Cláudia Almeida de Aguiar Tavares

ORQUESTRAÇÃO INTELIGENTE EM CLOUD NATIVE

Relatório de Estágio no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Doutor Fernando Pedro Lopes Boavida Fernandes e à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

julho de 2022

Cláudia Almeida de Aguiar Tavares

ORQUESTRAÇÃO INTELIGENTE EM CLOUD NATIVE

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Cláudia Almeida de Aguiar Tavares

ORQUESTRAÇÃO INTELIGENTE EM CLOUD NATIVE

Relatório de Estágio no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Doutor Fernando Pedro Lopes Boavida Fernandes e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

julho de 2022

Agradecimentos

À minha família,

Aos meus orientadores, Professor Doutor Fernando Pedro Lopes Boavida Fernandes,

Luís Miguel Batista Rosa e Luís Filipe Vieira Cordeiro,

A todos aqueles que me acompanham e estão sempre presentes.

Resumo

A crescente inovação tecnológica e digitalização dos setores económicos e indústrias potencia a transição de infraestruturas informáticas para a nuvem, uma vez que as suas vantagens cada vez mais se tornam conhecidas, inclusive para uso pessoal. Esta transição também acarreta algumas preocupações de segurança, devido à sua característica de *multi-tenancy*, característica que aumenta a necessidade de automação dos processos, e consequentemente a diminuição da interação humana. Neste sentido, surge a necessidade de implementação de uma Orquestração Inteligente e Automatizada de ambientes multi-domínios, onde seja possível avaliar as necessidades do ambiente, e agir em conformidade com as mesmas.

Neste âmbito, a OneSource integra o projeto de investigação Europeu CHARITY para o desenvolvimento de um ambiente de Orquestração Inteligente que permita a implementação de serviços XR. De acordo com este enquadramento, foi proposto o estágio para o desenvolvimento e validação de mecanismos que permitam a orquestração de diversos domínios em nuvem, no qual surge este relatório, que documenta o trabalho desenvolvido no âmbito da unidade curricular "Dissertação/Estágio" do Mestrado em Segurança Informática da Universidade de Coimbra em parceria com a OneSource.

Este trabalho permitiu explorar a implementação de domínios em nuvem, através de OpenStack, e a implementação de mecanismos de automação, através da execução de *playbooks* Ansible que permitam facilitar a orquestração multi-domínios. Estes *playbooks* foram integrados na ferramenta AWX, de forma a permitir uma gestão simplificada dos mesmos. A orquestração de Clusters também possui um peso bastante significativo, e como tal, foi realizada a exploração da ferramenta Rancher, bem como de mecanismos de automação nestes ambientes. Adicionalmente, foi elaborada a monitorização do ambiente através de Prometheus e Grafana, bem como foram elaborados Operadores que permitem a migração automatizada de microserviços. Além disto, este trabalho permitiu também a integração de cenários multi-cluster, tendo em consideração os seus desafios.

Como forma de conclusão deste trabalho, foi realizada a integração de todos os mecanismos referidos no mesmo ambiente e foi realizada a comparação da performance de Clusters em *edge* e em nuvem. Esta integração foi realizada de forma bem-sucedida, no entanto, é de realçar que uma das conclusões retiradas é o facto de a implementação de Clusters em nuvem possuir níveis de latência significativos no *deployment* de microserviços. Como tal, torna-se importante utilizar estratégias de pré-pull de imagens, de forma a diminuir esta latência.

Palavras-Chave

Orquestração de microserviços, Orquestração Inteligente, Orquestração Automatizada, Multi-domínios, Serviços XR, Zero-touch, Computação em Nuvem, Monitorização, Operadores k8s, Comunicação multi-Cluster

Abstract

The growing technological innovation and digitalization of economic sectors and industries boosts the transition from IT infrastructures to the cloud, as its advantages are increasingly becoming known, including for personal use. This transition also raises some security concerns, due to its multi-tenancy characteristic, a characteristic that increases the need for process automation, and consequently the reduction of human interaction. In this sense, there is a need to implement an Intelligent and Automated Orchestration of multi-domain environments, where it is possible to assess the needs of the environment, and act accordingly.

In this context, OneSource integrates the European research project CHARITY for the development of an Intelligent Orchestration environment that allows the implementation of XR services. According to this framework, an internship was proposed for the development and validation of mechanisms that allow the orchestration of several domains in the cloud, in which this report appears, which documents the work developed within the scope of the curricular unit "Dissertation/Internship" of the Master in Computer Security from the University of Coimbra in partnership with OneSource.

This work allowed exploring the implementation of cloud domains, through OpenStack, and the implementation of automation mechanisms, through the execution of Ansible playbooks that facilitate the multi-domain orchestration. These playbooks were integrated into the AWX tool, in order to allow a simplified management of them. Cluster orchestration also has a very significant weight, and as such, the Rancher tool was explored, as well as automation mechanisms in these environments. Additionally, monitoring of the environment was carried out through Prometheus and Grafana, as well as Operators that allow the automated migration of microservices. In addition, this work also allowed the integration of multi-cluster scenarios, taking into account their challenges.

As a conclusion of this work, the integration of all referred mechanisms was carried out in the same environment and a comparison of the performance of clusters in edge and in cloud was carried out. This integration was carried out successfully, however, it should be noted that one of the conclusions drawn is the fact that the implementation of Cloud Clusters has significant levels of latency in the deployment of microservices. As such, it is important to use image pre-pull strategies in order to reduce this latency.

Keywords

Microservices Orchestration, Intelligent Orchestration, Automated Orchestration, Multi-Domain, XR Services, Zero-touch, Cloud Computing, Monitoring, K8s Operators, Multi-Cluster Communication

Índice

Acrónimos

API Application Programming interface. 3, 4, 13, 14, 16, 21, 23, 26, 27, 32, 36, 40, 46, 47, 49, 54, 58, 59, 60, 61, 62, 64, 73, 77, 78, 79, 84, 99, 100, 101, 103, 113

AR Augmented Reality. 3, 9

CDN Content Delivery Network. 23

CI/CD Continuous Integration/Continuous Delivery. 33, 41

CIDR Classless Inter-Domain Routing. 101

CLI Command Line Interface. 21

CNCF Cloud Native Computing Foundation. 2, 76

CNI Container Network Interface. 100, 101, 104

CPU Central Process Unit. 17, 18, 21, 48, 54, 56, 58, 64, 65, 77, 79, 80, 81, 91, 92, 93, 94, 95, 96, 110

CRUD Create, Read, Update and Delete. 1, 8, 21, 27, 34, 37, 45, 54, 62, 63, 64, 73

CSR Certificate Signing Requests. 85

DNS Domain Name System. 24, 53

DoS Denial-of-Service. 10

DSL Domain-Specific Language. 24

GKE Google Kubernetes Engine. 24

E2E End-to-end. 29, 30, 31, 33

EC2 Elastic Compute Cloud. 21, 35, 57

ENI Experiential Networked Intelligence. 32

FIDO Fast IDentity Online. 28

HDFS Hadoop Distributed File System. 16

HTTP Hypertext Transfer Protocol. 18, 20, 78, 91, 104

IaaS Infrastructure as a Service. 4, 19

ICMP Internet Control Message Protocol. 103, 104

IPC Inter-process communications. 82

ID Identificador. 44, 45, 54, 57, 63, 68, 78, 82, 83, 88, 93, 99

IoT Internet of things. 3, 12, 34, 55

IP Protocolo da Internet. 13, 24, 66, 68, 69, 77, 91, 101

IPv4 Protocolo de Internet versão 4. 24, 104

IPv6 Protocolo de Internet versão 6. 24, 101, 103, 104

MANO Management and Orchestration. 33

MR Mixed Reality. 9

NIST National Institute of Standards and Technology. 20

NFV Network Function Virtualisation. 33, 34

PNF Physical Network Functions. 29

PPDR Public Protection and Disaster Relief. 3

QoE Quality of experience. 4

RGPD Regulamento Geral sobre a Proteção de Dados. 27, 50

RKE Rancher Kubernetes Engine. 76, 79, 86, 87, 106, 108, 109

SDN Software Defined Network. 34

SSH Secure Shell. 24, 54, 68, 94, 96

SSL Secure Sockets Layer. 63, 71

TCP Transmission Control Protocol. 91, 101, 103, 104

TI Tecnologias de Informação. 1, 2, 22, 24, 36

UC Universidade de Coimbra. 1

UDP User Datagram Protocol. 101, 103, 104

vCPU Virtual Central Process Unit. 21, 63

VK Virtual-kubelet. 100

VNF Virtual Network Functions. 29

VR Virtual Reality. 2, 9, 10

XR Extended Reality. 3, 4, 9, 10, 16, 32, 33, 35, 38, 49, 50, 51, 52, 53, 54, 80, 88, 90, 93, 101, 109, 112

Lista de Figuras

Lista de Tabelas

Capítulo 1 Introdução

Com a necessidade de desenvolver infraestruturas que automatizem processos de escalonamento e implementação, surge o conceito de Orquestração de *containers*, que visa facilitar a gestão de infraestruturas constituídas por múltiplos ambientes distintos. Com esta abordagem, processos de implementação e escalonamento tornam-se automatizados e com uma menor necessidade de interação humana.

Em paralelo, abordagens *Cloud Native* têm sido cada vez mais estudadas e utilizadas para diversas funcionalidades, desde o uso pessoal para *backup* de ficheiros, como para conexão entre múltiplas organizações em simultâneo. Apesar das vantagens desta abordagem, surgem também preocupações referentes à sua segurança, as quais devem ser tidas em consideração.

Tendo estas abordagens como ponto de vista inicial, torna-se necessário criar relações entre infraestruturas de baixa latência de forma a facilitar as necessidades de aplicações emergentes. O principal objetivo passa por alcançar benefícios de inteligência e orquestração automatizada em nuvem, de forma a permitir a realização de diversos processos com o mínimo esforço humano possível. Esta necessidade de uma Orquestração Inteligente surge, uma vez que a interação humana levanta uma maior possibilidade de erros, o aumento de custos, e o aumento do tempo necessário, não só no treino como nas operações técnicas a realizar.

Um ponto que não pode ser esquecido é o facto de quando se falar em múltiplas organizações, levanta-se a necessidade de orquestração de múltiplos domínios, domínios estes que podem ser em nuvem, *edge* ou recursos de rede. Esta diversidade de domínios acarreta a necessidade de orquestração dos mesmos, de forma a tornar processos de implementação, escalonamento e processos CRUD o mais simplificados possível.

Desta forma, surge o presente relatório, realizado no âmbito da unidade curricular de Dissertação/Estágio do 2º ano do Mestrado de Segurança Informática da Universidade de Coimbra (UC), no ano letivo de 2021/2022. Este relatório visa descrever o estágio realizado pela aluna Cláudia Tavares na empresa OneSource, com foco na Orquestração Inteligente em *Cloud Native*, tendo em consideração as abordagens atuais e as problemáticas e vantagens de cada uma.

1.1 Entidade de Acolhimento - Onesource

A OneSource é uma empresa de TI fundada em 2001, especializada em diversas áreas como comunicação de dados, segurança, *networking* e gestão de sistemas, incluindo consultadoria, auditoria, *design*, desenvolvimento e administração de soluções

especializadas [1]. Desta forma, as suas especializações podem ser destacadas em quatro categorias:

- Consultoria - Oferece serviços de consultoria e auditoria nas áreas de segurança, sistemas de informação, *networking* e infraestruturas de TI.
- Sistemas customizados - Desenvolvimento de aplicações personalizadas e sistemas de informação e integração dos mesmos.
- Gestão de TI - Forte experiência no *design*, implementação e administração de redes corporativas e sistemas de informação, inclusive *outsourcing* e monitorização.
- Investigação - Esta é uma empresa *spin-off* da Universidade de Coimbra. Os serviços de investigação incluem projetos de transferência de tecnologia, serviços de desenvolvimento de produtos, integração de sistemas, como tal participa em diversos programas de investigação, inclusive programas nacionais.

1.2 Contextualização do problema

Cloud Native, segundo CNCF ("*Cloud Native Computing Foundation*"), é uma abordagem que visa facilitar as organizações na criação e execução de aplicações escalonáveis em ambientes modernos e dinâmicos [2]. Através do uso de técnicas como microserviços e malhas de serviços o sistema torna-se pouco acoplado, resiliente, observável e de fácil orquestração. E quando combinado com automação, facilita possíveis modificações necessárias de uma forma mais frequente com a diminuição do esforço humano.

No entanto, a orquestração de clusters de aplicações nestes ambientes não é automatizada e requer ainda a intervenção humana para a configuração de serviços, o que acarreta custos, tempo e possíveis erros, que são proporcionais à escala dos sistemas envolvidos. Como tal, surgem abordagens com principal foco na automação de clusters de aplicações nativas em nuvem, onde será considerada a abordagem Zero-touch, em atividades de provisionamento, *backup*, restauro e escalonamento, de forma a ter em consideração a grande escala de sistemas envolvidos e colmatar a necessidade de envolver o fator humano. Assim, é aumentada a agilidade no desenvolvimento, integração e instalação do sistema.

Este relatório surge no âmbito de alguns projetos europeus, financiados pelo programa "European Union's Horizon 2020 research and innovation programme" que conta com a empresa OneSource como parte integrante do consórcio. Os projetos em questão são: Projeto CHARITY; Projeto 5G-EPICENTRE; Projeto FUDGE-5G.

O projeto CHARITY tem como principal objetivo a criação de uma relação entre infraestruturas de baixa e alta latência de forma a facilitar as necessidades de aplicações emergentes, através do alcance dos benefícios da inteligência e orquestração automatizada em nuvem, em *edge* e recursos de rede [3].

Este projeto visa o atendimento a aplicações emergentes e como tal os casos de uso passam por três categorias: Aplicações holográficas em tempo real (como exemplo de um concerto holográfico ou uma reunião holográfica), Treino virtual imersivo (como por exemplo Treino VR Médico) e Aplicação interativa de realidade mista (por exemplo jogos

colaborativos), desta forma, deve ser tido em consideração os requisitos deste tipo de aplicações.

O projeto 5G-EPICENTRE tem como principal objetivo a construção de uma plataforma 5G ponta a ponta, adaptada às necessidades do mercado de segurança pública e resposta a emergências [4]. Desta forma, este projeto visa abordar vários cenários operacionais do PPDR, desde dispositivos IoT que permitem validar o estado de saúde de indivíduos, dispositivos AR vestíveis, cuidados cirúrgicos de emergência assistidos por AR, entre outros.

O projeto FUDGE-5G visa conceber, avaliar e demonstrar uma arquitetura 5G, soluções e sistemas nativos em nuvem conceitualmente novos, unificados e protegidos baseados em serviços para redes privadas, de forma a permitir interoperabilidade e customização para setores verticais de indústria [5]. Desta forma os casos de uso que serão usados como forma de validação, passa pela implementação de rede 5G para PPDR, escritório 5G virtual para hospitais e ambientes de Indústria 4.0, envolvendo todos os requisitos e desafios de cada caso de uso.

1.3 Definição do problema

Este trabalho visa responder às necessidades de Orquestração de múltiplos domínios, de uma forma unificada e autónoma, através de mecanismos de Orquestração Inteligente que permitam a recolha de métricas e a tomada de decisões conforme as mesmas. Desta forma, a solução proposta irá permitir a integração de múltiplos domínios de uma forma automatizada, assim como a orquestração de microserviços de forma abstrata para os diversos domínios.

Assim, a Orquestração dos diversos domínios deve ser vista como uma Orquestração inteligente e devem ser implementados os mecanismos e APIs necessárias para a interação com os diversos domínios ser realizada de forma autónoma, independentemente do tipo de domínio e da sua localização geográfica. Desta forma torna-se possível diminuir a necessidade de interação humana, de forma a automatizar processos de implementação, escalonamento e provisionamento, e que estes mesmos processos sejam possíveis em qualquer domínio, independentemente do tipo de domínio e da sua localização.

Quando abordados ambientes em nuvem é importante ter em consideração as suas preocupações quanto à segurança, uma vez que estamos a falar de locais acessíveis por diversos utilizadores, que como tal levantam problemáticas de *multi-tenancy*. Este fator torna-se ainda mais preocupante quando se aborda ambientes multi-domínios, como tal, esta preocupação deverá ser uma constante no trabalho a realizar.

Além disto, não podem ser esquecidos os requisitos das aplicações XR, uma vez que a baixa latência é um ponto fulcral no seu uso, e tendo em conta a contextualização do problema e o seu enquadramento no âmbito dos projetos mencionados, as implementações realizadas devem ter este aspeto em consideração.

1.4 Objetivos do Trabalho

O presente estágio pretende atingir os seguintes objetivos no decorrer do trabalho:

- Levantamento do estado de arte relativamente à orquestração inteligente de clusters.
- Definição do caso de uso para a demonstração da orquestração inteligente (como por exemplo segurança e QoE).
- Especificação e implementação dos componentes necessários para a implementação de um orquestrador inteligente.
- Integração dos componentes no projeto em que se enquadra e validação do seu funcionamento e performance.

1.5 Plano de Trabalhos

O plano de escalonamento do trabalho é dividido em duas secções principais, a primeira referente ao 1ºSemestre (período entre setembro de 2021 e janeiro de 2022) e a segunda referente ao 2ºSemestre (período entre fevereiro e julho de 2022).

1.5.1. Semestre 1

O plano de escalonamento dos trabalhos pretendidos é dividido nos seguintes tópicos, referentes apenas ao 1ºsemestre:

- T1.1 – Análise do estado da arte e dos requisitos do sistema.
- T1.2 – Identificação dos casos de uso a aplicar o orquestrador inteligente.
- T1.3 – Preparação de protótipo para um ambiente ilustrativo da aplicação.
- T1.4 – Desenho inicial da solução a desenvolver.
- T1.5 – Preparação do relatório intermédio.

De forma mais detalhada, pode ser visualizada a Tabela 1.1 e a Tabela 1.2, que representam o diagrama referente ao planeamento de trabalho entre setembro de 2021 e 17 de janeiro de 2022, data da defesa intermédia.

O período representado na Tabela 1.1 contemplou o estudo da contextualização teórica (Serviços XR e os seus requisitos, arquiteturas microserviços e *service mesh*, bem como uma análise do Kubernetes e dos seus componentes). Além disto foi feita uma análise do estado da arte relativamente a ferramentas que permitam solucionar a problemática de orquestração multi-domínios e técnicas existentes na literatura. Ferramentas e projetos também foram abordados, de forma a analisar possíveis abordagens para Provedor IaaS, para Orquestração multi-domínio e para Orquestração de cluster, e de forma a perceber a vantagem das mesmas e como seriam implementadas.

Durante o período ilustrado na Tabela 1.2 foram implementadas e testadas as ferramentas de Provedor IaaS e as ferramentas de orquestração multi-domínio, de forma, a prever um ambiente com múltiplos domínios em nuvem e a sua orquestração de forma unificada. Além disto, foram também analisadas possíveis APIs para a interação entre o Orquestrador Inteligente e os múltiplos domínios. Findada esta tarefa, foi desenvolvido um desenho inicial da solução a desenvolver, de acordo com os requisitos da mesma.

Tabela 1.1- Diagrama de Gantt relativo à primeira parte do 1ºsemestre

	Semana 1 01/09-11/09	Semana 2 11/09-18/09	Semana 3 18/09-25/09	Semana 4 25/09-02/10	Semana 5 02/10-09/10	Semana 6 09/10-16/10	Semana 7 17/10-23/10	Semana 8 23/10-30/10	Semana 9 30/10-06/11
Contextualização Teórica									
Análise de Artigos e Projetos									
Análise de Ferramentas de Provedor IaaS e de Ferramentas de Orquestração Multi-domínio									
Análise de Ferramentas de gestão de Cluster									
Implementação e Teste de Ferramentas de Provedor IaaS e de Ferramenats de Orquestração Multi-domínio									
Análise de APIs para integração entre o Orquestrador Inteligente e os diversos domínios									
Desenho inicial da solução									
Escrita do relatório relativamente à defesa intermédia									
Preparação da Apresentação									

Tabela 1.2- Diagrama de Gantt relativo à segunda parte do 1ºsemestre

	Semana 10 06/11-13/11	Semana 11 13/11-20/11	Semana 12 20/11-27/11	Semana 13 27/11-04/12	Semana 14 04/12-11/12	Semana 15 11/12-18/12	Semana 16 18/12-25/12	Semana 17 25/12-01/01	Semana 18 01/01-08/01	Semana 19 08/01-17/01
Contextualização Teórica										
Análise de Artigos e Projetos										
Análise de Ferramentas de Provedor IaaS e de Ferramentas de Orquestração Multi-domínio										
Análise de Ferramentas de gestão de Cluster										
Implementação e Teste de Ferramentas de Provedor IaaS e de Ferramenats de Orquestração Multi-domínio										
Análise de APIs para integração entre o Orquestrador Inteligente e os diversos domínios										
Desenho inicial da solução										
Escrita do relatório relativamente à defesa intermédia										
Preparação da Apresentação										

O trabalho descrito pode ser visualizado com mais detalhe no Capítulo 4, onde é feita a descrição do mesmo. Além disto, é importante referir que o trabalho foi refletido na escrita de documentação para o projeto CHARITY.

De acordo com o planeamento inicial, representado com cor amarela, o trabalho decorreu conforme o espectável, apenas com a exceção da etapa “Análise de Artigos e Projetos”, a qual teve uma duração prolongada, tendo sido efetuado em simultâneo com a posterior análise de ferramentas. Além disto, a etapa de “Desenho Inicial da Solução” foi antecipada, e como tal, não foi efetuada apenas em duas semanas, mas sim, à medida que foram efetuadas as implementações e testes das diversas ferramentas. Estas alterações necessárias no decorrer do trabalho estão representadas a uma cor distinta, no entanto é de realçar que estas alterações não foram significativas para o trabalho planeado.

1.5.2. Semestre 2

O plano de escalonamento dos trabalhos pretendidos é dividido nos seguintes tópicos, referentes apenas ao 2ºsemestre:

- T2.1- Especificação e testes para o orquestrador inteligente a desenvolver.
- T2.2- Preparação do ambiente do sistema.
- T2.2- Implementação das restantes funcionalidades de acordo com o desenho e as especificações técnicas.
- T2.3- Integração com outros serviços e validação do sistema nos cenários do projeto em que se enquadra.
- T2.4- Elaboração de documentação, incluindo o relatório de estágio, documentos técnicos e manuais de utilização.

De forma mais detalhada, pode ser visualizada a Tabela 1.3 e Tabela 1.4, que representam o diagrama onde inclui o progresso semanal de cada atividade, referente ao planeamento de trabalho entre fevereiro de 2022 e junho de 2022, data da defesa final.

Ao longo do período representado na Tabela 1.3 foi realizada a implementação de múltiplos domínios que são geridos com base em ferramentas de orquestração multi-domínios, onde foi testada a modificação de domínios, nomeadamente a criação e remoção dos mesmos. Além disto, foi implementada a ferramenta que permite a orquestração de cluster de *containers*, a qual, numa primeira fase foi testada sem a integração de domínios, e numa segunda fase com a integração do provedor implementado anteriormente. Durante este período foram também implementados mecanismos que permitem realizar atividades de forma autónoma e sem a necessidade de acesso direto à ferramenta de Orquestração multi-domínio. Desta forma, a orquestração do ambiente é feita de forma unificada e simplificada.

Durante o período ilustrado na Tabela 1.4, foi realizada a integração dos domínios implementados com a ferramenta de orquestração de clusters, onde foi realizada a sua monitorização. Foram implementados mecanismos de monitorização e segurança que permitem avaliar o estado do ambiente, quer ao nível dos domínios, como ao nível dos clusters. Após isto, foi realizada a integração dos diversos mecanismos no ambiente implementado de forma a validar o sistema. No decorrer de todo o semestre foi realizada a escrita do relatório, bem como a documentação dos resultados obtidos, conforme as etapas.

