelhança dos recursos e carga. Quando ocorre a primeira análise tanto o serviço *s_utente*

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
1ª reconfiguração	5	2.8	8.5
2ª reconfiguração	3	3.0	1.0
3ª reconfiguração	5	2.8	3.3
4ª reconfiguração	3	2.9	1.1

Tabela 4.9: Reconfigurações do serviço *s_medicamento* sob o volume de carga 1

Reconfigurações	Instâncias	vCPU (total)	Memória (total)
2ª reconfiguração	3	0.6	0.9
3ª reconfiguração	2	0.7	1.2
5ª reconfiguração	6	1.2	1.8

Tabela 4.10: Reconfigurações do serviço *s_utente* sob o volume de carga 1

como o serviço *s_medicamento* não são reconfigurados uma vez que o seu valor de utilidade é positivo, embora não muito superior a zero, o que provoca a manutenção das suas configurações anteriores.

Devido ao aumento da carga, até à segunda análise a utilidade de ambos os serviços deteriora-se, especialmente a do serviço *s_utente* consequência do número médio de pedidos por instância ser elevado.

Quando ocorre a primeira reconfiguração o serviço *s_utente* sofre um ligeiro aumento do número de recursos enquanto que os recursos do serviço *s_medicamento* são aumentados drasticamente. Este aumento ocorre devido ao crescimento do número de pedidos e à consequente deterioração do valor das métricas. Contudo, a atribuição de recursos poderia ter sido mais gradual e não o foi devido a dois factores: ao facto de existirem poucos valores no intervalo de dados utilizado para o cálculo da utilidade óptima da nova configuração, e devido à parametrização do Analisador nomeadamente, ao factor de suavização utilizado para calcular a utilidade da configuração proposta tendo em conta a utilidade das configuração existentes na Base de Dados (ver fórmulas 3.13 a 3.17). Estas situações ocorrem com frequência durante a fase inicial devido à falta de dados existentes na base de dados de Desempenho, entidade onde o Analisador se baseia para propor configurações.

Durante a terceira análise os recursos do serviço *s_medicamento* são consideravelmente diminuídos por forma a reduzir o custo. O serviço *s_utente* não sofre alterações devido ao bom desempenho da utilidade.

Aos 40 minutos ocorre a terceira análise e ambos os serviços sofrem um aumento do número de recursos consequência da deterioração do valor de utilidade. Na última reconfiguração os recursos do serviço *s_medicamento* são reduzidos para controlar o custo e porque através do conhecimento armazenado na Base de Dados de Desempenho o Analisador sabe que com a carga vigente é possível ter valores de utilidade semelhantes com uma configuração mais económica. Por fim são atribuídos mais recursos ao serviço

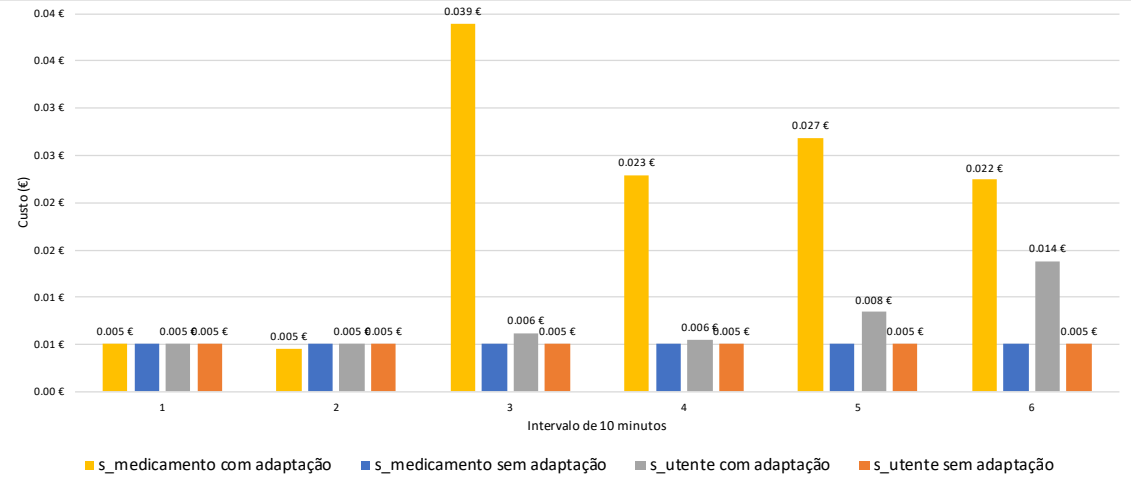

Figura 4.10: Evolução do custo dos serviços *s_medicamento* e *s_utente* quando sujeitos a cargas homogéneas

s_utente para fazer face ao aumento constante do número médio de pedidos por instância.

No final do teste é possível concluir que a adaptação produziu melhores resultados do que os serviços sem adaptação. Embora a única utilidade verdadeiramente satisfatória tenha sido a do serviço *s_utente* uma vez que é positiva, a utilidade do serviço *s_medicamento* é a melhor possível tendo em conta os recursos disponíveis.

O gráfico 4.10 demonstra a evolução do custo associado à execução dos serviços em análise e reflecte o aumento e diminuição dos recursos alocados aos diferentes serviços ao longo do teste.

É possível concluir com os testes anteriores que a adaptação com múltiplos serviços apresenta desempenhos consideravelmente superiores aos serviços sem adaptação, confirmando uma vez mais o correcto funcionamento do sistema. No entanto importa relembrar que devido à periodicidade das análises, o sistema fica sujeito a situações semelhantes como a que ocorreu entre os 18 e 24 minutos do teste com cargas heterogéneas, onde devido à redução dos recursos pelo bom desempenho da fase anterior o desempenho da fase seguinte é deficiente, pois o serviço não tem recursos suficientes para suportar a carga.

Outro facto importante diz respeito ao valor de utilidade do serviço *s_medicamento* ser sempre negativo mesmo com adaptação, tal deve-se ao número máximo de recursos possíveis de alocar que é claramente insuficiente para suportar as cargas utilizadas, sendo importante alargar as instâncias de um serviço a mais do que um Pod.

Resultados melhores poderão não ter sido alcançados devido a outros factores e elementos envolvidos no processo de adaptação, que porventura condicionaram os valores obtidos. Na arquitectura actual do Pod de Serviço existe apenas um Proxy, que devido ao volume de dados gerado, poderá não ter capacidade de distribuir todos os pedidos pelas várias instâncias do serviço. Uma solução seria realizar uma adaptação das instâncias do proxy consoante a carga deste.

