1 2 9 0

UNIVERSIDADE D
COIMBRA

Henrique Teixeira Fonseca

Sistema de Apoio à Decisão, Deteção de Erros e Formação Médica em Terapêuticas Injetáveis

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor António Dourado Pereira Correia e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Henrique Teixeira Fonseca

Sistema de Apoio à Decisão, Deteção de Erros e Formação Médica em Terapêuticas Injetáveis

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de *Software*, orientada pelo Professor Doutor António Dourado Pereira Correia e o Engenheiro Elsio Cardoso, da MedicineOne, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2023

Agradecimentos

Primeiramente, gostaria de agradecer à MedicineOne, por me terem acolhido nos seus escritórios, e investido em mim como estagiário e informático, permitindo-me experiênciar algo mais do que apenas prática no mercado de trabalho: um gosto pelo trabalho que se faz, e pelas vidas que se mudam com o mesmo.

Um obrigado muito grande à Rute Esteves, Gestora Legal e dos Recursos Humanos da MedicineOne, pela disponibilidade, e por me ter oferecido a oportunidade de estagiar nesta empresa, verificando quais os projetos poderiam beneficiar de um estagiários, e submetendo uma nova proposta para o DEI, após eu não ter sido selecionado para nenhum estágio profissional na primeira fase de candidaturas.

Gostaria de agradecer aos meus orientadores, Elsio Cardoso, engenheiro na MedicineOne, e António Dourado, professor no Departamento de Engenharia Informática (DEI) da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC), não só pelo auxílio quer no desenvolvimento da prova de conceito, e na redação deste documento, mas também pela compreensão e apoio que tiveram para comigo em certos momentos.

Um enorme obrigado ao Pedro Ferreira, engenheiro da MedicineOne, pela disponibilidade, vontade de ajudar, e pelo gosto em ensinar e motivar.

Agradeço também à Ana Filomena Isabel e ao Ricardo Vieira pelo auxílio da correção de erros ortográficos, de conteúdo e de estrutura deste documento, e por se demonstrarem disponíveis e interessados em oferecer esse mesmo auxílio.

Gostaria, por fim, de agradecer à minha mãe, Ana Paula Fonseca, à minha irmã, Patrícia Fonseca, aos meus amigos mais próximos, Tiago Menezes, Bernardo Rodrigues e Ana Beatriz, e aos seus amigos/colegas de faculdade Francisco Bugalho, Pedro Tiago, Diogo Jordão, Nuno Silva, José Reis, José Miguel e Bruno Gandres, por me auxiliarem em manter o foco no estágio, durante este ano menos positivo na minha vida.

Resumo

Este documento relata o projeto de mestrado do autor, para o qual foi necessário definir uma arquitetura *cloud* de forma a implementar uma prova de conceito do módulo de apoio à decisão, deteção de erros e formação médica de terapêuticas injetáveis do produto da MedicineOne: o M1. A pesquisa foi maioritariamente efetuada através da análise da documentação de três *cloud providers* pertencentes a três empresas distintas: AWS da Amazon, Azure da Microsoft e GCP da Google. É definido que o *cloud provider* a utilizar será o AWS, uma vez que, além de algumas vantagens sobre os restantes, é o *cloud provider* utilizado em outros produtos da MedicineOne. Será também definido um plano para a implementação de uma prova de conceito desta solução, através da sua arquitetura, produtos *cloud* a utilizar e divisão de tarefas de implementação por fases. Todas estas decisões serão justificadas e todo o processo de implementação será detalhadamente explicado ao longo deste documento. Por fim, serão efetuados comentários, uns de forma a visar o futuro deste projeto após o término do estágio curricular, e os restantes em forma de conclusões e críticas a todo o processo de estágio efetuado. O projeto foi concluído com sucesso uma vez que os requisitos funcionais foram cumpridos. No entanto, não se encontra totalmente de acordo com o que seria o seu maior potencial. Nesse sentido, verifica-se a falta de certas validações no código, bem como, o uso de uma biblioteca de forma a facilitar a transição de objetos entre camadas da arquitetura.

Abstract

This document's scope is to report a master's degree project in which the author aims to define a cloud architecture that serves the purpose of hosting a concept proof of the module of decision support, error detection and medical instruction on injectable therapies that belongs to MedicineOne's product: M1. The research's data was mainly gathered on the documentation of the three biggest cloud providers on the market, which are AWS, from Amazon; Azure, from Microsoft; and GCP, from Google. It will be decided that the cloud provider to use is AWS. Beyond it's own advantages over other cloud providers, it's also already used with some of other MedicioneOne's products. There will also be the definition of a plan to follow for the implementation of a proof of concept for this project, which is divided between the cloud arquitecture, cloud tecnologies to use, and implementation tasks division. All the decisions will be justified, and all the implementation processes will be carefully explained. By the end of the document, some comments will be made about the future of this project, after the end of the internship. Also, there will be some remarks and conclusions to be taken about the whole process of the internship. The project was successfully concluded, since all the functional prerequisites were accomplished. Nevertheless, it is not fully what was intended. In that sense, one can tell there are some missing validation to do in the project's code, as well as the usage of a library in order to facilitate the transition between objects from different layers of the architecture.

Palavras-Chave

Cloud Providers, Perfusões, Amazon Web Services (AWS), Container as a Service (CaaS), Escalabilidade, Segurança, Base de Dados, Arquitetura, Docker, Fargate

.

Conteúdo

1 Introdução **1**
1.1 Enquadramento e motivações . . . 1
1.2 Objetivos . . . 2
1.2.1 Objetivos Pessoais . . . 2
1.2.2 Objetivos do Projeto . . . 2
1.3 Estrutura do documento . . . 2

2 Metodologia e Planeamento **5**
2.1 Metodologia de Gestão de Projeto . . . 5
2.1.1 Scrum . . . 5
2.1.2 Comunicação . . . 7
2.2 *Guidelines* . . . 7
2.3 Requisitos . . . 9
2.3.1 Requisitos Funcionais . . . 9
2.3.2 Requisitos Não Funcionais . . . 10
2.4 Gestão de Riscos . . . 11
2.4.1 Métricas . . . 11
2.4.2 Identificação e análise dos riscos . . . 13
2.5 Planeamento . . . 13
2.5.1 Primeiro Semestre . . . 13
2.5.2 Segundo Semestre . . . 14

3 Conceitos Introdutórios sobre *Cloud* e Estado da Arte **15**
3.1 Conceitos Introdutórios sobre *Cloud* . . . 15
3.1.1 IaaS . . . 15
3.1.2 CaaS . . . 15
3.1.3 PaaS . . . 15
3.1.4 SaaS . . . 16
3.1.5 *Serverless Computing* . . . 16
3.1.6 *Hybrid Cloud Approach* . . . 16
3.2 Estado da Arte . . . 17
3.2.1 Mercado Atual . . . 17
3.2.2 Principais *Cloud Providers* . . . 17
3.3 *Cloud Providers* . . . 17
3.3.1 *Cloud Providers* Líderes do Mercado . . . 18
3.3.2 AWS vs. Azure vs. GCP . . . 19
3.3.3 Outros *cloud providers* analisados . . . 28
3.4 Sumário e Discussão do capítulo . . . 29

4 Arquitetura **31**
4.1 *Onion Architecture* . . . 31
4.2 Arquitetura para produtos *cloud native* . . . 33
4.3 Arquitetura Final . . . 34
4.3.1 Produtos de Segurança . . . 34
4.3.2 Produtos de Rede . . . 35
4.3.3 Produtos de Computação . . . 35
4.3.4 Produtos de Armazenamento . . . 36

4.3.5 Produtos de Integração . . . 36
4.3.6 Produtos de Gestão . . . 36
4.4 Sumário e Discussão do Capítulo . . . 37

5 Desenvolvimento do Projeto **39**
5.1 Preparação e Tutoriais . . . 39
5.2 Migração da Base de Dados . . . 39
5.3 Implementação do código do projeto . . . 40
5.4 Criação do *container* Docker . . . 41
5.5 Ambiente AWS . . . 41
5.5.1 Ambiente Inicial AWS . . . 41
5.5.2 *Set-up* da AWS CLI e AWS Tools for PowerShell . . . 42
5.5.3 Registo do *container* . . . 42
5.5.4 Criação do *cluster* Fargate . . . 42
5.5.5 Migração da Base de Dados Local (PostgreSQL) para o AWS RDS . . . 44
5.5.6 Criação de um NLB e serviço Fargate . . . 46
5.5.7 API Gateway, Amazon Cognito e AWS Lambda . . . 49
5.5.8 Correções na Infraestrutura . . . 50
5.6 Sumário e Discussão do Capítulo . . . 51

6 Testes **53**
6.1 Planeamento de Testes . . . 53
6.2 Execução de Testes . . . 53
6.3 Sumário e Discussão do Capítulo . . . 54

7 Conclusão **55**
7.1 Análise do Planeamento . . . 55
7.2 Trabalho Futuro . . . 57
7.3 Comentários e Conclusões . . . 57

Anexo A Plano e Casos de Teste **67**

Acrónimos

AKS Azure Kubernetes Service.

Amazon RDS Amazon Relational Database Services.

API Application Programming Interface.

ASHP American Society of Health System Pharmacists.

AWS Amazon Web Services.

CaaS Container as a Service.

CHNM Código Hospitalar Nacional do Medicamento.

CIDR Classless Inter-Domain Routing.

CLI Command Line Interface.

CPU Central Processing Unit.

DDD Domain Driven Design.

DDoS Distributed Denial of Service.

DEI Departamento de Engenharia Informática.

DNS Domain Name Service.

DTO Data Transfer Object.

EC2 Amazon Elastic Compute Cloud.

ECR Elastic Container Registry.

ECS Elastic Container Service.

EHR Electronic Health Record.

EKS Elastic Kubernetes Service.

FCTUC Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

GCP Google Cloud Platform.

GKE Google Kubernetes Engine.

GPUs Unidades de Processamento Gráfico.

I/O Input e/ou Output.

IaaS Infrastructure as a Service.

IAM AWS Identity and Access Management.

IBM International Business Machines Corporation.

IDE Integrated Development Environment.

IP Internet Protocol.

K8s Kubernetes.

MEI Mestrado em Engenharia Informática.

MIGs Managed Instance Groups.

NLB Network Load Balancer.

OCI Oracle Cloud Infrastructure.

OSI Open Systems Interconnection.

P2S Point-to-Site.

PaaS Platform as a Service.

S2S Site-to-Site.

SaaS Software as a Service.

SDK Software Development Kit.

SLA Service Level Agreement.

SO Sistema Operativo.

SOs Sistemas Operativos.

SSMS SQL Server Management Studio.

TCO Total Cost of Ownership.

UC Universidade de Coimbra.

URL Uniform Resource Locator.

VM Máquina Virtual.

VPC Virtual Private Cloud.

VPN Virtual Private Network.

Lista de Figuras

2.1 Caso de Segurança . . . 10
2.2 Caso de Resiliência e Tolerância a Falhas . . . 11
2.3 Caso de Escalabilidade . . . 11
2.4 Matriz de riscos [14] . . . 12
2.5 Plano de Trabalhos . . . 14

3.1 Regiões geográficas com zonas de disponibilidade AWS [34] . . . 18
3.2 Regiões geográficas com zonas de disponibilidade GCP [35] . . . 19
3.3 Estrutura de rede AWS [37] . . . 19
3.4 Estrutura de rede Azure [37] . . . 20
3.5 Estrutura de rede GCP [37] . . . 20
3.6 ligação *on-premise* AWS [37] . . . 20
3.7 Modelo Responsabilidade Partilhada Amazon Web Services (AWS) [45] . . . 25
3.8 Modelo Responsabilidade Partilhada Azure [46] . . . 25
3.9 Modelo Responsabilidade Partilhada Google Cloud Platform (GCP) [47] . . . 26

4.1 Esquema de Onion Arquitecture . . . 33
4.2 Arquitetura Final [85] . . . 34

5.1 Estrutura da Solução . . . 40
5.2 Dockerfile . . . 41
5.3 Comandos para Registo da imagem Docker no ECR . . . 42
5.4 Etapa 1 da criação do *cluster* Fargate . . . 43
5.5 Etapa 2 da criação do *cluster* Fargate . . . 43
5.6 Etapa 2 da criação da tarefa Elastic Container Service (ECS) . . . 44
5.7 Configurações de armazenamento da instância Amazon Relational Database Services (Amazon RDS) . . . 45
5.8 Registo do servidor Amazon RDS . . . 46
5.9 Propriedades NLB . . . 47
5.10 Conexão entre a tarefa Fargate e o Network Load Balancer (NLB) . . . 48
5.11 Confirmação da ligação entre no NLB e a tarefa Fargate através do Target Group . . . 48
5.12 Recursos no API Gateway . . . 50
5.13 Configurações de Integração . . . 50
5.14 Segredo para acesso à Base de Dados RDS . . . 51
5.15 Sucesso de um pedido de teste no API Gateway . . . 51
5.16 Sucesso de um pedido de teste no Postman . . . 52

6.1 Exemplo de Unit Test . . . 54

7.1 Plano para o futuro da arquitetura do projeto . . . 56

Lista de Tabelas

3.1 Tabela de protocolos para *load balancing*, AWS vs. Azure vs. GCP 21
3.2 Apoios ao SQL Server 21
3.3 Monitorização e Disponibilidade 22
3.4 Escalabilidade dos serviços de bases de dados 22
3.5 *Backups* das bases de dados 22
3.6 Segurança das bases de dados 23
3.7 Otimizações das VMs dos cloud providers 23
3.8 Disponibilidade de acordo com os Service Level Agreement (SLA) 24
3.9 Modelo de Responsabilidade Partilhada 24
3.10 Produtos de K8s 28

Capítulo 1

Introdução

Este documento foi redigido no âmbito da disciplina de Dissertação/Estágio, que se insere no Mestrado em Engenharia Informática (MEI), especificação em Engenharia de Software, da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (FCTUC).

O estágio decorreu durante o ano letivo de 2022/2023, sob a supervisão e acompanhamento do Professor António Dourado Pereira Correia do Departamento de Engenharia Informática (DEI) da Universidade de Coimbra (UC), e do Engenheiro Elsio Cardoso da MedicineOne, empresa que facultou o estágio em questão.

1.1 Enquadramento e motivações

É frequente que utentes de um determinado serviço de saúde necessitem de medicação que atue de imediato no seu sistema. Em algumas situações, estes mesmos utentes poderão estar impossibilitados de ter medicamentos administrados via oral. Há, portanto, fatores que implicam a administração de soluções medicamentosas diretamente na corrente sanguínea dos utentes.

A administração destes medicamentos diluídos em solventes, tais como, soro, glucose, etc., pode ser apelidada de perfusão. Sabendo que podem existir diversos fatores que podem colocar em risco a saúde do utente, estes fatores de risco devem ser alvo de análise, antes de cada procedimento. Alguns desses fatores são:

- Temperatura;
- Condições de luz;
- Outras medicações às quais o utente possivelmente esteja sujeito;
- Estabilidade físico-química da solução (ex: compatibilidade dos medicamentos);
- etc.

O M1 [1], é o produto mais comercializado da MedicineOne. Tendo em conta as considerações referidas anteriormente, este produto, comumente presente em hospitais do setor português privado da saúde, inclui no seu módulo de internamento a possibilidade de prescrever perfusões. Neste modelo de prescrição, após a seleção dos medicamentos e das suas especificações, o *software* informa o utilizador de incompatibilidades, riscos e outros avisos, os quais o prescritor deve ter em consideração antes de tomar a decisão final. Além dos médicos prescritores, este *software* auxilia, não só farmacêuticos no momento da preparação da solução, como enfermeiros no momento da administração.

Sendo que parte da missão [2] da MedicineOne é o auxílio a profissionais de saúde na sua atividade, e que um dos valores [2] da empresa é a colaboração entre estes, um dos objetivos da

empresa para com este projeto é a partilha desta solução para outros Electronic Health Record (EHR), que são aplicações que registam o percurso de saúde de utentes e disponibilizam essa informação para outros profissionais de saúde. É esta disponibilização de informação o cerne do que foi o objetivo deste estágio.

1.2 Objetivos

Nesta secção, estão descritos os objetivos pessoais do autor deste documento, bem como os objetivos do projeto relativos ao seu estágio.

1.2.1 Objetivos Pessoais

Na sua ótica pessoal, o autor desta dissertação, doravante referido como estagiário aquando das suas contribuições para o estágio, considera que, a prioridade durante o estágio foi demonstrar a si mesmo e aos que o rodeiam, que é um bom profissional através de uma boa ética profissional e empenho, e, em simultâneo, sempre disposto a melhorar. É, ainda, importante para o mesmo, adquirir novas capacidades, tanto *hard* como *soft skills*, bem como consolidar as que já possui.

No âmbito profissional, o objetivo centra-se na implementação da solução que melhor se adequa, não só à MedicineOne, mas também a todas as empresas que possuam um EHR e que necessitem de aceder às soluções que o M1 possui e que a MedicineOne esteja disposta a disponibilizar. De uma forma sucinta, o autor espera conseguir auxiliar o maior número de profissionais de saúde possível, através do acesso à área de perfusões do M1.

1.2.2 Objetivos do Projeto

O objetivo do estágio consiste na implementação de uma Application Programming Interface (API) *cloud native* que tire partido das tecnologias mais recentes de *cloud*. Assim sendo, através da invocação de micro-serviços, todas as soluções do M1, e outros EHR, terão a possibilidade de aceder ao sistema de apoio à decisão de administração de medicamentos via intravenosa. Para alcançar este objetivo, será necessário analisar não só a existência de soluções semelhantes, mas também qual(ais) a(s) tecnologia(s) *cloud*, ou *cloud providers*, que melhor se agilizam com as necessidades do *software* M1.

Por fim, será necessário implementar um editor de conteúdo, de forma a que profissionais de saúde possam atualizar a informação relativa ao sistema de informação em questão.

1.3 Estrutura do documento

Este documento contém na sua estrutura sete capítulos, cada um dos quais inclui a descrição detalhada das etapas do estágio.

- No **capítulo 2**, é apresentada a metodologia de gestão de projeto à que o estagiário terá sido introduzido durante o segundo semestre, assim como o planeamento das tarefas do mesmo;
- No **capítulo 3** são apresentados conceitos introdutórios necessários para compreensão da informação pesquisada pelo autor, nomeadamente sobre *cloud*. São, também apresentados, o estado da arte de sistemas de apoio à decisão clínica e *cloud providers* líderes do mercado, e definidos os conceitos e as *guidelines* às quais o estagiário teve acesso. Neste seguimento, é efetuada seguidamente uma comparação detalhada e extensiva entre os *cloud providers* líderes do mercado. Por fim, são apresentados dois *cloud providers* que poderiam ter outros benefícios para o projeto, além dos benefícios oferecidos pelos *cloud providers* líderes de mercado;

- No **capítulo 4** é apresentada a estrutura do *software* da MedicineOne, o M1, e apresentada a arquitetura *cloud native* na qual o projeto se enquadra, assim como todos os produtos que a mesma utiliza juntamente com a justificação para o uso dos mesmos.
- No **capítulo 5** está descrito detalhadamente todo o trabalho de desenvolvimentos que o estagiário efetuou, encontram-se especificadas todas as tecnologias utilizadas, e são explicados todos os conceitos mais específicos que não foram apresentados nos capítulos anteriores;
- No **capítulo 6** apresentam-se as métricas, metodologias de testagem e os testes efetuados pelo estagiário;
- No **capítulo 7**, por último, é efetuada uma discussão sobre o futuro do estagiário e do projeto de estágio, realizando uma breve conclusão;

Capítulo 2

Metodologia e Planeamento

Neste capítulo será, inicialmente, apresentada a metodologia de gestão de projetos a qual o estagiário e autor deste documento terá aplicado no desenvolvimento do seu projeto, o Scrum. Seguidamente, será apresentado o plano de tarefas pelo qual o mesmo se guiou. Por fim, será feita uma identificação e análise de riscos do projeto em questão.

2.1 Metodologia de Gestão de Projeto

2.1.1 Scrum

O Scrum é uma *framework* leve que auxilia equipas de desenvolvimento a melhor identificar, gerir e acompanhar objetivos/tarefas durante todo o processo de desenvolvimento de um *software* através do uso de certas práticas. Esta *framework* prioriza ciclos curtos de *feedback*. É por isso considerada uma *framework* ágil de gestão de projetos para a obtenção e registo de nova informação e, consequentemente, respetivas melhorias contínuas.

Artefactos do Scrum

Existem três artefactos de grande importância dentro desta *framework*: ***backlog* do produto**, ***backlog* do sprint** e **incremento do produto**.

- ***Backlog* do Produto**: é o conjunto total de tarefas a serem efetuadas no produto, incluindo *features*, melhorias e correções de erros. Esta lista pode, e deve, revista e melhorada constantemente. Por outras palavras, o *backlog* do produto consiste em todos os objetivos que a equipa tem para o produto final;

- ***Backlog* do *Sprint***: é o conjunto de tarefas a serem efetuadas até ao final do próximo ciclo de trabalho. Estas tarefas são provenientes do *backlog* do produto e, caso não sejam terminadas antes do ciclo de trabalho correspondente, serão adicionadas de novo ao mesmo, ou adicionadas ao próximo ciclo de trabalho. O ato de criar ciclos de trabalho, ou *sprints*, consiste na decisão de quais as tarefas que serão movidas do *backlog* do produto para o *backlog* do *sprint*;

- **Incremento do Produto**: é o resultado do término de tarefas, ou seja, as mesmas já não serão encontradas em nenhum dos *backlogs*, caso sejam terminadas num *sprint*. Isto resulta na adição de novas funcionalidades ou correções no produto atual (incremento do produto).

Equipa do Scrum

Como referido anteriormente, o Scrum é uma *framework* de auxílio à gestão de projetos e, consequentemente, à equipa de desenvolvimento do mesmo. Geralmente, as equipas num projeto em que se utiliza o Scrum, são pequenas e sem hierarquias. Estas equipas devem incluir membros de forma a que o conhecimento e as capacidades necessárias para completar o projeto estejam presentes. Assim, a própria equipa, em conjunto, deve ser *self-managed*, isto é, a equipa deve decidir quais as tarefas a efetuar, quando, como, e a quem serão atribuídas. Existem, no entanto, três tipos de posições a tomar dentro de uma equipa Scrum: **dono do projeto**, **mestre do Scrum** e **programadores**:

- **Dono do projeto**: corresponde a uma única pessoa - não a uma entidade -, que define os objetivos e prioridades a implementar no produto. Desta forma, é responsável por pedir a criação de tarefas no *backlog* do produto, porém, deve garantir que esses objetivos são claros e precisos;
- **Mestre do *Scrum***: corresponde, também, a um único indivíduo, que guia tanto o dono do projeto, como os programadores, a entenderem e a praticarem esta metodologia ágil. Deve ainda guiar na direção de constante melhoria. O mestre do *Scrum* é responsável, ainda, por garantir que os processos respeitam os tempos estabelecidos previamente, e que a equipa é, efetivamente, *self-managed*;
- **Programador**: são responsáveis por efetuar as tarefas que constam no *backlog* do produto, e por tentar adaptar o seu trabalho de forma a que, a cada *sprint*, terminem as tarefas que constem no seu *backlog*. Antes de uma tarefa ser classificada como terminada, os programadores são responsáveis por garantir a qualidade do resultado dessa mesma tarefa.

Eventos do *Sprint*

Como já foi referido anteriormente, os *sprints*, ou ciclos de trabalho, são as unidades básicas do Scrum. São eventos de duração fixa entre duas e quatro semanas utilizadas para criar consistência. Assim que um *sprint* termina, outro é iniciado e é dentro de cada *sprint* que são efetuadas operações como refinamento do *backlog* do produto, planeamento do *sprint* seguinte, revisão do decorrer do *sprint*, etc.:

- **Planeamento do *Sprint***: ocorre no início de cada *sprint*. Determinam-se quais as tarefas a serem efetuadas durante esse *sprint* para criar objetividade na equipa e existir um caminho definido para os objetivos desse mesmo *sprint*;
- **Refinamento do *Backlog***: processo em que o dono do produto e a equipa de desenvolvimento trabalham para que as tarefas presentes no *backlog* sejam bem definidas e estimadas de acordo com a sua complexidade. O objetivo é de manter as expectativas de todos os interveninentes equiparadas. Reuniões para este propósito devem ser agendadas uma vez por semana ou uma vez por *sprint*. O *backlog* do produto deve manter-se detalhado e estimado;
- **Scrum diário**: reunião diária que visa manter todos os membros da equipa de desenvolvimento atualizados acerca das tarefas efetuadas e a efetuar. Pode ocorrer troca de ideias quanto a certas tarefas, o que pode auxiliar os responsáveis pelas mesmas;
- **Revisão do *Sprint***: a equipa do Scrum apresenta o trabalho desenvolvido durante o *sprint* aos *stakeholders*. Estas reuniões devem ocorrer uma vez por *sprint*, previamente às reuniões de retrospetiva do *sprint* e com duração entre 30 a 60 minutos. Nestas reuniões, obtém-se *feedback* por parte dos *stakeholders*, determina-se se o objetivo do *sprint* foi corretamente atingido, e prepara-se o próximo *sprint*;
- **Retrospetiva do *Sprint***: reuniões de uma a duas horas, nas quais se discutem os sucessos e insucessos do *sprint*. Através da sua análise, procuram-se formas de melhorar o decorrer dos próximos *sprints*, bem como atitudes que devem ser mantidas.