Tabela 1.3- Diagrama de Gantt relativo à primeira parte do 2ºsemestre

	Semana 1 29/01-05/02	Semana 2 05/02-12/02	Semana 3 12/02-19/02	Semana 4 19/02-26/02	Semana 5 26/02-05/03	Semana 6 05/03-12/03	Semana 7 12/03-19/03	Semana 8 19/03-26/03	Semana 9 26/03-02/04	Semana 10 02/04-09/04	Semana 11 09/04-16/04
Implementação de domínios em nuvem											
Implementação de ferramenta de orquestração de Clusters											
Realização de ações de monitorização dos Clusters implementados											
Implementação de containers Kubernetes através dos provedores de nuvem											
Implementação de mecanismos que permitam atividades CRUD em domínios especificos											
Monitorização dos Clusters entre domínios											
Implementação de mecanismos de segurança											
Integração dos diversos mecanismos no Ambiente implementado											
Documentação dos Resultados											
Escrita do relatório de estágio											
Preparação da Apresentação											

Tabela 1.4- Diagrama de Gantt relativo à segunda parte do 2ºsemestre

	Semana 12 16/04-23/04	Semana 13 23/04-30/04	Semana 14 30/04-07/05	Semana 15 07/05-14/05	Semana 16 14/05-21/05	Semana 17 21/05-28/05	Semana 18 28/05-04/06	Semana 19 04/06-11/06	Semana 20 11/06-18/06	Semana 21 18/06-25/06	Semana 22 26/06-04/07
Implementação de domínios em nuvem											
Implementação de ferramenta de orquestração de Clusters											
Realização de ações de monitorização dos Clusters implementados											
Implementação de containers Kubernetes através dos provedores de nuvem											
Implementação de mecanismos que permitam atividades CRUD em domínios especificos											
Monitorização dos Clusters entre domínios											
Implementação de mecanismos de segurança											
Integração dos diversos mecanismos no Ambiente implementado											
Documentação dos Resultados											
Escrita do relatório de estágio											
Preparação da Apresentação											

De acordo com o planeamento inicial deste semestre, representado com cor amarela, o trabalho decorreu conforme o espectável, apenas com a exceção da etapa "Implementação de containers Kubernetes através dos provedores de nuvem", a qual teve uma duração prolongada. Além disto, a etapa de "Implementação de mecanismos que permitam atividades CRUD em domínios específicos" foi antecipada. Estas alterações estão representadas a uma cor distinta, no entanto é de realçar que estas alterações não foram significativas para o trabalho planeado.

1.6 Estrutura do relatório

O presente relatório está dividido em capítulos, os quais podem ser descritos da seguinte forma:

- Capítulo 1 – Introdução ao estágio curricular, com a apresentação da entidade de acolhimento, bem como a contextualização do problema a resolver e os objetivos do estágio.
- Capítulo 2 – Contextualização dos temas fundamentais para o desenvolvimento do projeto, como arquitetura de microserviços, *containers*, orquestração e as vantagens de *service mesh*. Será também detalhado as diversas características do uso de um ambiente em Kubernetes e os desafios criados nestes ambientes, assim como será abordado o conceito de Engenharia do Caos. Além disto, neste capítulo será feita a descrição e análise de diversas abordagens implementadas por outros autores com vista a responder às problemáticas de Orquestração entre domínios, e à Orquestração Inteligente. Neste capítulo são referidos os componentes que irão corresponder à arquitetura a desenvolver, e são mencionadas as abordagens iniciais, numa fase antes de testes.
- Capítulo 3 – Apresentação da metodologia utilizada, de forma a abordar uma proposta inicial da solução, onde é realizado o seu enquadramento, bem como são analisados os requisitos do ambiente, tanto no ponto de vista de requisitos não funcionais como requisitos funcionais. Também a descrição da arquitetura a é apresentada.
- Capítulo 4 - Descrição do trabalho exploratório efetuado e das provas de conceito que foram realizadas, de forma a perceber quais seriam as ferramentas a utilizar e de que forma poderiam ser integradas.
- Capítulo 5 - Apresentação do trabalho realizado com base na problemática de Orquestração multi-domínios, onde é apresentada a abordagem utilizada e os mecanismos de automação que foram implementados para a orquestração dos mesmos, bem como testes funcionais à solução apresentada no capítulo.
- Capítulo 6 - Descrição do trabalho realizado no âmbito da Orquestração de clusters, com a integração de ambientes multi-domínios, onde é realizada a implementação de clusters e a sua monitorização. Além disto, são apresentados os mecanismos que permitem orquestrar este ambiente de uma forma eficiente.
- Capítulo 7 - Apresentação das conclusões retiradas no decorrer do trabalho e do trabalho futuro a realizar.

Capítulo 2 Contextualização Teórica e Estado da Arte

Orquestração de *containers* é um tema que tem ganho uma elevada escala. Este conceito aparece com inúmeras vantagens, ao nível de gestão de infraestruturas constituídas por um ou por diversos ambientes, que podem ser semelhantes ou bastante diferentes. A utilização de Orquestradores permite que aplicações baseadas em microserviços sejam implementadas de forma automatizada, geridas de forma unificada e que sejam escaladas conforme as necessidades.

Outra temática abordada neste capítulo é a orquestração de múltiplos domínios de forma unificada, uma vez que este será um requisito para o trabalho a ser desenvolvido, desta forma, são analisadas possíveis abordagens para a criação de ambientes em nuvem e para a sua orquestração, onde são abordados também os problemas de segurança associados a estes ambientes *multi-tenancy*.

O presente capítulo constitui a base de conhecimento inicial ao desenvolvimento do projeto, onde são abordados os conceitos que serão utilizados para a implementação da solução de orquestração inteligente e projetos que são utilizados como referência. A necessidade desta base de conhecimento, deve-se à constante evolução de tecnologias de orquestração, onde se torna relevante estudar conceitos como microserviços, *containers*, *service mesh*, orquestração, processos e ferramentas que possibilitem a Orquestração Inteligente, bem como a Orquestração Automatizada. Neste âmbito, é também discutida a abordagem Zero-touch, uma vez que permite uma maior tomada de decisões inteligentes e que será um conceito fulcral em todo o desenvolvimento do trabalho. Estes conceitos tornam-se relevantes para estudo, de forma a perceber o funcionamento dos mesmos, as suas características e de que forma podem ser implementados e melhorados na temática da Orquestração Inteligente, ao longo do trabalho a desenvolver.

Além disto, neste capítulo é também abordado o tema dos serviços XR, uma vez que a arquitetura a desenvolver no âmbito do projeto CHARITY deve ter suporte à implementação destes mesmos serviços. Os casos de uso do projeto visam a implementação de serviços XR, e como tal, deve ser tida em conta as suas características e requisitos.

2.1 Serviços XR

Extended Reality (XR) é um conceito que abrange todos os ambientes reais e virtuais alterados por computadores, desde o conceito de Realidade Aumentada (AR), Realidade Virtual (VR) e Realidade Mista (MR).

Realidade aumentada é a sobreposição de conteúdo gerado por computador no mundo real, sem o reconhecimento dos objetos físicos. Isto significa que o utilizador visualiza o mundo real com informações percetivas geradas virtualmente, criando uma camada de sobreposição ao mundo real. Um dos casos de uso deste tipo de Realidade é em jogos de AR, onde são utilizadas imagens dos espaços dos utilizadores.

Realidade virtual engloba todas as experiências virtualmente imersivas, isto é, todas as experiências que substituam completamente a perceção do utilizador sobre o mundo real para um mundo virtual, provocando ao utilizador o sentimento de estar num local diferente. Neste sentido, surgem exemplos de uso como aulas de treino baseado em plataformas VR, que permite revelar vários tipos diferentes de dados, como o ambiente ao redor.

Realidade mista consiste na interação entre o mundo real e o virtual, removendo os limites entre a interação dos dois por meio da oclusão, ou seja, os objetos virtuais podem ser visivelmente obscurecidos por objetos reais.

O progresso das aplicações XR tem aumentado significativamente devido aos avanços do *hardware* disponíveis, à atualização dos sensores utilizados e aos avanços das tecnologias gráficas de computação. Apesar disto, estas aplicações trazem alguns desafios relativos à segurança e privacidade uma vez que é efetuada a recolha e processamento de grandes quantidades de dados confidenciais (por exemplo, a localização do utilizador, biometria e informações sobre espaços privados). Desta forma, há requisitos que devem ser atendidos como a necessidade de identificação dos utilizadores, através de identificadores únicos e da gestão de identidades dos mesmos. Isto irá permitir a integridade e não repudiação de forma a garantir a não modificação dos dados e não possibilitar a negação de uma ação por parte dos utilizadores. Além disto, a identidade dos utilizadores permite garantir a confidencialidade e *unlinkability*, de forma a proteger os recursos confidenciais e não ser capaz de identificar utilizadores conforme os dados que circulem. Também a autorização, autenticação e o controlo de acessos torna-se possível através da gestão de identidades, de forma a identificar os utilizadores e analisar as suas permissões, bem como deverá haver a disponibilidade do serviço sempre que o mesmo seja requisitado pelo utilizador.

Neste sentido, podem ser distinguidas as seguintes categorias de principal atenção, nomeadamente a proteção de *inputs* e *outputs*, a proteção do acesso de dados, proteção de interações e de dispositivos [6], conforme representado na Figura 2.1. Isto porque esta tecnologia é vulnerável a ameaças conhecidas, desde *malwares*, *ransomwares*, *Denial-of-Service (*DoS) e ataques *Man-in-the-Middle*.

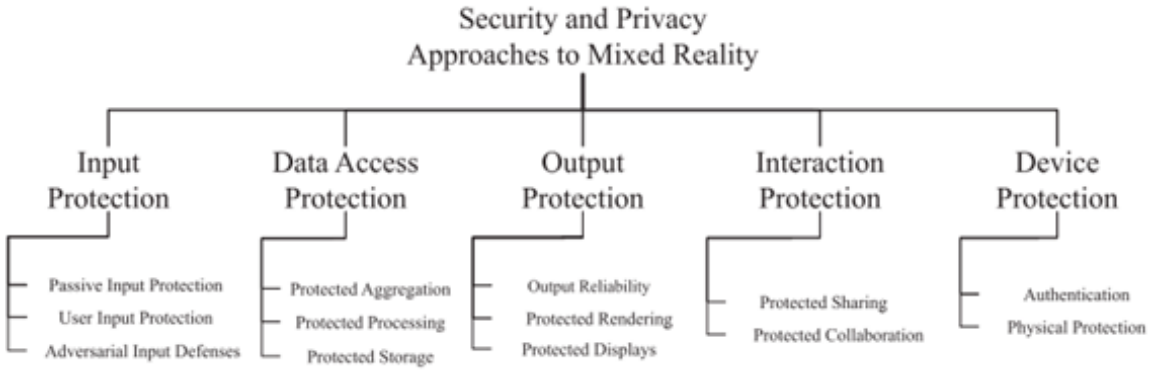

Figura 2.1- Mapeamento entre abordagens de Segurança e Privacidade de aplicações XR [6]

Além disto, não pode ser esquecido que as aplicações XR requerem baixa latência, o que normalmente é afetada pela introdução de mecanismos de segurança no sistema, como tal, as ferramentas de segurança usadas devem ser tidas em consideração e analisadas de forma a não prejudicar a latência do sistema. Isto significa que não devem ser implementados todos os mecanismos possíveis, mas sim os mecanismos necessários para os aspetos que se deseja proteger.

2.2 Orquestração de containers

Automação é o processo que efetua a distribuição de *containers* de forma automática entre os clusters, aumentando assim a disponibilidade do sistema e a performance das aplicações. Em contrapartida, a Orquestração, foca-se na automação de processos com várias etapas em diferentes sistemas, acabando por tornar possível a implementação da mesma aplicação em diversos ambientes diferentes sem ser necessário voltar a projetar a mesma.

A utilização de orquestração de *containers* traz diversas vantagens, quer ao nível da eficiência do sistema como de segurança:

- Alocação de recursos – Permite a definição de capacidades dos recursos;
- Monitorização de Integridade – Permite a criação de serviços e aplicações que podem abranger diversos *containers*, programar o seu uso, escalar os mesmos e gerir a sua integridade no decorrer do tempo;
- Balanceamento de carga – Permite a portabilidade de cargas de trabalho e a migração de aplicações de forma rápida e automatizada;
- Escala e/ou remoção de *containers* – Permite o escalonamento e remoção de *containers* de forma automatizada.

No entanto, a maior vantagem do seu uso prevalece no facto de oferecerem uma visualização leve e *sandboxing* (mecanismo que permite a separação de programas em execução) através de espaços de rede definidos e grupos de controle que permitem esta mesma separação. Isto permite mitigar falhas ou vulnerabilidades que possam ter tendência para se propagar. Apesar disto, é de realçar que os *containers* apenas podem ser corridos em sistemas operativos Linux, o que acaba por ser uma grande limitação ao seu uso.

Os Orquestradores de *containers* podem ser desenvolvidos com base na arquitetura de microserviços, arquitetura onde cada microserviço é criado individualmente, com flexibilidade na escolha de linguagem de programação ou de outras ferramentas, como tal são projetados de forma independente e com a capacidade de se comunicar entre si. Além disto, a compilação dos *containers* pode ser efetuada através de *build-chains* (sequência de compilações interconectadas por *snapshots* que permite a revisão de sincronizações) e pela produção de binários executáveis que permitam garantir uma maior portabilidade.

A arquitetura de microserviços, tem desvantagens na sua utilização, como o facto de requerer ser bem planeada e documentada para a sua utilização (até porque cada parte da aplicação pode ser desenvolvida com linguagens de programação distintas). Isto acarreta problemas nomeadamente na sua segurança, uma vez que a sua natureza é bastante ampla, logo a superfície de ataque também se torna mais ampla. Neste sentido, torna-se bastante importante a sua correta definição e documentação. Apesar disto, esta arquitetura possui diversas vantagens comparando com a arquitetura monolítica (onde as aplicações têm todas a sua base de código no mesmo lugar e todas as suas funcionalidades definidas no mesmo local, não havendo separação de lógica em múltiplos locais) [7]:

- Em caso de necessidade de escalonamento, este processo pode ser feito num serviço individual, ao invés de ter de escalar o ambiente completo;
- Sistemas totalmente independentes, embora se complementem entre si;
- Facilidade de publicação da aplicação e da execução de testes individuais a cada parte distinta da aplicação;

- Ausência de ponto único de falha, que no caso da arquitetura monolítica é um dos maiores problemas, uma vez que em caso de falha de alguma das partes, todo o sistema poderia ficar inacessível.

Os microserviços são processos pequenos, independentes e altamente desacoplados, o que permite implementar aplicações complexas com maior facilidade [8]. Estas características facilitam a utilização destas abordagens para implementação de *software* necessário em clusters locais de baixo custo e a possível integração com dispositivos IoT (dispositivos de baixos recursos) [9].

De forma a implementar microserviços, podem ser utilizados *containers* (muito habitualmente comparados com a Virtualização, apesar de serem tecnologias complementares). Com o seu uso não se torna necessário ter um *Hypervisor*, uma vez que os diversos *containers* vão partilhar o Sistema Operativo da máquina, e posteriormente apenas são isolados os processos da aplicação do restante. Desta forma, o uso de Virtualização é uma solução mais pesada, na medida em que as suas capacidades e recursos são limitados, enquanto, no caso de *containers*, como são executados de forma nativa no Sistema, podem se salientar as seguintes vantagens:

- Implementação rápida, uma vez que os *containers* incluem os requisitos mínimos de execução da aplicação, como tal reduzem o seu tamanho;
- Portabilidade, visto que a aplicação e todas as suas dependências podem ser incorporadas dentro de um único *container*;
- Manutenção simplificada, devido à não dependência das aplicações;
- Imagens otimizadas, devido ao seu reduzido tamanho, que acaba por facilitar a entrega rápida e a redução do tempo de implementação.

Uma vez que cada *container* corresponde a uma pequena parcela responsável por uma aplicação, torna-se necessário pensar em formas de melhorar o seu desempenho, de forma a automatizar os seus processos e a sua orquestração. Neste contexto, é importante realçar que existe diferenças quando se debate a orquestração de serviços *stateless* e serviços *stateful*.

Em serviços *stateless* o servidor apenas processa solicitações com base nas informações retransmitidas em cada solicitação, não dependendo de registos anteriores, como tal não é necessário manter o estado das mesmas. Da mesma forma, isto permite que as solicitações possam ser processadas por múltiplos servidores distintos, uma vez que cada um irá ter as informações necessárias para processar a solicitação, aumentando a resiliência e a elasticidade dos serviços.

Por outro lado, os serviços *stateful* processa as solicitações com base nas informações retransmitidas em cada solicitação, mas também com base nas solicitações anteriores, como tal, é necessário o armazenamento das mesmas. Neste caso não é possível que diversos servidores processem as solicitações, a não ser que as informações de estado sejam partilhadas entre os mesmos. Este tipo de serviços acarreta desafios relativamente ao isolamento de recursos e ao armazenamento de serviços, uma vez que é necessário armazenar as informações e em simultâneo garantir a segurança das mesmas [10].

Assim, enquanto os serviços *stateless* podem ser executados de maneira resiliente, os serviços *stateful* tornam este processo mais complexo, tal como debatido no artigo [11], onde os autores investigam o suporte atual a este tipo de serviços de forma a simplificar a sua orquestração.

2.2.1. Ambiente Kubernetes

Na categoria de Orquestração destaca-se o Kubernetes tal como referido no artigo [12] onde foram comparados diversos orquestradores. Esta é uma ferramenta *open-source* de orquestração de *containers* que será utilizada no decorrer do trabalho, com o objetivo de hospedar serviços na nuvem e em *edge*. De forma a entender o funcionamento desta ferramenta é importante perceber quais são os seus componentes e de que forma se interligam, conforme se pode visualizar na Figura 2.2.

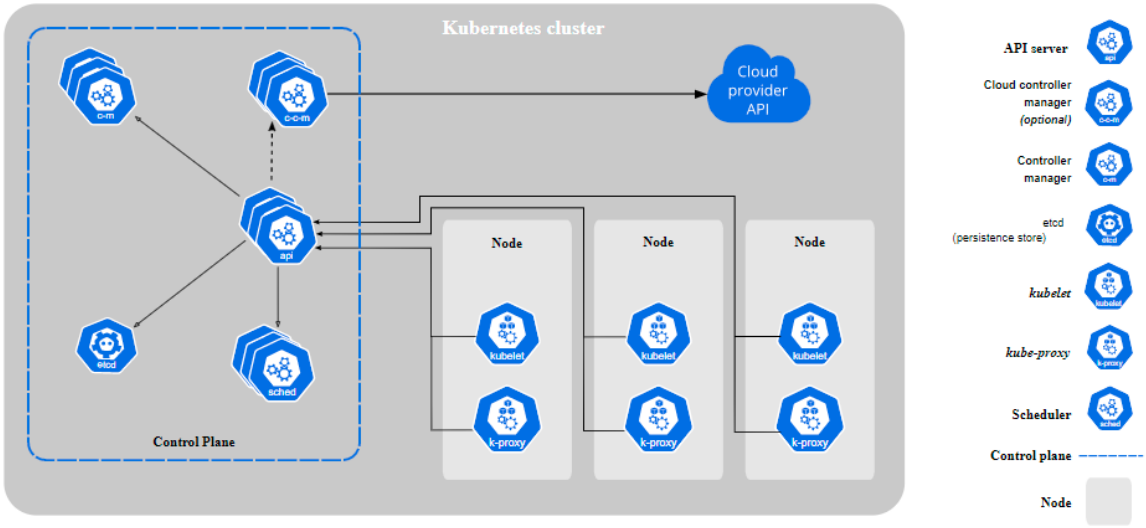

Figura 2.2- Representação dos componentes de um cluster Kubernetes [13]

O componente mais abrangente é o cluster, conjunto de máquinas (designadas por nós) que executam aplicações em *containers*. Dentro de cada nó pode haver mais do que um *pod*, que permite o encapsulamento de *containers* e o acesso aos mesmos (que não podem ser acedidos de forma direta). Desta forma, todos os *containers* existentes num *pod* possuem o mesmo endereço IP, assim como o mesmo nome de *host*.

Cada nó possui um agente *kubelet*, responsável por garantir que os *containers* foram iniciados e que estão em execução num *pod*. Além disto, em cada nó também há um *proxy* de rede (*kube-proxy*) que mantém as regras de rede, permitindo a comunicação de rede com os *pods* a partir de sessões de rede, quer dentro ou fora do cluster. E embora não seja possível visualizar pela Figura 2.2, cada nó tem um *container runtime*, o *software* responsável pela execução dos *containers*.

Por outro lado, o componente Plano de Controle faz a gestão dos nós e dos *pods* do cluster, como tal, é responsável pela tomada de decisões globais, deteção e resposta a eventos. Este componente pode ser executado em apenas uma máquina, no entanto, se a tolerância a falhas ao sistema e o aumento da sua disponibilidade for um requisito, é uma boa prática executar este componente em diversas máquinas. Este componente é constituído por diversos subcomponentes, que geralmente são inicializados sem a execução de *containers*:

- Servidor API – Componente responsável por expor a API Kubernetes, sendo o *frontend* do plano de controle para a utilização do cliente, permitindo a consulta e manipulação do estado dos diversos objetos do cluster. Em caso de múltiplos acessos, podem ser criadas novas instâncias deste componente, de forma a aumentar a disponibilidade do mesmo.

- Cloud Controller Manager – Este componente é opcional, e como tal apenas é utilizado em caso de haver conectividade com nuvem. Desta forma incorpora a lógica de controle específica da nuvem, vinculando o cluster à API do provedor. Este componente pode ser escalonado horizontalmente (isto é, pode ser criada uma nova instância) de forma a melhorar o desempenho e a tolerância a falhas.
- Kube Controller Manager – Componente responsável pela execução de processos de controlador, que analisa diversos eventos (como o caso de algum nó ficar indisponível ou a criação de *pods* que sejam necessários para a realização de tarefas únicas).
- Etcd – Serviço de armazenamento de metadados permanente e altamente disponível, que contém todo o armazenamento de apoio a dados do cluster. Este componente é bastante importante e como tal deve haver um plano de *backup* para estes dados, em caso de qualquer eventual falha. Este serviço faz uso de RAFT, como algoritmo de consenso, de forma a permitir que quando uma ação é solicitada, todos os membros do *etcd* possam permitir esta ação, e em caso afirmativo, a ação é gravada em *log*. Desta forma, há um aumento do nível de tolerância a falhas, uma vez que a (N/2)+1 dos membros do *etcd* terá se permitir a ação a realizar, sendo N o número total dos membros [14]. Assim, é também importante manter este componente atualizado e todos os seus membros com a mesma versão, não só porque isto causa impacto na capacidade de *backup*, como também evita a disparidade na tomada de decisões.
- Kubelet e kube-proxy – Componentes existentes também em cada nó do cluster, com responsabilidades de verificação da correta inicialização e execução do *container* e responsável pela comunicação entre os diversos *pods*.
- Scheduler – O principal objetivo deste componente é a análise dos *pods* recém-criados, que ainda não têm nó atribuído e efetuar essa mesma atribuição para posterior execução. Essencialmente esta atribuição é dependente dos requisitos de recursos, uma vez que cada *pod* possui diferentes requisitos e é necessário analisar quais são os nós válidos para cada *pod* (em caso de não haver compatibilidade de requisitos com nenhum nó, o *pod* vai permanecer em espera até possível atribuição). Após a análise dos nós viáveis, estes são pontuados, de forma a perceber qual o mais vantajoso para o *pod* em questão (em caso de haver mais do que um nó com pontuações mais altas é feita uma escolha aleatória entre os mesmos) e é enviada a notificação de decisão ao Servidor API.

O Kubernetes é um sistema de orquestração, que visa a automação de processos como a implementação, dimensionamento e a sua constante gestão, o que traz bastantes benefícios para os seus utilizadores, no entanto quando aumenta o número de *containers*, maior se torna a sua necessidade de automação.

De forma a reduzir a necessidade de intervenção humana nos processos de implementação e escalonamento existe os Operadores, uma extensão de *software* que usa recursos personalizados para gerir a aplicação e os seus componentes, isto é, através de recursos criados por pessoas especializadas (por exemplo, se estivermos a falar de uma Base de Dados MySQL, este recurso é desenvolvido por experientes nesta Base de Dados), efetua o mesmo mecanismo de *Control Loop* do Kubernetes, isto significa que observa os objetos no cluster, verifica as suas alterações e toma as ações necessárias para cada situação.