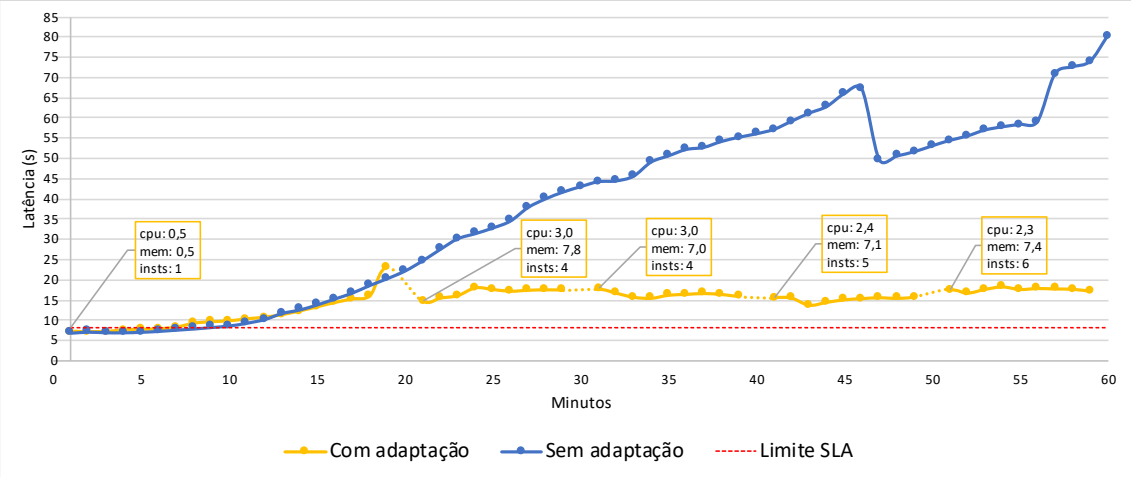

Figura 4.11: Evolução da latência do serviço *s_medicamento* sob o volume de carga 3

Outro factor importante a referir tem que ver com o facto de as reconfigurações estarem restringidas a um limite de instâncias e de recursos, significando isto que o nível de utilidade máximo consoante diferentes cargas está restringido à partida. Como os testes foram realizados em momentos diferentes do dia e em dias diferentes não é possível garantir que a disponibilidade da infraestructura onde os *Pods* de serviço estão alojados (Azure) é idêntica, o que pode também justificar oscilações nos resultados obtidos.

4.3.2 Evolução da Latência

A latência de um serviço é calculada através da contabilização do tempo que decore desde a recepção de um pedido até à sua resolução. Esta métrica está profundamente relacionada com a qualidade de serviço dos utilizadores finais e por isso mesmo é essencial demonstrar o seu desempenho tendo em conta os diferentes volumes de carga utilizados. O principal objectivo com a presente secção consiste em comprovar que a adaptação realizada pelo sistema autonómico é capaz de melhorar a latência dos serviços em comparação com o que seria expectável sem adaptação. Os resultados aqui demonstrados têm em conta os testes realizados na secção 4.3.1.1.

4.3.2.1 Testes com 1 Serviço

O primeiro teste é referente ao serviço *s_medicamento* e ao volume de carga 3. Como é possível observar no gráfico 4.11 a latência do serviço sem adaptação deteriora-se com o aumento do número de pedidos efectuados, contrariamente ao serviço com adaptação que devido às sucessivas reconfigurações (consultar a tabela 4.4) consegue manter uma latência constante e consideravelmente inferior. No entanto como foi referido na secção anterior, o valor de latência obtido é constantemente superior ao limite definido no SLA, tal deve-se à insuficiência de recursos e instâncias disponíveis para fazer face à carga do serviço.

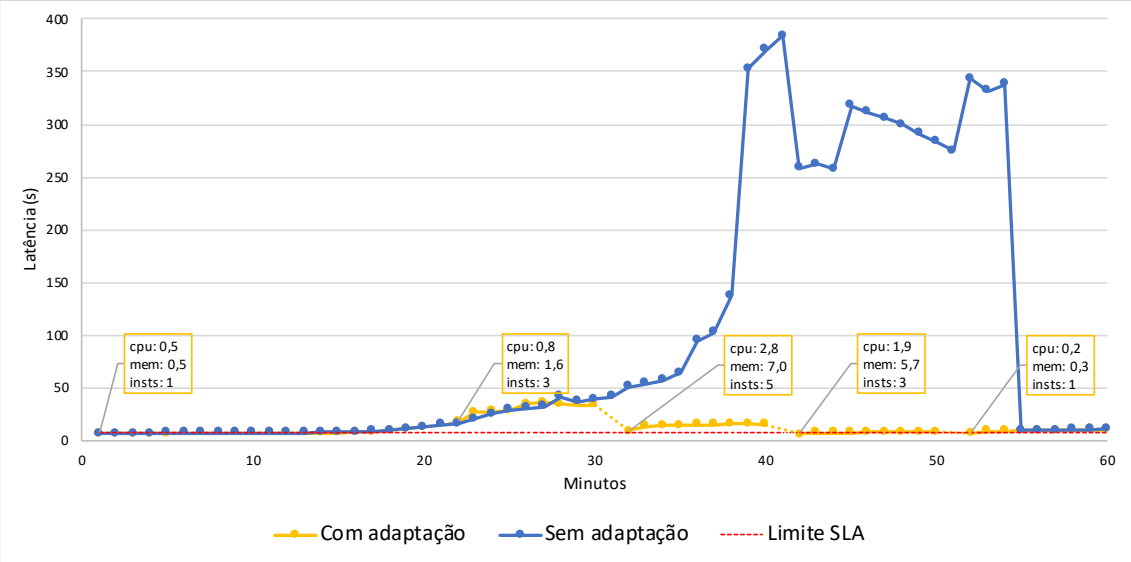

Figura 4.12: Evolução da latência do serviço *s_medicamento* sob o volume de carga 2

O segundo teste considera novamente o serviço *s_medicamento*. O volume de carga associado a este teste consistia no aumento brusco do número de pedidos. Como é possível constatar através do gráfico 4.12, a latência de ambos os serviços é semelhante até ao minuto 30, altura em que ocorre a reconfiguração do serviço com adaptação. Nos minutos seguintes percebe-se que a latência do serviço sem adaptação deteriora-se drasticamente chegando a valer 40 vezes mais do que a latência do serviço com adaptação, que se mantém constante e regulada de acordo com o limite SLA definido.

Por fim o último caso analisado tendo em conta um só serviço, diz respeito ao serviço *s_patologia* sob o volume de carga 1. Neste caso em concreto, como o objectivo inicial do teste era comprovar que o sistema conseguiria reduzir os recursos do serviço se a sua utilidade assim o permitisse, a latência do serviço com adaptação é consideravelmente superior à do serviço sem adaptação que dispõe de mais recursos. No entanto a latência permanece abaixo do limite definido no SLA (consultar tabela 4.3) e tem um custo bastante inferior ao serviço sem adaptação (consultar gráfico 4.6).

4.3.2.2 Testes com 2 Serviços

O primeiro teste de latência com dois serviços é demonstrado no gráfico 4.14 e tem em conta os serviços *s_medicamento* e *s_utente*, ambos sob cargas heterogéneas nomeadamente o volume de carga 3 e o volume de carga 2.