2.1.2 Comunicação

A comunicação, durante o primeiro semestre, foi limitada a três membros (além de questões referentes a contratos e integração na empresa): o autor do documento e estagiário, o orientador de estágio Eng. Elsio Cardoso, para questões referentes ao estágio, e a formadora Sara Fernandes, para questões referentes às funcionalidades do produto, o M1. A comunicação foi efetuada através do **Microsoft Teams**, por *chat* ou por vídeo-chamada. A maioria da comunicação geral dentro da empresa é efetuada via **Outlook**, e também é uma possibilidade em momentos de contacto com vários membros. Durante o segundo semestre, os meios de comunicação não se alteraram, tendo o estagiário, no entanto, comunicado com outros trabalhadores da MedicineOne a fim de definir requisitos corretamente, obter acesso a documentos e obter permissões de utilização de *software*, utilizado dentro da MedicineOne, informações necessárias para o bom desenvolvimento do projeto.

Para organização do Scrum, é utilizada a ferramenta **Jira Software** [3].

2.2 *Guidelines*

Após uma introdução à aplicação M1 da MedicineOne e à linguagem C# usadas na implementação da mesma, foram oferecidas ao estagiário algumas *guidelines* de orientação à pesquisa necessária para a definição da arquitetura do sistema a implementar. Assim, o autor apresentará essas mesmas *guidelines* por tópicos e resumirá o propósito de cada uma das mesmas.

- **Micro-Serviços** [4]: uma arquitetura de micro-serviços consiste na divisão do *software* em pequenas aplicações independentes que comunicam entre si. O objetivo da MedicineOne a longo prazo, é transformar o seu principal produto, o M1, de uma arquitetura monolítica (sistema único com um processo) para uma arquitetura de micro-serviços, pretendendo atingir esse mesmo objetivo através do isolamento progressivo de funcionalidades. Assim, o módulo de aconselhamento à administração de medicamentos via intravenosa, o foco deste estágio, será um dos primeiro módulos a transformar num micro-serviço hospedado na *cloud*. Isto permite, não apenas aumentar a resiliência e, portanto, a disponibilidade deste módulo para o M1, mas também disponibilizá-lo para outros Electronic Health Record (EHR), o que foi definido como requisito previamente. O facto de transformar este módulo num micro-serviço e disponibilizá-lo na *cloud*, permitirá a possibilidade de adaptar automaticamente a sua escalabilidade (recursos) à quantidade de pedidos efetuados para o mesmo;
- **Base de dados *open-source*** [5]: uma base de dados *open-source* é qualquer base de dados disponível para *download*, modificação, distribuição e reutilização. Um exemplo destas bases de dados seria o PostgreSQL [6], uma base de dados *object-relational* disponível para todos os possíveis utilizadores. Sendo que o módulo de perfusões necessita de ter armazenados e constantemente atualizados os dados sobre, entre outras características, todos os medicamentos disponíveis no mercado, as suas interações, condições ideais de administração, etc., o mesmo necessita também de uma base de dados que seja capaz de lidar com uma grande afluência de pedidos de acesso a tabelas com grandes quantidades de dados. A *Big Data* [7], uma base de dados em PostgreSQL, adequa-se ao projeto;
- ***Auditing***: é o processo constante de análise e verificação da necessidade e qualidade dos serviços, em temas como desempenho, riscos, segurança, comunicação, gestão de mudanças/manutenção, etc. Através do registo, controlo e análise de dados gerados por um *cloud audit*, os responsáveis pela aplicação podem avaliar o estado da aplicação quanto às características que a mesma possui ou deve alcançar, podendo ainda definir planos, se necessários, para melhorar ou adaptar essas características;
- ***Logging***: é o ato de manter informação, quer características de eventos, erros, ou apenas operações efetuadas pelo *software*. Esta ação é aproveitada, em *development*, maioritariamente para deteção e correção de erros. É também bastante usada em produção, tanto para registar eventos como transferências de dados, comunicações e interações;

- **Segurança**: em computação *cloud*, é necessário definir várias normas de segurança, quer sejam sobre a infraestrutura, código ou dados. Existem várias formas de segurança, sendo todas complementares e não substituíveis:
  - **Confidencialidade** - evitar divulgação indesejada de dados;
  - **Integridade** - evitar alteração indesejada de dados;
  - **Disponibilidade** dos dados - apenas disponibilizar os dados necessários e a entidades que necessitem e tenham direito ao mesmos;
  - **Autenticação** - validação de quem está a utilizar o software;
  - **Autorização** - definição de acessos a uma entidade ou a um conjunto de entidades;
  - **Auditoria** - registo e análise de vários tipos de informação que a aplicação pode e consegue registar;
  - **Gestão de sessões, configurações e exceções**.

  Existem alguns tópicos referentes a segurança na *cloud*, cada um deles com vários serviços disponíveis, sendo eles:
  - **Gestão de identidade e acesso**: semelhante a autenticação e autorização;
  - **Deteção**: gestão de vulnerabilidades, acessos, etc.;
  - **Proteção de rede**: contratos e configurações para ligações seguras ao serviço;
  - **Proteção de dados**: criação e gestão de chaves de controlo, encriptação, etc.;
  - **Resposta a incidentes**: investigação e correção de lacunas de segurança;
  - **Conformidade**: acordos de segurança e *auditing*.

- **Escalabilidade** [8]: o conceito de escalabilidade, em computação *cloud*, é referente à medida em que um *software* consegue lidar com altas cargas de utilizadores, ou seja, a capacidade de adaptar os seus recursos a diferentes (principalmente elevadas) quantidades de pedidos efetuados. O objetivo da escalabilidade é manter a *performance* do programa, e uma das formas mais comuns de efetuar uma avaliação de *performance* é medir os tempos de resposta em várias situações. Existem dois métodos para aumentar a escalabilidade, sendo estes **escalabilidade vertical** e **escalabilidade horizontal** [9]:
  - **Escalabilidade vertical**: método que consiste no aumento das capacidades do *hardware*, semelhante a um *upgrade* ao sistema utilizado (instância);
  - **Escalabilidade horizontal**: método no qual são adicionadas novas instâncias e através do qual se distribui a carga de utilizadores pelas mesmas, isto é, são adicionadas e executadas novas instâncias da aplicação em diversos servidores. Isto implica a necessidade de sincronização entre as instâncias, nomeadamente, em momentos de escrita e leitura de dados. Geralmente, este processo é menos custoso, mas depende sempre do número de instâncias necessárias.

  Existem também três formas de aplicação dos métodos de escalonamento anteriores: **manualmente**, através de **agendamento** e **automaticamente** [10]:
  - **Escalonamento manual**: o escalonamento manual é aquele que implica maior uso de recursos de mão de obra. Quando a escassez (ou excesso) de recursos dedicados ao *software* devido ao aumento (ou diminuição) de utilizadores regulares ocorre, aplica-se uma das restantes duas formas de escalonamento;
  - **Escalonamento agendado**: como o nome infere, o escalonamento agendado implica a adaptação dos recursos dedicados ao *software* numa certa data ou até durante um determinado período de tempo. Para isto, é necessário o acompanhamento e a estatística de ocorrências de picos altos e baixos do número de utilizadores, que levará à tentativa de identificação de padrões destas ocorrências;
  - **Escalonamento automático**: o escalonamento automático, também como o nome indica, é a aplicação de um método de escalonamento quando o uso de recursos como Central Processing Unit (CPU), memória e rede são elevados. Normalmente é aplicado escalonamento horizontal e aplicados métodos de *load-balancing*.

De salientar que manter as capacidades de armazenamento de dados através do *software* é também de grande importância em questões de escalabilidade.

- **Resiliência** [11]: resiliência é a capacidade de um *software* resistir a falhas ou erros mantendo as suas funcionalidades ativas e disponíveis. Nos casos em que manter as funcionalidades se torna inevitável, o objetivo é recuperar as mesmas no menor tempo possível. Uma arquitetura de micro-serviços permite certas vantagens no isolamento das funcionalidades no que toca a falhas, nomeadamente, se serviços não dependerem de outros, apenas falharão caso sejam afetados diretamente. Alguns dos exemplos de causas de falhas são, por exemplo, latência inesperada, *overload* de pedidos, falhas de *hardware*, entre outros;

- **Docker** [12]: em computação *cloud*, o Docker é uma ferramenta utilizada para automatizar o *deployment* de aplicações num ambiente de gestão de *containers*. Um *container* do Docker é uma unidade de *software* que contém o código e todas as dependências necessárias para executar o programa contido nessa unidade, tornando mais fácil e fiável a sua transição entre ambientes. Este fator pode ser útil, como por exemplo, nas operações de escalonamento referidas previamente;

- **Amazon Web Services (AWS) vs. Microsoft Azure vs. Google Cloud Platform (GCP)**: foi ainda pedido ao autor para efetuar uma comparação entre os três *cloud providers* com maior cobertura do mercado, tendo em conta, principalmente, as *guidelines* anteriores. De notar que, apesar de a linguagem usada pelo produto M1 e, consequentemente, o módulo de perfusões do mesmo, ser C#, uma linguagem desenvolvida pela Microsoft, o *cloud provider* utilizado até ao momento pela MedicineOne não é o Microsoft Azure, mas sim o AWS.

2.3 Requisitos

Nesta secção serão apresentados os requisitos funcionais e não funcionais do projeto. Os **requisitos funcionais** foram definidos pelo autor através da análise de documentação e do código do M1, uma vez que o âmbito do projeto inclui manter as funcionalidades do produto em questão. Já os **requisitos não funcionais**, foram definidos com o auxílio do orientador Eng. Elsio Cardoso.

2.3.1 Requisitos Funcionais

Antes de introduzir as funcionalidades a transferir para micro-serviços, é necessário identificar algumas questões quanto à implementação. O produto (M1) já contém as funcionalidades implementadas, tanto em *frontend* como em *backend*, pelo que o primeiro objetivo do projeto é definir quais os *endpoints* (pedidos ao servidor) de *backend* a transitar para a um ambiente *cloud* e, posteriormente, efetuar a adaptação no M1. As funcionalidades a serem transitadas para este ambiente são as atualmente presentes num módulo de *backend* do M1 denominado Clinical Brain.

A visão da MedicineOne para este módulo é que o mesmo contenha funcionalidades que necessitem de armazenamento e processamento de dados sendo que estes terão de ser atualizados/adicionados e processados constantemente. Atualmente, o Clinical Brain, além das funcionalidades de prescrição de medicamentos (externa e interna), contém ainda um motor de regras de validação farmacêutica de prescrições, calculadoras médicas de diversas especialidades (calculadoras de riscos), disponibilização de evidências clínicas/estudos (que serão disponibilizados por fornecedores internacionais), entre outros. Para armazenar e aceder à informação destas funcionalidades, o módulo Clinical Brain acede a três bases de dados, sendo denominada de Trissel a referente às interações entre medicamentos e soluções medicamentosas.

Assim, será necessário disponibilizar estas funcionalidades, através de tecnologias *cloud*, não só às instâncias do M1 atualmente disponibilizadas a clientes, mas também a outros EHRs através de um modelo de subscrição.

2.3.2 Requisitos Não Funcionais

Antes de introduzir os requisitos não funcionais, é de referir que os mesmos são adaptados ao facto de o projeto consistir na implementação de uma prova de conceito e não num produto final, adaptando-se, desta forma, a uma complexidade inferior à necessária para um ambiente de produção.

Segurança

A segurança consiste, entre outras características, na capacidade do sistema bloquear o acesso não autorizado a terceiros tanto a funcionalidades, como a dados, o que pode ser alcançado através de processos de autenticação e autorização como uso de *firewall*s, contratos/acordos de segurança para ligações seguras, tal como referido na sub-secção 2.2. Em casos de falhas no impedimento de acesso a dados, estes devem estar encriptados de forma a não serem úteis a quem tenha efetuado estas tentativas de acesso. Por fim, é necessário garantir que, a Application Programming Interface (API) tem um sistema de *auditing* e *logging* que permita identificar e registar ataques ao sistema, de modo a permitir criar novas formas de proteção da mesma. Este requisito não funcional tem prioridade elevada no projeto, uma vez que casos como a alteração dos dados pode ser determinante no tratamento dos utentes. Na figura 2.1 está exemplificado o caso em que o sistema deve ter ações de segurança.

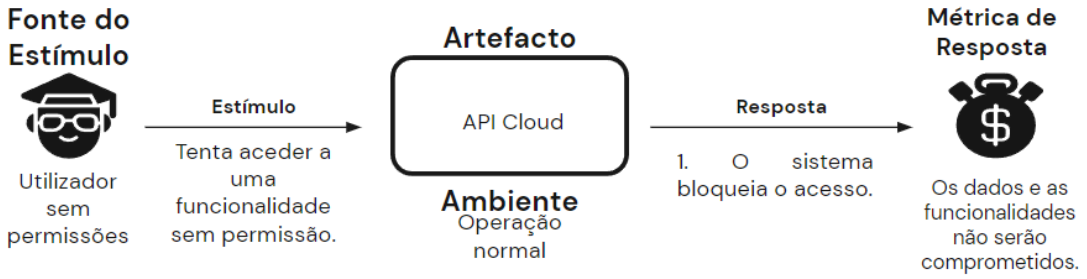

Figura 2.1: Caso de Segurança

Resiliência

Resiliência de *software* é a capacidade de o mesmo resistir a falhas/adversidades, ou seja, é a capacidade de manter o normal funcionamento ou de o recuperar. Pode ser caracterizada através de métricas como a percentagem de tempo disponível, uma determinada medida de tempo ou o tempo de recuperação através de mecanismos de *failover* [13]. Como já foi referido, na subsecção 2.2, irá ser implementada uma arquitetura de micro-serviços e serão utilizados *containers*, o que pode auxiliar neste requisito, nomeadamente através de características como *health check*, isolamento de funcionalidades, orquestração de *containers* ou até mesmo o uso de várias regiões de disponibilidade (redundância de dados e funcionalidades). Na figura 2.2, pode ser observado um dos casos de resiliência que a API *cloud* deve seguir.

Escalonamento

Como já referido, será implementada uma API *cloud* utilizando micro-serviços e *containers*, sendo este o ambiente ideal para escalar os mesmos, consoante a sua necessidade, ou seja, número de pedidos de acesso aos serviços. Tecnologias *cloud* têm vários produtos que auxiliam no escalonamento de aplicações, inclusivamente orquestradores de *containers*, tópicos a abordar posteriormente neste documento. No entanto, mesmo tendo as ferramentas ideais à disposição, o projeto consiste apenas na criação de uma prova de conceito e não na implementação de um produto final, como foi também já referido, sendo então as métricas destes requisitos adaptadas a este fator,

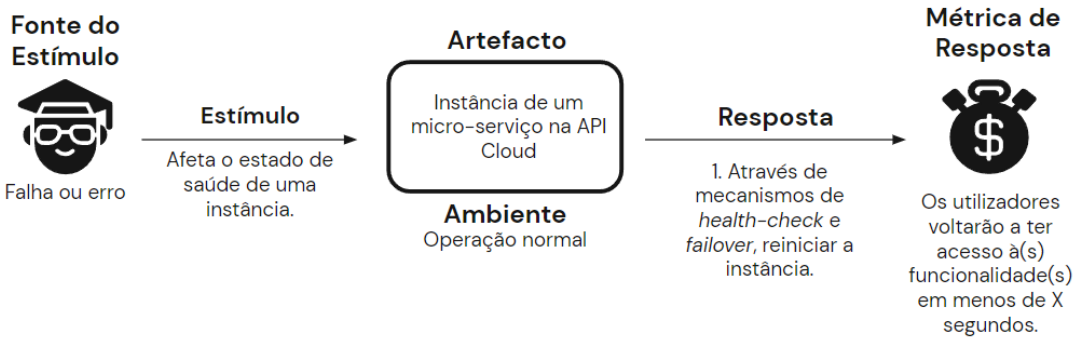

Figura 2.2: Caso de Resiliência e Tolerância a Falhas

de modo a, também, limitar o *budget* deste projeto ao menor custo possível. Na figura 2.3, pode ser visualizado o caso de escalonamento que o API *cloud* deve seguir.

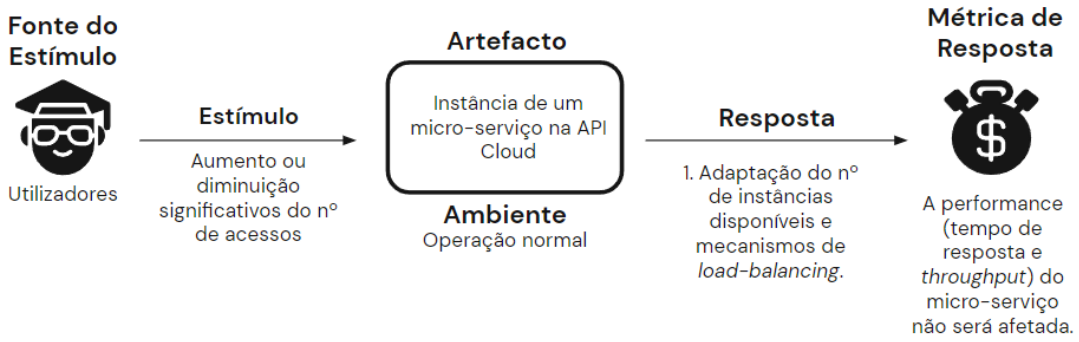

Figura 2.3: Caso de Escalabilidade

2.4 Gestão de Riscos

Nesta secção irão ser identificados e analisados riscos que podem comprometer o bom decorrer da implementação da solução do projeto. Esta identificação e análise terá uma descrição prévia das métricas necessárias para as mesmas e serão seguidas da criação de um plano de prevenção e/ou mitigação dos mesmos, ação que poderá ser fulcral para atingir o sucesso do projeto.

2.4.1 Métricas

Tanto a probabilidade de um risco ocorrer, como o seu impacto, podem ser divididos em três métricas: **elevada**, **média**. ou **baixa**. No entanto, há que especificar o significado de cada uma destas métricas em cada um dos casos.

Probabilidade do risco ocorrer

- **Baixa**: A probabilidade do risco ocorrer é inferior ou igual a 33%;
- **Média**: A probabilidade do risco ocorrer encontra-se entre 34% e 66%, inclusivamente;
- **Elevada**: A probabilidade do risco ocorrer é igual ou superior a 67%.

Impacto do risco

- **Baixo**: Os objetivos do projeto poderão ser atingidos facilmente, caso o risco ocorra;

- **Médio**: Os objetivos do projeto poderão ser atingidos, caso o risco ocorra;

- **Elevado**: Haverá dificuldade em atingir os objetivos do projeto, caso o risco ocorra.

É ainda necessário definir uma matriz de riscos, com zonas de risco que definem se é necessária a criação de um plano de mitigação para cada risco. Essas zonas estarão identificadas pelas cores vermelho e amarelo na figura 2.4.

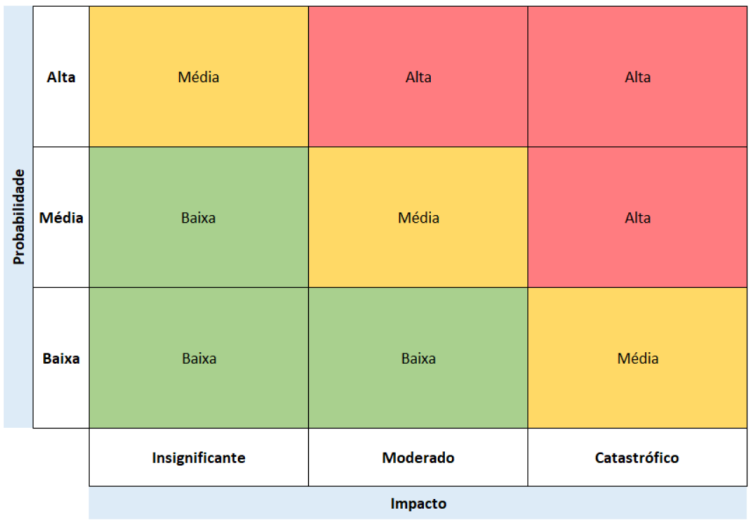

Figura 2.4: Matriz de riscos [14]

2.4.2 Identificação e análise dos riscos

O passo seguinte é a identificação e análise dos possíveis riscos de acordo com as métricas já referidas.

- **1. Tentativas de acesso indevido (Segurança)**: utilizadores tentam aceder a serviços sem permissões, de forma a obterem dados ou tirarem partido dos mesmos:
  - **Probabilidade**: média;
  - **Impacto**: alto;
  - **Plano de contingência**: aplicar na arquitetura características como autenticação e/ou autorização, isolamento dos micro-serviços (menor número de comunicação entre os mesmos possível), *firewall*, encriptação dos dados em transmissão (protocolos), e *gateway* de entrada;
  - **Plano de mitigação**: incluir na API produtos de *auditing* e *logging*.
- **2. Serviço ficar indisponível (Resiliência)**: a instância de um serviço não estar disponível ou em condições corretas de funcionamento, devido a erros/falhas:
  - **Probabilidade**: média;
  - **Impacto**: alto;
  - **Plano de contingência**: inclusão de mecanismos de *health check*, orquestração de *containers* e *failover*, na arquitetura.
- **3. Exceder o *budget* (Escalonamento)**: o estagiário não possui muita experiência no uso de tecnologias *cloud*, ou da linguagem C#, podendo, então, exceder algum limite imposto pela empresa:
  - **Probabilidade**: média;
  - **Impacto**: médio;
  - **Plano de mitigação**: durante a definição da arquitetura, efetuar uma análise de custos, e definir um limite de pedidos/clientes para efetuar testes, de modo ao projeto não ter custos excessivos (sendo o projeto a conceber apenas a uma prova de conceito). Em simultâneo, tentar optar por utilizar serviços gratuitos disponibilizados pelo *cloud provider*.

2.5 Planeamento

2.5.1 Primeiro Semestre

O plano de trabalhos para o primeiro semestre consistiu em:

- Introdução ao M1;
- Introdução à linguagem C#;
- Introdução e análise das tecnologias *cloud*;
- Elaboração do relatório intermédio e seleção do *cloud provider* mais adequado, devendo o mesmo assegurar *performance*, segurança e escalabilidade.

2.5.2 Segundo Semestre

O plano de trabalhos para o segundo semestre consistiu nas seguintes tarefas:

- Definição dos requisitos funcionais do projeto;
- Estudo dos serviços da AWS a usar (aprimoramento da arquitetura);
- Desenvolvimento da API *cloud* que permitirá aos M1 e outros EHR obter a lista de eventuais alertas para uma determinada combinação de medicamentos e solventes;
- Desenvolvimento do editor de conteúdos da informação de base aos sistemas de alerta e apoio à decisão;
- Elaboração do relatório final.

De forma a apresentar os planos de tarefas de forma mais visual, o autor definiu o seguinte gráfico de Gantt, visível na figura 2.5, com a escala de datas aproximadas referentes a cada tarefa.

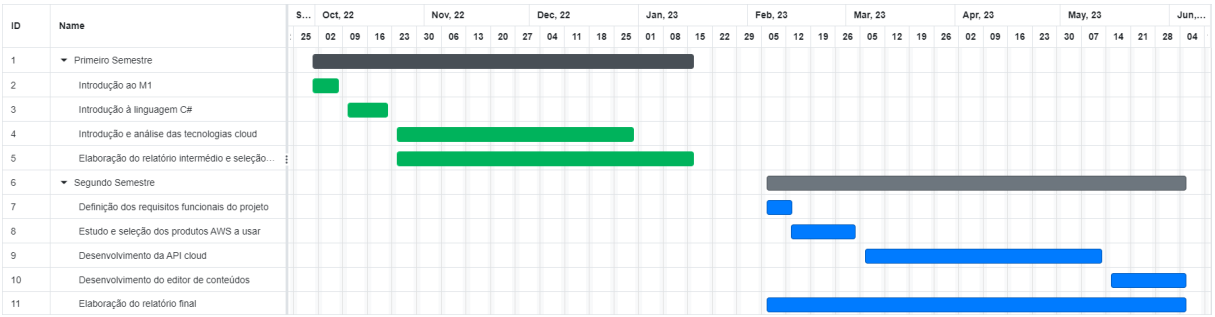

Figura 2.5: Plano de Trabalhos

Capítulo 3

Conceitos Introdutórios sobre *Cloud* e Estado da Arte

3.1 Conceitos Introdutórios sobre *Cloud*

Um *Cloud Service Provider*, ou apenas *cloud provider*, consiste numa empresa cujo negócio se baseia na disponibilização de uma plataforma *cloud*, ou seja, disponibiliza servidores *online* onde pode ser introduzido *software* de clientes para que o mesmo esteja disponível para utilizadores em várias partes do globo [15]. Estes serviços têm vários métodos de pagamento de custos para os clientes, e modelos de adesão para os utilizadores do *software* publicado nestes servidores. Os serviços disponíveis por estas empresas consistem em:

3.1.1 IaaS

É um serviço *cloud* que disponibiliza recursos de computação, armazenamento, navegação e de rede (ferramentas como *firewalls*, *routers* virtuais, *software* de gestão de largura de banda de rede), através de um método de pagamento, como por exemplo, *pay-as-you-go*, ou seja, o cliente paga pelo serviço previamente, sendo que o uso deste serviço apresenta certos limites definidos. Assim que estes limites são atingidos, o cliente deixa de poder usufruir dos mesmos até efetuar um novo pagamento [16];

3.1.2 CaaS

É um serviço de *cloud* que se foca na gestão e no *deployment* de aplicações utilizando *containers* que envolvem o código, bem como todas dependências necessárias para a sua execução. Este serviço pode auxiliar a aumentar os níveis de segurança, de escalonamento, mas principalmente de portabilidade da aplicação em causa. Exemplos de ferramentas que permitem este serviço são o Docker e o Google Kubernetes. Container as a Service (CaaS) encontra-se definido como algo entre o Infrastructure as a Service (IaaS) e o Platform as a Service (PaaS) [17].