2.2.2. Service Mesh

A Orquestração de *containers* requer a constante necessidade de comunicação entre cada elemento de uma aplicação ou sistema. Neste sentido, com esta necessidade, num ambiente de microserviços que tende a tornar-se cada vez mais complexo, acaba por ser importante o uso de *service mesh*, principalmente quando se fala de aplicações nativas em nuvem, que englobam um grande número de serviços, com possibilidade de expansão e uma possível elevada distribuição geográfica.

Esta malha é uma camada de infraestrutura de comunicação existente entre múltiplos serviços que compõem uma aplicação/sistema. Isto facilita e torna mais eficiente a partilha de dados entre componentes, inclusive permitindo a identificação de falhas na comunicação. Desta forma, é possível ser feito o encaminho de pedidos efetuados, em caso de um dos serviços não se encontrar disponível (evitando assim a falha de disponibilidade da aplicação).

Service mesh é composta por diversos *proxies* de rede (Envoy), responsáveis pelo encaminhamento das comunicações. Estes *proxies*, designados por "*sidecars*", são executados em paralelo a cada serviço, e não dentro do mesmo, o que permite que se afirme que a *service mesh* é constituída por diversos *proxies* que se interligam entre si, como é comprovado pela Figura 2.3, onde podemos visualizar a ligação entre todos estes elementos. Em simultâneo, os *proxies* podem conter políticas específicas de encaminhamento, que são definidas no Plano de Controle, de forma a abstrair estas regras dos microserviços [15].

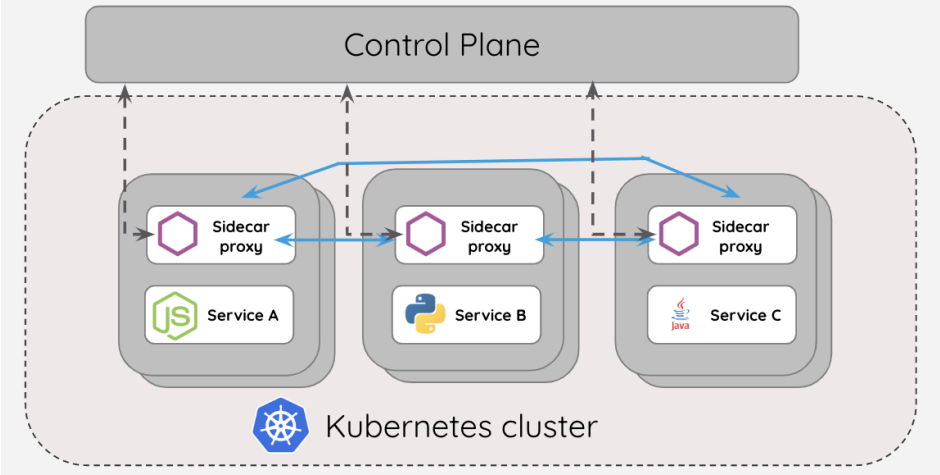

Figura 2.3- Representação de service mesh num cluster Kubernetes [16]

Além desta malha efetuar a identificação de falhas em microserviços, também utiliza métricas de desempenho (como por exemplo o registo do tempo que demora um serviço a voltar a estar disponível em caso de falha) que ajudam na eficiência e confiabilidade das comunicações efetuadas, uma vez que podem ser escritas regras de comunicação em função dos valores das métricas. Por exemplo, pode ser calculado o tempo de espera médio para efetuar outra tentativa de comunicação a um serviço indisponível, o que liberta o sistema de tentativas mal sucedidas.

No entanto, há algumas características que esta malha deve prever, como o suporte a alta performance (tanto o componente *proxy*, como o plano de controle devem ter este requisito em consideração), a adaptabilidade (deve haver facilmente a adaptação a plataformas de orquestração nativas em nuvem, de forma a ser configurável e facilmente extensível) e deve

prever alta disponibilidade (de forma a reduzir o risco de tomadas de decisões imprudentes e a isolar o impacto de casos de indisponibilidade) [17].

Atualmente o número de plataformas de *service mesh* disponíveis para uso, tem aumentado uma vez que aborda os principais desafios da concretização da arquitetura de microserviços, permitindo realizar as comunicações de forma mais automatizada. Uma das principais plataformas é o Istio, plataforma referida como solução para a problemática de instrumentação de *containers* e de avaliação de desempenho do *endpoint* da API no artigo [18]. Desta forma, Istio permite controlar a comunicação entre microserviços, através de balanceamento de carga, autenticação das comunicações, aplicação de políticas e da monitorização com o objetivo de fornecer uma comunicação segura e eficiente entre serviços. Em conjunto com o Istio, pode também ser integrado o Cilium, um projeto criado com o objetivo de fornecer rede, segurança e observabilidade do ambiente. Assim, o Clilium permite a definição de políticas de rede para tráfego protegido por mTLS sem a alteração do código das aplicações e das configurações dos serviços [19].

Além disto, podem ser também utilizadas ferramentas que permitam facilitar e suportar a comunicação entre os componentes, bem como a recolha e agregação de métricas de diferentes ativos e ferramentas de monitorização. Um exemplo de uma destas ferramentas é o Apache Kafka, que permite que os componentes possam trocar mensagens de forma assíncrona com um *bus* comum, onde cada componente pode publicar e consumir dados sempre que seja necessário [20]. Isto também permite a exposição de métricas observadas, que possam ser consumidas por ferramentas de monitorização. No mesmo sentido, outra alternativa passa pelo uso de Apache Flume, que também permite a recolha e agregação de dados em HDFS (“Hadoop Distributed File System”) [21], um sistema de arquivos distribuído que tem como vantagens a sua tolerância a falhas e a sua elevada taxa de transferência de dados.

2.2.3. Avaliação de Segurança do Ambiente

De forma a avaliar e testar um sistema de *software* que se encontra em fase de produção, pode ser utilizada uma abordagem de Engenharia do Caos, abordagem que será utilizada no decorrer do trabalho, de forma a prever a segurança do sistema. A segurança é um ponto fulcral em qualquer sistema, e tendo em conta que o trabalho irá ter suporte com serviços XR é importante ter em atenção aspetos de segurança e privacidade dos utilizadores.

Desta forma, esta abordagem visa quebrar o próprio sistema, num ambiente controlado, como objetivo de perceber quais são os pontos fracos do mesmo, através da injeção de falhas no ambiente. Este mecanismo traduz-se na introdução de forma consciente e deliberada de falhas ou mecanismos que possam causar o stress da aplicação/sistema. Isto permite avaliar a resposta do sistema em caso de entradas inesperadas por parte do utilizador, mudanças nas condições ditas normais do ambiente ou até mesmo falhas nas dependências. Desta forma, o sistema não se torna imprevisível e permite a correção de possíveis vulnerabilidades antes da sua implementação. O seu uso pode ter como objetivo o teste à camada de infraestrutura (de forma a perceber como o sistema reage em caso de falha de instâncias virtuais, por exemplo) ou à camada de rede (com o objetivo de validar a não existência de pontos únicos de falha e simular degradações da rede).

Relativamente à injeção de falhas, podem ser utilizadas ferramentas como o Gremlin, apresentado no artigo [22], onde os autores identificam esta ferramenta como solução para

testar sistematicamente as capacidades de tratamento de falhas de microserviços. Esta ferramenta foca-se na injeção das seguintes falhas:

- Simulação de cargas elevadas de CPU e/ou simulação de picos de tráfego;
- Desligar máquinas, de forma a visualizar a reação das suas dependências;
- Injeção de latência entre serviços;

Além das injeções de falhas disponibilizadas por esta ferramenta, podem também ser realizados outros testes, dependendo da arquitetura do sistema e dos objetivos do mesmo, como a simulação de falhas de um componente ou a interrupção de sincronizações entre os relógios do sistema (que muitas vezes provoca falhas nas comunicações). Esta sincronização é fundamental para aspetos de gestão, proteção, planeamento de uma rede, uma vez que permite a determinação de quando os eventos se irão realizar. Além disto, e uma vez que o tempo do sistema é o único ponto de referência entre todos os dispositivos na rede, torna-se bastante complicado fazer qualquer tipo de rastreamento se os registos de data e hora nos *logs* forem imprecisos [23].

Esta Engenharia deve ter sida em conta como uma forma de monitorização e avaliação de segurança contínua, isto é, este processo deve ser realizado de forma contínua no tempo e sempre que for detetada alguma vulnerabilidade ou algum ponto que deva ser melhorado, deve ser feita a sua correção ou a melhoria da segurança no sistema e posterior reavaliação. Isto pode ser traduzido como um ciclo de vida cíclico, conforme está representado na Figura 2.4.

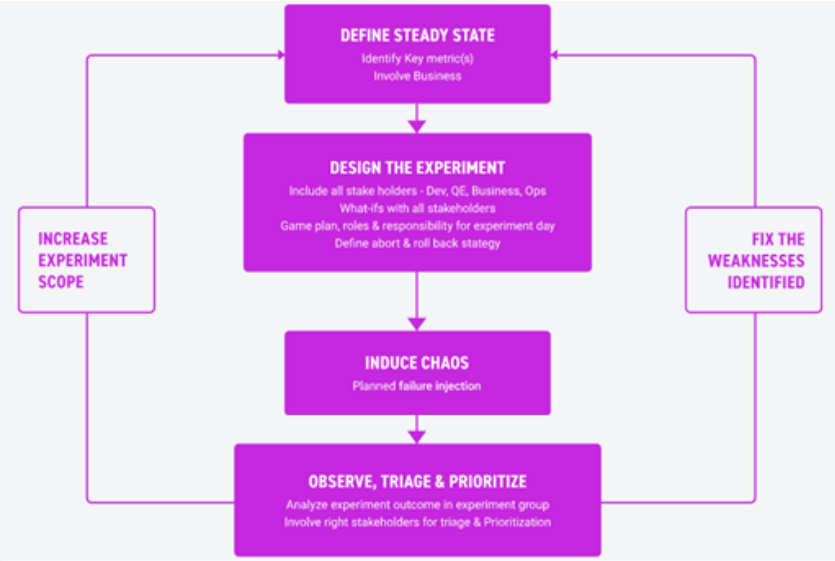

Figura 2.4- Ciclo de vida de Engenharia do Caos [24]

Neste sentido, e tendo em consideração o trabalho a desenvolver, é importante que seja realizada uma monitorização do ambiente, de forma a validar as respostas em caso de situações anómalas. Desta forma, é importante que sejam testadas várias injeções de falha, como o caso de falha de um dos componentes e interrupção de sincronizações entre os relógios do sistema, uma vez que estamos a falar de um ambiente com fortes dependências entre componentes. Também é importante que haja uma baixa latência nas comunicações, e como tal, a injeção de falhas de forma a testar a resposta em caso inverso deve ser tida em consideração, bem como o teste de casos de picos de tráfego.

Além disto, no sentido de fornecer suporte para avaliar e monitorizar a infraestrutura, aplicações e serviços do ambiente são consideradas algumas ferramentas, tais como, Prometheus e Grafana e Falco, ferramentas com objetivos distintos, mas que se forem utilizadas em conjunto, é possível abranger mais pontos de segurança.

O Prometheus é uma ferramenta que permite a monitorização de sistemas, serviços e aplicações, bem como o alerta de eventos. Esta ferramenta faz uso de exportadores que permitem exportar e enviar métricas dos componentes do ambiente, através de HTTP (por padrão) para um servidor central, Servidor Prometheus. Após esta exportação e envio, as métricas podem ser observadas através da criação de *queries* PromQL que permitem a escrita de gráficos ou a recolha de valores de forma a monitorizar o ambiente. Associado à visualização dos dados, é frequentemente usado o Grafana, uma vez que fornece uma maneira mais flexível de consultar as métricas (através de *queries* PromQL) e a sua visualização, uma vez que possui *dashboards* pré-definidas e configuradas, bem como a possível criação de novas *dashboards* [25].

O Falco é um projeto de segurança de tempo de execução que permite a deteção de comportamentos inesperados, quer seja, por eventos de escalonamento de privilégios, ou por leitura/escrita de ficheiros de forma suspeita [26]. Assim, quando estes eventos são detetados, é possível a emissão de alertas imediatos. Os comportamentos considerados suspeitos são geridos através de um conjunto de regras que fazem a verificação do comportamento do k*ernel* e verificam se algumas das regras foi quebrada.

De forma a facilitar a construção e execução facilitada de sistemas distribuídos elásticos e tolerante a falhas, uma possível implementação é o Apache Mesos, que permite a abstração de CPU, memória e armazenamento. Assim, permite uma simplificação de alocação de recursos e de extensibilidade permanente para execução de novas aplicações [27].

2.3 Ambientes nativos em nuvem e orquestração multi-domínios

Ambientes nativos em nuvem são abordagens que permitem construir e executar aplicações, com a exploração de características de flexibilidade, escalabilidade e resiliência. Os ambientes em nuvem podem ser privados, se forem fornecidos por uma rede privada e apenas destinados a utilizadores específicos, podem ser públicos se forem fornecidos por fornecedores externos através da internet e são disponibilizados a todas as pessoas que o desejem, ou híbridos se forem uma mistura destes dois tipos de nuvem.

Estes ambientes são assentes em cinco pilares distintos: *Design* moderno, microserviços (permite que os serviços sejam executados desacopladamente, o que se reve na agilidade dos mesmos), *containers* (permite que as ações efetuadas em serviços sejam mais facilitadas), serviços de *backend* (como serviços de armazenamento de dados, monitorização e serviços de identidade) e automação (permite a otimização na entrega contínua dos serviços e no desenvolvimento dos mesmos) [28].

Os autores do documento [29] analisam o problema da orquestração de múltiplos provedores e múltiplos domínios, primeiramente através da definição do problema e de seguida, abordando possíveis abordagens para solucionar o mesmo. Uma das principais problemáticas revê-se na colaboração dos diversos operadores e da orquestração dos diversos domínios

numa única operadora de rede, uma vez que é necessário que haja conhecimento dos recursos e topologias de diversos ambientes, e isto não pode ser efetuado entre provedores. A solução preferencial referida pelos autores passa arquitetura representada na Figura 2.5, onde é referida a existência de três interfaces principais. A primeira, representada como "IF1" é exposta ao utilizador, de forma a solicitar a implementação de serviços. A interface "IF2" conecta os orquestradores, de forma a permitir a operação em vários provedores. E por fim, a interface "IF3" tem o objetivo de abstrair a tecnologia e os detalhes da implementação, de forma a suportar a orquestração.

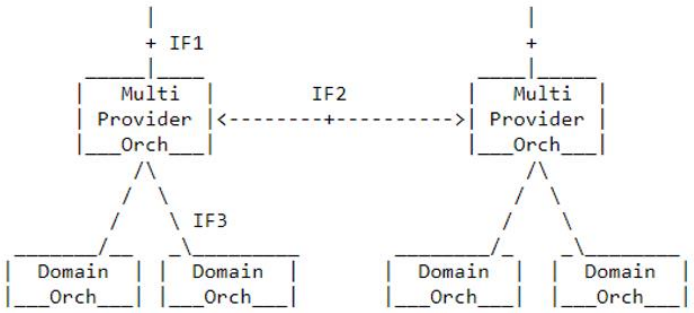

Figura 2.5- Arquitetura de provedores multi-domínio [29]

2.3.1. Infraestrutura de Computação em Nuvem

De forma a implementar múltiplos domínios, podem ser implementados provedores IaaS, que permitam a criação de ambientes isolados e independentes. Neste sentido, foram analisados trabalhos já existentes na literatura, de forma a perceber que soluções é que são utilizadas e quais as conclusões retiradas por cada autor. No trabalho [30], os autores utilizam a ferramenta OpenStack como solução para implementações de infraestruturas de nuvem, onde concluíram que a ferramenta é uma mais-valia para estas soluções, devido à sua modularidade e à disponibilização de diversos projetos robustos, com diferentes funcionalidades que podem ser implementados conforme as necessidades do utilizador. No mesmo sentido, no artigo [31], os autores apresentam soluções como OpenStack, OpenNebula e Eucalyptus. Neste artigo, os autores focam-se em pontos como *open-source* e suporte para novos padrões para a decisão da ferramenta mais vantajosa. Neste sentido, a conclusão revê-se no uso de OpenStack, uma ferramenta importante no *design* de computação em nuvem, uma vez que utiliza os recursos existentes para uma implementação modular, além de ser totalmente *open-source* e com suporte a padrões como OpenFlow. OpenFlow é um protocolo de comunicação que permite o acesso ao plano de encaminhamento de dispositivos de rede, isto permite a gestão das tabelas de encaminhamento dos mesmos (através da adição, modificação ou remoção de regras), por via remota reduzindo a necessidade de configuração manual, e não alterando a configuração dos dispositivos de rede [32].

Tendo estes estudos como ponto de vista inicial, OpenStack é uma plataforma que será analisada para solucionar a necessidade de implementação de infraestruturas em nuvem. Esta é uma plataforma que permite a criação e orquestração de nuvens públicas e privadas, através do uso de recursos virtuais agrupados. Ao contrário das plataformas de gestão de virtualização, o OpenStack executa uma combinação de ferramentas que criam um ambiente de nuvem que atende a cinco critérios do *National Institute of Standards and Technology*

(NIST) para a computação em nuvem: rede, recursos agrupados, interface de utilizador, provisionamento de capacidade e alocação ou controle automático de recursos [33].

De forma simplificada, esta plataforma pode ser visualizada como um conjunto de comandos, agrupados em projetos que têm a capacidade de retransmitir tarefas. Dos múltiplos projetos utilizados nesta plataforma que permitem o bom funcionamento da orquestração de nuvens, podem ser destacados os seguintes [34]:

- *Nova* é a ferramenta de acesso e gestão total para os recursos computacionais, incluindo criação, exclusão e programações dos mesmos.
- *Neutron* é o responsável pela ligação de redes a outros serviços da plataforma. Esta arquitetura exige diferentes nós de forma a conter interfaces de rede nas diferentes redes (rede de gestão, na rede de túneis e na rede externa).
- *Swift* é um serviço de armazenamento de objetos capaz de recuperar objetos de dados não estruturados. Este serviço é facilmente escalável e de alta disponibilidade, de forma a ser capaz de armazenar e recuperar elevadas quantidades de dados.
- *Cinder* suporta armazenamento de blocos persistentes, sendo blocos possíveis de montar numa máquina e rapidamente ter a flexibilidade de serem desmontados e montados novamente numa máquina distinta.
- *Keystone* é um *frontend* HTTP que oferece autenticação e autorização a todos os serviços, e possui o registo dos *endpoints* para os mesmos.
- *Glance* armazena e recupera imagens de disco de máquinas virtuais. Projeto idealizado com os principais requisitos de disponibilidade, tolerância e recuperação a falhas.

Estes componentes comunicam entre si, conforme representado na Figura 2.6, onde *Horizon* é a implementação da *dashboard* desta plataforma e como tal fornece uma interface de utilizador *web* para os diversos projetos. O projeto *Keystone*, tal como referido fornece autenticação e autorização aos restantes, de forma a aumentar a integridade do sistema e dos seus intervenientes. O projeto *Glance* fornece imagens às várias instâncias e armazena as diversas imagens no *Swift*, que por sua vez recebe também os *backups* necessários por parte do projeto *Cinder* que fornece também os volumes necessários para as várias instâncias. O projeto *Neutron* fornece conexão à rede para as várias instâncias e por fim, o projeto Nova provisiona as diversas instâncias conforme os requisitos. Além disto, é feita referência ao projeto *Ceilometer* que monitoriza *Glance* e Nova, uma vez que este é um projeto de recolha de dados [35].

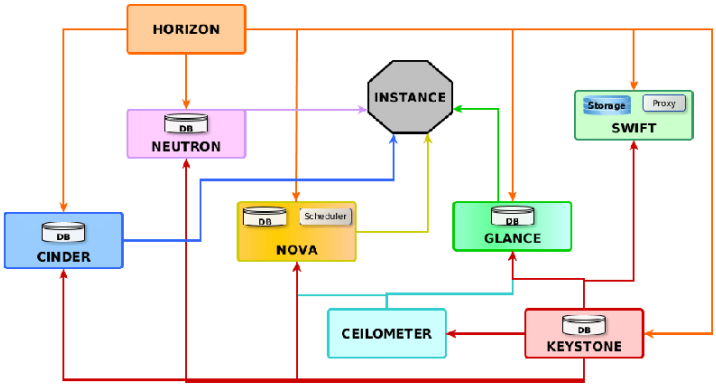

Figura 2.6- Representação das comunicações entre projetos [36]

Devido à dificuldade de gestão de implementações OpenStack, muitas vezes é feita a sua comparação com outras ferramentas, nomeadamente MicroStack e CloudStack. A primeira, é

uma distribuição de OpenStack que permite a diminuição do tempo e dificuldade de implementação, apesar de possuir os projetos e bibliotecas utilizadas em OpenStack. No mesmo sentido, CloudStack inclui todos os recursos necessários (orquestração de computação, rede como serviço, gestão de utilizadores e uma API nativa completa e aberta) para a implementação e orquestração de infraestruturas na nuvem. Além disto, CloudStack permite também que sejam implementadas nuvens híbridas, uma vez que a API é compatível com AWS EC2, por exemplo [37].

Por outro lado, no artigo [38] foi apresentada a solução Elastic Compute Cloud (EC2), analisado o seu desempenho e custo de execução de aplicações de fluxo de trabalho científico em nuvem. Os autores concluíram que esta é uma boa plataforma para implementação de aplicações de fluxo de trabalho, apesar de a aquisição de recursos para execução de tarefas ainda ser um ponto que necessita de melhorias. Da mesma forma, no artigo [39] os autores tiveram como objetivo o estudo do impacto da virtualização no desempenho da rede do *data center* implementado com esta solução. As conclusões registadas apresentam algumas falhas a nível de desempenho em caso da sua implementação num *data center*. Além disso, é de referir que uma desvantagem deste provedor é o facto de pertencer à Amazon e como tal acarreta custos, ao contrário do OpenStack que é uma plataforma *open-source*.

EC2 é uma alternativa ao OpenStack. Este é um serviço *web* proveniente da Amazon que possibilita capacidade computacional segura e redimensionável na nuvem, de forma a tornar a computação em nuvem mais acessível e facilitada. Assim sendo, permite a criação de diversas instâncias com diferentes Sistemas Operativos e a gestão de permissões de acesso à rede [40].

Com esta alternativa é possível definir o tipo de instâncias e determinar o local de execução das mesmas, compostos por regiões e zonas de disponibilidade, que permitem o isolamento entre regiões. Além disto, é possível monitorizar as instâncias, de forma a provisionar as suas capacidades computacionais, otimização de CPU (especificação de um número de vCPUs inicial ou a desativação do recurso de *multithread* em caso de não haver essa necessidade), auto escalonamento (redução ou ampliação automática da capacidade, de acordo com as condições definidas), bem como a hibernação das mesmas e posterior reinício [41].

Além disto, plataformas como Citrix, Vmware Vcenter e Red Hat virtualization permitem a gestão centralizada de máquinas virtuais, *hosts* e todos os componentes do ambiente.

2.3.2. Orquestração entre Domínios

Muitas vezes, podem ser utilizados diferentes provedores de nuvem devido à redução de depêndencia de fornecedores únicos, flexibilidade de custos e a possíveis políticas locais de armazenamento de dados (por exemplo, os dados podem ter de ser armazenados no mesmo país, ou na unidade de nuvem fisicamente mais próxima, acabando por reduzir possíveis latências). Desta forma, é importante que haja um elemento de orquestração das diversas nuvens que forneça funções ao Operador de nuvem, que poderão posteriormente ser executados na API do provedor ou nos serviços de CLI do mesmo, tornando possível operações *Create*, *Read*, *Update* e *Delete* (CRUD) aos diversos subcomponentes, de forma a tornar mais eficiente a orquestração do ambiente, conforme as diversas necessidades. Além disto, os desafios deste tipo de orquestração devem ser tidos em consideração, conforme referidos no artigo [42].