De uma forma geral a latência dos serviços com adaptação é consideravelmente inferior à dos serviços sem adaptação. No caso específico do *s_medicamento* a latência do serviço com adaptação só é superior entre o minuto 18 e 24 devido à alocação de poucos recursos na primeira reconfiguração o que causou um desempenho inferior comparativamente ao serviço sem adaptação. No que diz respeito ao valor limite acordado no SLA do *s_medicamento* para a a métrica latência é possível visualizar que o mesmo só é respeitado

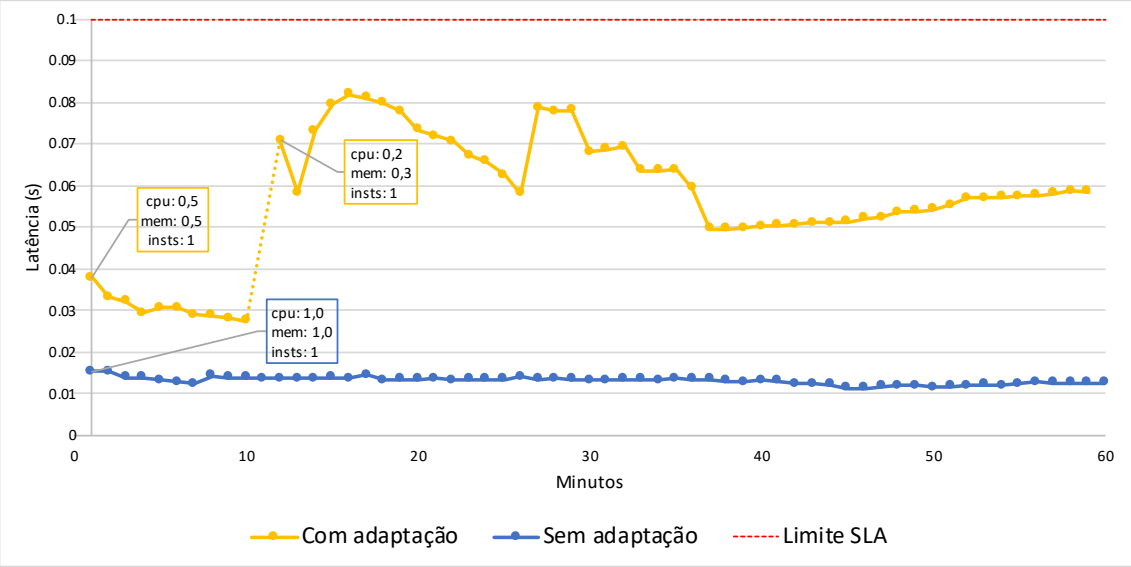

Figura 4.13: Evolução da latência do serviço *s_patologia* sob o volume de carga 1

nos primeiros 9 minutos, nos minutos seguintes o valor obtido é consideravelmente superior. Tal deve-se, como já foi explicado anteriormente, ao facto de não existirem recursos e instâncias disponíveis suficientes que possibilitem a obtenção de um valor melhor.

O serviço *s_utente* para além de apresentar um desempenho igualmente satisfatório comparativamente ao serviço sem adaptação, que aquando do pico de carga vê a sua latência crescer exponencialmente, obtém também valores satisfatórios em relação ao limite SLA acordado, sendo apenas superior ao mesmo durante o pico de carga, muito embora com uma diferença muito inferior à do serviço sem adaptação.

Com este teste conclui-se que o sistema autonómico tem um desempenho superior ao sistema antigo, conseguindo adaptar-se às cargas a que fica sujeito por forma a minimizar a latência dos serviços. Contudo no que diz respeito aos valores acordados no SLA não é capaz de permanecer abaixo dos mesmos, devido aos poucos recursos e instâncias que dispõe no momento de reconfiguração dos serviços

O último teste da presente secção foi efectuado recorrendo aos serviços do teste anterior, mas utilizando volumes de carga homogéneos mais concretamente o volume de carga 1.

Os resultados obtidos são demonstrados no gráfico 4.15. É possível constatar que os valores de latência do serviço *s_utente* são semelhantes com e sem adaptação, embora ambos apresentem configurações e custos muito diferentes (consultar respectivamente os gráficos 4.9 e 4.10). Tal deve-se aos factores considerados durante o cálculo do novo estado, isto é, se o único elemento considerado para o cálculo da utilidade fosse a latência, o custo do serviço com adaptação seria muito menor, pois este com menos recursos também conseguiria atingir o mesmo nível de latência, à semelhança do serviço sem adaptação. A justificação para o custo elevado do serviço com adaptação prende-se com o facto da utilidade ser composta por outras métricas e uma em particular os *pedidos médios por instância*,

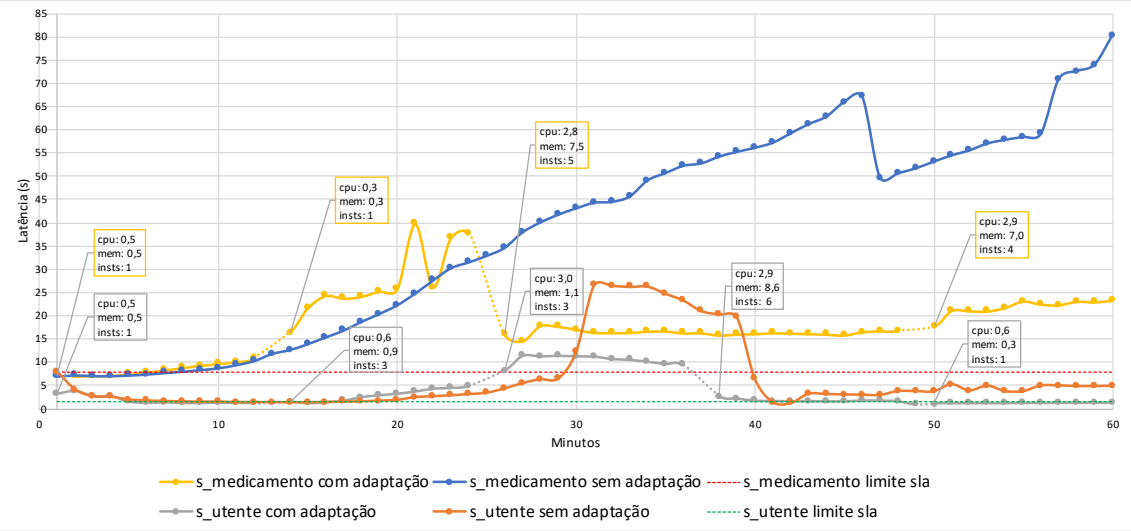

Figura 4.14: Evolução da latência dos serviços *s_medicamento* e *s_utente* sob cargas heterogéneas

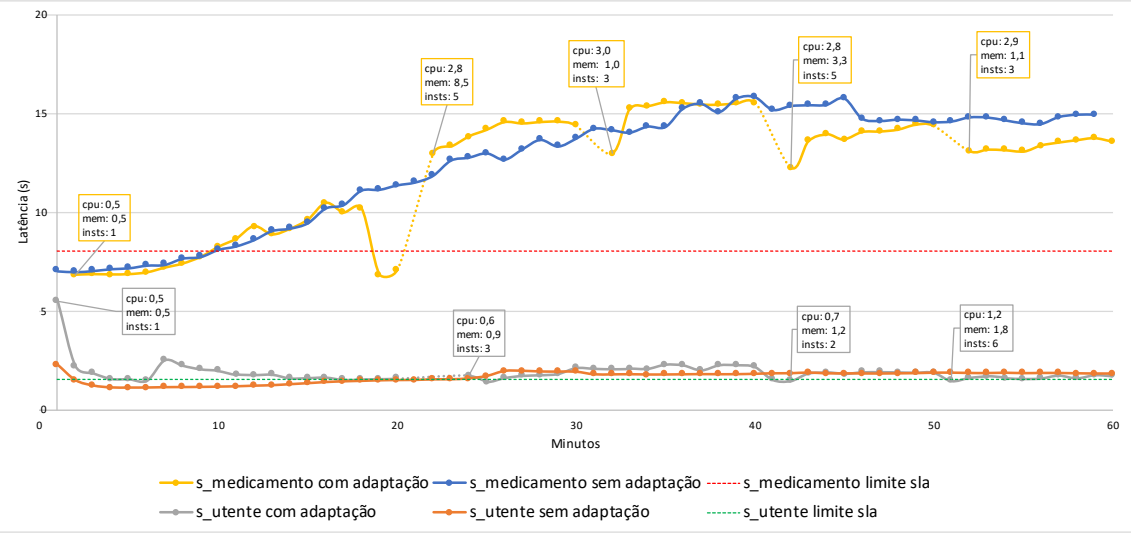

Figura 4.15: Evolução da latência dos serviços *s_medicamento* e *s_utente* sob cargas homogéneas