3.1.3 PaaS

Consiste na disponibilização de um ambiente completo de desenvolvimento e distribuição (*deployment*) para aplicações quer de complexidade baixa ou elevada. Além de incluir os mesmos recursos de infraestrutura que o IaaS, inclui também recursos como ferramentas de desenvolvimento, *middleware* e serviços de gestão de bases de dados, permitindo assim sustentar todo o processo de desenvolvimento e ciclo de vida de aplicações *web* [18];

Kubernetes (K8s)

Antes de abordar especificamente este tema, é necessário referir que este não consiste num sistema tradicional de PaaS, apesar de apresentar algumas semelhanças. Originalmente, o *deployment* de aplicações era efetuado para servidores físicos, o que poderia levar a complicações relativamente à gestão de recursos, isto é, situações em que aplicações ocupavam a maior parte dos recursos do servidor, o que resultava em *downtime* das restantes aplicações.

A primeira solução criada para este problema foi a virtualização, ou seja, a jutilização de Máquinas Virtuais (VMs) que auxiliassem na criação e *deployment* e hospedassem e dividissem os seus recursos entre cada aplicação. A segunda solução que viria a auxiliar neste problema foi a criação de *containers*. Estes são semelhantes a VMs, no sentido em que cada um o seu próprio sistema de ficheiros, parte de memória e CPU. No entanto, *containers* permitem a partilha do mesmo Sistema Operativo (SO) entre as aplicações. Outra vantagem do uso de *containers* é a sua independência de infraestrutura, sendo portáteis para distribuição entre Sistemas Operativos (SOs) e *cloud*.

É aí que entra a Kubernetes (K8s)[19], uma plataforma que gere *containers* como unidades únicas de uma aplicação, de forma a evitar problemas como *downtime*. Além disso, K8s trata ainda de gerir *containers* de forma a automatizar:

- *Load-balancing*;
- Escalonamento;
- *Failover*;
- Orquestração de armazenamento;
- *Self-healing*;
- entre outros aspetos importantes.

3.1.4 SaaS

Permite conectar aplicações baseadas em *cloud* através da Internet, tais como serviços de email, calendário, Microsoft 365, entre outros, possibilitando a empresas alugar esses serviços. Toda a infraestrutura, *middleware*, dados e *software* estão alocados em centros de dados do *provider* do serviço. A única preocupação do cliente, é a distribuição dos mesmos pelos utilizadores. Poderão, no entanto, existir vários níveis de contratos, com diferentes níveis de disponibilidade e/ou segurança [20].

3.1.5 *Serverless Computing*

O *Serverless Computing* visa facilitar o desenvolvimento de aplicações, removendo do cliente toda a componente de gestão de infraestrutura e tornando-a da responsabilidade do *cloud provider*. O *software* é executado, obviamente, por um servidor, que apenas retira as tarefas de gestão de infraestrutura aos *developers*, podendo os mesmos focarem-se noutras tarefas, aumentando a sua produtividade. Assim, a escalabilidade do *software* é dinâmica e a gestão dos recursos é mais eficiente [21].

3.1.6 *Hybrid Cloud Approach*

Contraditoriamente ao que o nome indica, o método híbrido de *cloud* não implica o uso de serviços de vários *cloud providers*. Esse conceito é conhecido como *hybrid multicloud*, uma derivação do mesmo. *Hybrid Cloud* implica o uso de serviços de *cloud* públicos [22], serviços de *cloud* privados [23], e uma infraestrutura *on-premise*, sendo os dois últimos reservados para um único cliente.

3.2 Estado da Arte

Neste capítulo, irão ser efetuadas duas análises: uma **análise ao estado da arte**, onde serão apresentadas as soluções existentes para o problema da informação de incompatibilidade entre medicamentos e uma **introdução relativa aos principais *cloud providers*** do mercado. Por fim, serão apresentadas algumas *guidelines* definidas para uso no projeto, mas principalmente, para guia da pesquisa necessária.

3.2.1 Mercado Atual

Sistemas de apoio à decisão clínica não são um conceito novo. Existem plataformas que implementam e distribuem produtos com o intuito de auxiliar profissionais de saúde em decisões, como diagnósticos. Livros e bases de dados como o "Handbook on Injectable Drugs"de Lawrence A. Trissel [24], são considerados como referência para o tema de interações entre medicamentos, e onde se baseiam os sistemas da American Society of Health System Pharmacists (ASHP) [25] para enfermeiros. São exemplos produtos como o UpToDate [26], o DynaMed [27] e o BMJ Best Practise [28].

Não há muita informação sobre produtos como estes presentes na Internet, mas é possível, no entanto, comprovar que a maioria destes produtos contém aconselhamento clínico para, pelo menos, trinta especialidades médicas. O DynaMed, por exemplo, tem presente no seu *website*, uma secção equivalente ao módulo de perfusões da M1, com as interações entre medicamentos. No entanto, não há referência a este se limitar/focar apenas em medicamentos injetáveis e não é possível o acesso a estas informações sem subscrição à plataforma. De acordo com um artigo do Jornal Médico [29], é defendido que a informação do DynaMed é bastante detalhada e de confiança, e baseada em artigos científicos publicados diariamente. Neste artigo consta que, a cada cinco minutos, é publicado um novo artigo científico sobre medicina.

Ainda falando sobre o atual mercado, é de relembrar que, a MedicineOne já possuiu uma parte do mercado público de centros de saúde, relativamente a *software* clínico, e que foi forçada a adaptar-se ao setor privado devido ao reforço do estado português a estes estabelecimentos, com *software* adquirido ou desenvolvido pela mesma entidade. Este fator parece ser determinante na definição do mercado alvo desta adaptação do módulo de perfusões a um ambiente *cloud native*, mas não o é.

Sendo que o módulo referido terá um modelo de subscrição, os clientes podem adquirir acesso ao módulo e integrá-lo nas suas próprias plataformas, poupando custos de manutenção do seu próprio produto. É da responsabilidade da MedicineOne adaptar o valor da subscrição e manter a qualidade do serviço em questão, entre outros fatores, de forma a ggarantir a oportunidade de ser um forte concorrente a plataformas já existentes, que são privadas, ou seja, não são adaptáveis a serem integradas noutros produtos e/ou EHR.

3.2.2 Principais *Cloud Providers*

No mercado atual de *cloud providers*, há três empresas dominantes em relação às restantes: a **Amazon** com o seu produto **AWS**, a **Microsoft**, com o **Microsoft Azure** e a **Google** através do **GCP**. Estas três empresas representaram, em 2022, 66% dos rendimentos do mercado de *cloud providers*, existindo, ainda assim, uma diferença significativa entre as mesmas [30].

3.3 *Cloud Providers*

Nesta secção, serão efetuadas **duas análises** distintas, mas coincidentes no seu objetivo: uma comparação extensiva dos três *cloud providers* líderes de mercado, guiada pelas *guidelines* introduzidas na secção 3.3, e uma análise mais sucinta a outros dois *cloud providers* que foram considerados

como possíveis alternativas para a implementação do ambiente *cloud* para o módulo de perfusões do M1.

3.3.1 *Cloud Providers* Líderes do Mercado

Nesta secção do documento será feita uma comparação extensiva dos três *cloud providers* líderes do mercado, pertencentes à Amazon [31], Microsoft [32] e Google [33]. No entanto, é necessário introduzir inicialmente os *cloud providers* individualmente, apresentado algumas das suas mais importantes características.

AWS

A AWS é a plataforma de serviços de computação *cloud* pertencente à Amazon e é, atualmente, a líder de mercado, e pelo décimo ano consecutivo. Possui mais de 200 *datacenters* por todo o mundo e 96 **zonas de disponibilidade**, que são *datacenters* isolados e independentes. Orquestram dentro de uma região geográfica específica e estão divididos em 30 regiões. Estas zonas de disponibilidade facultam os serviços AWS a 145 países. Na figura 3.1 são visíveis, a verde, as regiões geográficas onde a Amazon já construiu zonas de disponibilidade, e a vermelho, as próximas a serem criadas.

Figura 3.1: Regiões geográficas com zonas de disponibilidade AWS [34]

Azure

A Azure é a plataforma de serviços de computação *cloud* da Microsoft, a segunda classificada na corrida ao mercado de *cloud providers*. No modelo da Azure, cada região tem no mínimo três zonas de disponibilidade, com o objetivo de aumentar a resiliência dos serviços. A Microsoft, atualmente, tem os seus serviços em 60 regiões, e possui mais de 180 zonas de disponibilidade, cobrindo apenas, no entanto, cerca de 140 países.

GCP

Por último, o terceiro líder do mercado de *cloud computing*, o GCP, pertencente à Google. Das três empresas, a Google foi a última a juntar-se ao mercado em questão. Neste caso, o conceito de zonas de disponibilidade denomina-se apenas por zonas, mas por efeito de facilidade para o leitor, o autor continuará com a terminologia usada até agora. A Google também pretende atingir o objetivo de possuir três zonas de disponibilidade por região, tal como a Microsoft. Neste momento, existem 106 zonas de disponibilidade divididas em 35 regiões. Estas podem ser visualizadas na figura 3.2.

Figura 3.2: Regiões geográficas com zonas de disponibilidade GCP [35]

3.3.2 AWS vs. Azure vs. GCP

Virtual Private Cloud (VPC)

Dentro de cada zona de disponibilidade existem VPCs [36] ou, no caso da Azure, VNet, que são serviços de hospedagem de *software* privados, ou seja, não estão na *cloud* pública. VPCs podem comunicar entre regiões através de um processo denominado VPC *peering*, sendo a única condição que cada VPC tenha blocos Classless Inter-Domain Routing (CIDR) diferentes, de forma a permitir a criação, comunicação e partilha de dados de instâncias do mesmo *software* em regiões diferentes. Os *cloud providers* implementam este conceito de forma um pouco diferente dos restantes, no entanto, o conceito é o mesmo. No AWS e Azure, VPCs existem dentro de regiões e contêm *subnets*. No AWS, como camada a envolver as *subnets*, distinguidas pelo seu acesso à internet como públicas ou privadas, encontram-se as referidas zonas de disponibilidade. Já no GCP, a VPC é global, contendo regiões que, por sua vez, contêm *subnets*, sendo que todas as instâncias e recursos podem comunicar entre si. Esta informação pode ser melhor analisada nas figuras 3.3, 3.4 e 3.5.

As ligações entre redes virtuais, nos três *cloud providers*, não permitem *transitive peering*, ou seja, se a VPC A comunica com a VPC B e a VPC B comunica com a VPC C, então a VPC A não pode comunicar com VPC C. Assim, o uso de *gateways* é introduzido para melhorar o sistema de comunicação.

Figura 3.3: Estrutura de rede AWS [37]

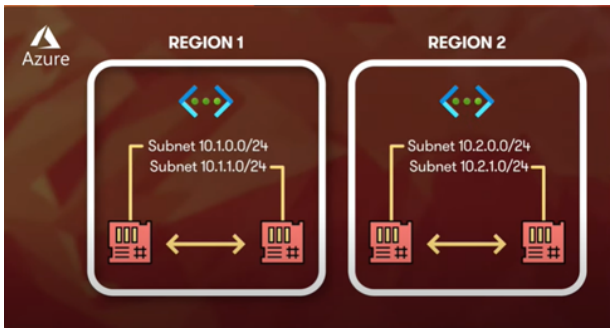

Figura 3.4: Estrutura de rede Azure [37]

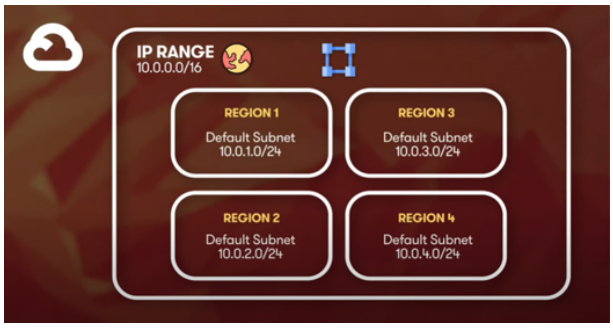

Figura 3.5: Estrutura de rede GCP [37]

No AWS é implementado um *transitive gateway* para ligar VPCs dentro de uma região, e implementado *inter-region peering* permite a comunicação de VPCs entre diferentes regiões. Para ligações a aplicações externas à *cloud*, ou seja, *on-premise*, são utilizadas ligações Virtual Private Network (VPN). Esta última forma de comunicação pode ser visualizada na figura 3.6.

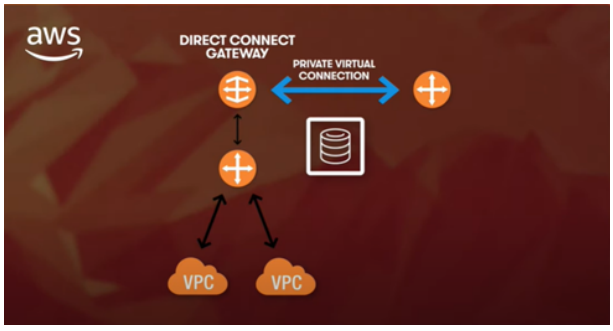

Figura 3.6: ligação *on-premise* AWS [37]

O processo é bastante semelhante no Azure, onde uma VNet é utilizada como HUB, ou *network switch*, que efetua as comunicações, quer entre redes como para o exterior. Para comunicações *on-premise*, é também utilizada uma ligação VPN. Para ligações a circuitos privados *third-parties*, o Azure disponibiliza o serviço ExpressRoute [38], com ligações redundantes e seguras.

No GCP o cenário muda um pouco, uma vez que VPCs estão contidas num conceito denominado de projetos e apenas podem comunicar com VPCs de outros projetos através de VPC *peering*. No entanto, existem VPCs partilhadas, que podem ser acedidas em vários projetos. O GCP oferece dois *gateways* para comunicações: Cloud VPN para ligações seguras e públicas e Cloud Interconnect [39], equivalente ao ExpressRoute do Azure.

Load Balancing

Load balancing é um conceito que visa o equilíbrio de pedidos efetuados a um sistema com várias instâncias, de forma a evitar a sobrecarga do mesmo. Sendo que o futuro que a MedicineOne visa para este projeto considera a possibilidade de o disponibilizar a uma escala global, este é um conceito que deve ser analisado para a escolha do *cloud provider* a selecionar. De notar que todos os *cloud providers* em análise têm um sistema de Domain Name Service (DNS).

Tabela 3.1: Tabela de protocolos para *load balancing*, AWS vs. Azure vs. GCP

AWS		**Azure**		**GCP**	
Network Load Balancer	Layer 4 TCP/SSL	Azure Load Balancer	Layer 4 TCP/UDP	Interno	Regional, Pass-through, TCP ou UDP, Proxy HTTP (S)
Application Load Balancer	Aplication Layer – Layer 7	Azure Application Gateway	Aplication Layer - Layer 7		

É de salientar, na tabela 3.1, os processos de *load balancing* de rede e da aplicação do GCP, ou seja, apenas para clientes da mesma VPC em que se encontra o *software*.

Bases de dados

Em ambientes *cloud*, os serviços de bases de dados podem ser considerados IaaS, PaaS ou Software as a Service (SaaS), diminuindo o nível de configurações e responsabilidades por esta ordem. O motor de base de dados do M1, de momento, é o Microsoft SQL Server, uma base de dados relacional, que pode ser usada como IaaS, utilizando serviços como Amazon Elastic Compute Cloud (EC2) [40], Azure Virtual Machines [41] ou o Compute Engine [42]. Pode também ser usado como PaaS, através de serviços como o Amazon Relational Database Services (Amazon RDS) [43] ou Azure SQL Managed Instances. Assim, o primeiro passo será analisar os três *cloud providers* quanto aos seus serviços de bases de dados em formato PaaS. De notar que não está a ser considerado *cloud-native* PaaS, visto que o M1 não foi desenvolvido para ser executado em apenas em ambientes *cloud*.

Tabela 3.2: Apoios ao SQL Server

	AWS	**Azure**	**GCP**
Microsoft SQL Server	✓ Tutoriais Compreensíveis ✓ Licensa de Mobilidade	✓ Tutoriais Compreensíveis ✓ Licensa de Mobilidade ⊕ Benefícios Híbridos de Azure (Auto-Patch & Auto-Packup)	✓ Tutoriais Compreensíveis ⊗ Licença de Mobilidade

Assim sendo, o primeiro ponto a analisar são os apoios ao motor da base de dados em questão, o SQL Server, na tabela 3.2. Como é de esperar, e imediato de comprovar pela na tabela seguinte, o Azure tem vantagem sobre os restantes *cloud providers*, visto ser um motor da Microsoft. De salientar que o GCP não possui licença de mobilidade para este motor de base de dados.

Tabela 3.3: Monitorização e Disponibilidade

	AWS	Azure	GCP
Monitorização	Cloudwatch	Azure Monitor SQL Insights SQL Analytics	Cloud Logging Cloud Monitoring
Visão de performance	Performance Insights	Intelligent Insights (SQL Server) Query Store Query Performance Insights Slow Query Logs	Query Insights
SLA de disponibilidade	99.95%	99.99%	99.95%
Failover Intra-Região	Automático	Automático	Automático
Failover Inter-Regiões	Automático	Automático	MySQL PostgreSQL

Quanto à **monitorização** e **visão da *performance*** das bases de dados, todos os *cloud providers* oferecem serviços responsáveis por esta operação, tal como é possível visualizar na tabela 3.3. Contudo, apenas o GCP oferece serviços para casos de *failover* inter-regiões para SQLServer e PostgreSQL, ou seja, não abrange bases de dados em SQL Server. O Azure oferece um nível de **disponibilidade** das suas bases de dados superior (0.04% é uma medida considerável quando se analizam estes valores de disponibilidade e a importância do serviço em questão). Estes dados podem ser verificados na tabela 3.3.

Tabela 3.4: Escalabilidade dos serviços de bases de dados

	AWS	Azure	GCP
CPU (VCore)	96	80	96
RAM (GB)	384	504	624
Armazenamento (TB)	16 (SQL Server)	8 (SQL Server) Auto Crescimento	30 Auto Crescimento
Downtime de Escalonamento Vertical	Alguns minutos	Segundos (SQL Server) 1-2 min. (Não SQL Server)	Até 5 minutos

Em termos de escalabilidade dos serviços de base de dados, todos os *cloud providers* têm as suas especificações com vantagens e desvantagens, podendo ser analisados na tabela 3.4. O AWS oferece o serviço com menos memória de processamento e não tem crescimento automático da memória total de armazenamento. No entanto, tem mais memória que o serviço da Azure. Além disso, o *downtime* em casos de *failover* de estruturas com escalonamento vertical é superior ao o do Azure, quando se fala de bases de dados SQL Server.

Tabela 3.5: *Backups* das bases de dados

	AWS	Azure	GCP
Backups Automáticos	Diários	Semanais com diferenciais de 12 horas 5-10 min. de logs de transações	Diários
Retenção de Backups	35 dias	35 dias (default) 10 anos (retenção a longo prazo, SQL Server)	365 dias
Restauro "Point-in-Time"	35 dias	35 dias <1h (Geo-Restore)	7 dias (PostgreSQL, MySQL)

Em termos de **backups** dos dados e estados das bases de dados, o AWS e o GCP oferecem um serviço diário e automático, enquanto que o Azure ocorre semanalmente, como ergistado na tabela 3.5. No entanto, para SQL Server, o Azure oferece uma retenção até 10 anos, uma quantidade de tempo muito superior à dos seus concorrentes. Esta informação pode ser visualizada na tabela 3.5.

Como visível na tabela 3.6, os serviços de segurança das bases de dados têm praticamente as mesmas especificações entre os três *cloud providers*, com o fator negativo que no AWS não se

Tabela 3.6: Segurança das bases de dados

	AWS	Azure	GCP
Firewall & VNET	✓	✓	✓
Encriptação em trânsito	✓	✓	✓
Encriptação em Rest	✓	✓	✓
Encriptação do lado do cliente	⊗	✓	✓
Audit Logging	✓	✓	✓
Autenticação Cloud	✓ IAM através da Amazon RDS Proxy ✓ AD Microsoft Gerida	✓ Azure RBAC ✓ Azure AD	✓ Cloud IAM através do Cloud SQL Proxy
Aconselhamento Integrado	⊗	✓ Azure Defender ✓ Data Discovery & Classification	⊗

encontra encriptação do lado do cliente, e o fator positivo para o Azure que contém serviços de aconselhamento integrados nos seus produtos.

Computação

Os serviços de computação *cloud* permitem hospedar e executar aplicações na *cloud*, retirando dos programadores responsabilidades como gestão do *hardware*, oferecendo maior segurança do *software* e dos seus dados, assim como facilitando o acesso do produto ao mercado. As categorias destes serviços foram definidas posteriormente neste documento (IaaS, PaaS e SaaS). Para este projeto, irá ser implementada uma solução CaaS, de modo a facilitar o escalonamento da aplicação.

Os serviços PaaS mais conhecidos do AWS, Azure e GCP, são, respetivamente, o Elastic Beanstalk[40], Virtual Machines [44] e o Compute Engine [42]. Todos estes serviços suportam os SOs Windows Servers e Linux.

Relativamente a **escalonamento**, o EC2 oferece escalonamento horizontal automático (EC2 Auto Scaling) e *features* como a Fleet Management, que automaticamente substitui instâncias indisponíveis (ou em estados de má operação), ou o Predictive Scaling, que adapta o número de instâncias de acordo com a análise de picos de uso em determinadas horas do dia. No Compute Engine, existem conjuntos de Máquina Virtual (VM) idênticas, localizadas numa ou várias zonas dentro da mesma região chamados Managed Instance Groups (MIGs), que permitem ações de escalonamento e podem ser geridos como uma única entidade. Também suportam escalonamento automático e agendamento de escalonamento. Possui ainda uma *feature* denominada *autohealing* que recria VMs que falham os testes de bom funcionamento. No Azure, os grupos de VM denominam-se Scale Sets e também incluem escalonamento automático, baseado no uso das VM. Também existe escalonamento agendado e um sistema de recuperação do número de instâncias, tal como o *autohealing*, denominado Instance Repair.

A tabela 3.7 contém as opções de otimizações das VM dos três *cloud providers* em análise. A partir desta, apenas se pode assumir uma desvantagem para o GCP, uma vez que não contempla a opção de otimizar o armazenamento nas VMs do Compute Engine.

Todos *cloud providers* apresentados têm planos de redução de **custos** das VMs, com compromissos de uso de um ou três anos. No AWS e no Azure, estes compromissos chamam-se Reserved Instances, e no GCP denominam-se Commited Use Discounts. No GCP, os descontos podem chegar a 70%.

Tabela 3.7: Otimizações das VMs dos cloud providers

Opções de Otimização	**Computação**	**Memória**	**Processamento Acelerado**	**Armazenamento**	**Explosividade**
AWS	✓	✓	✓	✓	✓
Azure	✓	✓	✓	✓	✓
GCP	✓	✓	✓	⊗	✓

Disponibilidade

Relativamente à **disponibilidade**, de acordo com os Service Level Agreement (SLA) dos *cloud providers*, é possível criar a tabela 3.8. De notar que em serviços com apenas um servidor, a Amazon garante muito menos disponibilidade comparativamente aos seus concorrentes.

Tabela 3.8: Disponibilidade de acordo com os SLA

	AWS	**Azure**	**GCP**
Multi-Availability Zone SLA	99.99%	99.99%	99.99%
Single Server Availability SLA	90% (por hora)	99.9% (SSD premium) 99.5 (SSD)	99.5%

Segurança

Os modelos IaaS, PaaS e SaaS, implicam responsabilidades diferente, facto que se mantém quando se refere ao tópico de segurança em ambientes de *cloud*.

Na tabela 3.9, encontram-se as propriedades pelos quais os *cloud providers* se responsabilizam por assegurar medidas de segurança, nos diferentes modelos de serviços *cloud*. Os três *cloud providers* em estudo tomam abordagens um pouco diferentes entre si, no que diz respeito à identificação de responsabilidades de segurança. No entanto, seguem os princípios da tabela anterior. Nas próximas três figuras, 3.7, 3.8 e 3.9, podem ser observadas as abordagens referidas.