Desta forma, o objetivo principal destes elementos passa pelo auxílio da orquestração de nuvem, proporcionando abstração e automação de forma a fornecer aos desenvolvedores uma forma de evitar a complexidade da nuvem, uma vez que quanto mais complexidade mais pontos se tornam necessários gerir e, consequentemente, mais possíveis pontos de falha.

No artigo [43], os autores implementaram a ferramenta Cloudify como solução para controlo unificado de plataformas comuns de computação em nuvem. Da mesma forma, apresentam bases de automação e gestão de cluster Cloudify, bem como fornecem mecanismos de recolha de dados de forma a avaliar o seu desempenho. Esta ferramenta foi implementada e analisada em conjunto com OpenStack, o que permite perceber a sua compatibilidade.

Cloudify é uma ferramenta *open-source* direcionada à orquestração de várias nuvens, incluindo abstração e automação, proporcionando que os utilizadores automatizem todos os ciclos de vida de uma aplicação, gerindo, detetando algumas falhas e manutenções contínuas. Esta ferramenta utiliza arquivos *blueprint* que são gravados em YAML, de forma a descrever a representação lógica de uma aplicação, como os seus componentes, a relação entre eles, a forma como são instalados, configurados, monitorizados e mantidos [44]. Com as configurações *blueprint* a manutenção das aplicações torna-se mais simplificada, uma vez que é mais simples de atualizar ou reconfigurar novas aplicações. Além disto, a ferramenta permite também a visibilidade dos dados por meio das métricas e dos resultados recolhidos e a adição de extensões através de *plugins*.

No artigo [45] esta ferramenta é comparada com Terraform, as quais se revelam opções bastante eficientes para a gestão de recursos de computação de provedores de nuvem. Esta ferramenta permite a definição de infraestrutura como código, através de uma linguagem de programação declarativa. Assim, permite a implementação e gestão dessa mesma infraestrutura em vários provedores de nuvem pública, assim como em plataformas de virtualização (o caso do OpenStack) através do uso de comandos. O uso desta ferramenta é bastante prático, uma vez que a sua leitura de arquivos de definição permite o provisionamento dos vários recursos do provedor especificado. O uso desta ferramenta é visível em diversos artigos, como [46], onde é utilizada em conjunto com Amazon Web Services, de forma a fornecer PaaS para análise de imagens médicas. Também no artigo [47] é utilizado Terraform com o objetivo de fornecer uma plataforma de processamento de *big data*, independentemente da nuvem em que se encontrem inseridos.

Outras plataformas, como ManageIQ são exploradas. Os autores do artigo [48] desenvolveram uma *testbed* onde a ferramenta ManageIQ foi implementada, em conjunto com Ansible e OpenStack, de forma a fornecer uma ajuda na orquestração de diferentes máquinas virtuais e *containers*. ManageIQ fornece flexibilidade para incluir novas ações de orquestração, através dos *playbooks* Ansible para adicionar/executar novas configurações ou implementações.

ManageIQ é uma plataforma *open-source* para TI híbrida (mistura de nuvens privadas e públicas) e infraestruturas virtuais que fornecem auto compreensão aos utilizadores finais e impõem políticas de conformidade, tudo numa única interface [49]. A principal característica desta plataforma é que o administrador pode manter um catálogo de pedidos solicitados por utilizadores. Estes pedidos são ações necessárias para o provisionamento de algum elemento que maioritariamente necessita de informação inicial (como por exemplo qual o tamanho do disco necessário ou a memória necessária). Quando os pedidos são atendidos são especificados na máquina de estado e pode ser feita a publicação do mesmo num catálogo de

serviços que permite que outros utilizadores façam a mesma solicitação [50]. Além disto, esta plataforma permite a autodescoberta para criação de políticas avançadas de segurança e conformidade, bem como a otimização por meio das métricas de todos os componentes que são usados para planear capacidades, executar cenários hipotéticos ou fazer recomendações de dimensões.

O Ansible é uma biblioteca que possui imensos estudos na literatura sobre as suas capacidades e benefícios, nomeadamente [51], onde a automação de aplicações é o foco principal, em atividades de provisionamento do ambiente, bem como de implementação de aplicação. Neste artigo, a solução apresentada corresponde à implementação de provedores Amazon, juntamento com Ansible como mecanismo de orquestração. Desta forma, os autores concluíram que a flexibilidade desta biblioteca facilita a automação de todo o procedimento de implementação de aplicações. No artigo [52] os autores implementaram esta ferramenta para gerir 10 laboratórios da Universidade de Tecnologia Brno numa única infraestrutura, com o objetivo de simplificar a configuração de tarefas que são realizadas diariamente. Desta forma, concluíram que Ansible permite uma gestão remota de forma segura e uma configuração de redes mais eficiente e prática.

Ansible é uma biblioteca capaz de implementar aplicações e serviços em nuvem automatizados, realizar a sua orquestração e a sua entrega contínua de forma automatizada, tal como é explorado no artigo [53]. Além disto, esta biblioteca apresenta ser uma opção para casos de recuperação de desastres uma vez que a utilização destas ferramentas automatiza este processo e o torna bastante mais rápido e eficiente do que em caso de operações humanas [54]. As funções desempenhadas por esta ferramenta utilizam componentes, como *plugins* de comunicação que executam a comunicação com o terminal ou *plugins* funcionais, inventário, API e módulos ao nível de *core*, funcionais ou personalizados. Além disto, o ficheiro *playbook* interage com o utilizador e com a ferramenta, uma vez que é neste que são armazenadas e executadas as diversas funções de configuração, de implementação e orquestração. Através deste manual é possível configurar tarefas em *hosts* determinados, com o utilizador pretendido e com a escrita dos comandos necessários para essa mesma tarefa, de forma a automatizar e diminuir a interação humana nestes processos.

No artigo [55] os autores descrevem a biblioteca Libcloud como forma de aumentar a flexibilidade associadas aos recursos do ambiente e de reduzir os custos associados à orquestração das infraestruturas. Desta forma, é projetada e implementada para atuação como nuvem privada para partilha de recursos entre diversos utilizadores. Libcloud é uma biblioteca que fornece uma API abstrata para interação com diversos provedores de nuvem, permitindo uma única escrita de código seja aplicada em diferentes provedores que estejam a ser utilizados. Esta biblioteca divide os recursos que podem ser geridos, em diferentes categorias [56]:

- *Cloud Servers* e *Block Storage* – Permite a gestão de servidores em nuvem e servidores virtuais, oferecidos por múltiplos provedores, conforme referido anteriormente.
- *Cloud Object Storage* e CDN – Permite a gestão de armazenamento de objeto em nuvem e gestão de CDN (rede distribuída geograficamente, responsável pela distribuição do conteúdo, de forma a melhorar o desempenho e segurança).
- *Load Balancers as a Service* – Permite a gestão de balanceadores de carga, responsáveis pela repartição de tráfego por várias máquinas, de forma a evitar a sobrecarga da mesma.

- DNS como serviço – Permite a gestão de DNS, capaz de fazer a tradução do nome de computador ou nome de serviço para o seu respetivo IP.
- *Containers as a Service* e *Container Engine* (GKE) – Permite a instalação e execução de *containers* baseados em plataformas virtualizadas, como Docker.

O estudo [36] apresenta uma comparação entre OpenStack e OpenNebula, no qual foi concluído que enquanto o OpenStack é uma solução para implementações complexas e robustas, OpenNebula é mais centralizado, sendo mais indicado para *data centers*, instituições públicas e privadas, sendo uma boa opção para gestão entre nuvens. Da mesma forma, no artigo [57] os autores apresentam e comparam estas duas ferramentas, tendo em conta características como arquitetura, *hypervisors* e segurança. A conclusão retirada pelos autores foi o facto de OpenNebula ser adequado para lidar com grandes quantidades de dados e para aproveitar as infraestruturas de TI existentes. Estes estudos revelam que o OpenNebula apresenta ser uma ferramenta bastante eficiente para gestão entre nuvens, por exemplo para gestão de infraestruturas como o OpenStack, que se enquadra na categoria de plataformas de virtualização, permitindo provisionar instâncias e os seus recursos.

Esta plataforma de computação em nuvem, OpenNebula, permite a gestão de infraestruturas heterogéneas de *data center* distribuído. Esta plataforma tem como principal objetivo a automação (processos de provisionamento de novos recursos de infraestrutura são processos automáticos), a portabilidade (implementação e migração de fluxos de trabalho entre diferentes provedores de nuvem) e a interoperabilidade (fluxos de trabalho podem ser localizados em diferentes provedores e podem ser utilizados para criar um serviço agregado). Desta forma, torna possível a independência do provedor, em simultâneo com o aumento da disponibilidade do serviço.

Outra alternativa às soluções apresentadas é Mist.io, que permite definir políticas de acesso para permissões de provisionamento de auto atendimentos aos utilizadores, otimiza a implantação de fluxo de trabalhos operacionais de arquiteturas em nuvem, e permite a receção de alertas sobre máquinas subutilizadas, para que seja possível a resolução dos problemas de forma mais rápida [58]. Esta plataforma oferece uma interface para visualização e gestão de infraestruturas e aplicações, permitindo visualizar as várias nuvens e o seu respetivo estado, as diversas máquinas e as imagens que podem ser utilizadas para inicializar instâncias. Permite também a visualização do estado de rede (que permite o controlo sobre a topologia e o endereçamento IPv4 ou IPv6), chaves SSH que podem ser importadas ou eventualmente criar novas e os *scripts* armazenados para automação das ações.

De forma a facilitar a construção e gestão de nuvens OpenStack, podem ser utilizadas ferramentas como Puppet, Chef e Juju. A primeira é uma ferramenta *open-source* que fornece ferramentas para automatizar a gestão, através da escrita de código em DSL ("*Domain-Specific Language*"). A plataforma é composta por vários componentes, o servidor, o agente e a base de dados, que em conjunto permitem a gestão, armazenamento e execução de código [58]. Esta plataforma possui integração com OpenStack de forma a facilitar o teste e validação às configurações do OpenStack, implementadas com os módulos Puppet, e inclusivamente de forma a facilitar a implementação de OpenStack [60].

Da mesma forma, Chef é uma ferramenta de automação que permite gestão de configuração, de forma a automatizar processos e tarefas em vários servidores e outros dispositivos de uma organização em etapas simples. Esta ferramenta permite também a construção e gestão de nuvens OpenStack [61].

Juju é um Charmed Operator Framework que permite a gestão de aplicações, infraestrutura e ambientes, de forma a garantir a redundância e resiliência de todo o ambiente. Assim, a sua utilização passa pelos componentes Charmed Operators e Charmed Operator Lifecycle Manager, que em conjunto permitem controlar todo o ambiente [62]. Esta ferramenta é bastante simples e permite a construção e gestão de nuvens, nomeadamente, nuvens OpenStack, vSphere ou manuais.

Além disto, a utilização de sistemas de *containers* tem sido exponencial e este fator deve-se à sua capacidade de empacotar, transferir e executar código de aplicações em diversos ambientes distintos, o que traz benefícios a nível de fluidez na gestão de aplicações. No entanto o uso de *containers* acarreta problemas na sua gestão, uma vez que o seu número tem aumentado e é necessário manter a comunicação entre os diversos. Além disto, processos como a sua implementação, escalonamento ainda não são processos automatizados por si só, o que carreta intervenção humana em cada cluster.

No processo de implementação e de atualização em caso de novas versões e em caso de serviços *stateless*, este processo é facilitado, uma vez que a sua atualização é feita de forma automatizada, sendo atualizado um *pod* de cada vez. Assim, não há o risco de afetar o acesso à aplicação, uma vez que a escolha do *pod* seguinte a ser atualizado está diretamente relacionado com a sua utilização no momento. Em caso de qualquer falha durante a atualização, a mesma pode ser rapidamente corrigida através do processo R*ollback*, que desfaz a publicação desejada. Além disto, este processo também pode ser realizado através da estratégia *Recreate*, que desliga todas as instâncias e cria novas instâncias com a nova atualização. No entanto, neste caso facilmente é percetível que há um período de tempo (entre o desligar as instâncias e a criação das novas) onde o acesso é perdido.

Em contrapartida, no caso de serviços *statefull* este processo é bastante mais complexo, uma vez que acarreta a necessidade de servidores de base de dados estarem sempre em sintonia para possível análise dos estados, como tal, devem ser constantemente verificados e sincronizados. Quando se fala em réplicas de bases de dados é importante não esquecer que estas não são idênticas, cada uma tem o seu próprio estado e a sua própria identidade e como tal, devem ser atualizadas através de uma certa ordem, de forma a não prejudicar o sistema. Além disto, e como há diferentes tipos de bases de dados não há uma solução padrão para este processo, é dependente do tipo de base de dados. Neste ponto entra a necessidade de intervenção manual para esta verificação e atualização, uma vez que as bases de dados são todas diferentes e podem ser de diferentes provedores, como tal, não há uma solução padrão para este processo.

No processo de escalonamento, apesar da existência do Controller Manager, quando um *pod* é bastante acedido e se torna necessário o seu escalonamento, não há operações automatizadas que criem as réplicas necessárias, sendo necessário a intervenção humana nesse mesmo sentido.

Neste sentido, o uso de ferramentas de orquestração de cluster de *containers*, como Kubernetes, vem facilitar esta gestão, de forma a localizar e supervisionar os diversos serviços. Da mesma forma podem ser utilizadas ferramentas que permitem a gestão de múltiplos clusters Kubernetes de forma a tornar a sua orquestração o mais eficiente possível, tal como representado na Figura 2.7, onde diferentes clusters são geridos através de Docker Network, que permite o provisionamento dos mesmos. Nesta esquematização, com o objetivo de aumentar a segurança baseada na identidade é utilizado o Vault, de forma a permitir controlar

o acesso a chaves de criptografia e segredos, através de uma autenticação em fontes confiáveis.

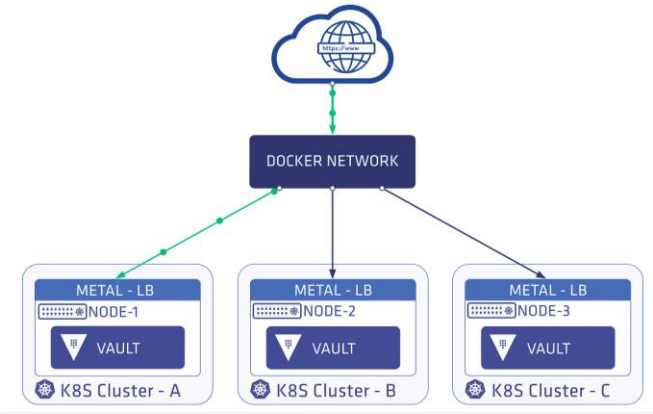

Figura 2.7- Esquematização de gestão de múltiplos clusters de containers, adaptado de [63]

No trabalho [64], com o objetivo de gerir configurações multi-cluster de Kubernetes, os autores propuseram a exploração de mecanismos básicos da ferramenta, bem como a implementação da ferramenta KubeFed num ambiente *multi-cloud*. No artigo [65], os autores utilizam a mesma ferramenta como ponto de partida para um controlo descentralizado, onde aproveitam o servidor Kubernetes API padrão para oferecer o suporte para a gestão de recursos. Nos dois artigos referidos, os autores fazem referência ao facto de esta ferramenta ainda se encontrar em fase de produção e testes, no entanto, não descartam a solução para quando o seu uso estiver operacional. Apesar disto, no artigo [66], os autores mostram que atrasos e falhas temporárias de rede podem levar à instabilidade de aplicações nesta ferramenta, o que no caso deste trabalho é um facto importante a ter em consideração.

Esta ferramenta, também conhecida como Kubernetes Cluster Federation permite a coordenação de configurações de vários clusters Kubernetes a partir de um único conjunto de APIs, ajudando na sua gestão através de mecanismos de baixo-nível que expressam qual deveria ser a configuração de cada cluster, conforme as suas características, tendo em conta principalmente características de geo-localização ou de recuperação de desastres [53]. A configuração desta ferramenta é baseada em dois tipos de informação, configuração do cluster (definição de quais são os clusters que devem ser geridos) e a configuração do tipo de API, que permite a visualização de modelos comumente utilizados, a localização dos recursos e a execução de possíveis modificações nos clusters.

Com o objetivo de criar um ambiente em nuvem privada com funções de rede virtualizada, os autores do trabalho [68] apresentam a implementação da ferramenta ONAP multi-K8s em conjunto com OpenStack. ONAP multi-k8s permite a gestão de múltiplos clusters, podendo ser dispersos geograficamente através de um *host* cluster.

De forma a gerir serviços baseados em Kubernetes e em ambientes em nuvem, os autores do artigo [69] demonstram a viabilidade do uso da pilha Rancher. Esta pilha de *software* tem o objetivo de permitir a utilização de containers, abordando questões de desempenho e de segurança da gestão dos mesmos, através da automação de configuração dos ambientes, permitindo negligenciar configurações especificas ou facilitar a (re)configuração de novas versões [70].

2.3.3. Segurança de ambientes multi-cloud

Em paralelo com o crescimento da utilização destes ambientes, também os ambientes multi-domínios são cada vez mais utilizados, grande parte dos projetos incorpora nuvens públicas e privadas de vários provedores, sendo ambientes com enorme grau de distribuição e consequentemente de complexidade, acrescendo também uma exigência maior de segurança e administração, desta forma o estudo da segurança destes ambientes já é bastante debatido em diversas especificações, como [71]. Neste sentido é importante ter em consideração que diferentes provedores podem utilizar métodos ou medidas distintas, uma vez que cada um possui as suas próprias APIs e processos, e como tal é importante que seja feita uma orquestração entre nuvens que contemple estas diferenças e seja capaz de reagir independentemente do provedor.

Relativamente à segurança de ambientes entre nuvens é fundamental não esquecer que são ambientes *multi-tenancy*, uma característica que tem sido estudada por diversos autores, como no artigo [72], onde são analisados os principais desafios e possíveis abordagens de diminuição de risco. Uma vez que múltiplos utilizadores partilham os mesmos recursos, existe uma preocupação acrescida, pois há um aumento de possibilidade de um dos utilizadores utilizar os dados de outros, quer de forma consciente ou de forma acidental. Esta preocupação deve ser tida em consideração pelo provedor do serviço e pelo próprio utilizador, as responsabilidades devem ser partilhadas e ambas as partes devem garantir a sua parte do acordo.

O armazenamento dos dados deve ser efetuado numa infraestrutura partilhada, mas sempre correspondendo aos requisitos regulamentares, nomeadamente tendo em conta o RGPD, uma vez que este regulamento não se aplica em todos os países. Além disto, é necessário ter em atenção que um utilizador pode prejudicar o restante ambiente se não forem tidas atenções relativamente à proteção de dados. Deve ser tida em consideração a partilha e exposição de possíveis dados confidenciais, a possível indisponibilidade ou perda do serviço, a necessidade de eliminação dos dados de utilizador por encerramento de acordo e a respetiva autorização que cada utilizador tem sobre os dados e sobre o sistema, tal como foi abordado no artigo [73], onde os autores implementaram uma plataforma com estes desafios em consideração. Neste sentido, é também importante a utilização de métodos criptográficos como por exemplo através da Encriptação Homomórfica, que apesar da sua elevada complexidade, apresenta ser uma opção vantajosa em casos onde há a necessidade de efetuação de operações nos dados [74]. É fundamental, também, efetuar controlo de acessos, segmentar implementações de nuvem e isolar os dados e aplicações que possam ser mais críticas, de forma a dificultar o acesso aos mesmos por partes não autorizadas.

Da mesma forma que os dados devem ser isolados e deve ser requisitada a autorização para acesso ou modificação dos mesmos, os recursos também devem ser uma preocupação, uma vez que uma interferência de sobrecarga numa máquina de um utilizador, pode causar impacto negativo nas restantes. Além disto, a atribuição de recursos deve ser feita de forma que o compromisso da camada de virtualização não permita que um utilizador altere a configuração de outra máquina.

Desta forma, torna-se relevante analisar possíveis ferramentas de orquestração entre domínios, de forma a facilitar atividades CRUD nos diversos, bem como a realizar atividades de segurança que permitam fazer a sua monitorização. Num ponto de vista primário é importante que seja feita a autenticação e autorização dos diversos utilizadores, de forma a

criar políticas de segurança e a efetuar o controlo de acessos a cada solicitação, independentemente do provedor que estiver a ser utilizado. Relativamente à autenticação, uma possível abordagem passa pela autenticação FIDO ("Fast IDentity Online"), um conjunto de padrões para autenticação forte de forma mais rápida e simples, através de uma terceira parte confiável (composto por um servidor *backend* e uma aplicação *frontend* que interage com o autenticador de utilizador). Assim, quando um utilizador se liga ao ambiente, o FIDO autenticador gera as credenciais do utilizador (constituídas por uma chave pública e uma chave privada).

No ponto de vista de proteção de dados é fundamental que seja efetuada uma classificação dos mesmos e que estes sejam armazenados conforme a mesma, de forma a facilitar o planeamento da distribuição geográfica dos dados de acordo com as obrigações de conformidade, e devem ser implementadas soluções de Prevenção de Perda de Dados. Além disto é importante que a segurança seja constante, manter fechadas as portas que não sejam necessárias, remover *software* desnecessário, utilizar o princípio do mínimo privilégio, monitorizar e tirar proveito de ferramentas que facilitem a visibilidade de todo o ambiente, de forma a detetar e responder a possíveis ameaças.

Com isto, é importante que seja feito o isolamento dos dados e que os mesmos sejam protegidos. Tal como referido anteriormente, o uso de Vault pode ser uma solução atrativa, uma vez que permite a gestão de *secrets* e a proteção de dados, através de controlo de acessos e encriptação. Esta ferramenta possui implementações sobre SGX (Software Guard Extensions), *software* que permite o tratamento de *hardware* confiável, oferecendo criptografia de memória baseada em *hardware*, que permite o isolamento de código de aplicação e dos dados de memória [75]. Da mesma forma, *softwares* como Arm trustzone e AMD SEV podem ser utilizados. Arm trustzone oferece uma abordagem de segurança com isolamento reforçado por *hardware* [76]. AMD SEV (Secure Encrypted Virtualization) faz uso de chaves para o isolamento do *hypervisor*, requerendo ativação no Sistema Operacional e no *hypervisor* [77].

Uma vez que ambientes em nuvem podem comunicar entre si, mas deve ser garantido um isolamento de cada nuvem, é importante garantir que o tráfego dentro do ambiente isolado não é intercetado. Não só é importante garantir que apenas são efetuadas as comunicações autorizadas, entre os dispositivos autorizados, como garantir que não são efetuadas interceções, com o objetivo de processar o tráfego e reenviar para o destino final. Neste sentido, é possível utilizar ferramentas que permitem não só a gestão de tráfego, como a utilização de mTLS pra fornecer encriptação às comunicações internas, tal como Istio, referido na Secção 2.2.2.

Além disto, torna-se também importante analisar o modelo Zero-trust. A confiança usual numa rede passa por confiar automaticamente em tudo o que se encontra dentro da mesma, no entanto isto causa alguns problemas, nomeadamente no caso de ser inserido algum dispositivo malicioso (ex: inserção de *pen drive* com vírus) em algum computador interno. Neste caso, como todos os dispositivos do interior da rede são confiáveis, não é efetuada uma verificação do dispositivo inserido. Com a aplicação do modelo Zero-trust a confiança não é concedida e é adotado o princípio de privilégio mínimo. Assim, todos os dados, ativos e serviços são considerados recursos e o acesso aos mesmos é concedido apenas por sessão, a qual deve ser avaliada antes de o acesso ser concedido [79]. Isto traduz-se na necessidade de todas as pessoas, bem como *hardware* serem analisados e verificados. Além disto, todo o

tráfego deve passar por validação para cada solicitação de acesso, independentemente de pertencer à mesma rede ou não.

No artigo [79] os autores discutem a definição da arquitetura Zero-trust e propõem um modelo de confiança para ambientes em nuvem que fazem uso de Kubernetes. No qual é utilizado um servidor *proxy*, responsável por redirecionar o cliente ao servidor de autenticação e autorização, através de controlo de acessos que definem se o utilizador tem permissões.