econtrar-se muito acima do limite definido, no serviço sem adaptação, muito embora a latência tenha permanecido constante, sendo este o factor que promove a reconfiguração do serviço com adaptação e aumenta o seu custo.

A latência do serviço *s_medicamento* com adaptação apresenta um comportamento bastante irregular. Nos primeiros vinte minutos, exceptuando o minuto 19 e 20, a latência de ambos os serviços cresce com o aumento do número de pedidos. Com a primeira reconfiguração são alocados mais recursos ao serviço com adaptação por forma a contrariar o aumento da latência como seria expectável, contudo o valor da latência cresce a um ritmo consideravelmente superior ao do serviço sem adaptação apesar das instâncias alocadas e do maior número de recursos. Uma possível explicação para este comportamento poderá

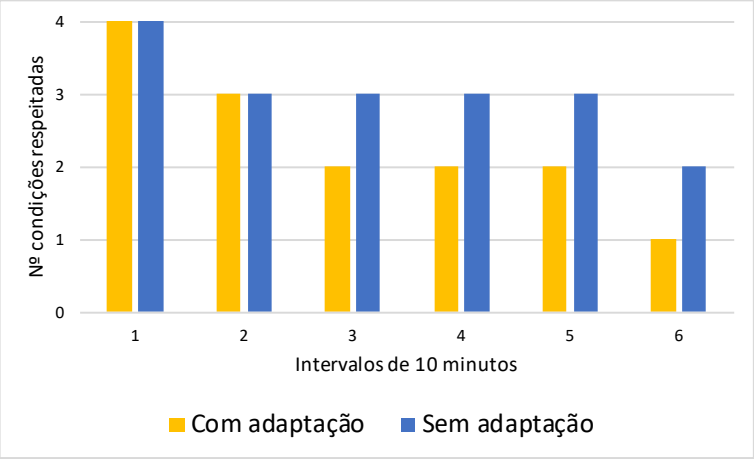

Figura 4.16: Evolução do cumprimento do SLA do serviço *s_medicamento* sob o volume de carga 3

relacionar-se com a instabilidade na plataforma Azure no período em que o teste foi realizado. No intervalo seguinte apesar do número de instâncias e recursos ser superior aos do serviço sem adaptação o valor de latência continua ligeiramente superior ao do serviço sem adaptação. Após a penúltima reconfiguração a latência do serviço com adaptação é finalmente inferior à do serviço sem adaptação e o mesmo se verifica de uma forma mais preponderante depois da última reconfiguração.

4.3.3 Cumprimento dos SLAs

Nesta secção é analisada a evolução do cumprimento dos SLAs definidos para cada serviço tendo em conta os cenários testados anteriormente. Através dos resultados aqui apresentados será possível perceber até que ponto os SLAs definidos são realistas face às exigências propostas e qual o desempenho do sistema autonómico no cumprimento dos SLAs.

O primeiro cenário a ter em conta compreende o serviço *s_medicamento* sob o volume de carga 3. O cumprimento dos SLAs é ilustrado no gráfico 4.16. É possível constatar que o número de condições respeitadas pelo serviço sem adaptação é superior ao serviço com adaptação. Tal deve-se às métricas *indisponibilidade* e *taxa de transações falhadas* que no caso específico do s_medicamento sempre que ocorre uma reconfiguração são consideravelmente afectadas.

Como foi referido a utilidade do serviço *s_medicamento* é maioritariamente negativa ao longo dos testes, tal facto é reflectido no gráfico em análise, onde após os 10 minutos iniciais a condição relativa à latência nunca é respeitada.

O gráfico 4.17 ilustra o serviço *s_medicamento* sob o volume de carga 2. Constata-se novamente que o serviço sem adaptação tem o maior número de condições respeitadas, devido ao impacto da reconfiguração nas métricas *indisponibilidade* e *taxa de transações*

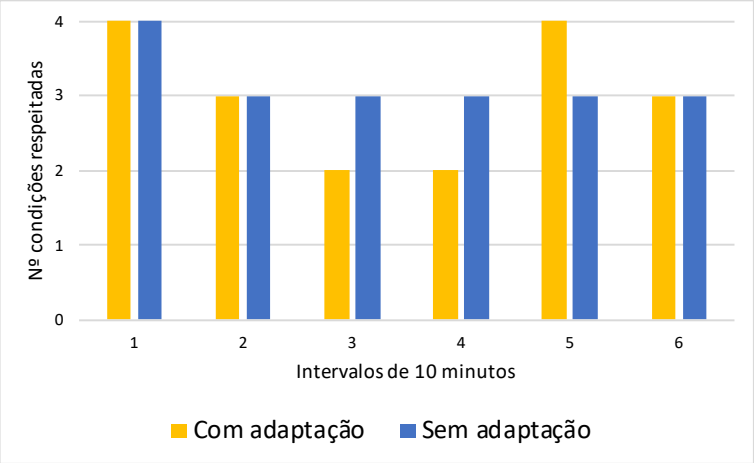

Figura 4.17: Evolução do cumprimento do SLA do serviço *s_medicamento* sob o volume de carga 2

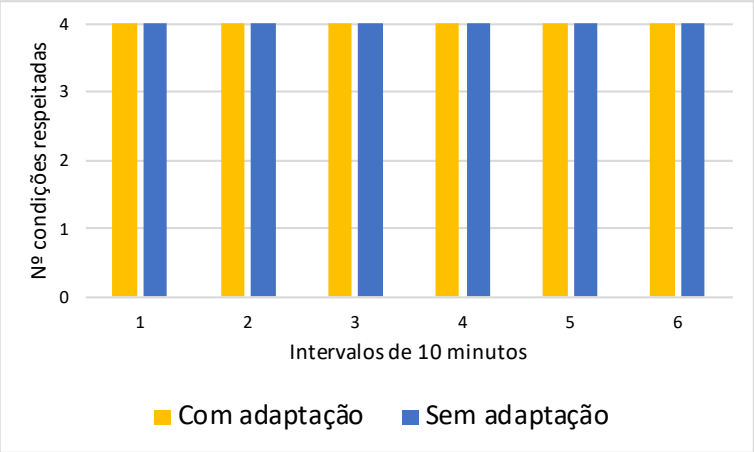

Figura 4.18: Evolução do cumprimento do SLA do serviço *s_patologia* sob o volume de carga 1

falhadas. No ponto 5 contudo, o serviço com adaptação respeita todas as condições estipuladas devido à capacidade da configuração proposta suportar a carga vigente e com isso possibilitar a obtenção do valor desejado de latência.