Tabela 3.9: Modelo de Responsabilidade Partilhada

	IaaS	**PaaS**	**SaaS**
Utilizadores	⊗	⊗	⊗
Dados	⊗	⊗	⊗
Aplicações	⊗	⊗	✓
Sistema Operativo	⊗	✓	✓
Redes Virtuais	⊗	✓	✓
Hypervisors	✓	✓	✓
Servidores e Armazenamento	✓	✓	✓
Redes Físicas	✓	✓	✓

Figura 3.7: Modelo Responsabilidade Partilhada AWS [45]

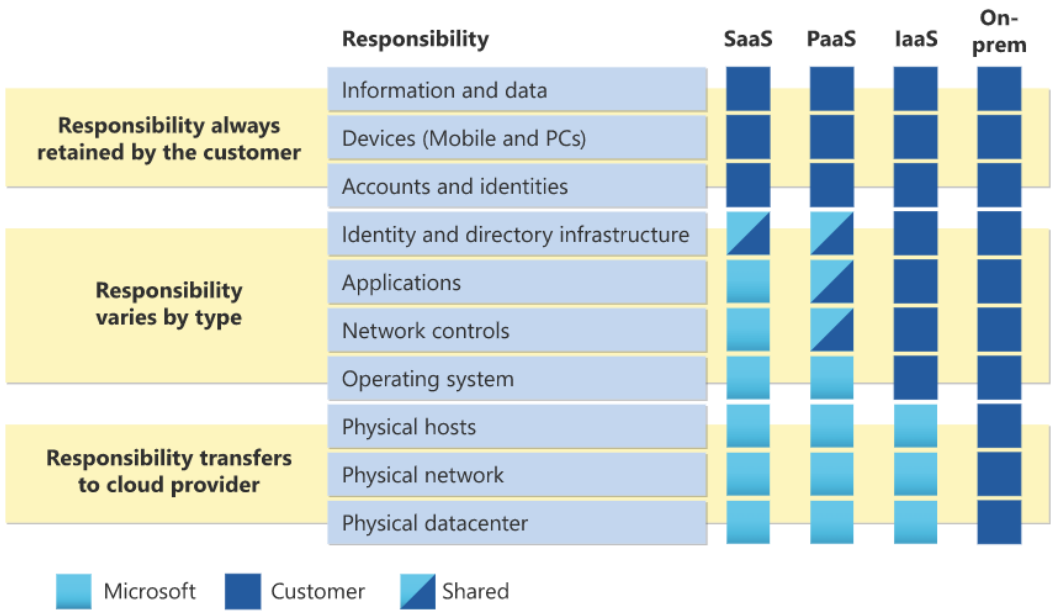

Figura 3.8: Modelo Responsabilidade Partilhada Azure [46]

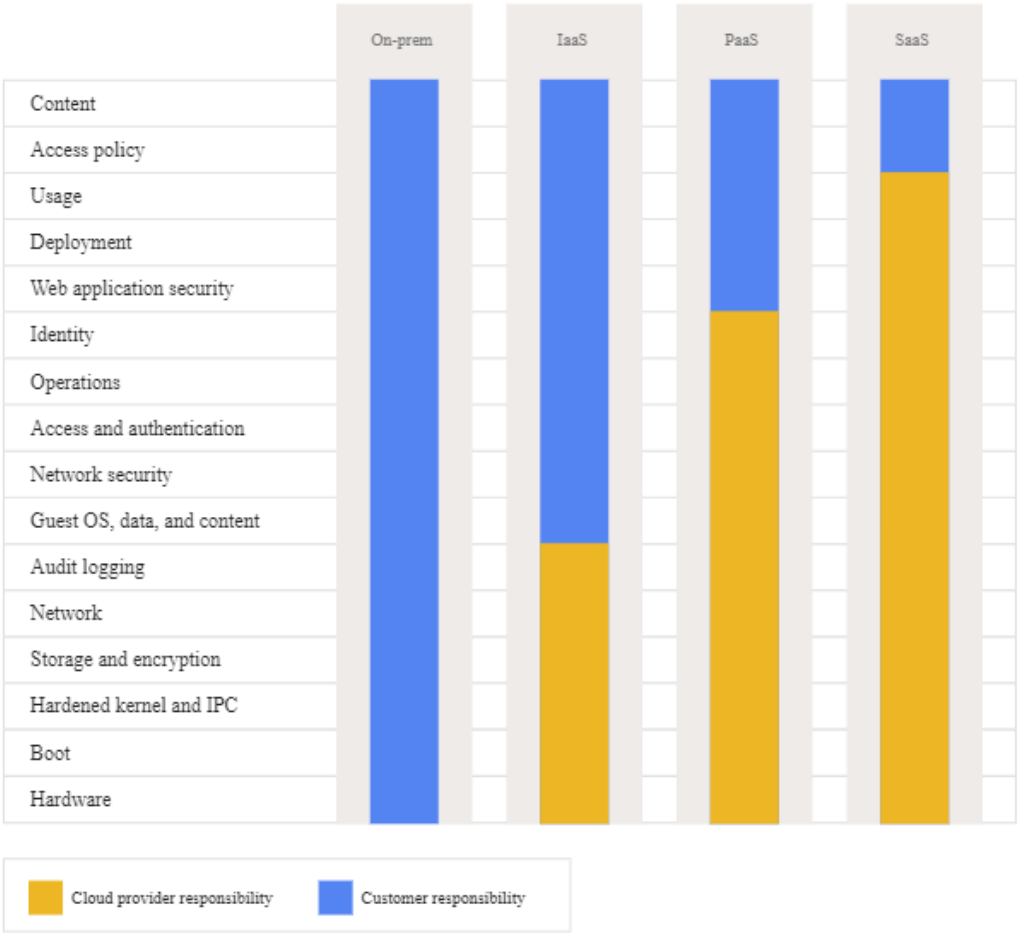

Figura 3.9: Modelo Responsabilidade Partilhada GCP [47]

Como é visível nas três figuras anteriores, as entidades donas dos *softwares* hospedados em ambientes *cloud* são responsáveis pelas contas dos utilizadores. No tópico de segurança, o termo a utilizar para esta responsabilidade é **gestão de acessos e identidade**. Existem vários produtos que visam garantir que as aplicações são seguras nos termos deste tópico e estas encontram-se presentes nos *cloud providers* em estudo, tais como:

- Autenticação Multi-Fator;
- Sessão única;
- Acessos baseados nas funções (do utilizador);
- Funções personalizadas.

No entanto, a gestão de acessos privilegiados varia quando se analisam diferentes *cloud providers*, sendo que nem o AWS nem o GCPapresentam qualquer funcionalidade para esta gestão. Terá então de ser introduzida por soluções terceiras através do *marketplace* do respetivo *cloud provider*. No Azure, a solução que trata esta funcionalidade é denominada Privileged Identity.

Em termos de **segurança dos serviços**, os *cloud providers* oferecem funcionalidades que cobrem vários aspetos. Para **proteção contra ataques Distributed Denial of Service (DDoS)**, o AWS oferece um produto denominado AWS Shield [48], enquanto que os produtos equivalentes do Azure e GCP são, respetivamente, o DDoS Protection [49] e o Google Cloud Armor [50]. Todos estes produtos têm por base os mesmos princípios. Quanto à **gestão de segredos**, os produtos do AWS, Azure e GCP são, respetivamente, o Secrets Manager [51], o Key Vault [52] e o Secret Manager [53]. Todos estes produtos permitem o armazenamento de "segredos"como palavras-passe, chaves de segurança e certificados digitais. Relativamente a **VPNs**, tanto o AWS como o Azure, suportam ligações Point-to-Site (P2S) como Site-to-Site (S2S), sendo o limite de ligações por portal VPN de 10 no AWS e 30 no Azure. Já o GCP apenas suporta ligações P2S.

Como ainda não foi determinado se a base de dados vai ser implementada num modelo IaaS ou PaaS, que é possível através do uso conjunto de ferramentas como o EC2 e o Amazon RDS, convém ainda analisar algumas vantagens de segurança quando a responsabilidade sobre os dados das organizações ou dos utilizadores é do *cloud provider*, ou seja, quando a base de dados está hospedada num produto PaaS como o Amazon RDS. Todos os *cloud providers* analisados possuem os seguintes protocolos de segurança implementados, neste tipo de produtos:

- Políticas de identidade e acesso;
- Regras de *firewall* incluindo lista branca de Internet Protocol (IP), ou seja, permissões de acesso para IP específicos, podendo assim permitir acessos apenas a IP públicos de organizações, por exemplo;
- Encriptação em trânsito, para verificar se as ligações às bases de dados são seguras;
- Encriptação em descanso, permitindo à base de dados *encription address*.

Todos os *cloud providers* possuem ainda produtos SaaS de serviços embutidos de segurança e conformidade, e que se regem pelos mais conhecidos padrões de conformidade como o ISO 27001 [54]. O AWS possui o Amazon Inspector [55], o Azure possui o Azure Security Center [56] e o GCP possui o Security Command Center [57], sendo que todos eles são produtos de *auditing* e aconselhamento de políticas de segurança a aplicar no ambiente *cloud* que estão a analisar.

Containers

Este conceito, introduzido previamente neste documento, consiste, resumidamente, de *packages* leves com todas as dependências necessárias para executar o *software* que o próprio *container* também envolve. Os três *cloud providers* possuem motores de *containers* para execução dos mesmos. É necessário, primeiramente, criar imagens (ficheiros estáticos de código executável) dos mesmos

e, seguidamente, registar essas imagens no *container registry*, que as controla e às suas versões. Exemplos destes *registries* são o AWS Elastic Container Registry (ECR) [58], o Azure Container Registry [59] e o GCP Artifact Registry [60]. No entanto, os *containers* utilizados de forma independente apresentam limitações, não possuindo replicação, auto-escalonamento, *auto-healing* ou *load-balancing*. É possível, no entanto, o uso de *Container Cluster Orchestrations*, como K8s, em conjunto com produtos de IaaS, como o EC2. Na tabela 3.10 podem ser visualizadas as soluções de K8s dos diferentes *cloud providers*, assim como algumas das suas propriedades.

Tabela 3.10: Produtos de K8s

	AWS **Elastic Kubernetes Service (EKS) [61]**	**Azure** **Azure Kubernetes Service (AKS) [62]**	**GCP** **Google Kubernetes Engine (GKE)[63]**
Upgrades	Alguns passos manuais	Automáticos	Automáticos
Nós de Clusters	VM, Unidades de Processamento Gráfico (GPUs) Bare Metals [64]	VM e GPUs	VM e GPUs
Linha de Comandos	Suporte Parcial	Suportada	Suportada
Mesh de Serviços [65]	App Mesh	⊗	Istio [66]
Nós	100	500	5000

Serviços e *Features*

Não estendendo muito este tópico, é necessário mesmo assim denotar a existência de alguns serviços, comuns entre todos os três *cloud providers*. Todos possuem Command Line Interface (CLI), vários Software Development Kit (SDK), incluindo o .NET [67], consolas (AWS Management Console, Azure The Portal e GCP Cloud Console) e ferramentas de desenvolvimento na própria *cloud* (AWS Cloud9, Azure Visual Studio Code online e GCP Cloud Code IDE).

3.3.3 Outros *cloud providers* analisados

O autor do documento foi ainda apresentado com a opção de investigar *cloud providers* além dos três mais dominantes do mercado. Assim, após uma breve pesquisa, o autor decidiu que os três outros alvos de pesquisa seriam os serviços de *cloud* da **IBM** e **Oracle**.

IBM Cloud

A International Business Machines Corporation (IBM) é uma empresa americana fundada em 1911 dedicada não só a *software* mas também a produção de *hardware* [68]. Uma das suas soluções de negócio é, precisamente, a hospedagem de *software* em ambientes *cloud* onde servem clientes de renome, nomeadamente, o Lloyds Banking Group [69] e a Coca-Cola [70].

Nos seus serviços *cloud*, a IBM tem o foco de reduzir custos e reduzir o tempo despendido pelos seus clientes na configuração e na manutenção do seu *software* [71]. Porém, estas vantagens são maioritariamente notórias na IBM Cloud Private [72], dedicada a um cliente único. Ainda assim, existem soluções *cloud* da IBM que permitem o objetivo da MedicineOne, isto é, compartilhar o seu módulo de perfusões com outras empresas que possuem EHR.

Após uma breve pesquisa, imediatamente se destaca um fator de comparação entre o IBM Cloud e o *standard* de ambientes *cloud*, o da AWS. A IBM possui um sistema de agendamento de tarefas que nenhum outro serviço *cloud* supera. No entanto, o autor não sente que este fator seja importante para a MedicineOne, ou pelo menos não o suficiente para ultrapassar a flexibilidade e a amplitude dos serviços e das ferramentas da AWS. A IBM afirma, no entanto, possuir serviços que são melhores no que diz respeito ao tratamento de *Big Data*, tendo feito inclusivamente um estudo em que refere possuir transferências de dados até três vezes mais rápidas do que a concorrência, e conseguir obter até 39% mais *throughput* [73][74].

Oracle Cloud Infrastructure

A Oracle é uma empresa multi-nacional norte-americana, e a maior empresa de gestão de dados do mundo. Na carteira de clientes incluem-se a Mazda [75] e o Deutsche Bank [76]. Esta informação não a torna digna de ser considerada como *cloud provider* para este projeto, de forma independente. A Oracle Database está disponível em diversos outros *cloud providers*, incluindo os três referidos no tópico anterior deste documento.

O modelo de negócio da Oracle Cloud Infrastructure (OCI) visa maior *performance*, custos mais baixos e maior facilidade de migração de aplicações para um ambiente *cloud* por comparaçºao aos seus concorrentes. A Oracle contém no seu próprio *website* uma referência aos seus pontos fortes em relação ao seu maior concorrente em funções de *cloud provider*, a AWS [77].

O primeiro ponto de referência é *software on-premise*, onde a Oracle evidencia a sua capacidade de oferecer os mesmos serviços, modelo de pagamento e métricas de disponibilidade nos centros de dados privados dos seus clientes, que oferece nos seus serviços públicos de *cloud*. Este fator não é muito importante de acordo com o objetivo para o M1, pelo menos num futuro próximo.

Uma vez que a Oracle é, como referido anteriormente, dona de uma base de dados de excelência, faz sentido que o segundo ponto de referência desta empresa seja assegurar melhor *performance*, escalabilidade e opções de serviços para bases de dados por comparação à concorrência. Ainda devido a este facto, a Oracle oferece Total Cost of Ownership (TCO) mais baixo para licenças da Oracle, ou até mesmo licenças próprias. Este último fator facilita a migração para a OCI e, para auxiliar, a Oracle tem ferramentas de automação para processos como este.

Como terceiro ponto de referência, a Oracle usa a segurança. Nos serviços da OCI, estão incluídas ferramentas como a OCI Security Zones - onde podem ser aplicadas regras de segurança personalizadas para transmissão e tratamento de dados de elevada importância -, ou o Oracle Cloud Guard - que analisa as definições de segurança da arquitetura e visa encontrar defeitos (a maioria dos defeitos de segurança de ambientes *cloud* são devido a erros humanos). A segurança, por predefinição, na OCI, é regida pelo lema "Zero confiança", facilitando assim o crescimento do *software* na medida em que haverá menos possibilidades de irregularidades de acessos à medida que o mesmo vai crescendo.

Ainda de referir que, apesar um dos objetivos da Oracle ser a redução de custos comparativamente a outros *cloud providers*, também se compromete a manter a qualidade dos serviços de rede, armazenamento e computação, sendo a *performance* dos mesmos consistente e previsível. A Oracle refere ainda que, através de um sistema de recompensas, um terço do dinheiro investido na manutenção dos seus serviços *cloud* é devolvido. De notar que, o preço dos serviços da Oracle é constante e independentemente da região do mundo em que o *software* está alocado.

3.4 Sumário e Discussão do capítulo

Primeiramente, é de reforçar a importância dos conceitos IaaS , PaaS e CaaS, uma vez que são conceitos aplicáveis a um ambiente *cloud* para o módulo de perfusões do M1. No entanto, o conceito de *container* é o de maior importância, uma vez que o uso do Docker foi definido como uma necessidade no projeto, colocando-se na categoria de CaaS.

Como referido, no mercado atual existe *software* que compete com o M1, não sendo este, no entanto, um fator determinante para o seu sucesso, uma vez que o objetivo é disponibilizar os seus módulos a esses mesmos concorrentes, colocando-os em ambientes *cloud*. Assim, foi necessário analisar um *cloud provider* para esta hospedagem.

Sendo que a MedicineOne pretende, como referido, alcançar o mercado internacional de *software* visado em auxiliar profissionais de saúde, e pretende disponibilizar módulos do M1 a outros EHRs, a cobertura global do *cloud provider* a selecionar é uma questão de referência necessária. Neste ponto, o AWS possui a maior cobertura, levando os seus serviços a mais cinco países que o seu concorrente mais direto.

Quanto a aspetos relativos ao motor de base de dados a utilizar, o SQL Server, imediatamente o

GCP não parece ser a melhor solução, dado não possuírem mecanismos de *failover* inter-regiões, nem restauro *point-in-time* para este motor de base de dados. Já o Azure parece ser, deste ponto de vista, a melhor opção, revelando melhores apoios e especificações para SQL Server, como seria de esperar.

Quando se analisam as otimizações de computação em VM, mais uma vez o GCP fica a perder para os seus concorrentes, não possuindo otimizações de armazenamento, enquanto que o AWS e o Azure possuem otimizações nas mesmas áreas.

No entanto, quando o tema é disponibilidade, o SLA do AWS é muito inferior quando se trata apenas de hospedagem apenas numa zona de disponibilidade, sendo a percentagem de tempo de disponibilidade garantida 90%, e medida à hora. Isto poderá ser um problema num momento inicial, a menos que a MedicineOne decida hospedar imediatamente o módulo de perfusões em mais do que uma zona de disponibilidade (o que poderá aumentar os custos). Caso seja este o caso, o valor do SLA aumentará para 99.99%, valor igual ao dos seus concorrentes.

Para serviços de *containers* (K8s), os três *cloud providers* analisados apresentam vantagens e desvantagens. *Upgrades* não são totalmente automáticos no AWS, já o Azure não permite a inclusão de redes de aplicações num *container*. Neste caso, o GCP aparenta ter mais vantagens que os concorrentes.

Apesar da melhor opção para uso como *cloud provider* para o módulo de perfusões do M1, de acordo com a pesquisa efetuada pelo estagiário, ser o Azure, devido ao facto de a MedicineOne já se encontrar a utilizar os produtos da AWS, será este o *cloud provider* utilizado. Este também tem refere vantagens quer em termos dos seus produtos e especificações, quer em termos de mercado.

Capítulo 4

Arquitetura

Neste capítulo será introduzida a **arquitetura** utilizada no M1, produto da MedicineOne referido anteriormente neste documento, assim como alguns dos conceitos inerentes à mesma. Posteriormente serão apresentados algumas decisões iniciais quanto ao ambiente *cloud* a implementar para o módulo de perfusões do produto em questão.

4.1 *Onion Architecture*

A Onion Architecture [78] foi criada com o objetivo de facilitar a manutenção/modificação de *software*, uma vez que, com o passar dos anos e com a mudança de engenheiros a trabalhar numa aplicação específica, esta pode-se tornar numa tarefa cada vez mais difícil (devido a situações como código ilegível, várias formas de organização de código devido a diferentes programadores trabalharem no mesmo código, etc.). Este objetivo é alcançado usando várias camadas, o mais independentes entre si possível, isolando assim as regras de negócio das regras de estrutura do *softawre*. De facto, uma das regras deste modelo é que nenhuma camada interior deve ter nenhuma informação sobre as camadas exteriores, ou seja, níveis acima de si mesma. Outra vantagem deste modelo é ser independente de linguagem de programação e *frameworks*.

Para facilitar esta individualidade das camadas a referir seguidamente, foi definido um conceito a ser usado não só entre programadores, mas também em todos em redor da aplicação em causa (linguagem ubíqua). Este conceito tem o nome de Domain Driven Design (DDD) [79], e implica o uso dos mesmos nomes/substantivos para as classes e entidades necessárias no código e o seu correspondente no negócio em si. Um dos conceitos chaves do modelo DDD é o Modelo do Domínio, que corresponde à entidade que incorpora todos os dados e funções de uma parte do negócio.

Assim, as três camadas definidas para esta arquitetura são: a **camada do domínio**, a **camada de aplicação** e a **camada de infraestrutura**.

- **Camada do domínio**: é a camada central deste modelo arquitetural, constituída por **modelos** e **serviços** de domínio, contendo toda a lógica e regras de negócio, não podendo efetuar operações de Input e/ou Output (I/O), facilitando assim a sua testagem.
  - **Modelos de Domínio**: como referido, a unidade central da camada de domínio, contendo dados e regras de negócio (operações). Idealmente cobririam todas as regras de negócio, mas isto nem sempre é possível, daí a existência de serviços de negócio;
  - **Serviços de Domínio**: criado para regras de negócio que não existem num, ou não se restringem a um único modelo de domínio. Podem ser definidos como funções, em termos de programação, nesta camada e não como modelos;
  - **Objetos de valor**: objetos que não contêm lógica nem identidade, apenas dados imutáveis. Podem estar contidos em modelos de domínio.

- **Camada de aplicação**: camada central da arquitetura, que contém o ambiente que envolve os modelos de domínio e contém as regras da aplicação, que correspondem aos casos de uso da aplicação. Estas regras são diferentes das regras de negócio, enquanto estas são executadas em qualquer caso, mesmo não existindo computadores, as regras de aplicação nem existiriam. As implementações destes casos de uso chamam-se serviços de aplicação.

  A camada de aplicação é responsável pela declaração de interfaces que contêm os métodos que permitem a execução de operações. Esta camada não é responsável por operações de I/O, mas é ainda importante referir o uso de Data Transfer Object (DTO), objetos com dados e formatos específicos utilizados para transferências de dados entre camadas, uma vez que a camada em questão pode chamar métodos de objetos que implementam interfaces que lhe dizem respeito, e estes objetos podem ter de efetuar operações de I/O;

- **Camada de infraestrutura**: é a camada externa da arquitetura em análise, responsável por todas as operações de I/O necessárias ao *software*. Possui toda a informação sobre as restantes camadas, uma vez que é a mais externa, e tem permissões para importar entidades provenientes das mesmas, não devendo, no entanto, efetuar lógica de negócio nem casos de uso. Esta camada é então, dito de outra forma, responsável por todas as API, *listeners* de eventos, e código que efetue operações de I/O, abstraindo a forma como a aplicação está exposta aos utilizadores.

  Nesta camada estão incluídos repositórios, que são classes que disponibilizam mecanismos de armazenamento e persistência. Cada modelo de domínio deve ter o seu próprio repositório, de modo a facilitar não só a leitura e organização do código, mas também para evidenciar a singularidade dos domínios. Evita também a preocupação da forma como os dados estão a ser armazenados (em memória, numa base de dados) durante o desenvolvimento do próprio código.

- **Ponto de entrada da aplicação**: normalmente a função *main* da aplicação, responsável por instanciar as dependências do código e injetá-las (*dependency injection*), quer sejam motores de base de dados, API, etc., sendo então, o único ponto no programa que deve criar objetos capazes de efetuar operações de I/O. Assim, a camada de infraestrutura pode criar instâncias dos objetos para efetuar estas operações, e o ponto de entrada da aplicação cria os clientes e apresenta-os à camada de infraestrutura. Isto permite também a troca de API sem alterações enormes de código;

Esta arquitetura é normalmente aplicada nas soluções da MedicineOne e deve manter-se assim quer no módulo de perfusões, ao ser colocado num ambiente *cloud*, quer na aplicação de atualização dos dados a implementar. Na figura 4.1 encontra-se um esquema exemplo desta arquitetura.

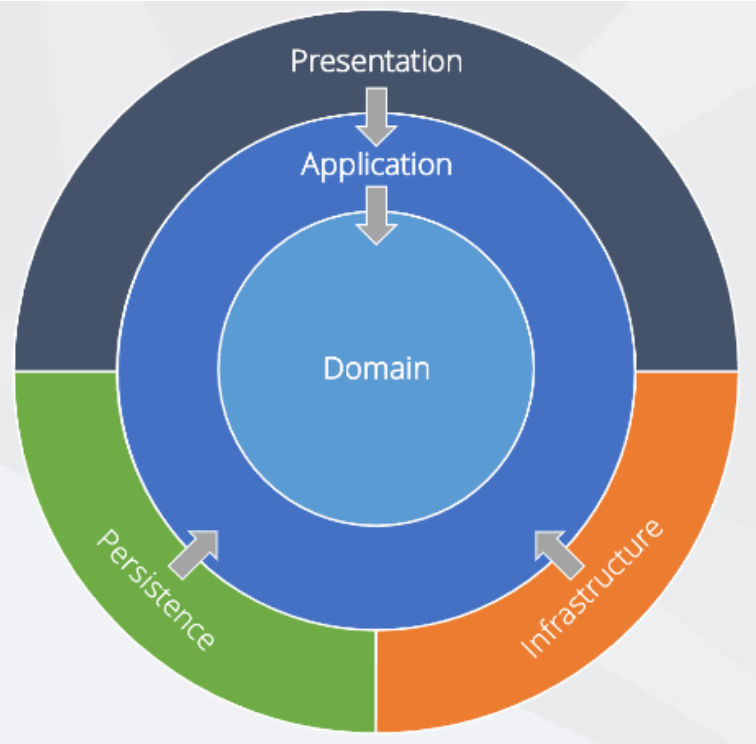

Figura 4.1: Esquema de Onion Arquitecture

4.2 Arquitetura para produtos *cloud native*

Como referido anteriormente, a arquitetura de ambiente *cloud* a implementar terá que ser uma **arquitetura de micro-serviços**, de modo ao módulo de perfusões ser escalável, resiliente e independente. Para facilitar o processo de *deployment* para condições de escalonamento horizontal, será utilizado o Docker, sendo, então, provável o uso um serviço como o EC2 [40], uma vez permitir o *deployment* de micro-serviços e ser compatível com os *containers* do Docker, e o Amazon RDC, para hospedar a base de dados, caso necessário.

Também como referido na secção 3.3 o serviço *cloud* tem que oferecer uma **base de dados *open-source***. No caso do M1, a o motor de base de dados atualmente utilizado é o SQLServer [80]. No entanto, durante este projeto, ao contrário do indicado anteriormente, será efetuada uma migração dos dados para uma base de dados PostgresSQL [6], devido a, entre outros aspetos, os seus custos mais baixos. O PostgresSQL está disponível em serviços AWS, como por exemplo, o Amazon RDS [81]. De notar que esta alteração na decisão do motor de base de dados, não influencia a escolha do *cloud provider*.

Em termos de ***auditing***, outro conceito apresentado previamente, a AWS contém soluções como o AWS Audit Manager [82] ou o Amazon CloudWatch[83], para analisar constantemente as características do *software* e análise do mesmo por parte da MedicineOne. A última solução permite também atos de ***logging***, através do Amazon CloudWatch Logs.