No entanto, num ambiente de nuvem, isto significa que a verificação de políticas de segurança deve ocorrer em todas as comunicações de rede entre serviços e *containers*. Isto traduz-se no desafio de lidar com todas os constituintes destes ambientes, uma vez que são ambientes complexos e em exponencial crescimento. Neste sentido, uma das soluções passa por utilizar o conceito de s*ervice mesh*, referido na Secção 2.2.2 , uma vez que permite a modificação de funções relacionados à orquestração da lógica da aplicação para a camada de infraestrutura, de forma a simplificar as mesmas e, consequentemente, ajudar na implementação do modelo Zero-trust em ambientes em nuvem.

2.4 Orquestração Automatizada

A abordagem Zero-touch visa aumentar a automação dos processos, através de uma análise constante ao ambiente, observação das suas necessidades e execução de operações essenciais, de forma automática, para a manutenção do bom funcionamento do mesmo (como operações de escalonamento, *backup*, restauro e provisionamento), sem necessidade de qualquer recurso humano. Esta abordagem foi desenvolvida segundo a arquitetura presente no documento [80], de forma a responder de forma positiva às necessidades dos utilizadores.

Nesta arquitetura diversos componentes são referenciados, de entro os quais, **Management Services** (representados como serviços ZSM), componente que oferece um conjunto consistente de recursos para fins de chamada e comunicação, permitindo um alto grau de automação e continuidade. Estes podem ser oferecidos para consumo por diversos consumidores de serviço, podendo ser combinados em novos sistemas de gestão.

Management Funcions (componente representado dentro do domínio) possui uma gestão *closed loops*, responsável pela automação de operações e como tal podem produzir ou consumir serviços, podendo ser um "*service provider*" ou "*service consumer*", respetivamente, ou também pode ser os dois em simultâneo, isto é, consumir e produzir serviços.

ZSM framework consumers, tal como o próprio nome indica este componente consome serviços, que tanto podem ser fornecidos pelo Management Domain, como pelo E2E Service Management Domain.

Domain Managed Infrastructure Resource é o componente responsável pelos recursos de infraestrutura, que podem ser físicos (ex: PNFs, como *routers*, *switches*, *firewalls*, balanceadores de carga), virtuais (ex: Serviços baseados em *software* ou VNFs) e/ou baseados em nuvem. Os produtores de serviços podem necessitar de comunicação com a Infrastructure Resource ou essa mesma comunicação ser feita de forma indireta.

Management Domain (pode ser dividido em vários "*subordinate*" *Management Domains* se as responsabilidades administrativas ou separação de interesses assim o exigir), tem a

principal função de agrupar os serviços (que podem ser consumidos apenas por consumidores autorizados dentro do domínio, não havendo exposição do serviço, ou serem expostos externamente apenas sob certas condições, designados por serviços condicionais) com os recursos necessários para o seu uso e opcionalmente pode fornecer meios de gestão do ciclo de vida dos mesmos. De um ponto de vista mais complexo, os seus serviços podem ser divididos em categorias, como a recolha de dados, a análise dos mesmos, controlo de cada entidade, suporte a serviços (permite a gestão de políticas), serviços de orquestração e inteligência (permite a inteligência de automação dos *closed-loops* através do apoio à decisão, tomada de decisão e por fim o planeamento da ação a executar.

E2E Service Management Domain, fornece gestão ponta a ponta de serviços voltados para o utilizador, no entanto não gere recursos de infraestrutura de forma direta. Este componente possui diversos serviços idênticos ao anterior, que podem ser mais uma vez categorizados em serviços de recolha de dados, de análise, suporte de serviços, orquestração e inteligência.

Data Services (representado dentro de um *Ma*nagement Domain ou E2E Service Management Domain, bem como na ligação entre domínios distintos) permitem meios consistentes de acesso a dados de gestão partilhados e a sua persistência por consumidores autorizados, tanto dentro ou entre domínios distintos, eliminando assim a necessidade de Management Functions controlar a persistência de dados. Desta forma, este componente possui as seguintes funcionalidades (no caso de Data Services entre domínios todas se tornam obrigatórias, ao contrário do caso de dentro de domínios, onde as funcionalidades são apenas opcionais):

- Gestão de armazenamento de dados, onde é feita a descoberta dos serviços de persistência de dados disponíveis;
- Ambiente de execução para o processamento dos dados;
- Através de diversos tipos de Bases de Dados distintos, permite o suporte a armazenamento persistente, assim como possíveis soluções de armazenamento para grandes quantidades de dados.

Integration Fabric é o componente que permite a fácil comunicação e interoperação de Management Functions, facilitando o controle e possível partilha de dados, recursos ou serviços de domínios. Como tal este componente possui as principais funções:

- Registo ou cancelamento de registos de serviços;
- Descoberta de serviços registados e a descoberta de como é feito o acesso aos mesmos;
- Suporta a comunicação entre dois ou mais Management Functions, quer seja de forma síncrona ou assíncrona;
- Invocação de forma indireta de serviços e a gestão das regras para este processo de invocação;
- Configuração de exposição dos serviços, uma vez que estes podem não estar expostos externamente ao domínio.

Este componente possui duas formas de implementação, podendo ser referente apenas a um domínio (neste caso apenas é obrigatório a funcionalidade de configuração de exposição dos serviços) ou entre domínios, Cross-domain Integration Fabric, que possui todas as funcionalidades referidas, com exceção da configuração de exposição de serviços, que uma vez que é uma funcionalidade requerida em cada domínio não é necessária essa redundância.

Domain Integration Fabric é uma entidade lógica que representa funções de gestão responsáveis pelo controle da exposição dos serviços para o exterior do domínio (como referido, podem possuir certas condições de exposição, ou não serem totalmente expostos ao exterior) e controla o acesso aos mesmos. Além disto, fornece serviços de integração adicionais para a gestão de funções do seu Management Domain. Este componente habilita o consumo de serviços, bem como a integração e comunicação com sistemas de gestão de terceiros, dai a sua oferta de serviços de orquestração, recolha de dados e análise.

Cross-domain Integration Fabric permite a facilitação do processo de fornecimento de recursos e acesso a *endpoints* em múltiplos domínios, por outras palavras, permite a partilha de dados entre domínios distintos, dai ser o componente representado entre o Management Domain e o *E2E Service Management Domain*.

Além das principais características desta arquitetura é fundamental ser pensada na sua segurança, como tal, esta pode ser dividida em dois principais pontos: Segurança e privacidade dos dados, quer na sua partilha, processamento ou armazenamento, bem como na segurança do próprio sistema, isto é, na segurança dos diversos componentes e na confiança relativa a cada um.

Cada gestão de domínio e gestão de serviços ponta a ponta é responsável por assegurar a devida proteção aos seus dados, tendo sempre como base as leis regulatórias de proteção de dados e privacidade aplicáveis. Nesta área podem ser definidos alguns pré-requisitos:

- Recolher, processar e armazenar os dados estritamente necessários para as operações;
- Gestão de credenciais, identidades e certificados permitindo gerir os acessos de cada utilizador;
- Na circulação e no armazenamento de dados é importante haver proteção dos mesmos contra acesso ou modificações não autorizadas;
- Na transmissão de dados, deve ser utilizada autenticação mútua em *endpoints*, de forma a garantir a integridade dos intervenientes;
- Uso de algoritmos de criptografia conforme a segurança que os mesmos estabelecem. Tendo isto em consideração e uma vez que qualquer algoritmo pode ser quebrado mais rapidamente do que se imagina, é importante haver a fácil migração para uma nova versão, sem causar impacto no funcionamento do sistema (por exemplo, não causar a indisponibilidade do serviço).

Relativamente à segurança de cada componente do sistema, é importante garantir que cada um é confiável (necessidade de mecanismos de controlo de acessos e não repúdio), quer seja por estar incorporado num domínio ou conjunto de domínios que são confiáveis por padrão ou através da verificação da sua confiabilidade. Quando um componente não faz parte de um domínio confiável, é necessária a sua verificação, decorrendo nos seguintes passos:

1. Verificação inicial antes da sua execução;
2. Monitorização contínua;
3. Auditoria periódica.

Se houver falha na verificação inicial, a função de gestão não terá permissão para ser executada no domínio. Caso a falha só seja detetada na monitorização ou auditoria, uma vez que a função já se encontra em execução, será imediatamente colocada em quarentena, de forma a mitigar o risco o mais rapidamente possível.

Além da abordagem Zero-touch, é também importante analisar ETSI ENI ("Experiential Networked Intelligence"), onde o objetivo passa pela introdução de métricas de otimização e ajuste da experiência do utilizador, através do uso de técnicas de Inteligência Artificial para aprendizagem de máquina e de raciocínio. Desta forma, é feito o reconhecimento e incorporação de conhecimentos adquiridos e em caso de necessidade de toma de decisões, as mesmas são efetuadas de forma autónoma.

O Sistema ENI define uma arquitetura de Bloco funcional que visa solucionar a problemática da necessidade de interação humana em processos de orquestração. Esta arquitetura é constituída por um API Broker, que serve como *gateway* que traduz a informação entre diferentes sistemas e por três tipos de blocos funcionais, os quais devem evoluir no decorrer do tempo, conforme a aprendizagem que vão realizando [81]:

- Bloco de Processamento e Normalização de dados - Responsável pela recolha de dados de múltiplos fontes de entrada, processamento dos dados e a tradução dos mesmos para a posterior utilização dos dados por outros blocos.
- Bloco de geração de recomendações ou comandos - Responsável por processar e converter os dados recebidos para a posterior perceção pelo Sistema. A conversão inclui a definição de protocolos e a alteração de codificação de dados.
- Bloco de Análise de dados - Responsável pela compreensão dos dados e processamento de novos dados (isto permite que os dados sejam compreendidos, previstos e executados de forma a influenciar as decisões). Este bloco deve também ser capaz de descrever o estado e o ambiente do sistema.

Através destes três blocos, o sistema efetua a tomada de decisão, com base nas informações disponíveis e processadas por cada bloco, assim como na análise do ambiente. Quando a decisão é tomada após a compreensão dos dados, a sua implementação pode não ser imediata se assim não for pretendido, dependendo da ação a realizar.

2.5 Orquestração Inteligente

Atualmente, quando se fala na implementação de um sistema de uma organização deve ser tido em consideração que o mesmo usufrui de componentes físicos, componentes virtuais e baseados em nuvem, tanto pública como privada. Neste sentido, as organizações conseguem aproveitar os benefícios de cada um dos componentes, surgindo a necessidade de execução em apenas um ambiente, de forma a não causar complexidade no sistema. Além disto, de forma a aumentar a automação e consequentemente diminuir a interação humana nas diferentes atividades realizadas, será tida em atenção a abordagem Zero-touch, apresentada na Secção 2.4, que visa monitorizar o ambiente, avaliar as suas necessidades e executar as operações que se mostrarem necessárias, com base nas métricas recolhidas.

Em paralelo com a utilização de diferentes ambientes é importante existir uma c*loud framework* uma vez que há a necessidade de criação de novos serviços e da constante estabilidade do sistema. Desta forma, deve fornecer uma infraestrutura de fácil e ágil escalabilidade, garantir a disponibilidade e o rápido provisionamento do sistema para todos os intervenientes.

Os serviços XR encontram-se contidos no XR Service Deployment Plane, administrados por diferentes domínios, que podem pertencer à mesma entidade ou a diversas entidades, ou inclusive terem naturezas diferentes (por exemplo nem todos necessitam de ser

exclusivamente domínios em nuvem). Cada domínio necessita de uma instância NFV MANO (*framework* desenvolvida com o objetivo de garantir implementações de NFV rápidas e confiáveis em escala) de forma a gerir a infraestrutura virtualizada e a orquestração dos recursos exigidos pela mesma. Esta instância permite a interação de recursos e serviços com os restantes componentes, a gestão do ciclo de vida e dos *workloads* em *containers* (relativamente à sua implementação, monitorização e ciclo de vida).

Cada domínio é monitorizado por Domain-Specific XR Service Monitoring and Reaction Plane de forma a acompanhar o consumo de recursos utilizados, processar os dados monitorizados e realizar a análise dos mesmos para posteriormente tomar decisões e realizar as ações que forem necessárias. Desta forma este plano deve conter um componente que faça a recolha de dados de forma filtrada, reduzindo o tamanho dos dados para apenas o necessário, evitando a recolha e armazenamento de dados duplicados ou não relevantes. Além disto, este plano deve seguir um *closed-loop* de forma a providenciar automação a este processo.

O ponto de entrada dos fornecedores e desenvolvedores dos serviços é o XR Service E2E Conducting Plane, responsável por gerir todo o ciclo de vida dos seus serviços, desde o registo das informações dos recursos e esquemas genéricos dos mesmos, como tal é capaz de receber uma solicitação de um fornecedor e/ou desenvolvedor e traduzir a mesma num projeto, selecionando os domínios onde os serviços serão alojados. Desta forma também é importante implementar um *closed-loop* e um CI/CD *pipeline* de forma a automatizar não só o processo de recomposição dos serviços, como automatizar a construção, implementação e monitorização das aplicações, assim, o *pipeline* permite a integração de um serviço fornecido com os restantes já existentes na plataforma.

O Domain-Specific XR Service Monitoring and Reaction Plane e o XR Service E2E Conducting Plane devem conter o componente Integration Fabric, existente também entre estes dois domínios, de forma a permitir a interoperabilidade das diversas comunicações entre funções. Este componente, tal como representado na Figura 2.8, deve conter também serviços de registo e de descoberta para adição dos serviços a um catálogo, para posterior descoberta e invocação.

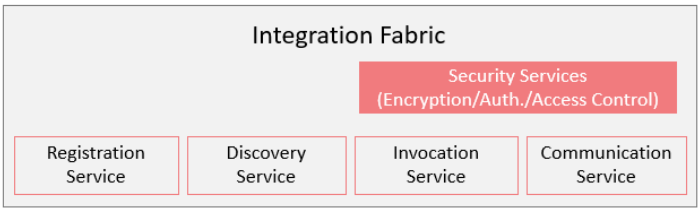

Figura 2.8- Componentes de Integration Fabric, adaptado de [82]

Tendo esta arquitetura em consideração, numa primeira fase foi realizado o estudo de componentes responsáveis por fornecer infraestruturas como serviço que permitem a criação e gestão de nuvens públicas e privadas. Numa segunda parte é importante haver uma orquestração entre nuvens, de forma a sincronizar todas as nuvens e possibilitar a sua adição ou remoção, bem como o acesso remoto a cada uma e a sua exposição. Posteriormente, foi analisada a possibilidade de gestão de cluster de *containers* de forma automatizada, que permita o escalonamento e conectividade entre as diversas máquinas virtuais de uma forma inteligente. Este estudo e análise foi realizado na Secção 2.3.

2.6 Projetos Europeus Relacionados

No âmbito do tópico de orquestração entre domínios pode ser destacado o projeto 5GEx, o qual possui o principal objetivo de permitir a orquestração entre domínios de serviços em várias administrações ou em administrações únicas de vários domínios [83]. Desta forma, foi desenvolvida uma plataforma com foco em casos de uso e cenários 5G que visa orquestrar infraestruturas como serviço de ponta a ponta em múltiplas operadoras.

Com o mesmo objetivo, foi desenvolvido o projeto 5G-TRANSFORMER, o qual visa uma implementação de orquestração entre múltiplos domínios, com recursos de escalonamento de serviços e aprimoramento do suporte para criação de novas instâncias em tempo de execução, de forma a não haver tempo de indisponibilidade dos serviços [84]. Além disto, a implementação visa também providenciar mecanismos de monitorização e gestão de políticas.

Os dois projetos mencionados são focados na orquestração de diversos domínios distintos, de forma a permitir a execução de atividades CRUD nos diversos. Estes domínios podem ser implementados com a ferramenta OpenStack, a qual é suportada em ambos os projetos.

No âmbito do tópico de Orquestração Inteligente existem diversos projetos que foram analisados como ponto de partida, como o projeto ANASTACIA, cujo principal objetivo é abordar questões de segurança cibernética desde o *design* dos sistemas baseados em IoT e arquiteturas de nuvem, através do desenvolvimento de uma plataforma que aborde todas as fases do ciclo de vida de desenvolvimento dos projetos e que seja capaz de tomar decisões automáticas [85]. Estas decisões são tomadas através do uso de tecnologias de rede como SDN, NFV, aplicações de segurança inteligentes e ferramentas de monitorização. A arquitetura que será utilizada no decorrer deste trabalho difere deste projeto, essencialmente no facto de, no projeto ANASTACIA apenas há a preocupação com um único domínio, e neste trabalho vão ser implementados diversos domínios, os quais devem ser geridos e mantidos.

Relativamente à orquestração de clusters de *containers* pode ser destacado o projeto COLA onde o resultado passa por uma *framework* para *containers* em Kubernetes, de forma a automatizar o processo de escalonamento e configurações de segurança, através de regras que são analisadas pela *framework* MiCADOscale e posteriormente se houver necessidade de realizar alguma operação, é enviada essa informação ao ambiente Kubernetes [86]. Este projeto tem foco na automação de Kubernetes, o que difere do trabalho proposto uma vez que o mesmo tem o objetivo de automação não só de redes orquestradas como de redes entre domínios.

No sentido de usufruir da arquitetura Zero-touch, o projeto INSPIRE-5Gplus pode ser destacado, uma vez que fornece proteção de gestão de infraestruturas de redes 5G em diversos domínios de forma inteligente [87], através do alinhamento com os princípios chave desta arquitetura.

Destaca-se ainda o projeto MonB5G, focado na orquestração Zero-touch em divisões de rede de grande escala, através da divisão do sistema de gestão centralizado em diversos subsistemas [88], de forma a distribuir a inteligência de tomada de decisões.

O projeto 5GZORRO tem o objetivo primário de definir uma arquitetura de nível de sistema que combine soluções de automação Zero-touch e tecnologias distribuídas para permitir uma combinação de múltiplas partes interessadas e composição de recursos e serviços em redes

5G [89], tendo uma arquitetura bastante semelhante à proposta deste trabalho, com a relevância também na automação de domínios distintos.

Tanto no projeto INSPIRE-5Gplus, como no projeto MonB5G e no projeto 5GZORRO a arquitetura que é utilizada no seu decorrer possui semelhanças com o trabalho, nomeadamente no facto de terem o seu principal foco na Inteligência e Automação, no entanto não se preocupam com a integração de serviços XR, o que se pode rever em latências mais elevadas do que aquelas que seriam consideradas suportadas pelos utilizadores.

2.7 Síntese de Capítulo

Atualmente, quando se fala na implementação de um sistema deve ser tido em conta que o mesmo usufrui de componentes baseados em nuvem, físicos e virtuais. Neste sentido, surge a necessidade de implementação destes componentes em apenas um ambiente, de forma a simplificar a gestão do sistema, evitando características de complexidade. Desta forma, a implementação de diferentes domínios pode ser realizada com o auxílio de ferramentas, como as abordadas na Secção 2.3.1, onde a Tabela 2.1 sumariza os estudos efetuados sobre as mesmas e as respetivas conclusões.

Tabela 2.1- Descrição dos estudos relacionados com a Infraestruturas de Computação em Nuvem

Artigo	Ferramentas abordadas	Conclusões
[30]	OpenStack	Ferramenta modular e com diversos projetos robustos disponíveis que podem ser implementados conforme as necessidades.
[31]	OpenStack	Ferramenta utiliza os recursos existentes para uma implementação modular, além de ser totalmente *open-source* e com suporte a padrões como OpenFlow.
[38]	Elastic Compute Cloud (EC2)	Plataforma para implementação de aplicações de fluxo de trabalho, apesar de a aquisição de recursos para execução de tarefas ainda ser um ponto que necessita de melhorias.
[39]	Elastic Compute Cloud (EC2)	Plataforma com falhas a nível de desempenho em caso de implementação em *data center*.

No entanto surge a questão de orquestração de múltiplos domínios, uma vez que podem ser bastante diferentes entre si, além disto, orquestrar todos os domínios de forma manual e de forma individual é um processo demorado e que pode ser melhorado em caso de implementação de ferramentas que permitam a sua orquestração de forma unificada, tal

como abordado na Secção 2.3.2, onde foram analisados os estudos apresentados na Tabela 2.2.

Tabela 2.2- Descrição dos estudos relacionados com a Orquestração entre Domínios

Artigo	Ferramentas abordadas	Conclusões
[43]	OpenStack Cloudify	Apresentação de bases de automação e gestão de cluster Cloudify, bem como fornecimento de mecanismos de recolha de dados de forma a avaliar o seu desempenho. Implementação compatível com OpenStack.
[48]	OpenStack Ansible ManageIQ	ManageIQ fornece flexibilidade para incluir novas ações de orquestração, através dos *playbooks* Ansible para adicionar/executar novas configurações ou implementações.
[51]	Amazon Ansible	Flexibilidade da biblioteca facilita a automação de todo o procedimento de implementação de aplicações.
[52]	Ansible	Permite uma gestão remota de forma segura e uma configuração de redes mais eficiente e prática.
[55]	Libcloud	Aumenta a flexibilidade associada aos recursos do ambiente e de reduzir os custos associados à gestão das infraestruturas.
[36]	OpenStack OpenNebula	OpenStack é uma solução para implementações complexas e robustas. OpenNebula é mais centralizado, sendo mais indicado para *data centers*, instituições públicas e privadas, sendo uma boa opção para orquestração entre nuvens.
[57]	OpenStack OpenNebula	Adequado para lidar com grandes quantidades de dados e para aproveitar as infraestruturas de TI existentes.

De forma a tornar mais eficiente a orquestração entre nuvens e o seu uso de forma a permitir operações mais automatizadas nos diversos domínios, a sua inserção em clusters (que podem ser geridos através de ferramentas como as referidas na Secção 2.3.2) pode ser uma solução válida, uma vez que, traz benefícios a nível de fluidez na gestão de aplicações devido à capacidade de transferência e execução de código de aplicações em diversos ambientes distintos, tal como referido ao longo deste capítulo. A síntese dos estudos analisados pode ser observada na Tabela 2.3, onde os mesmos são descritos.

Tabela 2.3- Descrição dos estudos relacionados com a Orquestração de cluster de containers

Artigo	Ferramentas abordadas	Conclusões
[64]	KubeFed	Ferramenta ainda em fase de produção e testes, no entanto, o seu uso releva-se vantajoso para quando estiver operacional.
[65]	KubeFed	Utilização como ponto de partida para um controlo descentralizado, onde o servidor Kubernetes API padrão é utilizado para oferecer o suporte para a gestão de recursos.

[66]	KubeFed	Atrasos e falhas temporárias de rede podem levar à instabilidade de aplicações.
[68]	OpenStack ONAP multi-K8s	Ferramenta para criação de ambiente em nuvem privada com funções de rede virtualizada.
[69]	Rancher	Viabilidade do uso da pilha Rancher.

Na Secção 2.4 foi discutida a temática da Orquestração Automatizada, onde foram estudadas abordagens importantes como ponto de vista inicial, que tornam percetivel a definição de possíveis implementações de funções de orquestração e de como pode ser feita a comunicação entre os diversos componentes. A especificação Zero-touch abordada permite que seja analisada uma abordagem de implementação de diferentes ambientes, onde há uma enorme escala de sistemas envolvidos e onde se torna importante mecanismos que facilitem o processo de orquestração, desenvolvimento e manutenção dos mesmos.

Tendo como base os tópicos mencionados e os respetivos projetos, pode ser analisada a Tabela 2.4, de forma a sumarizar os pontos que devem ser tidos em consideração, e os pontos distintos comparativamente com o trabalho a realizar.

Tabela 2.4- Descrição dos projetos relacionados e respetivas distinções para a arquitetura proposta

Projeto Europeu	**Pontos Semelhantes**	**Pontos distintos**
5GEx [83]	Focado em orquestração multi-domínio, de forma a realizar atividades CRUD nos diversos domínios. Suporte para provedores de infraestrutura em nuvem.	Não se foca na Orquestração Inteligente.
5G-TRANSFORMER [84]	Focado em orquestração multi-domínio, de forma a realizar atividades CRUD nos diversos domínios. Suporte para provedores de infraestrutura em nuvem.	Não se foca na Orquestração Inteligente.
ANASTACIA [85]	Utilização e suporte a provedores de infraestrutura em nuvem. Arquitetura dividida em diversos planos, tendo atenção às decisões autónomas e inteligentes.	Apenas se preocupa com um único domínio. O principal foco passa pelas questões de segurança dos serviços, e não pela orquestração dos mesmos.
COLA [86]	Estrutura multifuncional de orquestração e escalonamento automático para implementações do Kubernetes.	Foco na automação de Kubernetes, a nível de máquina virtual e de containers. A questão de domínios não é tida em consideração.