O próximo gráfico é referente ao serviço *s_patologia*. Como é possível observar todas as condições definidas foram respeitadas para ambos os serviços. Tal deve-se à eficiência do serviço em processar pedidos fazendo com que a carga a que foi sujeito não causa-se qualquer impacto na correcta operação do mesmo, contrariamente aos serviços *s_medicamento* e *s_utente*.

Os últimos dois gráficos são referentes aos testes efectuados com dois serviços em simultâneo. O gráfico 4.19 apresenta os resultados obtidos com os serviços *s_medicamento* e *s_utente* sob os volumes de carga 3 e 2 respectivamente. À semelhança dos gráficos analisados anteriormente o serviço *s_medicamento* com adaptação respeita um número menor de condições do que o serviço sem adaptação consequência das reconfigurações do

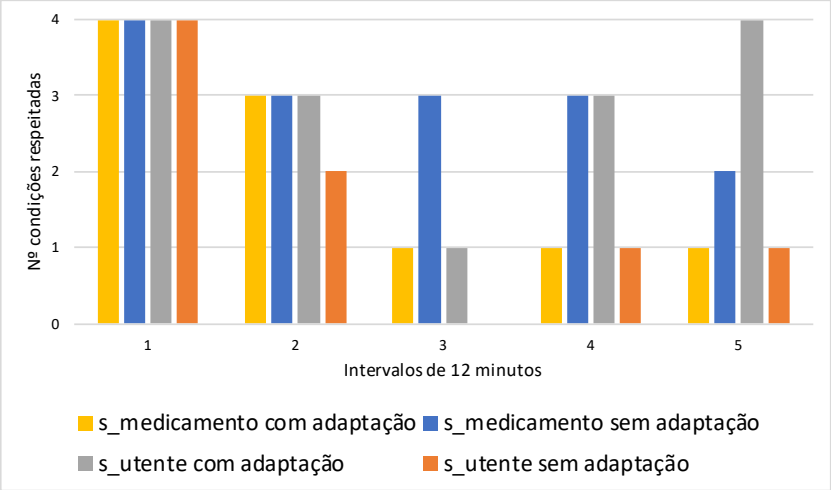

Figura 4.19: Evolução do cumprimento dos SLAs dos serviços *s_medicamento* e *s_utente* sob cargas heterogéneas

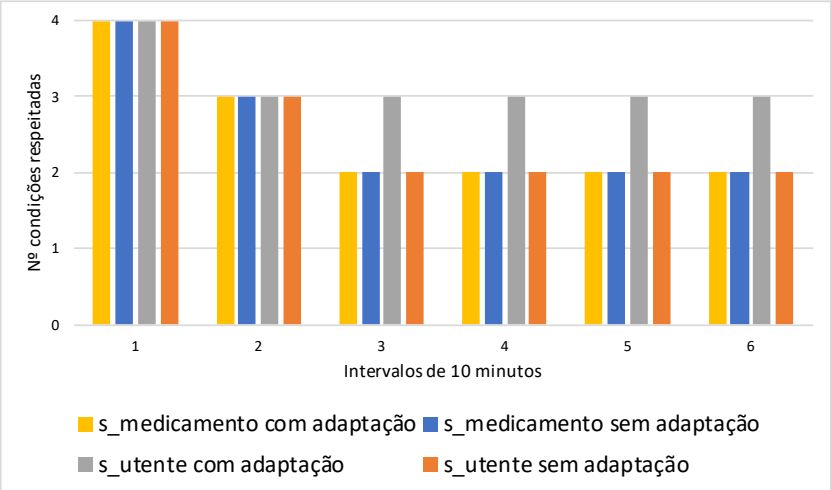

Figura 4.20: Evolução do cumprimento dos SLAs dos serviços *s_medicamento* e *s_utente* sob cargas homogénea

mesmo. Por outro lado o serviço *s_utente* com adaptação tem um desempenho superior ao serviço sem adaptação, com especial destaque para o ponto 3 onde nenhuma condição estabelecida foi respeitada pelo serviço sem adatpação.

O último teste com dois serviços em simultâneo é referente ao volume de carga 1 sob os serviços anteriores. Todos os serviços de uma forma geral têm um desempenho mediano respeitando em média 2 slas. Destaque para o serviço *s_utente* que apresenta um desempenho superior.

Os resultados analisados nesta secção permitiram visualizar o número de condições respeitadas nos SLAs durante a execução dos vários testes. Através dos gráficos obtidos é possível concluir que o valor definido para as métricas *indisponibilidade* e *taxa de transações falhadas* no caso concreto do serviço *s_medicamento* necessita de ser alterado, pois

o impacto da reconfiguração nas referidas métricas é considerável, muito embora o desempenho do serviço com adaptação seja superior ao do serviço sem adaptação, o que através da visualização dos valores das métricas anteriores poderá não ser perceptível. É também possível concluir que a avaliação do número de condições respeitadas por si só não é insuficiente para avaliar o desempenho do sistema, pois considerando o exemplo da latência existem diferentes níveis de "respeito", podendo o valor de latência de um serviço encontrar-se muito próximo ao limite definido enquanto que o valor da mesma métrica de outro serviço ser consideravelmente inferior e os mesmos serem avaliados com o mesmo desempenho por ambos respeitarem o mesmo limite.

Por fim a definição de SLAs com valores bem fundamentados é essencial para o correcto desempenho do sistema e para a correcta avaliação dos serviços.

4.4 Conclusão

Através dos vários testes realizados ao longo deste capítulo foi possível comprovar que o sistema desenvolvido melhora o desempenho global dos serviços significativamente e de uma forma dinâmica. Quer nos testes de utilidade como nos de latência destacou-se o desempenho superior dos serviços com adaptação autonómica, tal deve-se à incapacidade dos serviços sem adaptação conseguirem aumentar os recursos atribuídos quando se dá um aumento de carga.

Não só foi possível verificar o desempenho superior em situações de carga elevada como também, perceber que quando a carga é diminuta e não requer muitos recursos o sistema autonómico possui a sensibilidade, de reduzir os recursos atribuídos aos serviços e assim diminuir o impacto monetário dos mesmos, contrariamente aos serviços sem adaptação que apresentam o mesmo custo independentemente da carga vigente.

Contudo existem pontos a considerar, a adaptação ocorre periodicamente o que não é benéfico a longo prazo, pois um aumento de carga inesperado entre um intervalo de análises implicará a deterioração do serviço e o sistema nada poderá fazer para alterar esta situação. Cada reconfiguração implica em média um minuto de inactividade o que actualmente não é critico pois o número de requisições real dos serviços é tremendamente inferior aos testes aqui realizados, no entanto é algo a ter em conta.

Apesar dos pontos anteriormente referidos o sistema comportou-se como era expectável e poderá ser uma mais valia para a SPMS no que diz respeito ao controlo dos serviços que suportam as aplicações.