Quanto a segurança, existe uma lista de produtos disponíveis na AWS que se complementam e ajudam a assegurar certos níveis de segurança do *software*, dos utilizadores e dos seus dados. Será posteriormente necessário definir quais os serviços a utilizar e se os seus custos são aceitáveis para a empresa. [84]

4.3 Arquitetura Final

Iniciando o segundo semestre de estágio, o estagiário definiu uma arquitetura de produtos AWS que permitisse a hospedagem do serviço de perfusões e permitisse alcançar os objetivos definidos nos Requisitos Não Funcionais, presentes na secção 2.3.2 deste documento. Na figura 4.2, está representada esta mesma arquitetura, contendo presentes todos os produtos AWS a utilizar, e as suas interações. Na secção 4.3.1, será justificado o objetivo e o motivo da utilização destes produtos.

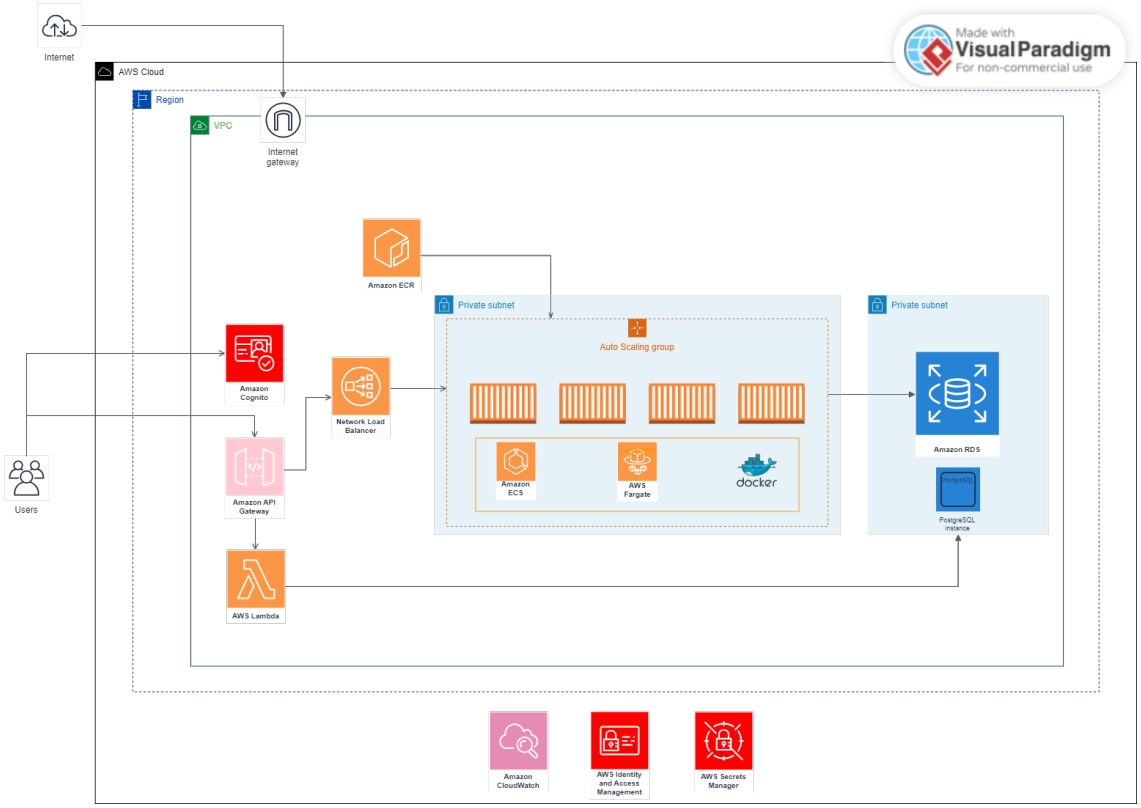

Figura 4.2: Arquitetura Final [85]

4.3.1 Produtos de Segurança

O produtos presentes nesta subsecção do documento visam garantir níveis de segurança que são *standard* na MedicineOne.

Amazon Cognito [86]

O AWS Cognito será utilizado para definir acessos a funcionalidades e recursos do micro-serviço. Este produto atribuirá permissões às aplicações cliente que beneficiam do sistema, e este produto irá gerir os acessos dessas contas. O AWS Cognito pode também controlar processos de registo, autenticação, além do controlo de acessos.

Amazon and Identity Access Manager [87]

O Amazon IAM, será utilizado apenas por curiosidade do estagiário, sendo o mesmo desnecessário à implementação do projeto. Este produto permite criar contas de utilizadores AWS, associados à conta que os criou. Permite também definir as permissões de acesso a produtos AWS desses

mesmos utilizadores. Este produto é útil em ambientes empresariais, por questões de gestão e de segurança.

AWS Secrets Manager [88]

O AWS Secrets Manager auxilia a gestão de acesso a produtos e partilha/encriptação de informação entre produtos AWS, informação esta que não deve ser acedida por clientes ou entidades exteriores à arquitetura. Neste projeto, o Secrets Manager será utilizado para partilhar os dados de acesso à base de dados (utilizador, password, *connection string*) para produtos como o Fargate, em forma de variáveis de ambiente.

4.3.2 Produtos de Rede

Internet Gateway

Um Internet Gateway, ou IGW, é um componente das VPCs que permite a conexão da mesma, e das suas *subnets*, à Internet pública, atribuindo endereços de IP públicos às mesmas e permitindo que a arquitetura receba os pedidos dos clientes.

4.3.3 Produtos de Computação

AWS Lambda [89]

O AWS Lambda é um serviço de computação *serverless* orientado a eventos, ou seja quando certos eventos da aplicação ou dos processos de rede e ligação ocorrem, e uma notificação dos mesmos é acionada, este produto permite executar código que executa tarefas paralelas ao serviço. Neste projeto, as funções Lambdas serão usadas para implementação da lógica de *endpoints* que visam a atualização da base de dados.

Amazon ECR [90]

O Amazon ECR, ou Elastic Container Registry, é um produto de registo de imagens *containers*, que será utilizado para registar as imagens de *containers* do Docker para posteriormente hospedar e escalar com o AWS Fargate, que será apresentado seguidamente. Será útil para facilitar a atualização e gestão do *software*, controlando as suas versões.

Amazon ECS

O Amazon ECR, ou Elastic Container Service, é um serviço de orquestração de *containers* , que facilita ou automatiza a hospedagem, o escalonamento, e a gestão das instâncias dos serviços.

AWS Fargate [91]

O AWS Fargate é um serviço de computação *serverless*, ou seja, retira do programador a responsabilidade de definir as propriedades do servidor onde o(s) seu(s) serviço(s) vai(vão) estar hospedados, compatível com o Amazon ECS. O serviço é executado conforme o uso, ou seja, apenas quando pedidos são efetuados para o serviço, é que o mesmo é executado e, posteriormente, o custo dessa execução é acrescentado. Isto pode permitir a redução dos custos da hospedagem do serviço, e permite um maior foco por parte dos programadores no desenvolvimento do serviço em si.

Network Load Balancer [92]

Um Load Balancer é uma ferramenta que distribui os clientes pelas várias instâncias saudáveis de um serviço, dividindo assim a carga de cada uma das instâncias e aumentando assim a disponibilidade do serviço. No entanto, um Network Load Balancer trabalha especificamente na quarta camada do modelo Open Systems Interconnection (OSI) [93], a camada de transporte, redirecionando assim os pedidos para a instância mais adequada, utilizando um dos seguintes protocolos de segurança de comunicação: TCP [94], UDP [95], TLS [96] ou TCP_UDP. Este balanço de utilizadores por instância pode ser efetuado em várias zonas de disponibilidade.

4.3.4 Produtos de Armazenamento

Amazon RDS [81] (Amazon Autora [97] for PostgreSQL [6])

O Amazon RDS é uma coleção de serviços de gestão de motores de bases de dados, de modo a facilitar a sua configuração, operação e escalabilidade. Existem sete modos de operação, mas apenas dois dos mesmos são visados para o uso de PostgreSQL, o sistema de base de dados relacional escolhido para este projeto. Estas duas opções consistem em:

- Hospedagem do PostgreSQL diretamente através dos Amazon RDS;
- Hospedagem através do Amazon Aurora, um sistema de gestão de bases de dados relacionais, disponibilizado pelo Amazon RDS, apenas para PostgreSQL e MySQL.

O Amazon Aurora facilita a gestão dos dados do projeto, através de replicação dos mesmos entre regiões, escalonamento automático das bases de dados, *backups* contínuos, entre outras propriedades do interesse da MedicineOne.

4.3.5 Produtos de Integração

Amazon API Gateway [98]

O Amazon Api Gateway, é um serviço que permite publicar, gerir e monitorizar APIs RESTful [99], ou seja, que funcionam como entradas para os serviços de *back-end*, permitindo comunicação bidirecional. Possui também funcionalidades de gestão de tráfego, controlo de acessos e controlo de versões. Da mesma forma que o Fargate, o custo do Amazon API Gateway é calculado puramente pelos acessos aos serviços.

4.3.6 Produtos de Gestão

Amazon CloudWatch [100]

O Amazon CloudWatch é um produto AWS que monitoriza infraestruturas *cloud*, armazena, analisa e permite a visualização de *logs* e métricas relativas a, como por exemplo, à *performance* do sistema e pode tomar ações relativas a essas análises, facilitando assim a gestão da infraestrutura.

AWS CloudTrail [101]

Da mesma forma que o Amazon CloudWatch, o Amazon CloudTrail é um produto de monitorização, mas visado à atividade dos utilizadores do serviço/infraestrutura, podendo registar eventos no Amazon CloudWatch Logs. Regista ainda anomalias de segurança como acessos não autorizados a informações e pode efetuar ações de resposta a essas mesmas anomalias.

4.4 Sumário e Discussão do Capítulo

Apesar deste ser um capítulo de apresentação de informação, neste caso, das arquiteturas e dos produtos a usar, é necessário justificar a utilização de algum destes recursos. A Onion Architecture, além de todos os benefícios descritos na secção 4.1, é, como referido, a arquitetura *standard* do Clinical Brain, produto da M1 no qual este projeto se baseia, daí ter sido decidio manter esta arquitetura. O mesmo motivo justifica o uso da linguagem C#, apesar da liberdade oferecida ao estagiário quanto ao uso da linguagem de programação, não faria muito sentido distinguir a mesma dos padrões da empresa.

O uso de Fargate, uma ferramenta de CaaS, deve-se não só à futura necessidade de escalabilidade do serviço, mas também à vantagem de não gerir servidores. O serviço de obtenção de interações entre medicamentos injetáveis consiste, após isolado do restante ClinicalBrain, de um pedido a uma base de dados, que, por muito complexo que o mesmo seja, não apaga que a lógica da aplicação que o executa não seja complexa. O fluxo de informação é direto (um cliente efetua um pedido e recebe uma resposta), não sendo então necessário implementar um serviço que estivesse disponível permanentemente num servidor. O Fargate oferece a vantagem de apenas executar a instância quando um pedido é efetuado e, consequentemente, os custos deste produto dependem do número de pedidos efetuados para o mesmo.

Capítulo 5

Desenvolvimento do Projeto

Neste capítulo, serão apresentados e explicados todos os procedimentos efetuados, pelo estagiário, neste projeto. Anteriormente aos procedimentos foco do projeto, foi efetuada uma migração da base de dados do projeto. Posteriormente, foram efetuados tutoriais relativos ao uso do Docker e dos AWS, como forma de preparação para a implementação da arquitetura presente na secção 5.3 deste documento. Por fim, foi efetuada a adaptação do código do ClinicalBrain (módulo do produto da MedicineOne, M1), encapsulamento do mesmo num *container* do Docker, e *deployment* para a *sandbox* [102] AWS da MedicineOne.

5.1 Preparação e Tutoriais

Para angariar alguma experiência com as tecnologias necessárias para implementação da arquitetura *cloud native* em AWS, o estagiário seguiu um tutorial [103], tutorial esse que inclui a utilização de código C# e tecnologias .NET, numa aplicação em Onion Arquitecture, e que possui ainda, já efetuados, os processos do Docker (dockerfile e imagem docker já criadas). Assim, o estagiário obteve uma explicação mais detalhada sobre o código, permitindo mesmo melhorar a sua compreensão do código do ClinicalBrain. Com o auxilio da documentação oficial do Docker, o estagiário conseguiu, também, uma melhor compreensão da criação e da formatação de um dockerfile [104], de um ficheiro docker-compose [105], de como criar o *container* de um projeto, e executá-lo, quer através do Integrated Development Environment (IDE) [106] Visual Studio, quer através da linha de comandos.

5.2 Migração da Base de Dados

Durante as primeiras duas semanas do segundo semestre do ano letivo, o estagiário e o seu orientador, agendaram uma reunião com membros da empresa que estiveram, a certo ponto da sua carreira, envolvidos no desenvolvimento do ClinicalBrain. Através desta reunião foram obtidas respostas a dúvidas que o estagiário necessitava esclarecer, dúvidas estas relativas às funcionalidades a colocar num ambiente *cloud*, e algumas especificações. Uma destas especificações é o facto de que, na MedicineOne, um dos requisitos ao transitar módulos do seu monólito em ambiente *cloud*, o motor de base de dados a utilizar é o PostgreSQL, sendo necessária uma migração das bases de dados do projeto, de SQL Server para uma base de dados do motor referido.

Assim, o estagiário seguiu os seguintes passos para efetuar esta tarefa:

- Obtenção dos *scripts* (sequências de comandos ou *queries*) SQL para criação das tabelas, através do SQL Server Management Studio (SSMS) [107], e execução dos mesmos no PostgreSQL;
- Instalação do DBeaver [108] para efetuar a migração automatizada dos dados;

- Criação das ligações entre os dados das tabelas (*Primary Keys* e *Foreign Keys*) no PostgreSQL;

- Edição da *query* de criação da *view* necessária para a obtenção dos avisos de interações entre medicamentos e toda a informação necessária ao M1 no momento do pedido.

Seguidamente o estagiário começou a implementação do código a colocar em ambiente *cloud*.

5.3 Implementação do código do projeto

Para recriar as funcionalidades do ClinicalBrain que interessam ao projeto, o estagiário criou uma nova solução no Visual Studio, mantendo a Onion Arquitecture. A solução consiste de oito projetos, que constam na figura 5.1

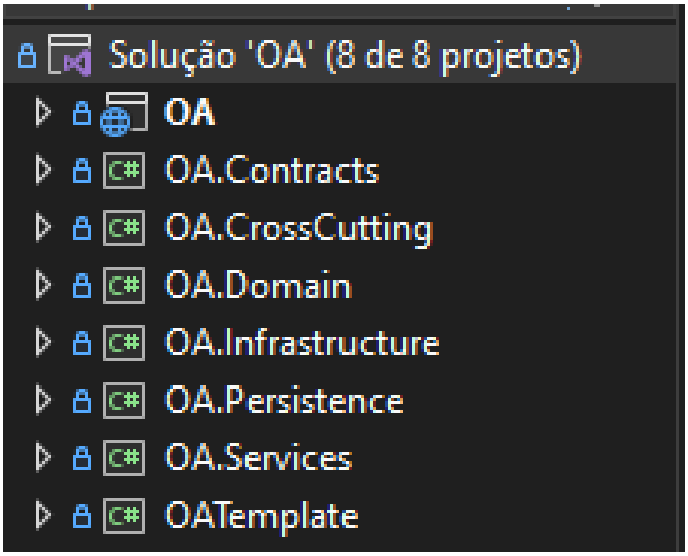

Figura 5.1: Estrutura da Solução

Como referido na secção 5.1, existem as camadas de infraestrutura e do domínio, notórias através de projetos com o mesmo nome. O projeto de serviços é, essencialmente, a camada de aplicação, onde efetivamente são implementadas e executadas as regras sobre as entidades do domínio. No projeto CrossCutting, estão definições de conexão à base de dados, e nos contratos as entidades a devolver aos clientes. A entrada para o aplicação e os *endpoints* (pontos de acesso para os clientes) são definidos na solução em si.

Para devolver avisos de interações entre medicamentos no momento de prescrição de soluções medicamentosas, foi necessário criar um *endpoint* que recebe uma lista de inteiros, correspondente aos Código Hospitalar Nacional do Medicamento (CHNM), que, através de uma *query* definida na camada de serviços, vai buscar toda a informação sobre as interações entre esses medicamentos. Caso algum dos medicamentos não exista, ou a lista esteja vazia, o *endpoint* retorna uma exceção, que equivale ao aviso do erro que o mesmo encontrou nos dados que recebeu. Apesar das estruturas dos dados a serem devolvidos do serviço para o *endpoint*, e do *endpoint* para o cliente serem iguais, não se reduz o número de entidades para uma, de forma a que as duas camadas sejam independentes. A transição de uma entidade para a outra deverá ser efetuada através de uma biblioteca de mapeamento de classes.

Após a implementação, e uma breve execução de testes por parte do estagiário, utilizando a ferramenta Postman [109], e a API Swagger [110], o estagiário procedeu para a implementação do *container* Docker do projeto.

5.4 Criação do *container* Docker

Para criação do *container* do Docker, é necessário primeiro criar dois ficheiros: Dockerfile e Docker-Compose.

O Dockerfile contém todos os comandos que poderiam ser executados numa linha de comandos/terminal, cujo objetivo é a criação de uma imagem de *container* Docker (modelo para criação de *containers*), inclusive a preparação de ficheiros necessários para a execução da aplicação no *container*, a diretoria onde o *container* vai ser definido, os portos[111] dos quais a aplicação vai receber informação, etc..

Já o ficheiro Docker-Compose, apesar de ser visado para serviços que utilizam vários *containers*, permitindo criar os mesmos através do mesmo ficheiro, foi utilizado neste projeto pois o estagiário tencionou aprender a estrutura destes tipo de ficheiros, ou seja, ficheiros YAML[112]. Apesar de não ser necessário ao projeto final, o estagiário colocou a base de dados PostgresSQL, criada anteriormente, noutro *container*, utilizando o docker-compose, e executou os dois *containers* em sintonia.

Na figura 5.2 pode ser visto o conteúdo do ficheiro Dockerfile implementado, que inclui a informação referida no segundo parágrafo desta secção.