INSPIRE-5Gplus [87]	Foco na Automação e Inteligência. Arquitetura baseada em Zero-touch que permite a orquestração de vários domínios, tendo em conta a automação através de *closed-loops*.	Não se preocupa com a integração de serviços XR, como tal, não tem em vista os requisitos destes serviços.
MonB5G [88]	Foco na Automação e Inteligência, através do uso de *closed-loops*. Utilização e suporte de provedores de infraestrutura em nuvem. Implementação de domínios independentes com funções de orquestração de serviços por domínio e entidades de orquestração de serviços ponto a ponto.	Não se preocupa com a integração de serviços XR, como tal, não tem em vista os requisitos destes serviços.
5GZORRO [89]	Foco na Automação e Inteligência. Utilização e suporte de provedores de infraestrutura em nuvem. Serviços de múltiplas partes interessadas e baseados na arquitetura Zero-touch. Múltiplos domínios, geridos através de serviços *Cross-Domain*.	Não se preocupa com a integração de serviços XR, como tal, não tem em vista os requisitos destes serviços.

Os projetos mencionados são importantes como enquadramento das soluções já existentes, no entanto, o principal ponto em que diferem dos requisitos deste trabalho, provém da integração de serviços XR, uma vez que estes serviços trazem requisitos como o caso de baixa latência, requisitos estes que também implicam uma maior preocupação no ambiente. No entanto, estes projetos abordam conceitos e arquiteturas como Zero-touch, que é também uma arquitetura base para o trabalho a ser desenvolvido, e desta forma fornecem bases para o estudo.

Da perspetiva da estrutura geral do documento, o capítulo que agora finda permite a contextualização do leitor perante a problemática da ocorrência da orquestração de clusters de aplicações, de forma a diminuir a necessidade de intervenção humana para a configuração de serviços, visto que acarreta custos, tempo e possíveis erros, que são proporcionais à escala dos sistemas envolvidos. Tendo isto em consideração, a proposta de implementação será apresentada no Capítulo 3.

Capítulo 3 Proposta de Implementação

Neste capítulo é abordado uma proposta de implementação, tendo como referência o projeto CHARITY, uma vez que este trabalho surge como resolução de problemas referenciados no mesmo, tendo em conta as suas características e requisitos.

Esta proposta de implementação é constituída pela definição de requisitos e pela definição e esquematização da arquitetura a desenvolver, respetivamente na Secção 3.4 e Secção 3.5. Este capítulo torna-se bastante importante uma vez que permite delinear o planeamento de trabalhos e a estrutura do documento, tendo em conta os requisitos da arquitetura.

3.1 Metodologia

A elaboração deste trabalho será possível devido à metodologia usada para a elaboração deste relatório. Primeiramente, é necessário ser realizado o enquadramento do trabalho, através de uma análise do projeto CHARITY, o qual será apresentado na Secção 3.2. Este trabalho surge como necessidade para a sua implementação da arquitetura definida no CHARITY, é neste projeto que se torna necessário a implementação de mecanismos que permitam a interação entre múltiplos domínios, e que os mesmos possam ser orquestrados por um orquestrador Inteligente. Neste sentido, é importante referir que o trabalho a desenvolver será realizado em conjunto com o trabalho dos parceiros, uma vez que este trabalho não se foca essencialmente na implementação do ciclo de recolha de métricas e tomada de decisões, mas sim com a interação do orquestrador com os diversos domínios existentes no ambiente.

Seguidamente, é fundamental que sejam analisados os casos de uso do trabalho, os quais são definidos na Secção 3.3. Da mesma forma, os requisitos são também definidos, de forma a serem integrados na arquitetura proposta. Os requisitos analisados são mencionados na Secção 3.4, onde são apresentados os requisitos genéricos da implementação a desenvolver, como a necessidade de o sistema ser simples, interativo e ter em consideração a possibilidade de inserção de novos serviços, ou a eliminação dos mesmos. Além disto, requisitos como a necessidade de implementação de domínios em nuvem, ou a implementação de mecanismos que permitam a interação entre o orquestrador inteligente e os diversos domínios, também são apresentados.

Com base nos requisitos apresentados, é proposta uma arquitetura, detalhada na Secção 3.5, onde os diversos componentes e serviços são mencionados, bem como os testes a serem realizados no decorrer do trabalho. Desta forma, é também referido como será feito a implementação dos diversos componentes e qual a vantagem da sua implementação. Esta arquitetura será implementada, testada e posteriormente avaliada.

Com base nesta metodologia, foi também delineado o planeamento de tarefas, de forma a organizar e prever o trabalho a ser realizado. Desta forma, a visualização das tarefas a realizar

será mais concisa e permitirá uma maior facilidade na definição de tempo, para o trabalho corresponder ao objetivo final do mesmo.

3.2 Projeto CHARITY

Este trabalho surge no âmbito do projeto CHARITY, de forma a solucionar a necessidade de criação de múltiplos domínios (neste caso, serão implementados como provedores de nuvem), a orquestração dos mesmos de forma unificada e a implementação de mecanismos ou APIs que serão expostas e que permitam a sua chamada para a realização de atividades inteligentes aos diversos domínios, conforme a ação a executar. Tendo em consideração que os parceiros irão implementar o algoritmo do orquestrador inteligente, o qual irá recolher as métricas necessárias e implementar a inteligência necessária para a realização de tomada de decisões. Desta forma, algumas das tarefas deste trabalho podem depender das implementações realizadas pelos parceiros.

Esta interação entre a tomada de decisão e a execução da mesma, é realizada através da chamada da API, que será um dos principais objetivos no decorrer deste trabalho. Um exemplo muito prático da interação entre a orquestração multi-domínios e o orquestrador passa por: Se o orquestrador inteligente tiver um pedido de implementação de um serviço, vai analisar os diversos domínios de forma a perceber as capacidades de cada um e analisar a compatibilidade com o serviço. Conforme o domínio escolhido, vai chamar a API necessária, a qual irá realizar os comandos e processos necessários para a interação com o domínio especificado.

O projeto CHARITY ("*Cloud or Holography and Augmented RealITY*") é um projeto financiado por European Union's Horizon 2020, que surge com o objetivo de usufruir dos benefícios da orquestração autónoma e inteligente da nuvem, *edge* e recursos de rede, para a implementação de aplicações emergentes em infraestruturas que cumpram os requisitos. Desta forma, podem ser destacados os seguintes pontos como requisitos para a implementação:

- Autonomia na orquestração de recursos de rede, computação e armazenamento em infraestruturas – Deve ser desenvolvida uma computação de alto nível e um orquestrador de rede que permita a realização de operações auto gerenciadas e otimizadas, de acordo com o desempenho dos diversos serviços e componentes, com a sua disponibilidade e a sua privacidade.
- Suporte para a orquestração de soluções de media avançadas – Deve suportar diversos dispositivos finais (uma vez que os acessos podem ser efetuados de múltiplos dispositivos distintos), tecnologias para (des)computação de nuvem e recursos de armazenamento inteligente (com adaptação baseada nas necessidades das aplicações e na capacidade geral da infraestrutura).
- Gestão de ciclo de vida de aplicações ponta a ponta – Deve suportar e combinar múltiplas instâncias de *framework* de computação de ponta, as quais devem ser geridas de forma inteligente e autónoma.
- Desenvolver aplicações e serviços altamente interativos e colaborativos – Deve suportar aplicações holográficas em tempo real, treino virtual imersivo e aplicações interativas de realidade mista.

Tendo como ponto de vista estes requisitos, foi desenvolvida uma arquitetura, representada na Figura 3.1, dividida em cinco camadas distintas, a camada inferior (constituída por dispositivos de utilizador final e várias nuvens), a camada de Função de Rede, a camada Network Slicing, a camada superior (dividida em Application Management Framework e Convergence & Abstraction Layer) e, por último, a camada de Segurança de rede e privacidade do utilizador (que abrange todas as restantes camadas). A camada de Função de Rede e dos componentes CI/CD permite a gestão de escalonamento, tanto vertical como horizontal, e a realocação de serviços sempre que for necessário. A camada Network Slicing visa a gestão de aplicações colaborativas e interativas, com base em processos inteligentes, de orquestração e recolha de dados. Application Management Framework e Convergence & Abstraction Layer permite o suporte a operações, a orquestração de rede multi-domínio e o isolamento e coordenação das diversas redes.

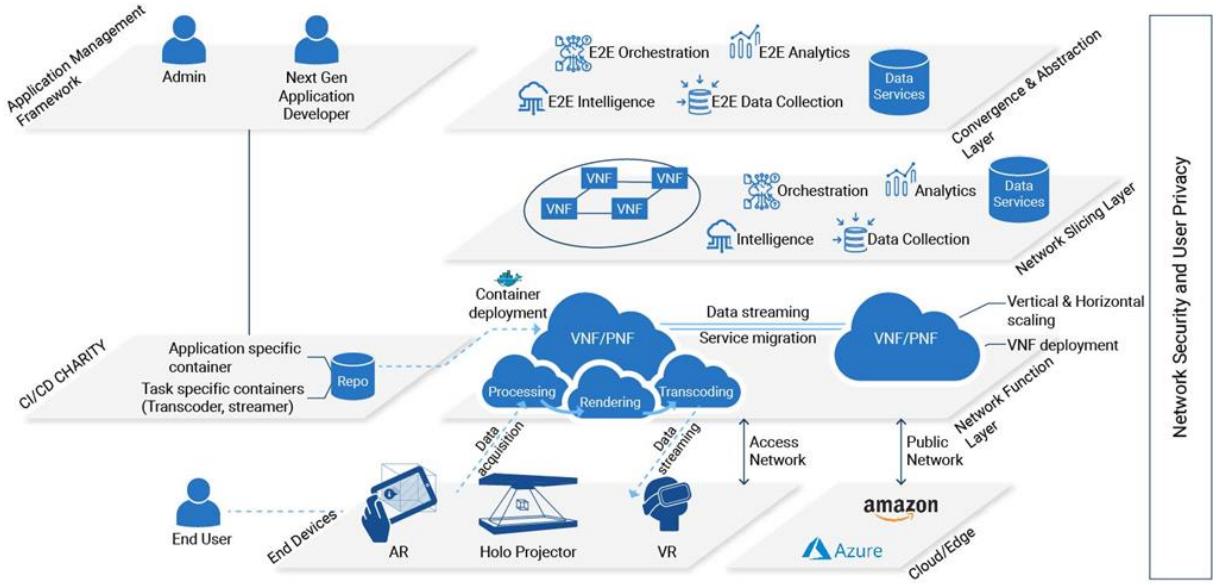

Figura 3.1- Arquitetura geral do CHARITY [3]

Numa primeira fase, neste projeto surge a necessidade de criar múltiplos domínios, domínios estes que devem ser geridos, de forma a tornar possível atividades de escalonamento e realocação de serviços no próprio domínio. Os domínios devem ser confiáveis e apenas devem estabelecer ligações com destinatários que mantenham uma relação de confiança. Isto permite proteger contra a inserção de domínios maliciosos ou de utilizadores que tentam estabelecer ligações não necessárias. Numa segunda fase, deverá ser implementado um mecanismo que permita a orquestração de múltiplos domínios de uma forma unificada. Desta forma, o mecanismo de orquestração deve permitir uma maior automação de processos de escalonamento e provisionamento, independentemente do domínio.

Posteriormente, serão implementados mecanismos de orquestração inteligente que permitem a tomada de decisões com base nas métricas recolhidas dos múltiplos domínios. Como tal, após a tomada de decisões é necessário executar as mesmas (estas ações podem passar pela criação de máquinas num domínio, implementação de serviços ou modificação de recursos num domínio, por exemplo).

3.3 Casos de Uso

De forma a contribuir para a implementação que irá ser realizada, bem como os testes que serão efetuados à solução, foram definidos casos de uso. Os casos de uso definidos consistem na execução de diversas atividades que deveram ser executadas de forma inteligente, e como tal, devem ser implementados na solução a desenvolver para posterior integração com os mecanismos de orquestração inteligente.

Tabela 3.1- Descrição dos Caso de Uso 1 - Provisionamento

Número de Caso de Uso	[CS1]
Nome do Caso de Uso	Provisionamento
Descrição	Criação de novas máquinas num determinado domínio

Tabela 3.2- Descrição dos Caso de Uso 2 - Escalonamento

Número de Caso de Uso	[CS2]
Nome do Caso de Uso	Escalonamento
Descrição	Aumento das capacidades de memória e processamento de uma determinada máquina

Tabela 3.3- Descrição dos Caso de Uso 3 - Implementação de serviços (1)

Número de Caso de Uso	[CS3]
Nome do Caso de Uso	Implementação de serviços (1)
Descrição	Implementação de serviços numa determinada máquina

Tabela 3.4- Descrição dos Caso de Uso 4 - Implementação de serviços (2)

Número de Caso de Uso	[CS4]
Nome do Caso de Uso	Implementação de serviços (2)
Descrição	Implementação de serviços num conjunto de máquinas

Tabela 3.5- Descrição dos Caso de Uso 5 - Realocação de serviços (1)

Número de Caso de Uso	[CS5]
Nome do Caso de Uso	Realocação de serviços (1)
Descrição	Migração de serviços entre máquinas do mesmo domínio

Tabela 3.6- Descrição dos Caso de Uso 6 - Realocação de serviços (2)

Número de Caso de Uso	[CS6]
Nome do Caso de Uso	Realocação de serviços (2)
Descrição	Migração de serviços entre máquinas de diferentes domínios

Tabela 3.7- Descrição dos Caso de Uso 7 – Backup (1)

Número de Caso de Uso	[CS7]
Nome do Caso de Uso	Backup (1)
Descrição	Efetuação de Backup de um determinado serviço

Tabela 3.8- Descrição dos Caso de Uso 8 – Backup (2)

Número de Caso de Uso	[CS8]
Nome do Caso de Uso	Backup (2)
Descrição	Efetuação de Backup do conteúdo de uma determinada máquina

Tabela 3.9- Descrição dos Caso de Uso 9 – Backup (3)

Número de Caso de Uso	[CS9]
Nome do Caso de Uso	Backup (3)
Descrição	Efetuação de Backup do conteúdo de um determinado domínio

Tabela 3.10- Descrição dos Caso de Uso 10 – Restauro (1)

Número de Caso de Uso	[CS10]
Nome do Caso de Uso	Restauro (1)
Descrição	Efetuação de Restauro de um determinado serviço

Tabela 3.11- Descrição dos Caso de Uso 11 – Restauro (2)

Número de Caso de Uso	[CS11]
Nome do Caso de Uso	Restauro (2)
Descrição	Efetuação de Restauro do conteúdo de uma determinada máquina

Tabela 3.12- Descrição dos Caso de Uso 12 – Restauro (3)

Número de Caso de Uso	[CS12]
Nome do Caso de Uso	Restauro (3)
Descrição	Efetuação de Restauro do conteúdo de um determinado domínio

É importante referir que a decisão dos casos de uso definidos está diretamente relacionada com o projeto CHARITY, e como tal, esta listagem pode sofrer alterações ao longo do desenvolvimento do projeto, de acordo com os requisitos que se revelem necessários.

3.4 Requisitos da Implementação

Uma abordagem que permita uma orquestração inteligente e que permita resolver a problemática de orquestração de múltiplos domínios necessita de ter algumas características em consideração, como a possibilidade de ocorrência de falhas, o que torna necessária ter algumas considerações. Além disto, devem ser tidos em conta requisitos relativamente à automação e criação de novos serviços. Neste sentido, os requisitos não funcionais da implementação podem ser visualizados na Tabela 3.13.

Tabela 3.13- Descrição dos requisitos não funcionais do trabalho

ID	Requisito	Prioridade
[R_NF1]	Resiliência da arquitetura, de forma a não haver a indisponibilidade dos serviços	Alta

[R_NF2]	Capacidade de retorno à operação normal depois da resolução de problemas de degradação no menor tempo possível	Alta
[R_NF3]	Possibilidade de novos serviços e capacidades, como tal a arquitetura deve ser extensível e escalável	Alta
[R_NF4]	Arquitetura modular (de forma a evitar o acoplamento forte e monolítico) e orientada a modelos (facilitando processos de portabilidade, reutilização e gestão)	Alta
[R_NF5]	Suporte para funções de gestão *stateless*	Média
[R_NF6]	Automação, através de processos de *closed-loop*	Alta
[R_NF7]	Distinção de dois tipos de gestão, de domínio e de serviços ponta a ponta (abrangendo gestão de múltiplos domínios)	Média

Tendo estes requisitos não funcionais como base, os requisitos iniciais pelos quais será regulada a proposta de implementação centram-se nos observados na Tabela 3.14, onde é possível também analisar o seu ID e a sua prioridade.

Tabela 3.14- Descrição dos requisitos funcionais do trabalho

ID	Requisito	Prioridade
[R1]	Implementação de domínios em nuvem, através da criação de máquinas virtuais com a instalação de provedores de nuvem	Alta
[R2]	Implementação de ferramenta de orquestração de multi-domínios que permita a orquestração eficiente de múltiplos domínios de forma unificada	Alta
[R3]	Criação de máquinas virtuais dentro de um domínio específico, análise do estado do mesmo, atualização ou eliminação (atividades CRUD), através de acesso direto ao provedor de nuvem	Média
[R4]	Implementação do ambiente em *containers* Kubernetes, representando clusters, através de uma ferramenta de orquestração de clusters	Alta
[R5]	Implementação de uma plataforma que permita a orquestração de diversos clusters Kubernetes de forma unificada	Baixa
[R6]	Implementação de mecanismos que permitam a criação de novas máquinas virtuais num domínio específico	Média
[R7]	Implementação de mecanismos que permitam a eliminação de máquinas virtuais num domínio específico	Média

[R8]	Implementação de mecanismos que permitam a atualização de recursos de uma máquina virtual num domínio específico	Média
[R9]	Implementação de mecanismos que permitam a migração de máquinas virtuais entre domínios	Média
[R10]	Implementação de mecanismos que permitam o isolamento de cada domínio	Média
[R11]	Implementação de mecanismos que permitam a verificação automatizada da relação de confiança entre domínios	Média
[R12]	Implementação de mecanismos que permitam a criação automatizada de políticas de segurança de um domínio específico	Média
[R13]	Implementação de mecanismos que permitam a atualização/alteração das políticas de segurança definidas para um domínio específico	Média
[R14]	Implementação de mecanismos que permitam a inicialização de *containers* num cluster específico	Média
[R15]	Implementação de mecanismos que permitam a paragem e eliminação de *containers* num cluster específico	Média
[R16]	Implementação de mecanismos que permitam a atualização de *containers* num cluster específico	Média
[R17]	Implementação de mecanismos que permitam a migração de *containers* entre nós de um mesmo cluster	Média
[R18]	Implementação de mecanismos que permitam a migração de *containers* entre clusters	Média
[R19]	Implementação de mecanismos que permitam a abstração de gestão de *containers* integrados em múltiplos clusters em simultâneo	Média
[R20]	Implementação de políticas de segurança que permitam limitar o acesso de cada *container* a informações e/ou serviços	Média
[R21]	Implementação de políticas de segurança que permitam limitar o acesso de cada *container* a recursos (como por exemplo, o uso de memória)	Média
[R22]	Implementação de políticas de segurança que permitam proteger o acesso a cada nó do cluster e à API	Média
[R23]	Implementação de mecanismos que permitam a atualização automatizada da versão das ferramentas utilizadas	Média

[R24]	Implementação de API para interação do orquestrador inteligente com a ferramenta de orquestração multi-domínios, de forma a facilitar a execução dos mecanismos implementados	Baixa
[R25]	Criação de novo serviço num cluster existente (escolha de cluster, criação e implementação do serviço no cluster)	Baixa
[R26]	Criação de novo serviço num novo cluster (criação de cluster, criação e implementação do serviço no cluster)	Baixa
[R27]	Implementação de mecanismos que permitam monitorizar o cluster, os seus nós e serviços relativamente a recursos físicos	Baixa
[R28]	Implementação de mecanismos que permitam monitorizar o cluster, os seus nós e serviços relativamente a tráfego de rede	Baixa
[R29]	Implementação de mecanismos que permitam monitorizar os serviços do cluster relativamente ao seu *deployment time*	Baixa
[R30]	Implementação e validação de integração de clusters Kubernetes em domínios em nuvem	Média

3.5 Arquitetura Proposta

Tendo em conta os requisitos referidos na Secção 3.4, ao longo deste trabalho será desenvolvida uma arquitetura onde serão implementados domínios distintos (conforme o requisito [R1]). Estes domínios serão implementados recorrendo a provedores de nuvem (por exemplo o OpenStack) que fornecem recursos de infraestruturas, permitem a criação e orquestração de nuvens públicas e privadas através do uso de recursos virtuais agrupados. Neste seguimento, surge a necessidade de analisar e testar as possibilidades de criação de instâncias dentro dos domínios implementados, bem como a sua eliminação e alteração de recursos.

A existência de múltiplos domínios, acarreta a necessidade da sua orquestração, de forma a tornar possível os acessos às diversas nuvens, eliminação ou adição das mesmas de uma forma unificada (conforme o requisito [R2] e [R3]). Neste sentido, será implementada uma ferramenta que permita esta mesma orquestração, de forma a sempre que se tornar necessário executar alguma ação em domínios, a mesma seja o ponto de interação entre os utilizadores e o respetivo domínio, de uma forma simples e eficiente. Além disto, esta ferramenta irá facilitar a monitorização dos diversos ambientes, através da recolha de métricas e da posterior análise por parte do orquestrador inteligente na tomada de decisões. A orquestração entre diferentes domínios é um ponto fulcral a ter em consideração, uma vez que a necessidade de orquestração de múltiplos domínios cada vez mais se torna essencial, devido ao aumento da utilização de diferentes domínios e da integração dos mesmos entre empresas, por exemplo. Este aspeto não é tido em consideração em múltiplos projetos, alguns dos quais mencionados na Secção 2.6, como ANASTACIA e COLA.

De forma a usufruir das vantagens de orquestração, nomeadamente de capacidades de escalabilidade, flexibilidade e automação, é importante que os diversos domínios sejam integrados em clusters (conforme o requisito [R4]). A orquestração de clusters deve abranger diversos domínios existentes, de forma a permitir uma maior facilidade, por exemplo em caso de lançamento de um novo serviço, onde podem ser analisados os vários domínios de forma a perceber qual poderá ter as necessidades requeridas.

A orquestração deve ser o mais inteligente possível, característica que distingue esta arquitetura das propostas nos projetos 5GEx e 5G-TRANSFORMER, mencionados na Secção 2.6. Neste sentido, este é um ponto de distinção destes projetos e que permite reduzir a necessidade da interação humana nas atividades que podem ser necessários, como por exemplo a nível de escalonamento ou provisionamento, reduzindo assim custos, tempo e possíveis erros humanos. Como tal a orquestração de domínios e *containers* deve ser capaz de permitir a recolha de métricas, as quais serão analisadas de forma a serem tomadas decisões pelo orquestrador inteligente. Estas decisões irão ser executadas com base em diversos mecanismos que serão implementados, conforme as necessidades dos envolventes (conforme o requisito [R6] a [R9] e [R14] a [R18]). Estes mecanismos devem permitir atividades como:

- Criação de novas instâncias em domínio específico;
- Eliminação de instâncias em domínio específico;
- Atualização de recursos de uma máquina virtual em domínio específico;
- Migração de instâncias entre domínios;
- Inicialização de *containers* em domínio específico;
- Paragem e eliminação de *containers* em domínio específico;
- Atualização de *containers* em domínio específico;
- Migração de *containers* entre domínios;
- Abstração de gestão de *containers* quando integrados em múltiplos domínios.