Capítulo 5

Conclusão e Trabalho Futuro

Neste último capítulo é apresentada a conclusão do trabalho desenvolvido e são descritas algumas das melhorias e adições que farão sentido para a evolução futura do sistema.

5.1 Conclusão

Com a expansão das soluções oferecidas pela SPMS e com o previsto aumento de utilizadores que interagem com as referidas soluções, será necessário reorganizar a infraestructura que suportará todo esse crescimento.

Esta reorganização não deve ser baseada na atribuição desmedida de recursos para que nada falhe, pois deste modo não só estará a ser desperdiçado dinheiro, como uma boa oportunidade para melhorar de forma inteligente a organização e arquitectura de sistemas já ultrapassados.

Foi com base na motivação anterior que o projecto agora desenvolvido, surgiu. A presente dissertação consistiu no desenvolvimento de um sistema capaz de adaptar autonomamente os serviços que o compõem, por forma a maximizar a utilidade global e assim melhorar a qualidade de serviço do utilizador final. Para tal, o sistema baseia-se no modelo MAPE-K que define os elementos essenciais para que possa ocorrer o fluxo de adaptação autonómico. A um nível mais prático o sistema baseia-se em Pods compostos por Contentores. Foram desenhados e implementados dois Pods específicos para esta solução, o Pod Autonómico que coordena a adaptação autonómica e o Pod de Serviço criado por cada serviço a monitorizar.

Embora o sistema tenha sido testado e implementado através da plataforma Azure, foi desenhado de maneira a ser modular, baseando-se em Contentores não dependendo de um Provedor de Recursos em específico para funcionar, sendo apenas necessário implementar o componente de ligação entre o Executor e o Provedor de Recursos a integrar.

No que diz respeito aos objectivos propostos na secção 1.2, todos foram endereçados. O primeiro problema mencionado tinha que ver com o facto de não ser possível definir métricas para o controlo dos serviços, através do sistema desenvolvido é agora possível especificar as métricas alto nível a serem consideradas no processo de adaptação autonómico dos serviços, significando não só que a qualidade de serviço é agora considerada mas também que é possível personalizar o significado da mesma para cada serviço, através da combinação de diferentes métricas alto nível.

O segundo ponto a endereçar consistia na forte dependência de Máquinas Virtuais e consequentemente do Provedor de Recursos para a gestão dos serviços. A solução desenvolvida não recorre a MVs, baseando-se em Pods e em Contentores que potenciam a independência do sistema de quaisquer Provedores de Recursos. Por forma a aumentar ainda mais esta independência, o Pod Autonómico possui todos os elementos necessários para gerir os restantes Pods não dependendo, de forma quase absoluta, de terceiros para o fazer.

O terceiro problema identificado relacionava-se com a falta de detalhe das métricas existentes. No sistema desenvolvido foram definidas várias métricas baixo nível sobre a execução dos serviços (consultar tabela 3.1) e quatro métricas alto nível: a taxa de transações falhadas, indisponibilidade, latência e taxa média de pedidos por instância. Para facilitar ainda mais a monitorização e controlo do sistema o Monitorizador integrado oferece uma dashboard que permite a consulta de todas as métricas expostas, de uma forma clara e interactiva.

O quarto problema referia-se ao facto da gestão dos serviços ser realizada de forma isolada sem considerar os restantes serviços e as ligações entre eles. O sistema desenvolvido resolve em parte o problema, pois durante o processo de adaptação e análise é considerado o valor de utilidade do sistema composto pelo valor de utilidade dos vários serviços. Desta forma são tidos em conta todos os serviços, e o estado de cada um é importante para melhorar o estado global. No que diz respeito às relações entre serviços, não foi implementado nenhum mecanismo que tenha em conta este factor importante, embora seja algo a considerar no futuro.

O último ponto tinha que ver com a dependência de intervenção humana para a gestão das entidades que suportam os contentores. O sistema desenvolvido não resolve o problema na totalidade, pois ainda é necessário fazer um primeiro deploy manual, aquando da adição de um novo serviço, mas a partir do momento em que existem Pods de Serviço o sistema é capaz de controlar de forma autónoma os recursos alocados a cada um.

Tendo em conta os problemas endereçados, o sistema desenvolvido vai de encontro às expectativas iniciais, oferecendo as funcionalidades essenciais e resolvendo a maioria dos problemas inicialmente identificados.

5.2 Trabalho futuro

Como foi descrito na secção anterior a solução desenvolvida oferece as funcionalidades essenciais inicialmente definidas, conseguido adaptar os serviços por forma a maximizar a sua utilidade minimizando os custos. No entanto muitas melhorias e adições poderão ser realizadas por forma a aumentar a utilidade futura do sistema. Esta secção encontra-se dividida em três subsecções, a primeira identifica algumas das melhorias e adições prioritárias, na segunda subsecção é explicado como poderá ser realizada a integração e implementação da solução desenvolvida na SPMS e por fim, na última secção são identificados os tópicos que devido à sua complexidade poderão ser relevantes para uma investigação futura.

5.2.1 Melhorias e novas funcionalidades

São agora listadas algumas das melhorias e novas funcionalidades a realizar:

- Permitir que as instâncias de um serviço não estejam limitadas a um só Pod, isto é, no momento de reconfiguração devem ser criados tantos Pods quantos aqueles que forem necessários e possíveis, por forma a melhorar o desempenho do serviço. Esta funcionalidade implicará entre outras alterações, um Proxy capaz de redirecionar os pedidos entre Pods do mesmo serviço e um registo de IPs e de estado dos Pods para averiguar o seu estado.

- Armazenar os IPs dos vários serviços activos numa cache. Actualmente já são considerados os IPs, mas a forma de obtenção dos mesmos é ainda dependente do Provedor de Recursos. Para uma solução capaz de funcionar em simultâneo com vários Provedores de Recursos é necessário controlar os IPs através do Pod Autonómico.

- Armazenar os SLAs dos serviços activos numa cache, pois sempre que os Gestores dos Pods de Serviço guardam o estado dos seus serviços necessitam de comunicar com o Gestor de Slas, o que pode não ser escalável. Através da cache seria possível reduzir o número de pedidos efectuados ao Gestor de SLAs.

- Alterar o processo de Análise. Actualmente as análises realizada ao sistema ocorrem periodicamente, tal não é um bom princípio como foi referido na secção 4.4, pois basta ocorrer um pico de carga ou uma diminuição da mesma entre os intervalos das análises e o Analisador não se conseguirá adaptar com prontidão. Uma solução poderia passar pela utilização de um sistema à base de alertas, onde um novo componente responsável por avaliar alertas despoleta uma análise do sistema se determinados acontecimentos ocorrerem.

- Recorrer à base de dados de Conhecimento para optimizar o Pod Autonómico, melhorando desta forma o desempenho de todos os componentes que dele fazem parte e consequentemente a adapatção Autonómica.