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["OA/OA.csproj", "OA/"]
COPY ["OA.Contracts/OA.Contracts.csproj", "OA.Contracts/"]
COPY ["OA.Domain/OA.Domain.csproj", "OA.Domain/"]
COPY ["OA.Infrastructure/OA.Infrastructure.csproj", "OA.Infrastructure/"]
COPY ["OA.Service/OA.Services.csproj", "OA.Service/"]
COPY ["OA.CrossCutting/OA.CrossCutting.csproj", "OA.CrossCutting/"]
RUN dotnet restore "OA/OA.csproj"
COPY . .
WORKDIR "/src/OA"
RUN dotnet build "OA.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "OA.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "OA.dll"]
```

Figura 5.2: Dockerfile

5.5 Ambiente AWS

5.5.1 Ambiente Inicial AWS

Para inicializar a experimentação de produtos AWS, foi uilizada uma VPC atribuída automaticamente ao acesso da *sandbox* que foi atribuído ao estagiário pela MedicineOne. Esta VPC contém três *subnets* [113] e uma tabela de rotas (*default*), que fornecem indicações quanto ao endereçamento dos serviços a colocar na *cloud*, ou seja, as três *subnets* cobrem conjuntos de endereços IP diferentes, podendo os serviços ser divididos pelas mesmas evitando problemas de tráfego de pedidos. Estas *subnets* encontram-se dentro da mesma região, Oeste Europeu - Londres (eu-west-2).

5.5.2 *Set-up* da AWS CLI e AWS Tools for PowerShell

O primeiro passo para iniciar a implementação do ambiente *cloud*, passou por criar um utilizador no AWS Identity and Access Management (IAM), e criar acessos (ID e chave secretos) tanto à consola dos AWS (CLI), como para conectar a conta AWS através de uma linha de comandos/terminal, presente no computador do estagiário, o que será importante para o registo do *container* do Docker.

A este utilizador, foram ainda atribuídas permissões totais, quer do ECR, quer do Elastic Container Service (ECS), de modo a que o estagiário não tenha problemas referentes a permissões ao trabalhar com estes serviços.

5.5.3 Registo do *container*

Para registar o *container* do Docker que inclui o código do projeto e as suas dependências, é necessário entrar novamente na AWS, desta vez com a conta de utilizador criada na secção anterior, uma vez que lhe foram atribuídas permissões totais para os serviços a utilizar.

Ainda anteriormente ao ato de registar o *container*, é necessário criar um repositório de *containers*, através do ECR. Assim, o estagiário criou um repositório com todas as características padrão (*default*).

Após ter a aplicação à qual se pretende efetuar *deploy* nos AWS, dentro de um *container* docker, é necessário registar esse *container* na *cloud*. Para esse efeito, é utilizado o Amazon ECR [90], que permite a posterior utilização do ECS. O estagiário criou assim um repositório privado cuja única alteração às definições padrão foi a verificação ao submeter imagens.

Por fim, foi necessário efetuar autenticação na Powershell do Windows (em modo administrador), como o utilizador criado na consola AWS, e através de uma sequência de comandos, registar a imagem do *container* do projeto no repositório criado. Os comandos executados para registar o *container* podem ser visualizados na figura 5.3.

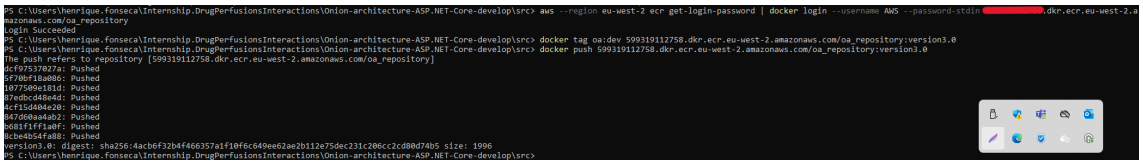

Figura 5.3: Comandos para Registo da imagem Docker no ECR

5.5.4 Criação do *cluster* Fargate

Para criação de um *cluster* de Fargate, (tecnologia que permite a execução de *containers*, através do ECS, sem a necessidades de configuração e escalonamento de VMs) o estagiário necessitou de executar três passos:

- Seleção a VPC e as *subnets* dessa VPC que o *cluster* irá utilizar, onde foram selecionadas a VPC e as *subnets* referidas na subsecção 5.5.1.
- Seleção do *container*, onde foi selecionado o *container* da aplicação, registado no ECR, e se mapeou o porto de entrada de dados (porto 80) e o protocolo utilizado. Estas configurações podem ser visualizados na figura 5.4;
- Configuração do ambiente, definições como o uso do Fargate, qual o sistema operativo a usar, tamanho e memória da tarefa, mas também definições de monitorização. Estas configurações podem ser visualizados na figura 5.5.

Figura 5.4: Etapa 1 da criação do *cluster* Fargate

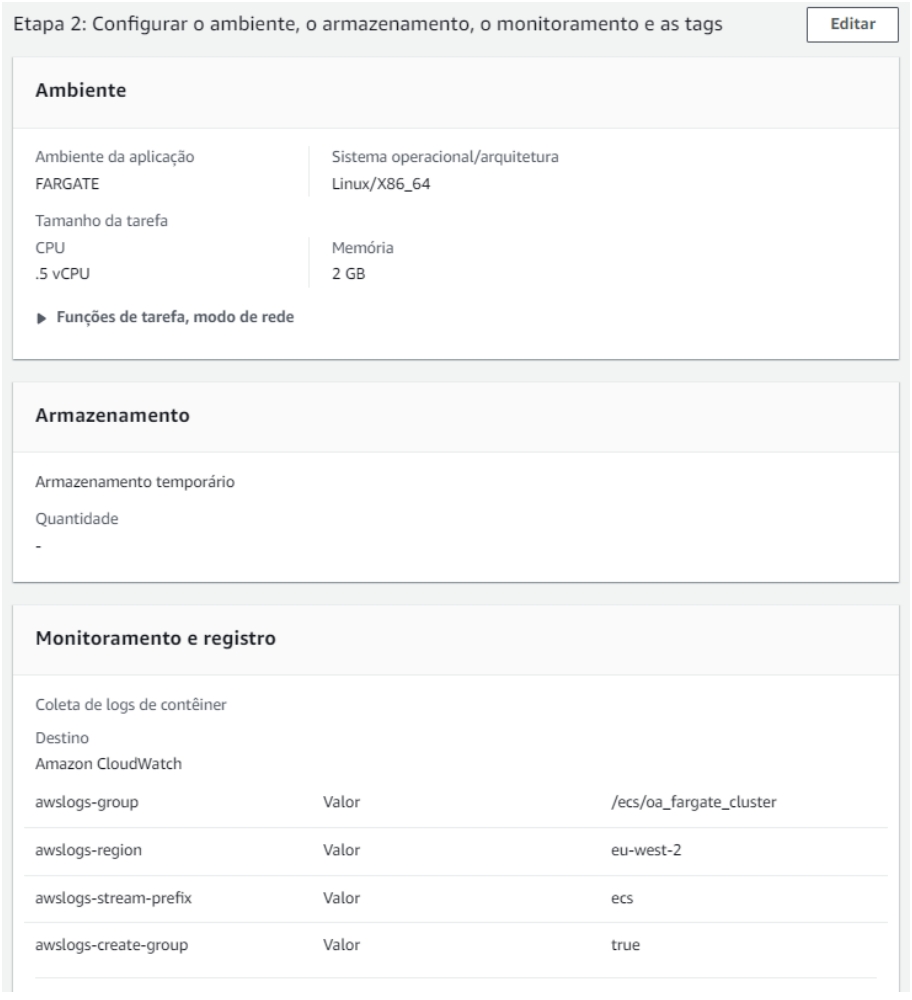

Figura 5.5: Etapa 2 da criação do *cluster* Fargate

Porém, a criação do *cluster* não é suficiente para a execução do *container* do Docker. Assim, o estagiário necessitou de criar uma tarefa do ECS. A criação desta tarefa define-se em dois passos:

- Definição do tipo de ambiente (Fargate) e versão;
- Seleção da VPC e s*ubnets* e do grupo de segurança, onde foi criado um novo grupo de segurança, com os protocolos e portos de transmissão de dados, porém, o grupo de segurança poderia ter sido criado previamente e apenas sido selecionado nesta etapa . A figura 5.6 demonstra esta etapa.

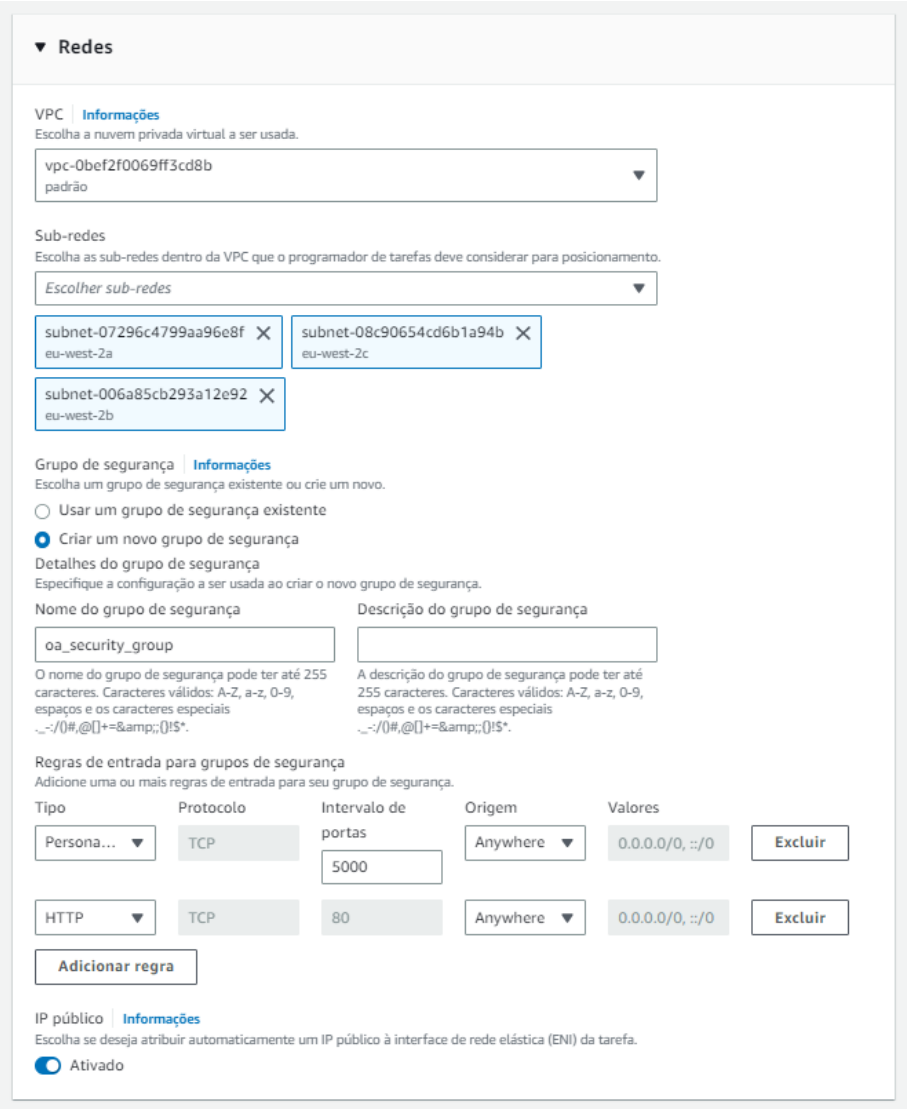

Figura 5.6: Etapa 2 da criação da tarefa ECS

Após a criação do *cluster* Fargate, upload do *container* Docker, e criação da tarefa ECS, é possível executar a aplicação, mas a mesma termina por ocorrer um erro ao ser efetuado um pedido ao *endpoint* da mesma. Este erro deve-se à incapacidade da aplicação se ligar a uma base de dados, levando isto ao próximo passo.

5.5.5 Migração da Base de Dados Local (PostgreSQL) para o AWS RDS

Para a execução da tarefa de migrar a base de dados local para o Amazon RDS, foi primeiro necessário adicionar permissões ao utilizador que o estagiário criou, para a mesma ferramenta.

Seguidamente, o estagiário começou por trabalhar imediatamente com o Amazon RDS, e depois executar as tarefas necessárias no PGAdmin [114], onde está a base de dados local.

Assim, para criação da instância Amazon RDS, foi necessário definir algumas configurações, como por exemplo, o tipo de motor de base de dados, ou seja, PostgreSQL, o tipo de nível de serviço, sem custos por 12 meses, e o identificador, utilizador e senha mestres da base de dados, colocando os mesmos valores presentes na base de dados local. As configurações de armazenamento, efetuadas dentro do limite sem custos, podem ser visualizadas na figura 5.7. Algumas outras configurações foram deixadas com os valores padrão, mas as configurações de conectividade são importantes de referir.

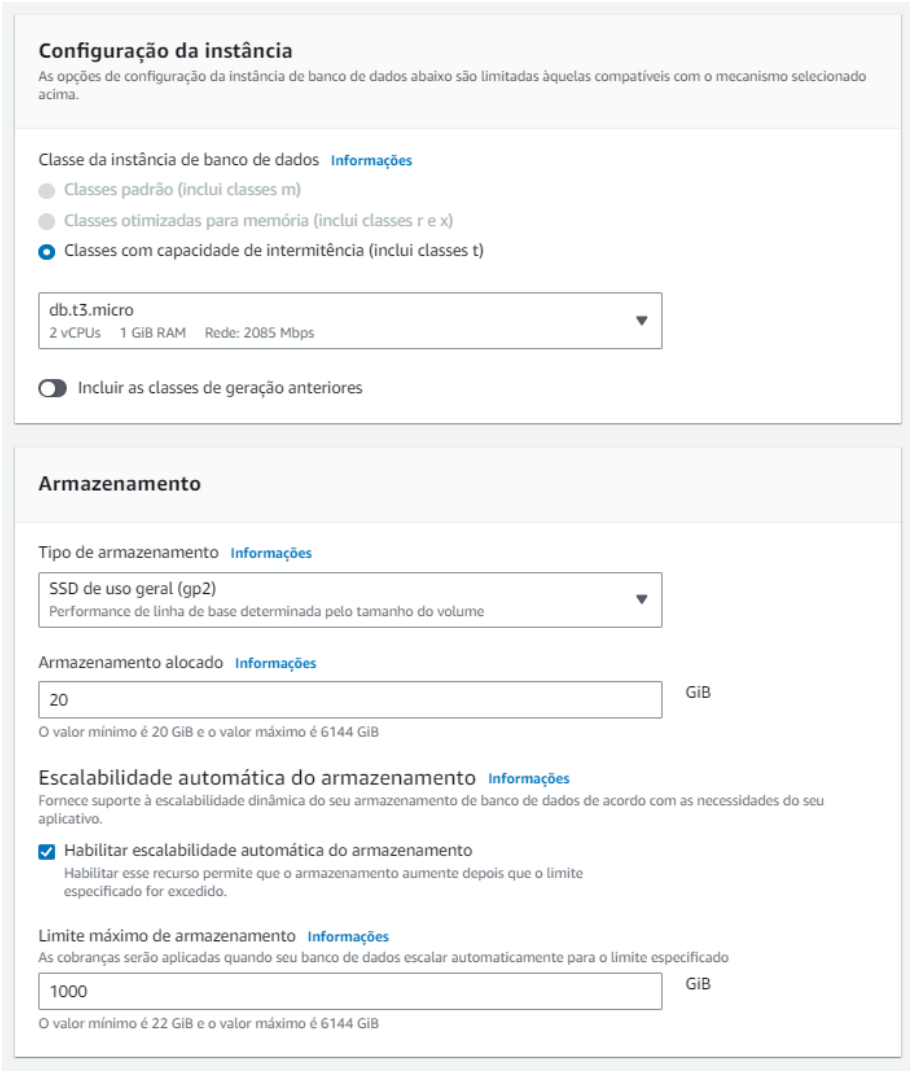

Figura 5.7: Configurações de armazenamento da instância Amazon RDS

Nas configurações de conectividade, o estagiário voltou a selecionar a VPC e as sub-redes criadas anteriormente, mas destas vez teve também a possibilidade de selecionar a zona de disponibilidade. Sendo que todo o projeto foi implementado na região com o código eu-west-2, ou seja, Londres, como referido na secção 5.5.1 deste capítulo, foi selecionada uma zona de disponibilidade presente nesta região, eu-west-2a. Foi definido que o acesso à instância não seria privado, nem limitado a instâncias de EC2, visto que o Fargate não se enquadra nessas instâncias, e criou-se também um novo grupo de segurança, com protocolos IPv4 e para todos os utilizadores, mas tornando uma possível alteração futura mais fácil de efetuar. O porto de acesso à instância, também é o mesmo que na base de dados local, o default do PostgreSQL, 5432.

Por fim, no PGAdmin, foi criado um *backup* da base de dados a migrar, de nome Trissel, e registou-se o servidor da instância Amazon RDS, utilizando o Uniform Resource Locator (URL) do mesmo,

como visível na figura 5.8. Posteriormente foi efetuado um *restore*, do *backup* efetuado anteriormente, mas no servidor Amazon RDS registado, conseguindo assim, ter a base de dados no ambiente *cloud*.

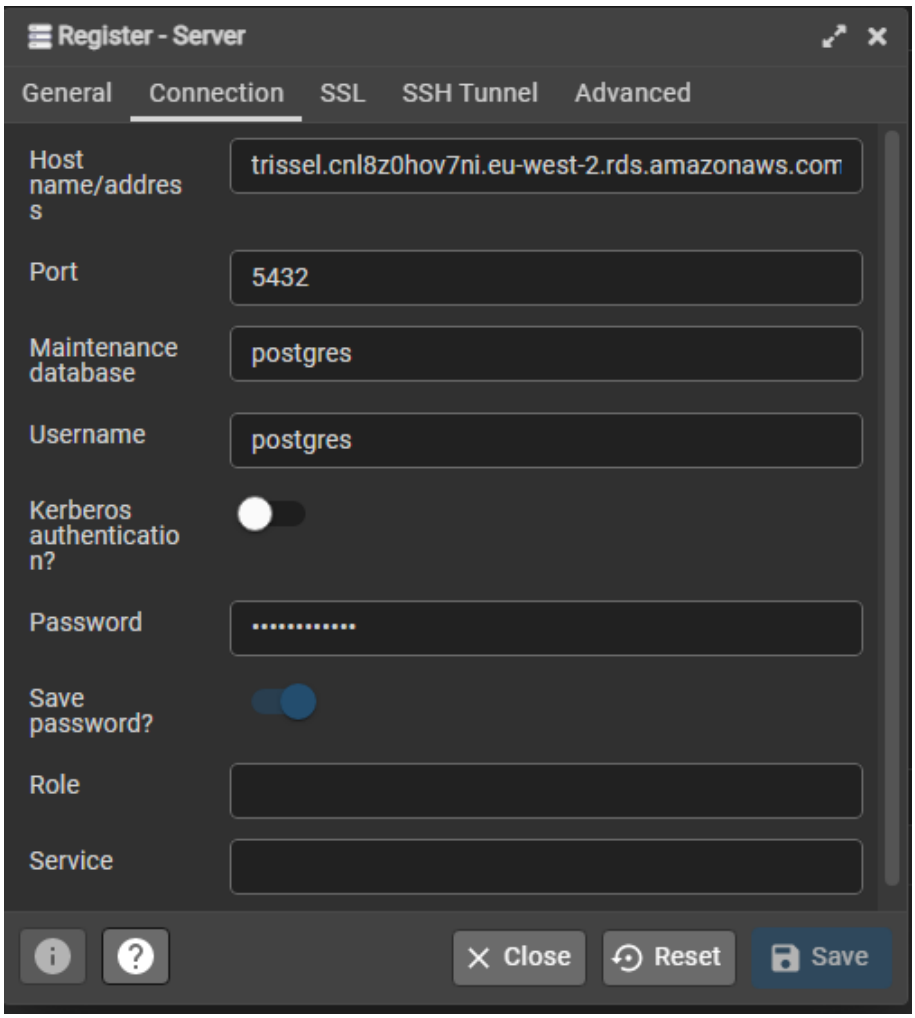

Figura 5.8: Registo do servidor Amazon RDS

5.5.6 Criação de um NLB e serviço Fargate

Dentro das necessidades do projeto, um Network Load Balancer (NLB) visa a distribuição do tráfego de pedidos efetuados a instâncias do Fargate. Assim, o estagiário começou por criar um Target Group, para definir que o destino dos pedidos será efetivamente a aplicação criada para o projeto, executada através das instâncias do Fargate. Neste Target Group, são defendidos os endereços IP que recebem os pedidos, o protocolo de comunicação (HTTP) e o porto (80). O destino (tarefa do Fargate) é definido apenas posteriormente. Assim, para a criação do NLB, foi necessário referir o mesmo como interno à VPC, uma vez que os pedidos serão recebidos provenientes do API Gateway, e não diretamente dos clientes. Foi necessário também associar o Target Group criado. Na figura 5.9 podem-se observar as propriedades do NLB referidas neste parágrafo.

Neste ponto, o estagiário encontrou a maior dificuldade do projeto, apesar de que a mesma se poderia ter verificado anteriormente. A imagem Docker submetida do ECR não passava nas revisões de saúde. Foi necessário revisar o código, atualizar a versão do .NET, da 5 para a 6, e atualizar todos os packages NuGet (pacotes de código adicionados ao projeto), inclusive os provenientes dos servidores da MedicineOne. Após, foi necessário atualizar o dockerfile e o docker-compose, de modo a que, aquando da criação da imagem, estivessem presentes as condições de acesso aos

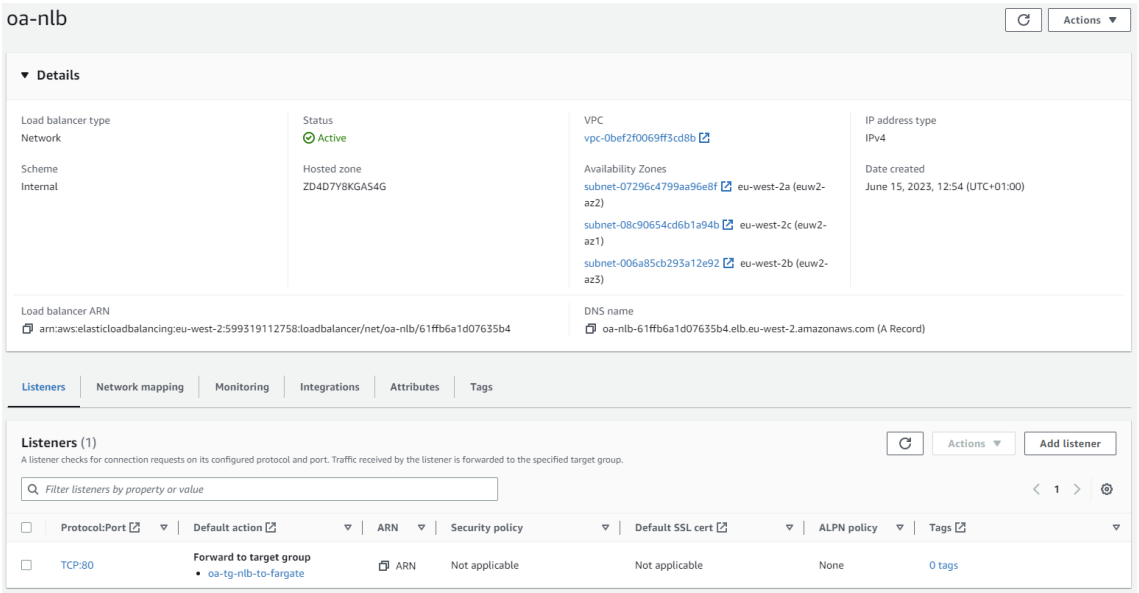

Figura 5.9: Propriedades NLB

servidores da MedicineOne (um *token* criado na plataforma da empresa, e nome de utilizador). Assim, depois destes processos, e de criar novamente a imagem do Docker do projeto e registar a mesma no ECR, o estagiário avançou para a criação do serviço Fargate, que indicou outro problema. Na criação do serviço, o mesmo também não passava nas verificações de saúde, devido ao excesso de utilização de CPU e memória, e acabando por terminar o processo e eliminando o serviço criado. Foi necessário, então, criar uma nova revisão de tarefa, aumentando os recursos de cada instância Fargate (4GB de memória e 1 vCPU, equivalente a 2 a 8 GB com incrementos de 1 GB). Por fim, o estagiário conseguiu avançar para uma criação de serviço Fargate que resultou em sucesso.

Finalmente, para a criação do serviço Fargate no *cluster* criado anteriormente, o estagiário selecionou a última revisão da tarefa intencionada (sendo que, nesta última revisão foram efetuadas as mudanças nas propriedades de performance referidas no parágrafo anterior). De seguida, foi necessário selecionar a VPC e as suas subnets, o grupo de segurança, o NLBe o Target Group, criando assim a associação da tarefa Fargate ao mesmo. Nas figuras 5.10 e 5.11, podem ser visualizadas, respetivamente, a atribuição do NLB à tarefa, no serviço Fargate, e a confirmação da ligação, posteriormente, no Target Group selecionado.

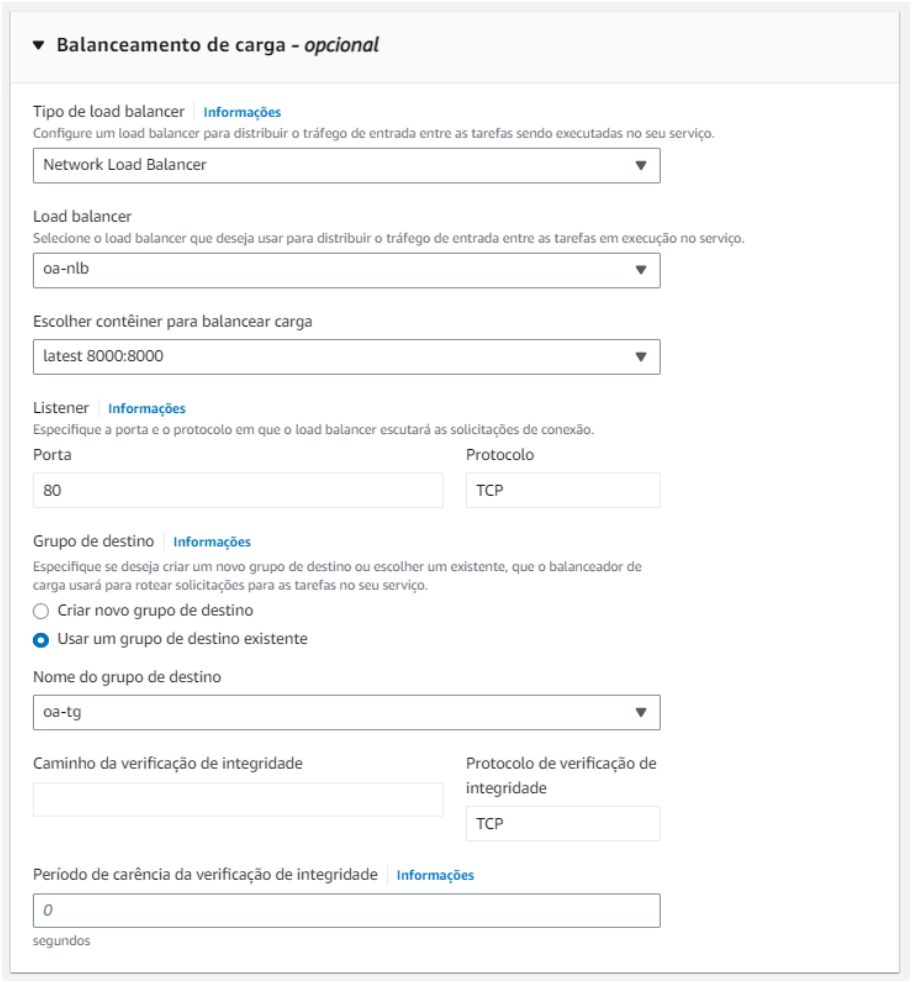

Figura 5.10: Conexão entre a tarefa Fargate e o NLB

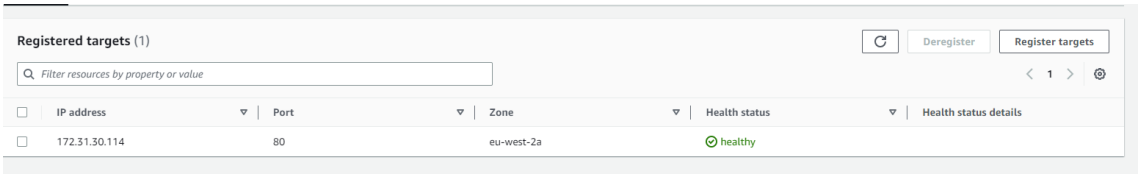

Figura 5.11: Confirmação da ligação entre no NLB e a tarefa Fargate através do Target Group

5.5.7 API Gateway, Amazon Cognito e AWS Lambda