Uma vez que o ambiente irá possuir múltiplos domínios, é importante ter em atenção os aspetos de segurança, uma vez que é necessário que os domínios sejam isolados (conforme o requisito [R10]), isto significa que cada domínio deve ter a sua própria rede e que o seu acesso seja controlado, deve haver políticas de segurança que não permitam acessos não necessários ao interior da rede do domínio. Além disto, é também importante que cada máquina dentro dos diferentes domínios tenha CPU dedicado, de forma que em caso de algum erro a este nível, o mesmo não prejudique as restantes máquinas do domínio.

Os domínios devem ser confiáveis, de forma a não ser possível uma interação com um domínio que não tenha essa característica. Desta forma, é importante que seja realizada uma verificação da relação de confiança entre domínios antes de ser efetuada alguma ação (conforme o requisito [R11]). Cada domínio, independentemente do provedor, deverá ter as suas políticas de segurança definidas e deverá ser considerado confiável para puder interagir com o restante ambiente.

Além disto, devem ser também definidas as políticas de segurança para cada domínio (conforme o requisito [R12]), conforme as suas necessidades, bem como a possível alteração das mesmas (conforme o requisito [R13]), uma vez que estas podem necessitar de ser alteradas no decorrer do tempo. Da mesma forma, devem ser implementadas políticas de segurança que permitam limitar o acesso dos diversos *containers* a dados e/ou informações, bem como os acessos de recursos (conforme o requisito [R20] a [R22]). Relativamente à

segurança de cada versão de ferramenta utilizada, quer seja o provedor, como Kubernetes, devem ser implementados mecanismos que permitam a sua atualização automatizada (conforme o requisito [R23]), de forma a evitar que sejam exploradas vulnerabilidades de versões descontinuadas.

De forma a permitir analisar as necessidades do ambiente e agir em conformidade com as mesmas, deve ser implementado um Domain-Specific XR Service Monitoring and Reaction Plane, tal como representado na Figura 3.2, constituído por três componentes principais. Estes componentes, designados por Analytics Engine, Decision Engine e Actuation Engine permitem analisar os dados e produzir alertas em caso de necessidade, usar o resultado para a tomada de decisões (por exemplo em caso de necessidade de escalonamento ou de adaptação de recursos), e por fim, decidir como fazer essa implementação, ou seja, analisar a decisão que foi gerada e executar as ações necessárias.

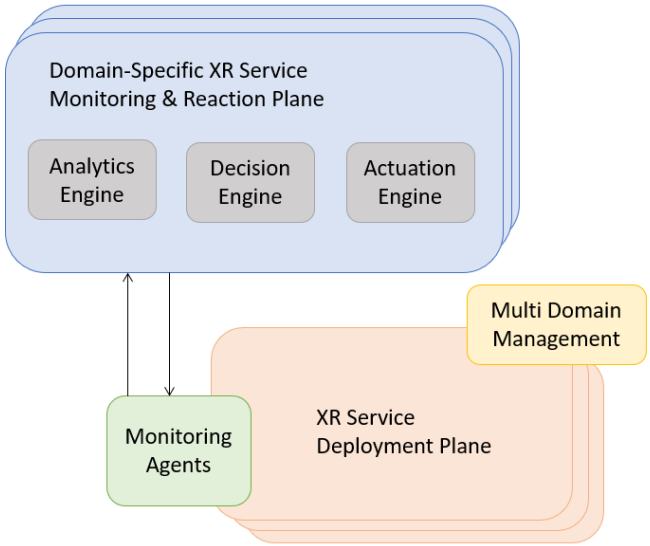

Figura 3.2- Integração entre os diversos domínios e o closed-loop

Neste sentido, é importante que o próprio mecanismo de *closed-loop* tenha em atenção a integração com serviços XR, de forma a não incrementar a latência nos diversos processos. Esta necessidade torna esta arquitetura distinta de alguns projetos mencionados na Secção 2.6, como INSPIRE-5Gplus, MonB5G e 5GZORRO.

A integração dos componentes mencionadas é efetuada através dos agentes de monitorização que são implementados nos diversos domínios (conforme o requisito [R27] a [R29]), e que comunicam com o algoritmo para a orquestração inteligente que será implementado por parceiros. O algoritmo irá analisar as métricas, fazer a sua filtragem e posteriormente irá tomar decisões e executar as mesmas através da chamada de API (conforme o requisito [R24]) que será implementada no decorrer do trabalho de forma a executar os mecanismos implementados e a permitir uma integração entre os múltiplos domínios e as atividades de orquestração inteligente. Nesta API apesar de facilitar a execução dos diversos mecanismos de forma automatizada, devem ser primeiramente registados os

domínios do ambiente de forma manual. Este algoritmo irá ser implementado como um *closed-loop*, o qual pode ser visualizado pela Figura 3.3.

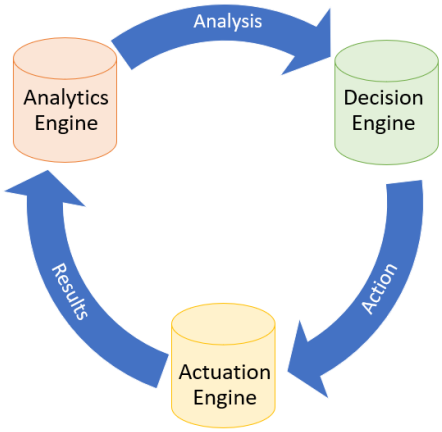

Figura 3.3- Esquema de closed-loop

Tendo em consideração a existência de diversos clusters surge a necessidade da orquestração dos mesmos (conforme o requisito [R5]), a gestão abstrata de múltiplos clusters (conforme o requisito [R19]), bem como a necessidade de configuração de características de segurança. Esta orquestração deve facilitar o processo de implementação de novos clusters, a sua orquestração e o lançamento de novos serviços, permitindo a sua inserção num cluster existente tendo em conta as capacidades e necessidades do mesmo (conforme o requisito [R25]) ou a criação de um cluster novo para a inserção do serviço (conforme o requisito [R26]). É importante referir que os clusters podem estar integrados em máquinas de um único domínio, ou em máquinas de diferentes de nuvem, e só nestes casos é que deve ser realizada uma comunicação entre provedores de nuvem, onde só devem ser aceites as comunicações estritamente necessárias. Assim, esta ferramenta deve comunicar com os provedores de nuvem de forma segura e apenas os utilizadores com permissões devem ser capazes de efetuar as atividades necessárias. No entanto, sempre que seja necessário efetuar alguma conexão entre estes dois componentes (quando for necessário implementar um novo cluster numa máquina de um domínio em específico), este processo deve ser efetuado de forma mais simples possível, embora manualmente. Após isto, todo o processo de orquestração poderá ser feito automaticamente, uma vez que a primeira relação já foi estabelecida.

Relativamente à segurança do ambiente é fundamental não ser esquecida, uma vez que estamos a falar de um ambiente com suporte a serviços XR, onde muitos dados são pessoais e sensíveis, nomeadamente, possíveis localizações e dados biométricos. Assim, é importante definir mecanismos que permitam a proteção destes mesmos dados, de forma a minimizar os riscos associados e aumentar a *unlinkability* e confidencialidade. Além disto, é importante também ter em consideração a sede das nuvens implementadas, uma vez que nem todos os países aplicam o RGPD, o regulamento do direito europeu sobre privacidade e proteção de dados pessoais. Da mesma forma, é importante haver mecanismos que permitem o tratamento do direito ao esquecimento, e como tal, deve ser possível analisar os locais de armazenamento de dados de utilizadores em todos os componentes. Em simultâneo, deve ser feita a monitorização e auditoria de forma regular para avaliar os mecanismos de segurança implementados e a sua eficiência.

No desenvolvimento do trabalho é importante que seja tida em consideração a disponibilidade dos componentes da arquitetura, uma vez que em caso de falha de algum

provedor de nuvem, todas as suas máquinas ficam indisponíveis e consequentemente todos os serviços implementados nas mesmas. A solução que visa solucionar este problema é a implementação de serviços em multi-domínios, assim, quando um domínio se encontrar indisponível, o serviço irá continuar acessível uma vez que está implementado em outro domínio em simultâneo. O mesmo acontece com os dados, ou seja, os mesmos devem ser armazenados em diferentes localizações (por exemplo: no caso de falarmos de uma aplicação multi-cluster, os seus dados podem ser armazenados em diversos clusters em simultâneo), de forma que em caso de indisponibilidade ou perda dos mesmos, seja possível aceder aos mesmos, através de outra localização. Da mesma forma, é necessário que as ferramentas de orquestração tenham esta mesma disponibilidade, uma vez que em caso de falha, apesar dos serviços continuarem disponíveis, a sua orquestração torna-se mais dificultada, e qualquer ação que seja necessária tem de ser realizada manualmente (situação que se quer evitar).

Além disto, no decorrer do trabalho será feita a integração de todos os mecanismos e a sua avaliação (conforme o requisito [R30]) de forma a permitir avaliar a performance das implementações realizadas. Algumas métricas serão tidas em consideração, por exemplo, o tempo de implementação e execução de cada mecanismo, de forma a perceber se a solução criada pode ser melhorada em função do tempo. Isto é importante uma vez que estes mecanismos devem ser executados em curtos intervalos de tempo, de forma a não atrasar os processos de automação.

Tendo em consideração a arquitetura proposta e os requisitos que a mesma vem a solucionar, é necessário ter em atenção parâmetros de QoS, nomeadamente valores de latência, velocidade das conexões envolventes e a perda de pacotes na rede. Estes valores tornam-se importantes, uma vez que a experiência do utilizador estará relacionada com a disponibilidade e latência nas respostas, uma vez que o ambiente visa fornecer suporte a serviços XR.

3.6 Síntese de Capítulo

O planeamento deste trabalho, é composto pela definição da metodologia aplicada, detalhada na Secção 3.1, a qual visa esclarecer o caminho traçado ao longo do estágio, de forma a ser alcançado o objetivo pretendido.

Primeiramente, o trabalho deve ser enquadrado, e como tal, surge a necessidade de descrever o projeto CHARITY, projeto do qual surge o trabalho a desenvolver. Neste sentido, na Secção 3.2 é detalhado os principais requisitos do ambiente, a sua arquitetura e a contextualização da necessidade da implementação que este trabalho visa responder.

Seguidamente, na Secção 3.4 são apresentados os requisitos da solução a desenvolver, quer sejam requisitos funcionais ou requisitos não funcionais. Estes requisitos são descritos e apresentados com base na sua prioridade no trabalho a desenvolver. Neste sentido, inicialmente torna-se necessária a implementação de mais do que um domínio para posteriormente implementar mecanismos que visam a sua orquestração de forma abstrata. O mesmo no caso da orquestração de containers, uma vez que estes podem ser implementados num único domínio ou em múltiplos domínios em simultâneo, característica que não deve ser um entrave para a sua orquestração. Assim, os requisitos visam não só implementar este ambiente, como testar mecanismos que permitem automatizar diversos processos (desde processos de escalonamento ou provisionamento, como relativos à segurança do ambiente).

Da mesma forma, na Secção 3.5 é feita a integração dos requisitos nos diversos componentes da arquitetura proposta, tendo em consideração a necessidade de múltiplos provedores de nuvem que representem os domínios em nuvem e a necessidade de um componente que permita a sua orquestração. Além disto, são também referidas as principais distinções da proposta descrita para as soluções descritas nos projetos europeus referidos na Secção 2.6. Nomeadamente a orquestração multi-domínio que é um dos principais objetivos da arquitetura proposta, a possibilidade de integração de serviços XR (os quais possuem requisitos próprios como a necessidade de baixa latência) e a preocupação com a inteligência e automação do sistema.

Assim, na perspetiva da estrutura geral do documento, o capítulo que agora finda permite a contextualização do trabalho, bem como permite perceber a solução proposta.

Capítulo 4 Prova de Conceito

Neste capítulo são abordadas as provas de conceito realizadas, tendo como referência os requisitos da implementação e a arquitetura proposta como solução para os mesmos. Desta forma, é apresentado o trabalho inicial, onde foram realizados alguns testes de forma a validar possíveis ferramentas, as suas características, a sua adaptação ao contexto e de que forma se pode beneficiar das mesmas na temática de orquestração inteligente.

Estas provas de conceito são constituídas por um trabalho exploratório inicial, onde foi feito o levantamento das ideias principais a ter em consideração, das necessidades do ambiente e das ferramentas que poderiam ser utilizadas. De seguida, são apresentadas as provas de conceito relativas a infraestruturas de computação em nuvem e a orquestração entre domínios, respetivamente na Secção 4.2 e Secção 4.3. Para finalizar, neste capítulo é apresentada uma conclusão dos resultados obtidos com as provas efetuadas, tendo em conta os requisitos que deverão ser tidos em consideração.

4.1 Trabalho Exploratório

Numa primeira fase foram estudados temas como Serviços XR, onde foram analisados os seus requisitos, tendo em consideração que o ambiente deverá prever a implementação destes serviços, que possuem requisitos de latência bastante definidos. Da mesma forma, e tendo em conta que o ambiente será constituido por diversos ambientes, bastante distintos entre si, e o conceito de orquestração possui inúmeras vantagens neste sentido, ao nível de gestão destas infraestruturas, foi feita uma investigação sobre este mesmo conceito. Dentro deste conceito foram destacados temas como microserviços e aplicações monolíticas, *containers*, *service mesh* e ambientes implementados em Kubernetes.

Desta forma, foram exploradas implementações em Docker e Kubernetes, onde foram realizadas atividades de implementação de clusters Docker em máquinas virtuais, e a criação e integração de um *container* DNS em ambiente Kubernetes, com o objetivo de fazer a tradução dos nomes para a correspondência dos mesmos aos respetivos certificados. Este trabalho reflete-se na integração em dois projetos europeus nos quais a empresa se incorpora, FUDGE-5G e 5G-EPICENTRE, e os quais tiveram como objetivo a exploração destes mecanismos. Assim, esta atividade permite a contextualização e a exploração das ferramentas, de forma a serem utilizadas no decorrer do trabalho.

Outro conceito bastante relevante e que teve atenção foi a especificação Zero-touch, que constitui um importante ponto de vista inicial, uma vez que apresenta possíveis implementações de funções de orquestração e de como pode ser feita a comunicação entre os diversos componentes. Desta forma, esta especificação permite que seja analisada uma abordagem de implementação de diferentes ambientes, onde se torna importante mecanismos que facilitem o processo de orquestração, de desenvolvimento e de manutenção dos mesmos.

Neste sentido, foi feita a investigação do conceito de *closed-loop*, o qual deve ser integrado com ferramentas de monitorização que permitam a recolha de métricas de diversos domínios.

Neste sentido e tendo em consideração os requisitos do sistema, também foram feitas análises à segurança de aplicações XR (no qual foram analisados temas como Zero-trust e Security as a Service) e o estudo de mecanismos de segurança de ambientes nativos em nuvem, uma vez que são pontos fulcrais que devem ser tidos em consideração ao longo do trabalho. Este trabalho refletiu-se na participação da escrita de documentação para o projeto CHARITY, nomeadamente o "Deliverable D2.1 - Edge and cloud infrastructure resource and computational continuum orchestration system", o qual ainda se encontra em desenvolvimento.

Ao longo da investigação dos diversos componentes e da integração entre os mesmos, inclusive da integração entre os múltiplos domínios, foram analisadas formas de comunicação entre os diversos componentes e de que forma seriam expostas as APIs necessárias para esta interação. Este trabalho refletiu-se na participação da escrita do "Deliverable D4.1 - Integration and Experimentation Plans", documentação para o projeto CHARITY, onde também foi feito um planeamento das fases seguintes do projeto, bem como a definição e descrição dos protótipos que serão implementados e avaliados.

Além disto, no decorrer da fase de implementações foi realizada a participação no *paper* "Towards Supporting XR Services: Architecture and Enablers", o qual visa projetar uma arquitetura com suporte a serviços XR, definindo algumas ferramentas a utilizar, as suas características e de que forma se poderá tirar o melhor partido das mesmas.

4.2 Infraestruturas de Computação em Nuvem

Nesta Secção será analisado e cumprido os objetivos da Tabela 4.1, os quais visam a implementação de domínios de nuvem através da instalação de OpenStack e posteriormente o teste e validação de criação de instâncias através do acesso a este provedor de nuvem.

Tabela 4.1- Descrição dos Requisitos Cumpridos na Secção 4.2

ID	Requisito	Descrição
[R1]	Implementação de domínios em nuvem, através da criação de máquinas virtuais com a instalação de provedores de nuvem	Implementação de domínios (representados por domínios OpenStack), tanto automaticamente como através de *playbooks*
[R3]	Criação de máquinas virtuais dentro de um domínio específico, análise do estado do mesmo, atualização ou eliminação (atividades CRUD), através de acesso direto ao provedor de nuvem	Criação e eliminação de instâncias dentro de um domínio específico, através de acesso direto ao provedor
[R10]	Implementação de mecanismos que permitam o isolamento de cada domínio	OpenStack permite a criação de instâncias com CPUs dedicados, com redes internas que permitem o isolamento das redes entre cada domínio

Para a criação de domínios distintos foram realizados testes de implementação ao OpenStack, o qual previa ser a opção a adotar. Normalmente a sua implementação envolve dois nós

distintos, sendo o primeiro um nó de Controlo (contendo o projeto Keystone, Glance, Neutron, Cinder e a base de dados) e o segundo um nó de Computação, que contém o componente Nova e o agente Neutron para gestão das máquinas virtuais e das configurações de rede das mesmas. No entanto, com o crescimento em escala da nuvem, um dos primeiros passos passa por ser a implementação do Neutron num nó dedicado, assim como o serviço Cinder que muitas vezes acaba por ser implementado em nós dedicados ao armazenamento.

Além disto, há dois pacotes que podem ser utilizados para a instalação e implementação desta plataforma de forma mais simples, como Packstack e Devstack. O primeiro usa módulos *puppet* para a instalação de diversos projetos OpenStack apenas em máquinas com Sistema Operativo RHEL ou CentOS. O segundo é um conjunto de *scripts* que permitem implementar OpenStack de forma bastante rápida. Esta instalação está disponível também para Ubuntu. Estes pacotes apesar de serem bastante úteis, uma vez que permitem a simplificação do processo de implementação e configuração do OpenStack, apenas se encontram disponíveis para alguns sistemas operativos.

Como alternativa, foi estudada a implementação de MicroStack (referência ao requisito [R1]), o qual é um *snap* de OpenStack, que integra serviços e bibliotecas de suporte originados no OpenStack (como tal, possui funcionalidades e comandos do OpenStack).

Esta ferramenta possui as funcionalidades e comandos do OpenStack e como tal é vista como uma implementação OpenStack. No entanto, oferece suporte à implementação de forma mais leve, o que a torna uma ferramenta mais utilizada para execução de protótipos e testes, mas também adequada para *edge*, IoT e dispositivos, devido aos seus requisitos mais leves, uma vez que engloba todos os serviços OpenStack e as respetivas bibliotecas de suporte num único pacote.

Neste sentido, e de forma a testar esta implementação, que representa o *deployment* de um domínio em nuvem, foram efetuados testes, relativamente à criação e acesso de instâncias, de forma a perceber como é que esta ação pode ser implementada posteriormente, através da plataforma de orquestração entre domínios. Assim, após a análise da *dashboard* do OpenStack, foram efetuadas algumas atividades, como:

- Criação de um novo *flavor*, que permite a definição da capacidade de computação, memória e armazenamento das instâncias;
- Criação de uma nova imagem, uma vez que por defeito, a única imagem que se encontra integrada é Cirros, uma distribuição Linux mínima projetada para uso como imagem de teste;
- Criação de novos pares de Chaves, que permitem o acesso SSH, por exemplo, através das mesmas;
- Criação e Acesso a Instâncias, através da *dashboard* do OpenStack, bem como através de acesso SSH (referência ao requsito [R3]).

Através da análise e dos testes efetuados, foi comprovado que MicroStack prevê ser uma alternativa para o uso de OpenStack, uma vez que utiliza as mesmas bibliotecas e serviços, como tal, tanto as suas funcionalidades, comandos e *dashboard* são iguais, tal como pode ser comprovado pela Figura 4.1. A maior distinção entre estas ferramentas prevalece no número de serviços que são disponibilizados, uma vez que esta ferramenta ainda se encontra num

estado inicial e como tal possui apenas alguns serviços (Glance, Horizon, Keystone, Neutron, Nova e Cinder), no entanto, estes são os serviços mais fulcrais para o objetivo pretendido.

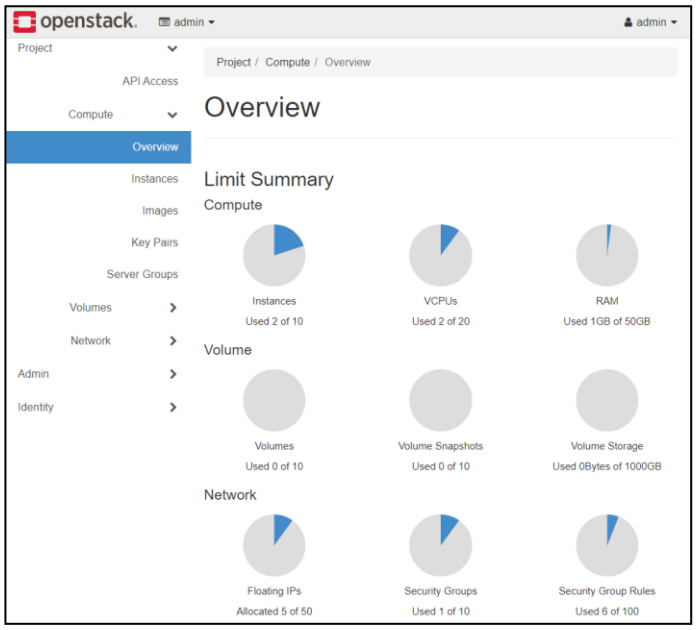

Figura 4.1- Dashboard Inicial do OpenStack

Apesar disto, é de referir que o Microstack ainda se encontra em desenvolvimento, e como tal, pode revelar-se bastante problemática pelos *bugs* e pela possível falta de documentação, apesar dos fóruns e da disponibilidade dos desenvolvedores em responder a qualquer e eventuais questões. Como tal, em caso de necessidade de alternativa a esta implementação, será utilizado o Apache Cloudstack.

De forma que cada domínio seja o mais seguro possível, e consequentemente que cada máquina possua uma maior segurança, é fundamental ter alguns fatores em consideração, como o isolamento de cada domínio (referência ao requisito [R10]). Cada domínio OpenStack deve possuir uma rede própria onde as suas máquinas devem comunicar, no entanto, para comunicações externas que possam ser necessárias deve ser utilizado *floating* IPs. O uso de *floating* IPs pode, inclusivamente, ser útil para casos de *failover* ou de atualização de serviços, uma vez que torna possível que outra máquina assuma este mesmo endereço. Além disto, é importante que os acessos à rede sejam controlados, através da definição de políticas de segurança que devem ser definidas para apenas aceitar a entrada de pacotes que sejam necessários.

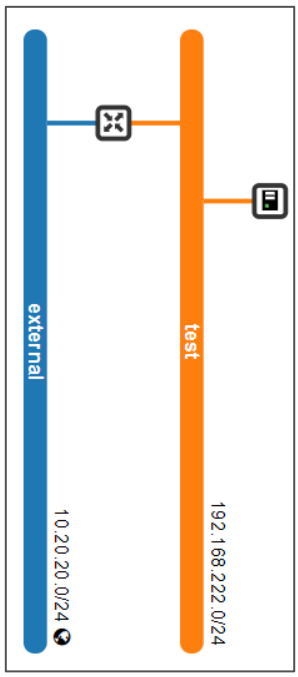

Figura 4.2- Topologia de rede do OpenStack

De forma a isolar possíveis problemas ao nível de CPU de cada máquina, as mesmas devem possuir CPUs dedicados, para que não seja partilhado qualquer erro que possa existir. Para a utilização de CPU dedicada é importante que cada *flavor* seja privado e apenas possa ser utilizado pelos domínios especificados na criação do mesmo, onde deve também ser introduzido o parâmetro *"hw:cpu_policy='dedicated'"* como propriedade do mesmo, para que seja feito o isolamento ao nível de CPU.