- Encontrar uma alternativa mais dinâmica para o controlo das instâncias activas de um serviço, por parte do Proxy e do Monitorizador.
- Colocar os servidores Prometheus fora dos Pods de Serviço, pois actualmente devido ao facto do Pod ser apagado sempre que ocorre uma reconfiguração a instância Prometheus também é apagada, o que faz com que não seja possível visualizar o histórico completo das instâncias.
- Optimizar o Analisador para que seja possível definir e considerar o peso das métricas. Esta alteração implica uma actualização do formato SLAng que permita a definição da importância das métricas. Seria também interessante permitir a definição da importância dos vários serviços e caso houvessem, as relações entre os mesmos, por forma a que estas pudessem ser consideras durante a adaptação.

Os pontos mencionados são as principais alterações e melhorias a realizar num futuro próximo para potenciar siginificativamente o sistema.

5.2.2 Implementação na SPMS

Uma vez que a integração do sistema desenvolvido com a SPMS foi considerada desde o início do projecto, muito do trabalho que de outra forma teria de ser agora realizado já foi concluído, nomeadamente:

- As tecnologias utilizadas são idênticas inclusive a plataforma onde os serviços são executados, o que facilita a incorporação dos mecanismos utilizados no sistema desenvolvido.
- Os serviços que serviram de base para a avaliação experimental são exactamente os mesmos serviços que são executados na SPMS, permitindo assim a fácil alteração dos restantes.

O trabalho já desenvolvido diminui muito o esforço associado à integração a realizar, no entanto, é ainda necessário muito trabalho para que a solução desenvolvida possa ser utilizada em produção. Numa primeira fase seria importante desenvolver testes unitários e testes de carga para atestar a veracidade dos serviços e a capacidade do sistema, face aos requisitos reais da SPMS.

Numa segunda fase seria importante proceder à escolha de um conjunto de serviços a alterar. A estes serviços seria adicionada a componente de monitorização à semelhança dos serviços utilizados na fase avaliação. Para cada um dos serviços teria que ser definido um SLA com os valores limites das métricas alto nível a monitorizar. Seria importante rever as métricas alto nível e se necessário definir novas métricas de acordo com as necessidades específicas da SPMS.

Era fundamental permitir a criação e remoção de mais Contentores e Pods por hora, sendo isto possível através de um pedido ao Azure na secção do ACI.

Tendo realizado todas as etapas anteriores poderia ser lançada uma versão piloto do sistema com os serviços modificados.

5.2.3 Investigação

Tal como foi explicado na secção 2.1.2, o sistema desenvolvido só alcança o penúltimo grau de autonomia designado "Autonómico". Para atingir o último nível e tornar-se totalmente autonómico teria que possuir algum tipo de inteligência, que lhe permitisse aprender com o tempo qual a melhor parametrização interna para produzir os melhores resultados. Esta funcionalidade é bastante complexa podendo relacionar-se com diversas áreas como a Inteligência Artificial e Big Data, decerto seria um tópico interessante para uma investigação futura.

Bibliografia

[1] G. Abbas, A. K. Nagar, H. Tawfik e J. Y. Goulermas. "Quality of service issues and nonconvex Network Utility Maximization for inelastic services in the Internet". Em: *17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2009, September 21-23, 2009, South Kensington Campus, Imperial College London*. 2009, pp. 1–11. DOI: 10.1109/MASCOT.2009.5366162. URL: https://doi.org/10.1109/MASCOT.2009.5366162.

[2] Y. woon Ahn e A. M. K. Cheng. "Automatic Resource Scaling for Medical Cyber-Physical Systems Running in Private Cloud Computing Architecture". Em: *5th Workshop on Medical Cyber-Physical Systems, MCPS 2014, Berlin, Germany, April 14, 2014*. 2014, pp. 58–65. DOI: 10.4230/OASIcs.MCPS.2014.58. URL: https://doi.org/10.4230/OASIcs.MCPS.2014.58.

[3] Y. woon Ahn, A. M. K. Cheng, J. Baek, M. Jo e H. Chen. "An auto-scaling mechanism for virtual resources to support mobile, pervasive, real-time healthcare applications in cloud computing". Em: *IEEE Network* 27.5 (2013), pp. 62–68. DOI: 10.1109/MNET.2013.6616117. URL: https://doi.org/10.1109/MNET.2013.6616117.

[4] A. Baptista, M. C. Gomes e H. Paulino. "Session-Based Dynamic Interaction Models for Stateful Web Services". Em: *Exploring Services Science - Third International Conference, IESS 2012, Geneva, Switzerland, February 15-17, 2012. Proceedings*. Vol. 103. Lecture Notes in Business Information Processing. Springer, 2012, pp. 29–43. DOI: 10.1007/978-3-642-28227-0_3. URL: https://doi.org/10.1007/978-3-642-28227-0_3.

[5] M. N. Bennani e D. A. Menascé. "Resource Allocation for Autonomic Data Centers using Analytic Performance Models". Em: *Second International Conference on Autonomic Computing (ICAC 2005), 13-16 June 2005, Seattle, WA, USA*. 2005, pp. 229–240. DOI: 10.1109/ICAC.2005.50. URL: https://doi.org/10.1109/ICAC.2005.50.

[6] W. Chen e J. Zhang. "An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements". Em: *IEEE Trans. Systems, Man, and Cybernetics, Part C* 39.1 (2009), pp. 29–43. DOI: 10.1109/TSMCC.2008.2001722. URL: https://doi.org/10.1109/TSMCC.2008.2001722.

[7] R. Dua, A. R. Raja e D. Kakadia. "Virtualization vs Containerization to Support PaaS". Em: *2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, March 11-14, 2014*. 2014, pp. 610–614. DOI: 10.1109/IC2E.2014.41. URL: https://doi.org/10.1109/IC2E.2014.41.

[8] A. G. Ganek e T. A. Corbi. "The dawning of the autonomic computing era". Em: *IBM Systems Journal* 42.1 (2003), pp. 5–18. DOI: 10.1147/sj.421.0005. URL: https://doi.org/10.1147/sj.421.0005.

[9] A. García-García, I. B. Espert e V. H. García. "SLA-driven dynamic cloud resource management". Em: *Future Generation Comp. Syst.* 31 (2014), pp. 1–11. DOI: 10.1016/j.future.2013.10.005. URL: https://doi.org/10.1016/j.future.2013.10.005.

[10] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar e R. Buyya. "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter". Em: *J. Network and Computer Applications* 45 (2014), pp. 108–120. DOI: 10.1016/j.jnca.2014.07.030. URL: https://doi.org/10.1016/j.jnca.2014.07.030.

[11] D. Garlan e B. R. Schmerl. "Model-based adaptation for self-healing systems". Em: *Proceedings of the First Workshop on Self-Healing Systems, WOSS 2002, Charleston, South Carolina, USA, November 18-19, 2002*. 2002, pp. 27–32. DOI: 10.1145/582128.582134. URL: http://doi.acm.org/10.1145/582128.582134.

[12] E. Gat. "On Three-Layer Architectures". Em: *Artificial Intelligence and Mobile Robots*. MIT Press, 1998.

[13] M. C. Gomes, H. Paulino, A. Baptista e F. Araújo. "Accessing Wireless Sensor Networks Via Dynamically Reconfigurable Interaction Models". Em: *IJIMAI* 1.7 (2012), pp. 52–61. DOI: 10.9781/ijimai.2012.176. URL: https://doi.org/10.9781/ijimai.2012.176.