Para criação e configuração do API Gateway, foi necessário, novamente, a edição das permissões AWS do utilizador criado, fornecendo-lhe assim total acesso a esta ferramenta.

Apesar de o API Gateway ser uma ferramenta que recebe pedidos independentemente de estar ou não associado a outras ferramentas (como por exemplo pedidos *mock*, ou simulações), é desejado que estes pedidos sejam recebidos de entidades autorizadas a tal, evitando assim acessos provenientes de *software* desconhecido. Assim, o estagiário decidiu, através do Amazon Cognito, permitir ao API Gateway filtrar o acesso aos recursos (neste caso, ao NLB), permitindo este acesso apenas a software autenticado.

Primeiramente, o cliente definiu, no Cognito, um grupo de utilizadores ao qual se deu o nome de "oa_clients", ou seja, um grupo para as aplicações que poderão aceder aos recursos através do API Gateway e, seguidamente, uma vez que a autenticação será para clientes em forma de aplicações externas e não utilizadores, foram definidas propriedades de integração de aplicações:

- **Domínio do Cognito:** onde se define o URL do domínio, ou seja, nome/*link* para o *website*;
- **Cliente da aplicação:** ao qual se deu o nome de "oa_client"permite o acesso de aplicações ao grupo de utilizadores, gerindo a criação de *tokens* (códigos) de autenticação e as suas propriedades (como a expiração da validade do *token*), atribuindo um identificador, uma chave secreta de cliente e fluxos de autenticação (métodos através dos quais as aplicações se podem autenticar);
- **Interface de hospedagem:** que permite a atribuição dos *tokens*, e a sua validação, durante o processo de autenticação das aplicações cliente;

A autenticação e permissão de acessos das aplicações cliente segue o protocolo padrão de autorização, o OAuth2.0 [115].

Seguidamente, foram criadas funções Lambda, através do AWS Lambda, cujas funcionalidades, de cada uma, eram a atualização dos conteúdos da base de dados criada na secção 5.5.5. as funções criadas correspondem aos métodos POST e PATCH, para, respetivamente, criação de novas entradas de medicamentos, interações e toda a informação anexada aos mesmos, e atualização de entradas já existentes. As funções Lambda foram implementadas utilizando a versão 3.10 da linguagem de programação Python [116].

Ainda antes de criar o API Gateway, foi criada um ligação VPC (VPC Link), para posteriormente, poder indicar ao *endpoint* de obtenção das interações entre medicamentos, que o destino do pedido, após ter sido aceite pelo API Gateway, é o NLB criado na secção 5.5.6. Esta ligação penas contém um identificador, e o alvo (o NLB).

Por último, o estagiário criou o API Gateway, visado para API Rest [117]. Após a criação do Gateway, foi necessário seguir vários processos de edição do mesmo para permitir que os pedidos dos clientes (autorizados) cheguem aos seus destinos.

- **Criação de *stages*:** foram criadas duas *stages*, para distribuir as alterações efetuadas nos recursos do API Gateway, tornando assim os mesmos acessíveis aos seus clientes. As *stages* criadas correspondem ao desenvolvimento do produto (dev) e a ambiente de produção (prod). Sendo o produto deste estágio, uma prova de conceito, a *stage* de desenvolvimento bastaria, uma vez que o seu objetivo é a testagem do produto, mas, num ambiente correto, seria também necessária a *stage* de produção, que seria correspondente à disponibilização do produto aos clientes;
- ***Authorizer*:** corresponde à integração do grupo de utilizadores criado anteriormente no Cognito com o API Gateway, garantido que o mesmo permite ao Cognito bloquear o acesso aos recursos a pedidos provenientes de entidades que não tenham autorização para aceder aos mesmos;

- **Recursos:** onde se definem os alvos dos pedidos a serem recebidos pelo API Gateway. Estes recursos são definidos pelo seu nome, que acrescenta ao URL do API Gateway (como por exemplo, o endereço para o recurso das interações acrescenta \interactions ao URL), e pelo método (GET, POST, PATCH, ...). Estes recursos podem ser visualizados na figura 5.12. Foi, também, dentro dos recursos, necessário configurar o método do pedido e a integração do mesmo, de acordo com as autorizações do Cognito e o VPC Link criado, respetivamente. As configurações de integração podem ser verificadas na figura 5.13.

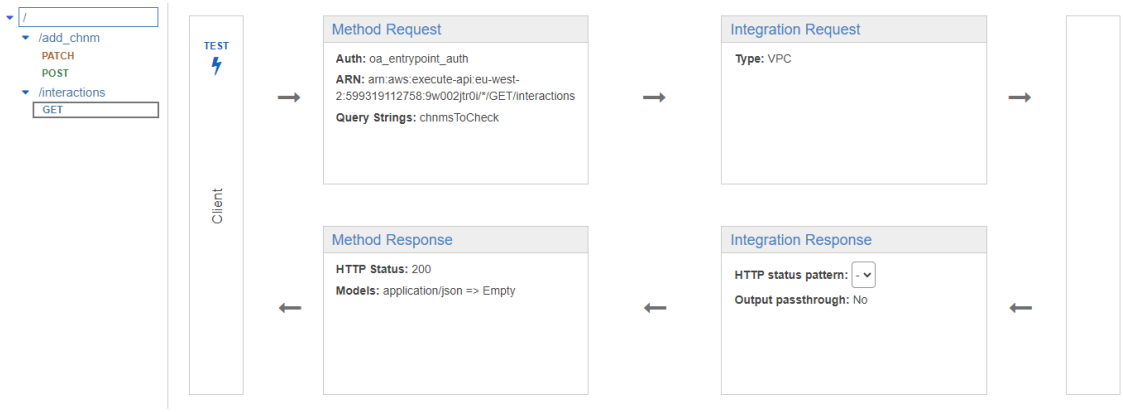

Figura 5.12: Recursos no API Gateway

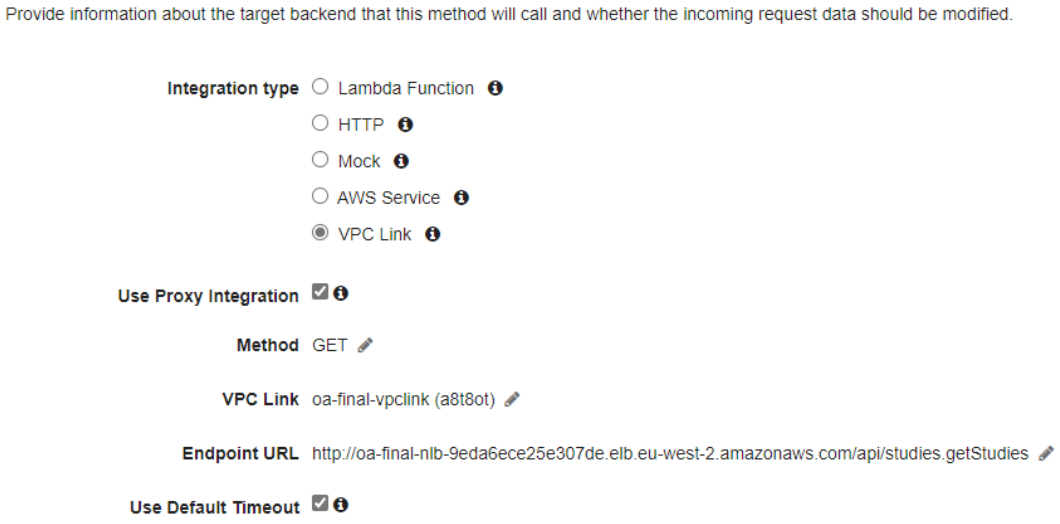

Figura 5.13: Configurações de Integração

5.5.8 Correções na Infraestrutura

Neste momento seria de esperar que todo o projeto estivesse em correto funcionamento, o que não foi confirmado. Através das consolas de *logs* dos produtos implementados (API Gateway, NLB, Fargate e RDS), o estagiário chegou à conclusão que os pedidos de teste, efetuados através do API Gateway, não eram redirecionados para a base de dados.

O primeiro passo que o estagiário seguiu, foi alterar os grupos de segurança utilizados, permitindo agora pedidos HTTP, com o protocolo TCP para o porto 80, provenientes de qualquer IP, e sem qualquer regra de output.

Seguidamente, foram adicionadas as variáveis de ambiente necessárias à conexão à base de dados, na imagem do Docker da tarefa Fargate criada na secção 5.5.4, através de uma revisão de tarefa.

Assim, primeiramente, criou-se um segredo no AWS Secrets Manager, como é visível na figura 5.14, com a informação necessária para esta conexão e, posteriormente, efetuou-se a revisão de tarefa do Fargate.

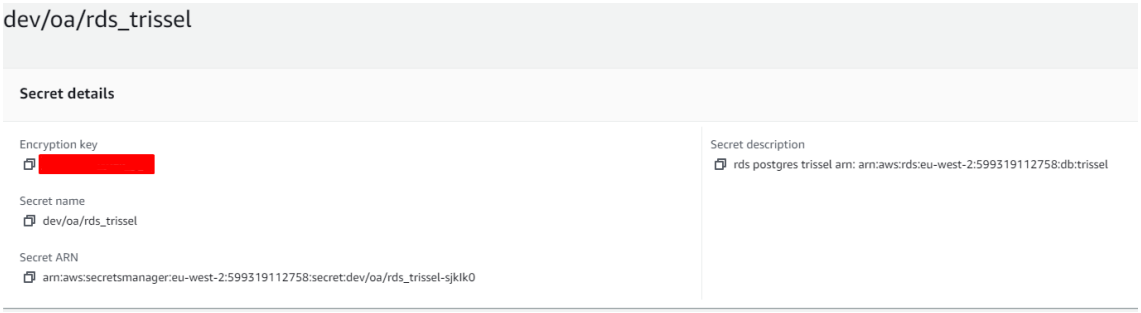

Figura 5.14: Segredo para acesso à Base de Dados RDS

Após se atualizar a versão de tarefa Fargate utilizada no serviço Fargate, o pedido continuava a não executar e percebeu-se, através dos *logs* da tarefa Fargate, que haveria um problema na aplicação. O erro indicava uma má utilização do *base path* da aplicação. No entanto, após *debugging*, o estagiário chegou à conclusão que o problema estava na configuração da API Swagger[110], e do *endpoint* definido para essa API. Após esta correção, e atualização do URL do pedido definido na ligação do API Gateway (para usar HTTP ao invés de HTTPS, uma vez que o ambiente de desenvolvimento não implica certificados SSL, que o HTTPS requer e o HTTP não), toda a arquitetura devolvia uma resposta correta. Esta resposta pode ser visualizada na figura 5.15.

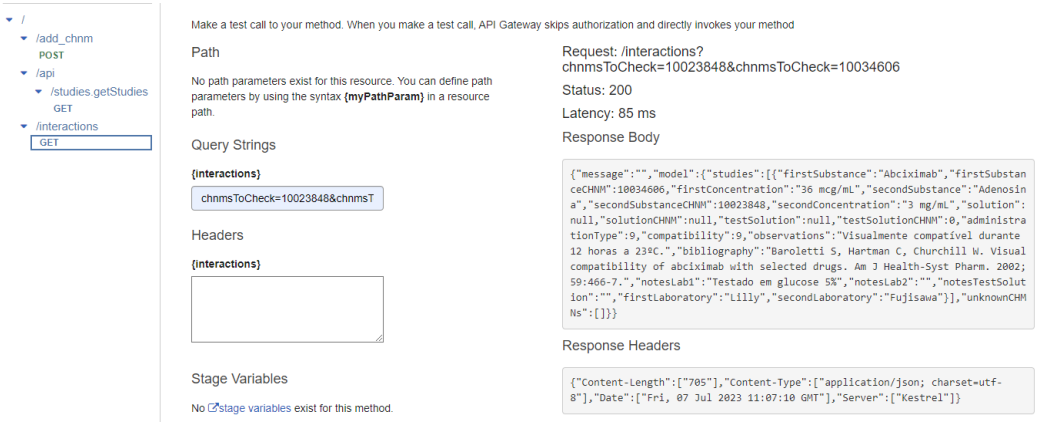

Figura 5.15: Sucesso de um pedido de teste no API Gateway

Posteriormente, o estagiário efetuou o mesmo pedido utilizando a aplicação Postman [109], obtendo sucesso novamente. A execução deste pedido pode ser visualizada na figura 5.16.

5.6 Sumário e Discussão do Capítulo

Neste ponto do projeto, foi considerado pelo estagiário e o seu orientador, que o mesmo estava preparado para o processo de testagem elaborado pelo estagiário e apresentado no capítulo 6.

De notar que, apesar de o desenvolvimento desta prova de conceito poder assemelhar-se a algo simples, não o foi. Nem todos os problemas com que o estagiário se encontrou estão relatados neste documento, apenas estão resumidos os mais complicados. Para ser um bom praticante de soluções em ambiente *cloud* é necessário muito conhecimento e planeamento de todos os produtos a usar em cada arquitetura definida. As configurações de cada produto utilizado podem afetar a correta interação com outros produtos, além de que, há produtos que só podem ser criados após outros, devido a dependências entre eles (exemplo: não se pode criar nem a imagem ECR, nem a tarefa Fargate, e nem o serviço Fargate, sem terminar a implementação da base de dados,

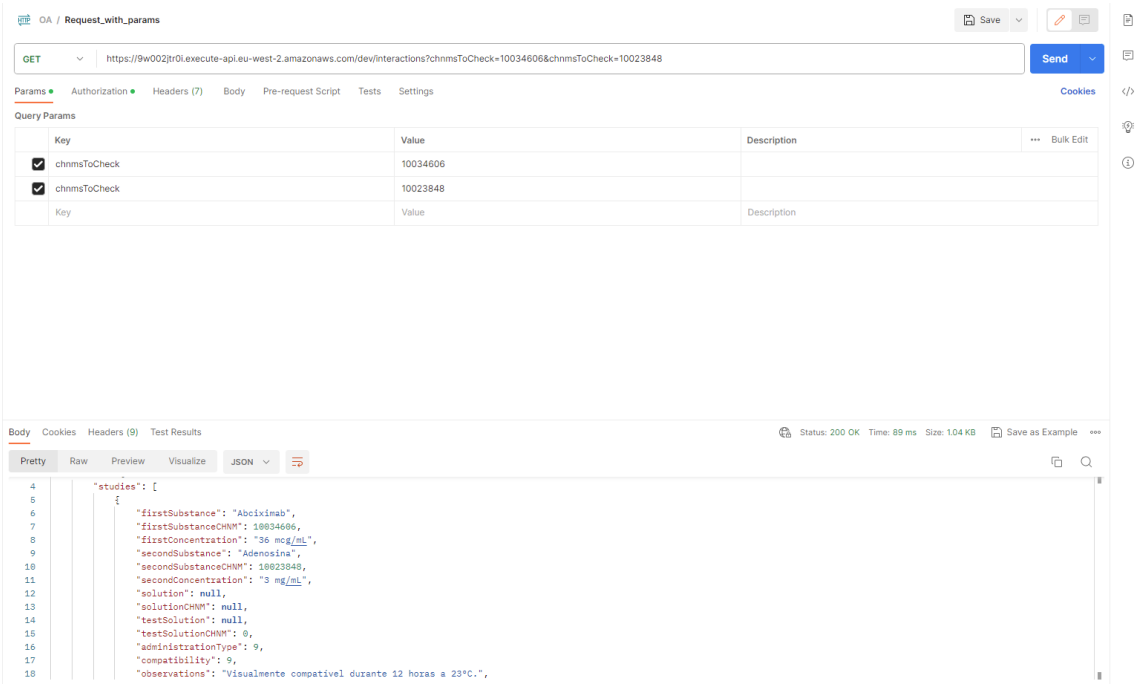

Figura 5.16: Sucesso de um pedido de teste no Postman

pois o código fonte da aplicação necessita de aceder à mesma, e todos estes três componentes, dependem do código fonte e do *container* do Docker).

Definitivamente a maior dificuldade na implementação desta prova de conceito, foi a falta de experiência, o que afetou/atrasou muito o planeamento e, consequentemente, todo o processo, havendo algumas pequenas alterações à arquitetura durante a implementação. Esta falta de experiência e conhecimento, teve outro efeito. O constante receio de ativar, inconscientemente, custos de produtos AWS demasiado elevados. Como por exemplo, algum *debugging* teria sido mais fácil de efetuar utilizando e configurando o AWS CloudWatch [83], obtendo melhores *logs*, mas o estagiário não quis ter custos que poderiam, possivelmente, ser evitados.

Capítulo 6

Testes

6.1 Planeamento de Testes

O planeamento dos testes foi efetuado pelo estagiário e aprovado pelo orientador. Antecipadamente, foi previsto que alguns testes iriam falhar, uma vez que o código da aplicação ainda não estaria completo. Sendo que o ambiente atual do projeto é de *development*, o requisito não funcional **escalonamento** não será testado neste ponto do projeto, apesar de ter sido verificado, pelo estagiário, que pelo menos dois pedidos conseguem ser efetuados em simultâneo, para garantir que a prova de conceito conseguirá, futuramente, ser melhorada para cumprir as necessidades da MedicineOne. Estes fatores serão abordados mais extensamente na secção 7.1.

Assim sendo, o estagiário preparou o ambiente de testes para que, posteriormente, o provador os consiga reproduzir. Os testes que foram realizados são divididos em quatro categorias, que serão descritas em seguida.

- **Testes Unitários**, que irão testar o correto funcionamento da principal função do programa implementado em C#;
- **Testes de Integração**, com o intuito de fazer testagem da integração de toda a aplicação e do *container* Docker;
- **Testes de Sistema**, que serão efetuados para verificar o correto funcionamento de toda a arquitetura *cloud*;
- **Testes de Aceitação**, cujo objetivo é testar os requisitos não funcionais definidos na secção 2.3.2.

Assim sendo, foi necessário criar um ambiente partilhado na ferramenta Postman [109], para que apenas seja necessário pelo provador alterar os *inputs* e crie novos *tokens* de autorização durante os testes de sistema e de aceitação. Adicionalmente, tornou-se importante criar um projeto MSTest [118], no projeto da aplicação, para efetuar o processo de testes unitários. Um exemplo de teste unitário pode ser visualizado na figura 6.1.

De acordo com o que foi referido anteriormente, não foi necessário criar ambientes adicionais para os testes de integração, uma vez que a aplicação já continha na sua implementada a API Swagger [110], que facilitou a execução destes testes.

6.2 Execução de Testes

A execução dos testes foi efetuada de acordo com o planeamento do estagiário, e todos os testes obtiveram os resultados previstos. Porém, nem todos considerados corretos, uma vez que os resultados estarem de acordo com o previsto, estes não cumprem os objetivos finais do projeto.