[14] M. C. Gomes, H. Paulino, A. Baptista e F. Araújo. "Dynamic Interaction Models for Web Enabled Wireless Sensor Networks". Em: *10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012, Leganes, Madrid, Spain, July 10-13, 2012*. IEEE Computer Society, 2012, pp. 823–830. DOI: 10.1109/ISPA.2012.120. URL: https://doi.org/10.1109/ISPA.2012.120.

[15] X. He, X. Sun e G. von Laszewski. "QoS Guided Min-Min Heuristic for Grid Task Scheduling". Em: *J. Comput. Sci. Technol.* 18.4 (2003), pp. 442–451. DOI: 10.1007/BF02948918. URL: https://doi.org/10.1007/BF02948918.

[16] *Healthcare Information Systems, Second Edition (Best Practices)*. Auerbach Publications, 2002. ISBN: 0849314984. URL: https://www.amazon.com/Healthcare-Information-Systems-Second-Practices/dp/0849314984?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0849314984.

[17] Horn. "Autonomic Computing: IBM's Perspective on the State of Information Technology". Em: *IBM Corporation (October 15, 2001)*. (Out. de 2001).

[18] M. C. Huebscher e J. A. McCann. "A survey of autonomic computing - degrees, models, and applications". Em: *ACM Comput. Surv.* 40.3 (2008), 7:1–7:28. DOI: 10.1145/1380584.1380585. URL: http://doi.acm.org/10.1145/1380584.1380585.

[19] *An Architectural Blueprint for Autonomic Computing*. Rel. téc. IBM, jun. de 2005.

[20] N. R. Jennings e M. J. Wooldridge. "Applications of Intelligent Agents". Em: *Agent Technology*. Ed. por N. R. Jennings e M. J. Wooldridge. Springer Berlin Heidelberg, 1998.

[21] A. Keller e H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Em: *J. Network Syst. Manage.* 11.1 (2003), pp. 57–81. DOI: 10.1023/A:1022445108617. URL: https://doi.org/10.1023/A:1022445108617.

[22] J. O. Kephart e D. M. Chess. "The Vision of Autonomic Computing". Em: *IEEE Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055. URL: https://doi.org/10.1109/MC.2003.1160055.

[23] J. Kramer e J. Magee. "Self-Managed Systems: an Architectural Challenge". Em: *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*. 2007, pp. 259–268. DOI: 10.1109/FOSE.2007.19. URL: https://doi.org/10.1109/FOSE.2007.19.

[24] K. Kuchcinski e R. Szymanek. *JaCoP Library User's Guide*. 2017. URL: http://jacopguide.osolpro.com/guideJaCoP.html.

[25] D. D. Lamanna, J. Skene e W. Emmerich. "SLAng: A Language for Defining Service Level Agreements". Em: *9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), 28-30 May 2003, San Juan, Puerto Rico, Proceedings*. 2003, p. 100. DOI: 10.1109/FTDCS.2003.1204317. URL: https://doi.org/10.1109/FTDCS.2003.1204317.

[26] K. Lin, J. Zhang, Y. Zhai e B. Xu. "The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA". Em: *Service Oriented Computing and Applications* 4.3 (2010), pp. 157–168. DOI: 10.1007/s11761-010-0063-6. URL: https://doi.org/10.1007/s11761-010-0063-6.

[27] G. Lodi, F. Panzieri, D. Rossi e E. Turrini. "SLA-Driven Clustering of QoS-Aware Application Servers". Em: *IEEE Trans. Software Eng.* 33.3 (2007), pp. 186–197. DOI: 10.1109/TSE.2007.28. URL: https://doi.org/10.1109/TSE.2007.28.

[28] B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir e T. F. Lawrence. "Taxonomy for QoS specifications". Em: *3rd Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97), 5-7 February 1997, Newport Beach, CA, USA*. 1997, pp. 100–107. DOI: 10.1109/WORDS.1997.609931. URL: https://doi.org/10.1109/WORDS.1997.609931.

[29] S. Singh, I. Chana e M. Singh. "The Journey of QoS-Aware Autonomic Cloud Computing". Em: *IT Professional* 19.2 (2017), pp. 42–49. DOI: 10.1109/MITP.2017.26. URL: https://doi.org/10.1109/MITP.2017.26.

[30] G. Tesauro, R. Das, W. E. Walsh e J. O. Kephart. "Utility-Function-Driven Resource Allocation in Autonomic Systems". Em: *Second International Conference on Autonomic Computing (ICAC 2005), 13-16 June 2005, Seattle, WA, USA*. 2005, pp. 342–343. DOI: 10.1109/ICAC.2005.65. URL: https://doi.org/10.1109/ICAC.2005.65.

[31] V. Tosic, K. Patel e B. Pagurek. "WSOL - Web Service Offerings Language". Em: *Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES 2002, Toronto, Canada, May 27-28, 2002, Revised Papers*. 2002, pp. 57–67. DOI: 10.1007/3-540-36189-8_5. URL: https://doi.org/10.1007/3-540-36189-8_5.

[32] H. N. Van, F. D. Tran e J. Menaud. "SLA-Aware Virtual Resource Management for Cloud Infrastructures". Em: *Ninth IEEE International Conference on Computer and Information Technology, Xiamen, China, CIT 2009, 11-14 October 2009, Proceedings, Volume I*. 2009, pp. 357–362. DOI: 10.1109/CIT.2009.109. URL: https://doi.org/10.1109/CIT.2009.109.

[33] W. E. Walsh, G. Tesauro, J. O. Kephart e R. Das. "Utility Functions in Autonomic Systems". Em: *1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA*. 2004, pp. 70–77. DOI: 10.1109/ICAC.2004.68. URL: http://doi.ieeecomputersociety.org/10.1109/ICAC.2004.68.

[34] M. Wooldridge e N. R. Jennings. "Intelligent agents: theory and practice". Em: *Knowledge Eng. Review* 10.2 (1995), pp. 115–152. DOI: 10.1017/S0269888900008122. URL: https://doi.org/10.1017/S0269888900008122.

[35] J. Xu, M. Zhao, J. A. B. Fortes, R. Carpenter e M. S. Yousif. "On the Use of Fuzzy Modeling in Virtualized Data Center Management". Em: *Fourth International Conference on Autonomic Computing (ICAC'07), Jacksonville, Florida, USA, June 11-15, 2007*. 2007, p. 25. DOI: 10.1109/ICAC.2007.28. URL: https://doi.org/10.1109/ICAC.2007.28.

[36] L. Zhang e D. Ardagna. "SLA based profit optimization in autonomic computing systems". Em: *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*. 2004, pp. 173–182. DOI: 10.1145/1035167.1035193. URL: http://doi.acm.org/10.1145/1035167.1035193.

Guilherme Rodrigues Alcobia Santos

Licenciado em Ciência e Engenharia Informática

Monitorização Autonómica de Contentores Docker e a sua Aplicação a Serviços da Saúde

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Setembro, 2018

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Guilherme Rodrigues Alcobia Santos

Licenciado em Ciência e Engenharia Informática

Monitorização Autonómica de Contentores Docker e a sua Aplicação a Serviços da Saúde

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Setembro, 2018

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Guilherme Santos 2018
Monitorização Autonómica de Contentores Docker e a sua Aplicação a Serviços da Sa 2018