```
[TestMethod]
0 referências | 0 alterações | 0 autores, 0 alterações
public void TestHandle5()
{

    List<int> inputs = new List<int>();
    inputs.Add(10034606);
    inputs.Add(0);
    inputs.Add(1);

    var studies = new GetStudiesQueryHandlerTest(connectionString).Handle(new GetStudiesQuery(inputs), new CancellationToken());

    Assert.AreEqual(null, studies.Result.Model, "Erro: pedido nao executou corretamente");
}
```

Figura 6.1: Exemplo de Unit Test

O provador apenas teve que preencher a coluna "Resultado Efetivo"em todas as tabelas presentes no documento no anexo A e assinalar o sucesso de cada teste na coluna "Sucesso".

6.3 Sumário e Discussão do Capítulo

Em concordância com aquilo que foi referido anteriormente, todos os testes indicam que a aplicação não está totalmente preparada. No entanto, considera-se que os resultados obtidos se podem classificar como positivos, uma vez que estão de acordo com aquilo que foi planeado no decorrer do projeto. Deste modo, o plano de testes encontra-se adaptado à fase em que o projeto se encontrada, sendo que, futuramente, este deve ser aprofundado de acordo com as alterações que manifestem essa necessidade.

Capítulo 7

Conclusão

7.1 Análise do Planeamento

Como afirmado no capítulo 6, o projeto foi concluído com sucesso, porém, não está de acordo com aquilo que seria o seu maior potencial. Nesse sentido, verifica-se a falta de validações no código e o uso de uma biblioteca para facilitar a transição de objetos entre camadas da arquitetura. No entanto, algumas funcionalidades, por exemplo, *logging*, foram excluídas deste projeto propositadamente. O projeto do ClinicalBrain utiliza uma tecnologia denominada Seq [119] que será introduzida ao estagiário no seu período de integração na equipa do ClinicalBrain. Esta tecnologia automatiza *logs* e alertas sobre eventos, tornando-se redundante a aplicação de outra metodologia de implementação de *logs* neste projeto.

No decorrer deste projeto e da forma como este se foi desenvolvendo, o orientando em conjunto com o seu orientador decidiram que o o editor de conteúdo da base de dados não seria desenvolvido, mudança esta aplicada através da metodologia Scrum. No entanto, foram implementadas funções AWS Lambda que correspondem aos *endpoins* que o mesmo editor de conteúdo utilizaria para efetuar as alterações na base de dados.

As funcionalidades essenciais do projeto foram cumpridas, mas os seus requisitos funcionais não o foram por completo. Apesar do nível de segurança pretendido para esta prova de conceito ter sido alcançado, o escalonamento e a resiliência não se encontram no melhor desempenho ou não foram comprovados até ao momento. No entanto, também foi definido que, para a resiliência, a próxima etapa seria a correção das falhas no código do projeto (falta de validações) e que, para alcançar ambos estes requisitos não funcionais e melhorar a segurança da arquitetura, será utilizado o AWS Well-Architected, para efetuar uma revisão da arquitetura. Além disso, foi ainda efetuada, durante o projeto, uma análise pessoal de como a arquitetura se deverá assemelhar num futuro próximo, sendo esta visível na figura 7.1.

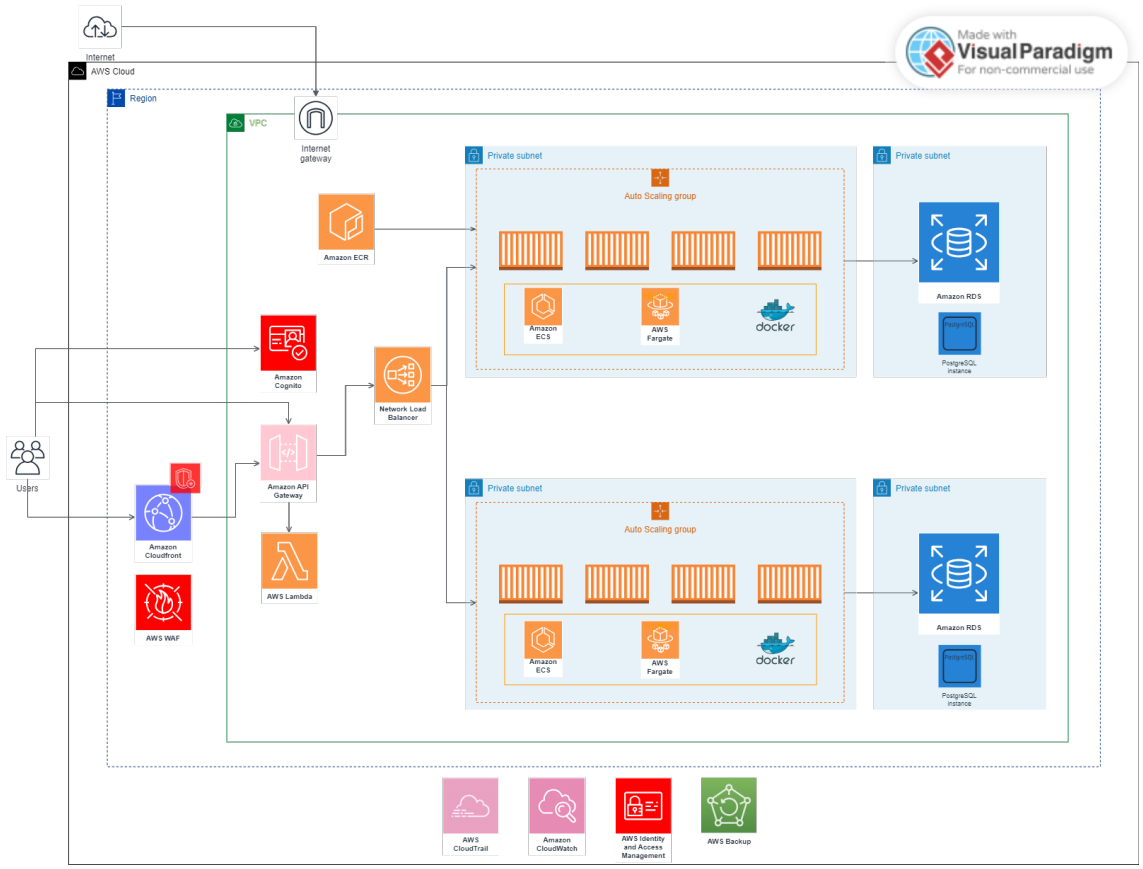

Figura 7.1: Plano para o futuro da arquitetura do projeto

De notar que, apesar de não estar representado na figura, 4.2, atualmente o serviço Fargate e a base de dados já são implementados em várias *subnets* (no máximo três). Isto pode ser explicado devido à falta apresentação de testes que o comprovasse e, deste modo, este dado não foi representado.

7.2 Trabalho Futuro

Antes de começar a desenvolver os objetivos futuros para o projeto, é necessário referir que o estagiário possui contrato como estagiário com a MedicineOne até Dezembro de 2023, tendo assim, se necessário, um maior período para continuar a desenvolver esta prova de conceito. Apesar de existir essa possibilidade, após discutir o seu futuro com os seus superiores, foi indicado ao estagiário que, o cenário mais plausível é que o mesmo seja integrado na equipa do ClinicalBrain, que trabalham neste momento para isolar todas as funcionalidade do monólito com o mesmo nome, em micro-serviços, tal como o projeto que o estagiário desenvolveu. Assim, o estagiário beneficiou deste projeto, como uma introdução ou aprendizagem do conhecimento necessário para a sua integração nesta equipa.

No entanto, caso seja determinado que o estagiário deve continuar a desenvolver este projeto em específico e não avançar de imediato para a equipa do ClinicalBrain, onde desenvolveria a mesma solução com acesso a mais recursos, o mesmo irá aplicar as alterações referidas na secção 7.1.

7.3 Comentários e Conclusões

Face ao trabalho desenvolvido durante o primeiro semestre neste projeto, o estagiário considera que o esforço alocado ao mesmo foi recompensador, mesmo que exista ainda bastante potencial de melhoria. Assim sendo, houve uma contínua busca por melhores resultados para aplicar no projeto e, nesta fase final, esta prova de conceito tornou-se uma mais valia para a aquisição de novas competências pessoais e profissionais. Ao longo do mesmo, os conhecimentos em C#, AWS (ou qualquer *cloud provider*) e Docker foram cada vez mais aprofundados, sendo que até ao início deste projeto, eram tecnologias praticamente novas para o estagiário.

Ao entrar em mais detalhe e fazendo uma revisão de todo o seu trabalho, o estagiário apercebe-se agora, naturalmente, que houveram oportunidades de utilizar ferramentas que facilitariam o seu avanço no projeto, como é o caso do Amazon CloudWatch [83], que poderia ter facilitado a deteção de problemas de arquitetura entre os vários produtos AWS utilizados.

Referências

[1] 1. Medicineone: M1. https://www.medicineone.net/m1, 2 Janeiro 2023.

[2] 2. Medicineone: Quem somos? https://www.medicineone.net/empresa, 2 Janeiro 2023.

[3] 34. Jira software. https://www.atlassian.com/software/jira, 9 Janeiro 2023.

[4] 10. What are micro-services. https://aws.amazon.com/microservices/, 2 Janeiro 2023.

[5] 11. What are open-source databases. https://aws.amazon.com/pt/products/databases/open-source-databases/, 2 Janeiro 2023.

[6] 12. What is postgresql. https://www.postgresql.org/about/, 2 Janeiro 2023.

[7] 13. What big data. https://www.oracle.com/pt/big-data/what-is-big-data/, 2 Janeiro 2023.

[8] 88. What is scalability in cloud computing. https://acloudguru.com/blog/engineering/scalability-cloud-computing, 3 Janeiro 2023.

[9] 15. Vertical vs. horizontal scaling. https://www.ibm.com/blogs/cloud-computing/2014/04/09/explain-vertical-horizontal-scaling-cloud/, 3 Janeiro 2023.

[10] 16. Manual, scheluded and automatic scaling. https://cloudcheckr.com/cloud-automation/horizontal-vertical-cloud-scaling/, 3 Janeiro 2023.

[11] 17. Cloud-native resilience. https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/resiliency, 3 Janeiro 2023.

[12] 18. Docker container. https://www.docker.com/resources/what-container/, 5 Janeiro 2023.

[13] 89. What is resiliency by ibm. https://www.ibm.com/br-pt/z/resiliency, 16 Fevereiro 2023.

[14] 87. Matriz de riscos. https://ferramentasdaqualidade.org/matriz-de-riscos-matriz-de-probabilidade-e-impacto/, 6 Fevereiro 2023.

[15] 4. What is a cloud service provider. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-cloud-provider/, 2 Janeiro 2023.

[16] 5. What is iaas. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/, 2 Janeiro 2023.

[17] 32. What is caas. https://www.redhat.com/en/topics/cloud-computing/what-is-caas, 8 Janeiro 2023.

[18] 6. What is paas. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/, 2 Janeiro 2023.

[19] 9. What is kubernetes. https://kubernetes.io/docs/concepts/overview/, 2 Janeiro 2023.

[20] 7. What is saas. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas/, 2 Janeiro 2023.

[21] 8. What is serverless computing. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-serverless-computing/, 2 Janeiro 2023.

[22] 31. What is public cloud. https://www.ibm.com/topics/public-cloud, 7 Janeiro 2023.

[23] 30. What is private cloud. https://www.ibm.com/topics/private-cloud, 7 Janeiro 2023.

[24] 48. Lawrence a. trissel. http://www.ashpmedia.org/bookstore/bio/Bio_Trissel.html, 2 Janeiro 2023.

[25] 47. Plataforma de drogas injetáveis. https://www.ashp.org/products-and-services/injectables?loginreturnUrl=SSOCheckOnly, 12 Janeiro 2023.

[26] 43. Uptodate. https://www.wolterskluwer.com/pt-br/solutions/uptodate/solutions-overview, 12 Janeiro 2023.

[27] 44. Dynamed. https://www.dynamed.com/, 12 Janeiro 2023.

[28] 45. Bjm best practise. https://bestpractice.bmj.com/info/, 12 Janeiro 2023.

[29] 46. Plataformas de apoio à decisão clínica. https://www.jornalmedico.pt/atualidade/37866-plataformas-de-apoio-a-decisao-clinica.html, 12 Janeiro 2023.

[30] 3. Statista: Cloud providers market 2022. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/, 2 Janeiro 2023.

[31] 49. Amazon. https://www.amazon.com/, 13 Janeiro 2023.

[32] 50. Microsoft. https://www.microsoft.com/, 13 Janeiro 2023.

[33] 51. Google. https://www.google.com/, 13 Janeiro 2023.

[34] 85. Infraestrutura global aws. https://aws.amazon.com/pt/about-aws/global-infrastructure/, 6 Fevereiro 2023.

[35] 86. Infraestrutura global gcp. https://cloud.google.com/about/locations?hl=pt-br, 6 Fevereiro 2023.

[36] 52. Vpcs. https://docs.aws.amazon.com/vpc/latest/userguide/configure-your-vpc.html, 13 Janeiro 2023.

[37] 84. Cloud providers networking. https://www.youtube.com/watch?v=o6l5Aw4Isns, 6 Fevereiro 2023.

[38] 53. Expressroute. shorturl.at/EIPW1, 14 Janeiro 2023.

[39] 54. Cloud interconnect. https://cloud.google.com/network-connectivity/docs/interconnect/concepts/overview, 14 Janeiro 2023.

[40] 59. Elastic beanstalk. https://aws.amazon.com/pt/elasticbeanstalk/, 14 Janeiro 2023.

[41] 60. Virtual machines azure. https://azure.microsoft.com/en-us/products/virtual-machines/#features, 14 Janeiro 2023.

[42] 58. Compute engine. shorturl.at/ejPY7, 14 Janeiro 2023.

[43] 55. Amazon rds. https://aws.amazon.com/pt/rds/, 14 Janeiro 2023.

[44] 57. Virtual machines. shorturl.at/eFP34, 14 Janeiro 2023.

[45] 81. Modelo responsabilidade partilhada aws. https://aws.amazon.com/pt/compliance/shared-responsibility-model/, 6 Fevereiro 2023.

[46] 82. Modelo responsabilidade partilhada azure. https://learn.microsoft.com/pt-pt/azure/security/fundamentals/shared-responsibility, 6 Fevereiro 2023.

[47] 83. Modelo responsabilidade partilhada gcp. https://cloud.google.com/architecture/framework/security/shared-responsibility-shared-fate?hl=pt-br, 6 Fevereiro 2023.

[48] 61. Aws shield. https://aws.amazon.com/pt/shield/, 14 Janeiro 2023.

[49] 62. Ddos protection. https://azure.microsoft.com/en-us/products/ddos-protection, 14 Janeiro 2023.

[50] 63. Google cloud armor. https://cloud.google.com/armor, 2 Janeiro 2023.

[51] 64. Secrets manager aws. https://aws.amazon.com/pt/secrets-manager/, 14 Janeiro 2023.

[52] 65. Key vault. https://azure.microsoft.com/en-us/products/key-vault/, 14 Janeiro 2023.

[53] 66. Secret manager gcp. https://cloud.google.com/secret-manager, 14 Janeiro 2023.

[54] 67. Iso 27001. https://www.27001.pt/, 14 Janeiro 2023.

[55] 68. Amazon inspector. https://docs.aws.amazon.com/managedservices/latest/userguide/inspector.html, 14 Janeiro 2023.

[56] 69. Azure security center. https://azure.microsoft.com/pt-pt/blog/azure-security-center-extends-advanced-threat-protection-to-hybrid-cloud-workloads/, 14 Janeiro 2023.

[57] 70. Security command center. https://cloud.google.com/security-command-center, 14 Janeiro 2023.

[58] 72. Elastic container registry. https://aws.amazon.com/pt/ecr/, 14 Janeiro 2023.

[59] 73. Container registry. https://azure.microsoft.com/en-us/products/container-registry, 4 Janeiro 2023.

[60] 74. Artifact registry. https://cloud.google.com/artifact-registry, 14 Janeiro 2023.

[61] 78. Elastic kubernetes service. https://aws.amazon.com/pt/eks/, 14 Janeiro 2023.

[62] 79. Azure kubernetes service. https://azure.microsoft.com/en-us/products/kubernetes-service/, 14 Janeiro 2023.

[63] 80. Google kubernetes engine. https://cloud.google.com/kubernetes-engine, 14 Janeiro 2023.

[64] 75. Bare metal servers. https://pt.wikipedia.org/wiki/Bare-metal_server, 14 Janeiro 2023.

[65] 76. Service mesh. https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh, 14 Janeiro 2023.

[66] 77. Istio. https://istio.io/, 14 Janeiro 2023.

[67] 71. .net. https://dotnet.microsoft.com/pt-br/, 14 Janeiro 2023.

[68] 19. Who are ibm. https://pt.wikipedia.org/wiki/IBM, 7 Janeiro 2023.

[69] 20. Lloyds bank. https://www.lloydsbank.com/, 7 Janeiro 2023.

[70] 33. Coca-cola. https://www.coca-cola.com/, 8 Janeiro 2023.

[71] 22. Ibm cloud objectives. https://www.ibm.com/docs/en/odm/8.10?topic=private-benefits-cloud, 7 Janeiro 2023.

[72] 21. Ibm cloud private. https://www.ibm.com/docs/en/cloud-private/3.1.1?topic=started-cloud-private-overview, 7 Janeiro 2023.

[73] 23. Ibm vs. aws. https://www.ibm.com/cloud/ibm-cloud-vs-aws, 7 Janeiro 2023.

[74] 24. Ibm vs. aws study. https://www.ibm.com/downloads/cas/OMN0KGWJ, 7 Janeiro 2023.

[75] 25. Mazda. https://www.mazda.com/, 7 Janeiro 2023.

[76] 26. Deutsche bank. https://www.db.com/index?language_id=1&kid=sl.redirect-en.shortcut, 7 Janeiro 2023.

[77] 27. Oci vs. aws. https://www.oracle.com/cloud/oci-vs-aws/, 7 Janeiro 2023.

[78] 35. Onion architecture. https://marcoatschaefer.medium.com/onion-architecture-explained-building-maintainable-software-54996ff8e464, 10 Janeiro 2023.

[79] 36. Domain driven design. https://en.wikipedia.org/wiki/Domain-driven_design, 10 Janeiro 2023.

[80] 38. Sqlserver. https://www.microsoft.com/pt-br/sql-server/sql-server-2019, 11 Janeiro 2023.

[81] 39. Amazon rds. https://aws.amazon.com/pt/rds/?did=ft_card&trk=ft_card, 11 Janeiro 2023.

[82] 40. Aws audit manager. https://aws.amazon.com/pt/audit-manager/, 11 Janeiro 2023.

[83] 41. Amazon cloudwatchr. https://aws.amazon.com/pt/cloudwatch/, 11 Janeiro 2023.

[84] 42. Produtos de segurança aws. https://aws.amazon.com/pt/products/security/?nc2=h_ql_prod_se_ic, 11 Janeiro 2023.

[85] 112. Visual paradigm. https://www.visual-paradigm.com/, 15 Março 2023.

[86] 93. What is amazon cognito. https://aws.amazon.com/pt/cognito/, 13 Março 2023.

[87] 119. Aws iam. https://aws.amazon.com/pt/iam/, 1 Abril 2023.

[88] 131. restapi. https://aws.amazon.com/pt/secrets-manager/, 15 Junho 2023.

[89] 96. What is aws lambda. https://aws.amazon.com/pt/lambda/, 13 Março 2023.

[90] 97. What is amazon ecr. https://aws.amazon.com/pt/ecr/, 13 Março 2023.

[91] 99. What is aws fargate. https://aws.amazon.com/pt/fargate/, 13 Março 2023.

[92] 100. What is a network load balancer. https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html, 13 Março 2023.

[93] 101. What is the osi model. https://www.geeksforgeeks.org/layers-of-osi-model/, 13 Março 2023.

[94] 102. What is tcp. https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/, 13 Março 2023.

[95] 103. What is udp. https://www.geeksforgeeks.org/user-datagram-protocol-udp/, 13 Março 2023.

[96] 104. What is tls. https://www.geeksforgeeks.org/transport-layer-security-tls/, 13 Março 2023.

[97] 120. Amazon aurora. https://aws.amazon.com/pt/rds/aurora/?pg=ln&sec=hiw, 14 Abril 2023.

[98] 107. What is amazon api gateway. https://aws.amazon.com/pt/api-gateway/, 13 Março 2023.

[99] 108. What is a restful api. https://aws.amazon.com/pt/what-is/restful-api/, 13 Março 2023.

[100] 109. What is aws cloudwatch. https://aws.amazon.com/pt/cloudwatch/, 13 Março 2023.

[101] 110. What is aws cloudtrail. https://aws.amazon.com/pt/cloudtrail/, 13 Março 2023.

[102] 113. Sandbox ou. https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/sandbox-ou.html, 1 Abril 2023.

[103] 114. Tutorial onion architecture & docker. https://code-maze.com/onion-architecture-in-aspnetcore/, 1 Abril 2023.

[104] 115. Dockerfile. https://docs.docker.com/engine/reference/builder/, 1 Abril 2023.

[105] 116. Docker-compose. https://docs.docker.com/compose/compose-file/build/, 1 Abril 2023.

[106] 117. Ide. https://aws.amazon.com/pt/what-is/ide/, 1 Abril 2023.

[107] 122. Ssms. https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16, 28 Abril 2023.

[108] 121. Dbeaver. https://dbeaver.io/, 28 Abril 2023.

[109] 123. Postman. https://www.postman.com/, 28 Abril 2023.

[110] 124. Swagger api. https://swagger.io/, 28 Abril 2023.

[111] 125. port. https://www.cloudflare.com/learning/network-layer/what-is-a-computer-port/, 7 Maio 2023.

[112] 126. yaml. https://shorturl.at/dLNTV, 7 Maio 2023.

[113] 118. Vpc subnets. https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html, 1 Abril 2023.

[114] 127. pgadmin. https://www.pgadmin.org/, 18 Maio 2023.

[115] 128. oauth2.0. https://oauth.net/2/, 15 Junho 2023.

[116] 129. python. https://www.python.org/, 15 Junho 2023.

[117] 130. restapi. https://www.ibm.com/topics/rest-apis, 15 Junho 2023.

[118] 132. mstest. https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest, 8 Julho 2023.

[119] 133. seq. https://datalust.co/seq, 8 Julho 2023.

Anexos

Anexo A

Plano e Casos de Teste

Julho de 2023

Plano e Casos de Teste

Sistema de Apoio à Decisão, Deteção de Erros e Formação Médica em Terapêuticas Injetáveis

HENRIQUE TEIXEIRAFONSECA

Conteúdo

1. Introdução 2
2. Âmbito 2
   a. Abrangência 2
   b. Objetivos de Qualidade 2
   c. Funções e Responsabilidades 2
3. Metodologia de testes 3
   a. Visão Global 3
   b. Níveis de Teste 3
   c. Triagem de Erros 3
   d. Critérios de Suspensão e de Resumo 3
   e. Integridade dos Testes 4
4. Entregáveis 4
5. Ambiente e Recursos 4
   a. Ferramentas 4
   b. Ambiente 4

1. Introdução

Este documento visa avaliar o módulo de Apoio à Decisão, Deteção de Erros e Formação Médica em Terapêuticas Injetáveis, que foi isolado da aplicação monolítica da MedicineOne, Clinical Brain, como um micro-serviço, e colocado numa arquitetura *cloud* no *cloud provider* Amazon Web Services (AWS). O micro-serviço, desenvolvido em C#, utilizando uma arquitetura denominada Onion-Arquitecture, foi colocado num *container* Docker, de forma efetuar a sua implantação no ambiente *cloud*. Uma vez que o módulo isolado consiste numa aplicação com um *endpoint*, a sua testagem deverá ser correta, mas não muito prolongada.

2. Âmbito

a. Abrangência

Como já foi supramencionado, o módulo a testar consiste numa aplicação w*eb* com apenas um *endpoint*, que será alvo de testes unitários e de integração com a aplicação e com a base de dados em PostgreSQL, que armazena a informação que visa responder.

A aplicação, por sua vez, irá incorporar os testes de sistema com o ambiente *cloud* na qual foi inserida, sendo que os pedidos provenientes do provador (*tester*) serão filtrados por vários produtos e configurações da AWS, tais como: API Gateway; Network Load Balancer; Instância Fargate; Security Groups e Validações do AWS Cognito.

b. Objetivos de Qualidade

O objetivo de qualidade dos testes tem como principal intuito validar a integração e o correto funcionamento de toda a arquitetura e da aplicação. Além disso, serve também para verificar se os requisitos do projeto estão funcionais ou não. Assim sendo, descreve-se de forma breve os requisitos e as suas funcionalidades, mencionados anteriormente.

- **Segurança:** verificar se as configurações do AWS Cognito e dos Security Groups de cada um dos produtos AWS utilizados, filtram pedidos indesejados;
- **Escalonamento:** verificar se o Fargate produz novas instâncias consoante a quantidade de utilizadores a efetuar pedidos;
- **Resiliência:** verificar se o Fargate produz novas instâncias, mantendo um número mínimo de instâncias saudáveis, definido no projeto.

c. Funções e Responsabilidades

De acordo com aquilo que são as funções e responsabilidades para efeitos de testagem, existe um Manager de Testes e um Provador, sendo estes Henrique Fonseca e Elsio Cardoso, respetivamente.

O Manager de Testes está responsável por tomar todas as decisões sobre o ambiente e a metodologia em que os testes serão efetuados, assim como, deve fornecer ao Provador toda a informação necessária para que este efetuar os testes corretamente. Quanto ao Provador, este tem como principal função efetuar os testes que foram planeados.

3. Metodologia de testes

a. Visão Global

A metodologia de gestão de projeto utilizada para o desenvolvimento do módulo especificado no início deste documento foi o SCRUM, uma metodologia ágil, permitindo assim constante reavaliação e acompanhamento dos objetivos semanais e, consequentemente, finais do projeto. Tendo em conta que, não foi implementada uma aplicação de atualização da informação da base de dados (BD), apenas os *endpoints* (em forma de funções Lambda na AWS) necessários para algumas das operações, que essa aplicação faria, os testes relativos a essa parte do projeto foram adaptados aos *endpoints* implementados.

b. Níveis de Teste

Os níveis de teste que foram realizados dividem-se em quatro categorias, descritas em seguida:

- **Testes unitários:** efetuados sobre as funções que efetua os pedidos à base de dados (quer na aplicação implementada, quer nas funções Lambda de atualização da informação);
- **Testes de integração:** da aplicação correspondente ao módulo de perfusões;
- **Testes de sistema:** de toda a arquitetura AWS, após registar a imagem Docker da aplicação na mesma;
- **Testes de aceitação:** para validar os requisitos não funcionais propostos.

c. Triagem de Erros

Na triagem de erros, os objetivos principais a realçar no decorrer deste procedimento de testagem são: avaliar o cumprimento dos requisitos não funcionas propostos e encontrar erros que deverão ser corrigidos futuramente.

d. Critérios de Suspensão e de Resumo

Idealmente, o procedimento de testagem deveria ser interrompido assim que um dos testes falhasse e, posteriormente, resumido quando submetido um novo código com correções sobre esse teste falhado. No entanto, devido à data de entrega do documento da dissertação ser num futuro próximo, não haverá interrupção, sendo que os testes falhados serão corrigidos em seguida, caso se verifique essa falha.

e. Integridade dos Testes

Todos os testes devem ser efetuados e devem ser concluídos com sucesso, para considerar este procedimento completo, ou seja, não deverão ocorrer erros durante o processo de testagem.

4. Entregáveis

O único entregável necessário, no final do processo de testagem, é este documento preenchido, com a especificação de *inputs*, *outputs*, códigos de erros e/ou algum método de registo dos erros (anexos com informação da consola do IDE, por exemplo). Assim sendo, foi entregue ao provador o documento que contém os requisitos não funcionais do projeto.

5. Ambiente e Recursos

a. Ferramentas

Para efeitos de testagem, as ferramentas utilizadas foram: Visual Studio 2022, Postman e AWS. Durante o processo de testagem, cada caso de teste terá na sua descrição as ferramentas a usar.

b. Ambiente

O ambiente de cada teste que foi executado será descrito em caso de teste, no entanto, não irá variar para os ambientes de Windows 11 e AWS.

6. Testagem

a. Casos de teste

i. Testes unitários

Os testes unitários serão executados apenas sobre uma função **Handle** da classe **GetStudiesQueryHandler**. Esta função executa uma *query* SQL na base de dados que visa obter toda a informação relativa à interação entre dois ou mais medicamentos, selecionados através do campo **CHMD_ID**. O *input* (medicamentos) deverá ser produzido criando um objeto da classe **GetStudiesQuery** e adicionando os valores à sua lista de inteiros **CHNMs**. A resposta será um objeto da classe **GetStudiesResult** que contém duas listas, uma com os resultados das interações (objetos da classe **Study**) e outra lista com inteiros correspondentes aos **CHNM_ID**s, que não estão presentes na base de dados.

O ambiente destes testes deverá ser em Windows 11, utilizando um projeto MSTest presente no código do projeto OA. Todo o projeto OA está presente no IDE Visual Studio.

Os resultados efetivos são definidos por *asserts*, onde são comparados objeto que se espera ser retornado, com o objeto que efetivamente a função **Handle** retorna. Estes objetos podem ser visualizados em formato JSON nos anexos deste documento.

Nota: foi criada uma classe, para efetuar estes testes (**GetStudiesQueryHandlerTest**), com a única diferença para a originar de que recebe a **ConnectionString** como uma string e não através da *interface* **IOptions<ConnectionStrings>.**

Teste	Descrição	Inputs	Resultado Esperado	Resultado Efetivo	Comentário	Sucesso
1	2 medicamentos	CHMD_IDs = 10034606, 10023848	*Assert* confirma que os dois objetos são iguais	*Assert* confirma que os dois objetos são iguais		✓
2	3 medicamentos	CHMD_IDs = 10034606, 10023848, 10035003	*Assert* confirma que os dois objetos são iguais	*Assert* confirma que os dois objetos são iguais		✓
3	2 medicamentos, 1 deles incorreto	CHMD_IDs = 10023848, 0	*Resposta é "null"*	*Resposta é "null"*	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
4	3 medicamentos, 1 deles incorreto	CHMD_IDs = 10034606, 10023848, 0	*Assert* confirma que os dois objetos são iguais	*Assert* confirma que os dois objetos são iguais		✓
5	3 medicamentos, 2 deles incorretos	CHMD_IDs = 10034606, 0, 1	*Resposta é "null"*	*Resposta é "null"*	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
6	2x o mesmo medicamento	CHMD_IDs = 10034606, 10034606	*Resposta é "null"*	*Resposta é "null"*	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
7	2x o mesmo medicamento + 1 medicamento	CHMD_IDs = 10034606, 10034606, 10023848	*Assert* confirma que os dois objetos são iguais	*Assert* confirma que os dois objetos são iguais		✓

ii. Testes de integração

1. Aplicação

Para testar a integração da função **Handle** com o restante código do projeto, executa-se o projeto e utiliza-se a API **Swagger** para executar pedidos à aplicação e à base de dados. Os testes devem ser executados em ambiente Windows 11, com recurso ao Visual Studio.

Teste	Descrição	Inputs	Resultado Esperado	Resultado Efetivo	Comentário	Sucesso
1	0 medicamentos	Não aplicável	Aviso de que são necessários 2 elementos no *array* de *inputs*	Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
2	1 medicamento	CHMD_ID = 10034606	Aviso de que são necessários 2 elementos no *array* de *inputs*	Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
3	2 medicamentos	CHMD_IDs = 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
4	3 medicamentos	CHMD_IDs = 10034606, 10023848, 10035003	Código 200, Response Body com um *array* com dois objetos Study	Código 200, Response Body com um *array* com dois objetos Study		✓
5	2 medicamentos, 1 deles incorreto	CHMD_IDs = 10023848, 0	Código 500, "Internal Server Error"	Código 500, "Internal Server Error"	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	✗

6	3 medicamentos, 1 deles incorreto	CHMD_IDs = 10034606, 10023848, 0	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
7	3 medicamentos, 2 deles incorretos	CHMD_IDs = 10034606, 0, 1	Código 500, "Internal Server Error"	Código 500	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
8	2x o mesmo medicamento	CHMD_IDs = 10034606, 10034606	Código 500, "Internal Server Error"	"Internal Server Error"	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
9	2x o mesmo medicamento + 1 medicamento	CHMD_IDs = 10034606, 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓

2. Docker

Para testar a integração do código com o *container* Docker, executa-se o ficheiro **docker-compose** e utiliza-se a API **Swagger** para executar pedidos à aplicação e à base de dados. Os testes devem ser executados em ambiente Windows 11, com recurso ao Visual Studio.

Teste	Descrição	Inputs	Resultado Esperado	Resultado Efetivo	Comentário	Sucesso
1	0 medicamentos	Não aplicável	Código 400, Aviso de que são necessários 2 elementos no *array* de *inputs*	Código 400, Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
2	1 medicamento	CHMD_ID = 10034606	Código 400, Aviso de que são necessários 2 elementos no *array* de *inputs*	Código 400, Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
3	2 medicamentos	CHMD_IDs = 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
4	3 medicamentos	CHMD_IDs = 10034606, 10023848, 10035003	Código 200, Response Body com um *array* com dois objetos Study	Código 200, Response Body com um *array* com dois objetos Study		✓
5	2 medicamentos, 1 deles incorreto	CHMD_IDs = 10023848, 0		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×

6	3 medicamentos, 1 deles incorreto	CHMD_IDs = 10034606, 10023848, 0	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
7	3 medicamentos, 2 deles incorretos	CHMD_IDs = 10034606, 0, 1		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
8	2x o mesmo medicamento	CHMD_IDs = 10034606, 10034606		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
9	2x o mesmo medicamento + 1 medicamento	CHMD_IDs = 10034606, 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓

iii. Testes de sistema

Para executar os testes de sistema, onde se testa toda a arquitetura AWS definida, utiliza-se a aplicação Postman. Para este fim, foi configurado um ambiente partilhado nesta aplicação para execução dos testes. Os pedidos são efetuados para o API Gatweay e, podem ou não necessitar, da geração de um *token* de autorização, também definido neste ambiente partilhado. O ambiente destes testes é o Windows 11, com recurso à aplicação Postman.

Teste	Descrição	Inputs	Resultado Esperado	Resultado Efetivo	Comentários	Sucesso
1	0 medicamentos	Não aplicável	Aviso de que são necessários 2 elementos no *array* de *inputs*	Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
2	1 medicamento	CHMD_ID = 10034606	Aviso de que são necessários 2 elementos no *array* de *inputs*	Aviso de que são necessários 2 elementos no *array* de *inputs*		✓
3	2 medicamentos	CHMD_IDs = 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
4	3 medicamentos	CHMD_IDs = 10034606, 10023848, 10035003	Código 200, Response Body com um *array* com dois objetos Study	Código 200, Response Body com um *array* com dois objetos Study		✓

5	2 medicamentos, 1 deles incorreto	CHMD_IDs = 10023848, 0		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
6	3 medicamentos, 1 deles incorreto	CHMD_IDs = 10034606, 10023848, 0	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
7	3 medicamentos, 2 deles incorretos	CHMD_IDs = 10034606, 0, 1		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
8	2x o mesmo medicamento	CHMD_IDs = 10034606, 10034606		Indefinidamente à espera de resposta	Não foi implementada forma de controlar o objeto retorno com array sem nenhum objeto Study	×
9	2x o mesmo medicamento + 1 medicamento	CHMD_IDs = 10034606, 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓

iv. Testes de aceitação

Os testes de aceitação visam a validação dos requisitos não funcionais do projeto que, neste caso, são a **segurança**, a **resiliência** e o **escalonamento**.

Nesta fase do projeto, verificou-se que ainda não seria adequado estudar de forma aprofundada o **escalonamento**, uma vez que é uma característica necessária ao ambiente de produção e não de desenvolvimento. Mesmo que tenha sido implementado um certo nível de escalonamento na arquitetura atual, não satisfará os níveis necessários para o produto final. A **resiliência** já foi afetada, o que foi verificado anteriormente em testes que não suportam pedidos com *inputs* errados, não havendo justificação para a continuar a testar com eventos mais complexos.

Deste modo, os testes que se seguem são relativos à **segurança**, mais especificamente, à autorização de acesso ao produto. Para este propósito, apenas devem aceder a esta autorização de acesso as entidades que possuam um *token* que verifique as suas permissões. O ambiente utilizado nestes testes consiste no uso do sistema operativo Windows 11, com recurso à ferramenta Postman.

Teste	Descrição	Inputs	Resultado Esperado	Resultado Efetivo	Comentário	Sucesso
1	Pedido com Token JWT válido	CHMD_IDs = 10034606, 10023848	Código 200, Response Body com um *array* com um objeto Study	Código 200, Response Body com um *array* com um objeto Study		✓
2	Pedido com Token JWT inválido	CHMD_IDs = 10034606, 10023848	Código 401, mensagem de token expirado	Código 401, mensagem de token expirado		✓
3	Pedido sem Token JWT	CHMD_IDs = 10034606, 10023848	Código 401, Mensagem “Unauthorized”	Código 401, Mensagem “Unauthorized”		✓