4.3 Orquestração entre Domínios

Nesta Secção será analisado e cumprido os objetivos da Tabela 4.2, os quais visam a implementação de uma ferramenta que permita a orquestração de múltiplos domínios de uma forma eficiente e simplificada.

Tabela 4.2- Descrição dos Requisitos Cumpridos na Secção 4.3

ID	Requisito	Descrição
[R2]	Implementação de ferramenta de orquestração de multi-domínios que permita a orquestração eficiente de múltiplos domínios de forma unificada	Implementação de Ansible

As ferramentas de orquestração multi-domínios são apresentadas na Secção 2.3.2, onde foram analisadas de forma a ser entendido o seu funcionamento e as suas vantagens, uma vez que será necessário implementar uma ferramenta que permita efetuar a orquestração entre diferentes domínios de uma forma unificada. Desta forma, foi efetuada uma análise e implementação de Cloudify, ManageIQ e Ansible como trabalho exploratório. Assim, foi feita a integração com a ferramenta OpenStack, através da definição de dois provedores na plataforma implementada e da posterior criação de múltiplas instâncias em domínios distintos. Esta implementação foi efetuada através do Docker, de forma a facilitar posteriormente o seu desenvolvimento em clusters Kubernetes, uma vez que Kubernetes permite a integração de *containers* Docker.

Desta forma, Cloudify apresenta algumas características que prevalecem mais vantajosas a nível de orquestração de *containers*, nomeadamente o facto de utilizar modelos *blueprint que* se encontram no formato YAML, que mesmo apesar de necessitarem de ser escritos para diversas soluções são bastante acessíveis e permitem que sejam implementadas soluções de maior dimensão através da repetição de implementações de projetos menores.

A ferramenta ManageIQ revelou-se bastante acessível e de fácil implementação e utilização, sem a necessidade de escrita de modelos ou de regras. Esta ferramenta também possui suporte para bastantes plataformas usuais, como Kubernetes, Ansible Tower, OpenStack, Google Cloud e Amazon Web Services, o que a torna uma opção bastante acessível.

Por fim, Ansible foi também analisado e revelou-se uma ferramenta bastante acessível, apesar de não manter o estado dos nós e não ser baseada na arquitetura Cliente/Servidor. Como tal, requer a necessidade de conexões abertas para cada nó de forma a implementar mudanças nos mesmos, o que por si só pode constituir um problema relativamente a possíveis pontos de entrada de ataques. No entanto, esta ferramenta revelou-se de fácil implementação e apesar de ser necessário a escrita de *playbooks* para a execução das diversas tarefas, a sua escrita é bastante simplificada com os modelos existentes. Os *playbooks* são escritos em linguagem YAML, nos quais é referenciado as máquinas para atuação e as diversas tarefas que se deseja executar. Na execução das tarefas, a ferramenta mostra a visualização da sua execução, permitindo perceber facilmente se alguma tarefa ficou pendente e qual foi o motivo. Além disto, para a escrita destes *playbooks*, existem dois modelos bastante úteis na gestão de provedores de nuvem, inclusive o modelo "*openstack.cloud.server*" e "*amazon.aws.ec2*" que permitem a escrita de tarefas personalizadas para provedores OpenStack e Amazon EC2, respetivamente.

De forma a analisar e explorar a ferramenta Ansible (referência ao requisito [R2]) primeiramente foi feito o estudo dos ficheiros de configurações, de forma a entender o seu funcionamento e a sua integração com diversas máquinas, tanto individualmente, como em conjunto, de forma a representarem um grupo específico. Além disto, foi estudada a linguagem dos *playbooks* desta ferramenta e foram efetuadas alguns *playbooks*, de forma a automatizarem tarefas, tais como:

- Visualização da *password* que permite o acesso á *dashboard* do OpenStack;
- Visualização de dados que permitem monitorizar as diversas máquinas, como o uso de memória, disco e CPU;
- Visualização de informações relativas aos diversos componentes do OpenStack (ex: versão do OpenStack, listagem das imagens disponíveis, listagem de instâncias);
- Alteração de algum dos componentes do OpenStack, desde a sua adição, eliminação ou alteração, conforme o requirido;
- Lançamento de novas instâncias, tendo em consideração as imagens que se encontram disponíveis no OpenStack;
- Criação de um novo Servidor OpenStack, onde apenas é necessário ter uma máquina disponível com os recursos necessários.

Neste sentido, na Figura 4.3 pode ser visualizado o *output* em caso de lançamento de uma instância com imagem "Cirros", a qual já se encontra integrada no OpenStack por padrão, como tal, apenas é efetuado o seu lançamento, sem a necessidade de criação de uma nova imagem.

```
root@template-debian-10:/etc/ansible# ansible-playbook -i hosts LaunchInstance.yml -u root -k
SSH password:

PLAY [Launch Instance Playbook - Starting Deploy] ******************************************

TASK [Gathering Facts] *********************************************************************
ok: [172.16.16.200]

TASK [Checking if is microstack] ***********************************************************
changed: [172.16.16.200]

TASK [Launch Cirros Instance] **************************************************************
changed: [172.16.16.200]

PLAY RECAP *********************************************************************************
172.16.16.200              : ok=3    changed=2    unreachable=0    failed=0
```

Figura 4.3- Lançamento de uma nova instância no OpenStack através do Ansible

Além disto, de forma a expor os mecanismos através de uma API foram analisadas possibilidades como Ansible Tower e AWX, de forma a perceber o seu uso e quais seriam as vantagens na sua utilização.

AWX é um Operador que fornece uma API, baseada nas necessidades de existência de um *frontend* em ambientes Ansible que permita a execução das diversas tarefas de uma forma simplificada. Com esta API, os mecanismos de orquestração multi-domínio tornam-se expostos para os processos de orquestração inteligente implementados pelos parceiros, conforme pretendido. Desta forma, quando o orquestrador inteligente pretende efetuar alguma ação com base na tomada de decisão efetuada, interage diretamente com esta API, a qual permite efetuar operações (como a migração de máquinas entre domínios ou a criação de máquinas virtuais) de uma forma mais autónoma, e sem a necessidade de acesso direto à ferramenta de orquestração multi-domínio e aos diversos domínios existentes no ambiente.

Esta API permite a integração com os *playbooks* criados de uma forma bastante eficiente, por exemplo, através de acesso manual (os *playbooks* são armazenados para uma diretoria pertencente ao AWX), acesso a arquivos remotos ou através do acesso a um repositório GitHub que possua os *playbooks* necessários. No caso foi feito o acesso através de um repositório GitHub privado, assim o Ansible não armazena localmente estes *playbooks*. De forma a sempre que for necessário correr alguma tarefa seja feita a verificação do repositório, é necessária a definição de credenciais de acesso ao repositório através de um *token* ou de credenciais de acesso ao mesmo.

Quando é feita a execução dos *playbooks*, o AWX permite a visualização dos estados de cada tarefa, de forma a perceber se todas as tarefas foram bem-sucedidas e em caso de falha, permite visualizar o motivo dessa falha. Esta API permite também a calendarização de *playbooks*, que pode ser muitas vezes necessária para validação dos recursos das diversas máquinas ou do posterior escalonamento e provisionamento das mesmas. Isto permite que uma tarefa possa ser realizada periodicamente, definindo o seu tempo de periocidade. Além disto, permite a criação e gestão de notificações, através de notificações de e-mail, de SMS ou por Slack. Assim, é possível configurar notificações que alertem sempre que um *playbook* inicie, quando for terminado com sucesso ou em caso de o mesmo ter terminado com alguma falha.

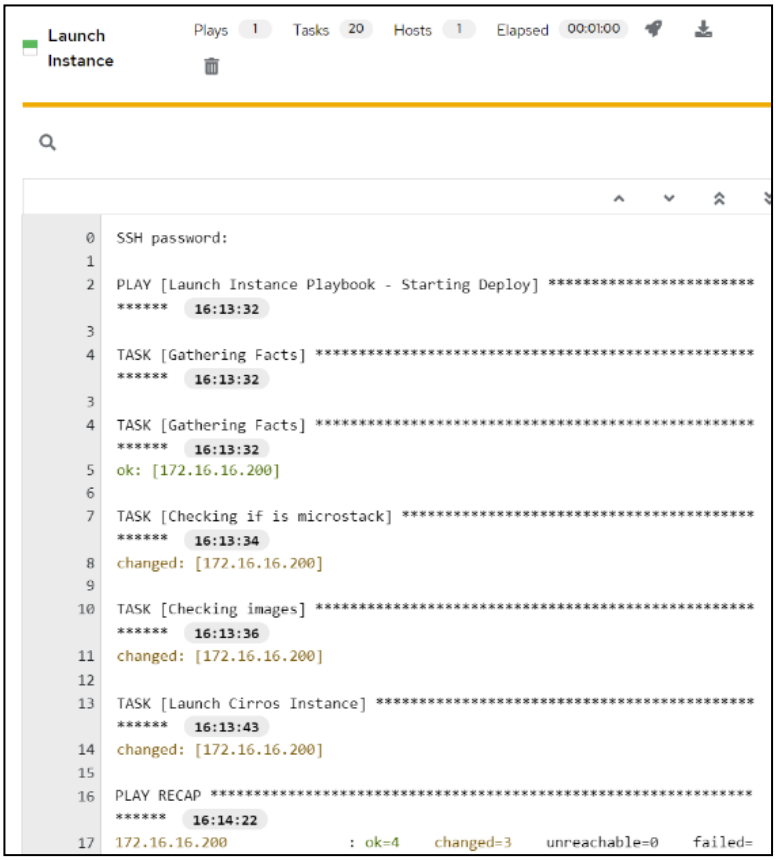

Figura 4.4- Execução de um playbook integrado no AWX

4.4 Síntese de Capítulo

Na Secção 4.1 é descrito o trabalho exploratório realizado, desde os temas que foram analisados e estudados, bem como as contribuições para projetos Europeus que foram efetuadas no longo desta fase de exploração.

Posteriormente, na Secção 4.2 e Secção 4.3 foram efetuadas provas de conceito sobre infraestruturas de computação em nuvem e orquestração entre domínios, respetivamente. Na primeira, foram exploradas as ferramentas que previam ser vantajosas para o seu uso como provedores de nuvem, onde o MicroStack preveu ser a opção a adotar. Neste sentido, foi realizada uma fase exploratória da ferramenta, bem como o teste de algumas atividades que serão necessárias no decorrer do trabalho, nomeadamente a criação e acesso a novas instâncias. Na segunda, foi realizada a exploração de diversas ferramentas de orquestração entre domínios, nomeadamente, Cloudify, ManageIQ e Ansible. Neste sentido, a ferramenta que demonstrou ser mais apelativa para o uso, principalmente pelo interesse de aprendizagem da mesma, foi Ansible. Desta forma, foi estudada a linguagem dos seus *playbooks* e a posterior implementação dos mesmos, de forma a testar a integração entre esta ferramenta e os provedores de nuvem. Além disto, na mesma secção foi explorada a API AWX, de forma a permitir uma maior abstração e eficiência na interação com os diversos domínios de nuvem.

Assim, da perspetiva da estrutura geral do documento, o capítulo que agora finda permite a contextualização das provas de conceito realizadas.

Capítulo 5 Orquestração multi-domínios

Neste capítulo é abordada a temática de orquestração multi-domínios, que permite a coordenação e gestão de todos os recursos que se encontram disponíveis em todos os domínios. Assim, o ponto de vista primordial passa pela diminuição da interação humana para processos de orquestração, e a automação destes mesmos processos. Tendo como referência os requisitos da implementação e a arquitetura proposta como solução para os mesmos este capítulo visa apresentar as abordagens e os mecanismos implementados.

Este capítulo é constituído por uma secção inicial onde é feita a apresentação da abordagem a utilizar, tendo em conta as ferramentas e de que forma é que as mesmas vão interagir entre si, e com os utilizadores. De seguida, na Secção 5.2 são apresentados os mecanismos de automação que foram implementados e os testes aos mesmos. Estes mecanismos estão inseridos num repositório GitHub, de forma a não serem localmente armazenados e a possibilitar o uso dos mesmos por qualquer interveniente.

Na Secção 5.2.1 são apresentados os testes funcionais efetuados aos mecanismos implementados, bem como é elaborada uma análise ao tempo de execução dos mesmos, de forma a avaliar se o tempo é constante ou se muda significativamente conforme o ambiente. Para finalizar, neste capítulo é realizada uma discussão dos mecanismos implementados e dos requisitos cumpridos, bem como uma conclusão dos resultados obtidos.

5.1 Abordagem Multi-domínio

Cada vez mais tem sido crescente a utilização e desenvolvimento de ambientes com múltiplos domínios, isto porque muitas vezes as empresas trabalham em comunicação com outras, ou pelo facto de usarem mais do que uma plataforma de virtualização ou provedores de nuvem. Neste sentido, a arquitetura implementada visa suportar esta diversidade no ambiente, e como tal, suportar a orquestração de múltiplos domínios distintos, localizados, possivelmente, em diferentes países.

Assim, como forma de representação de um provedor de nuvem foi selecionado o OpenStack, o qual visa permitir a criação e eliminação de diversas instâncias, as quais podem se orquestradas conforme as necessidades do ambiente.

Como forma de facilitar a orquestração de múltiplos domínios de nuvem (representados pelo OpenStack), é utilizada a ferramenta Ansible, em conjunto com AWX. A primeira permite a realização de múltiplas operações em OpenStack, de forma a facilitar a gestão do provedor, sem a necessidade de acesso direto ao mesmo. Esta gestão é efetuada através da escrita de *playbooks*. Em simultâneo, AWX permite que através dos *playbooks* escritos, os mesmos sejam geridos e executados através de uma API que permite uma orquestração mais eficiente e simplificada. A arquitetura descrita por estas ferramentas está representada na Figura 5.1, a qual demonstra que Ansible e AWX trabalham em conjunto para facilitar a orquestração de múltiplos domínios. Neste sentido, é possível, não só a eliminação e criação de um novo

domínio (através da criação ou eliminação de um provedor OpenStack), como é possível atividades CRUD em instâncias dentro de um domínio específico. É importante referir que, de momento, apenas serão implementados dois domínios distintos (representados por OpenStack).

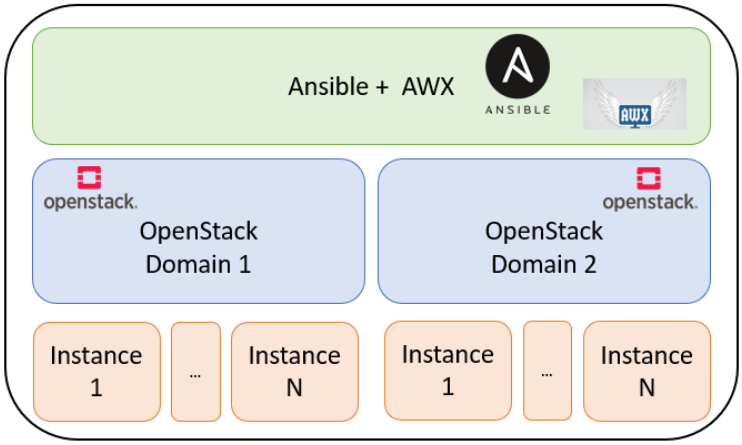

Figura 5.1- Representação da Arquitetura de Orquestração multi-domínios

AWX é um Operador Kubernetes, que permite a execução de diversas tarefas de forma transparente ao utilizador, através do uso de recursos personalizados que permitem gerir múltiplas aplicações e respetivos componentes.

Os diversos *playbooks* implementados não se encontram armazenados localmente no AWX, como tal, é necessária a conexão a um repositório git (ex: GitHub, GitLab) para acesso aos mesmos e posterior apresentação dos mesmos na API. Quando um utilizador deseja executar um dos *playbooks*, deve selecionar o mesmo na API e fazer o seu lançamento (passo 1). O AWX irá criar um operador (passo 2) responsável por executar o *playbook* selecionado e realizar o acesso à máquina onde se deseja executar o *playbook* (passo 3). Após a finalização das tarefas, o operador é terminado e removido. A representação dos diversos passos que constituem o processo de seleção e execução de um *playbook* no AWX estão representados na Figura 5.2.

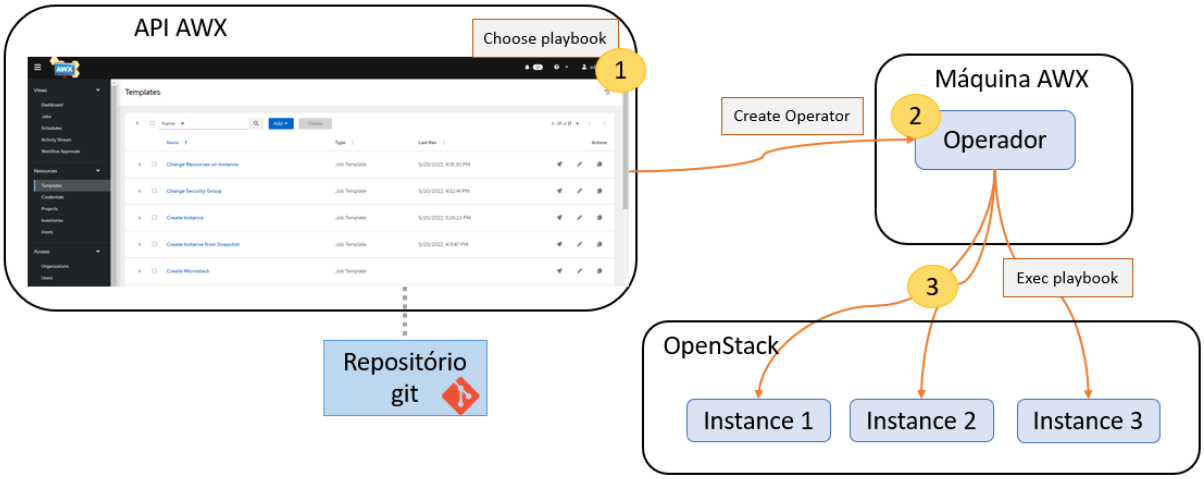

Figura 5.2- Representação dos passos para execução de um playbook

É de referir que é necessário que o servidor AWX tenha acesso a todas as máquinas necessárias para a execução das tarefas e que saiba as credenciais das mesmas.

5.2 Implementação de playbooks Ansible e Integração com AWX

Nesta secção será analisado e detalhado os *playbooks* da Tabela 5.1, os quais visam a implementação de mecanismos que permitam a automação de diversos processos referentes à orquestração multi-domínios, nomeadamente, atividades CRUD, atualização de versões das ferramentas e análise de utilização de recursos. Para cada *playbook* é também feita a referência ao requisito ao qual o *playbook* visa responder. No caso de não ser definido nenhum requisito, entende-se que o *playbook* foi implementado para facilitar processos internos.

Tabela 5.1- Descrição dos Requisitos Cumpridos no Capítulo 5

ID	Playbook	Descrição	Requisito
[P1]	"CreateInstance.yml"	Permite a criação de novas instâncias num domínio específico	[R6]
[P2]	"DeleteInstance.yml"	Permite a eliminação de instâncias existentes num domínio específico	[R7]
[P3]	"ChangeResources.yml"	Permite a edição de instâncias existentes (atualização de memória, disco e/ou vCPU) num domínio específico	[R8]
[P4]	"CreateSnapshot.yml"	Permite a criação de *snapshot* de uma instância existente, e posterior criação e instância através do mesmo	[R9]
[P5]	"CreateInstancefromSnapshot.yml"		
[P6]	"VerifyConfidence.yml"	Validação do certificado existente na máquina AWX para acesso à *dashboard* do domínio através de SSL	[R11]
[P7]	"CreateSecurityGroup.yml"	Permite a criação de um grupo de segurança e possível adição de regras	[R12]
[P8]	"ChangeSecurityGroup.yml"	Permite a modificação de regras de um grupo de segurança	[R13]
[P9]	"UpdateMachine.yml"	Permite a atualização das máquinas, atualização da ferramenta Rancher	[R23]
[P10]	"UpdateRancher.yml"		
[P11]	"CreateMicrostack.yml"	Permite a instalação de Microstack na máquina especificada	-

[P12]	"RestartMicrostack.yml"	Permite o reinício de Microstack na máquina especificada	-
[P13]	"RemoveMicrostack.yml"	Permite a remoção da instalação do Microstack	-

De forma à API AWX (referência ao requisito [R24]) puder ter acesso aos *playbooks* criados, os mesmos podem ser geridos manualmente ou através de um repositório GitHub. O primeiro caso, necessita que os ficheiros sejam armazenados numa subpasta do projeto AWX. No segundo caso, evita-se o armazenamento local dos ficheiros, e o acesso aos mesmos pode ser realizado através de um repositório GitHub, o qual pode ser público ou privado (no caso do repositório ser privado é necessário que seja criado um *token* para acesso ao mesmo).

Na implementação realizada, o AWX tem acesso a um repositório privado, onde estão todos os *playbooks* necessários. Assim, podem ser criados *jobs* através dos mesmos, onde é selecionado o *host* onde se quer executar as tarefas, e onde podem ser inicializadas variáveis que sejam necessárias para a execução do mesmo.

Neste sentido, e tal como referido na Secção 4.3, foram inicialmente realizados alguns testes ao funcionamento desta integração. Posteriormente, e de forma a implementar mecanismos de orquestração multi-domínios foram criados *playbooks* com o objetivo de automatizar e facilitar as atividades que possam ser necessárias nos diversos ambientes.

De forma a simplificar estes processos foi criado um mecanismo que permite recolher informações sobre o estado da máquina selecionada. Tal como demonstrado na Figura 5.3, este mecanismo permite observar a utilização de memória, disco e CPU. Assim, este mecanismo pode ser utilizado para avaliar o uso de recursos de uma máquina, de forma a perceber se a mesma está a fazer uma utilização correta dos mesmos.

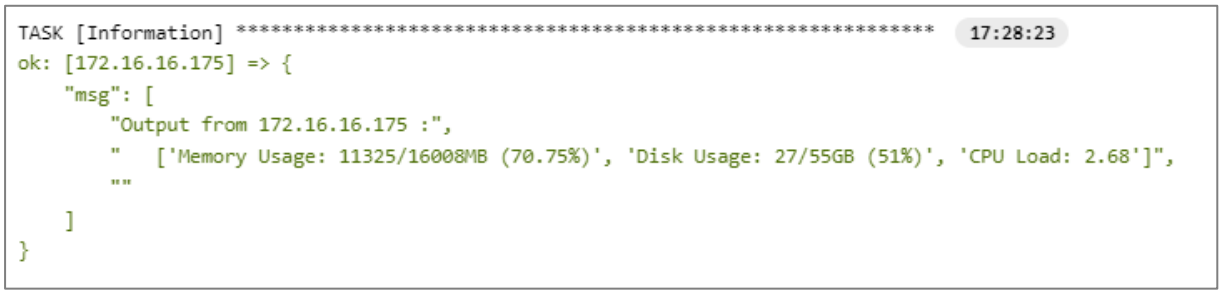

Figura 5.3- Output do playbook de recolha de informações sobre os recursos da máquina

Posteriormente, e de forma a simplificar a implementação de novos provedores de nuvem (que no caso, assume-se como instâncias OpenStack), foram criados mecanismos que permitem a inicialização de um novo provedor de nuvem, o seu reinício em caso de necessidade e a eliminação do mesmo. Assim, de forma a ser inicializado um novo provedor de nuvem apenas é necessário que a máquina seja previamente criada. No caso de eliminação ou reinício da ferramenta, apenas é necessário que esteja previamente instalada a mesma na máquina onde será executado o mecanismo. O reinício e a eliminação de provedores de nuvem são mecanismos que podem ser necessários em caso de algum incorreto funcionamento da ferramenta.

Da mesma forma, foram criados mecanismos que permitem realizar atividades CRUD em domínios de nuvem, nomeadamente a criação de instâncias (onde apenas é necessário definir

o nome da instância a criar, bem como a imagem e os recursos que irá utilizar - CPU, disco e memória), tal como representado na Figura 5.4 (referência ao requisito [R6]). Este mecanismo permite que esta ação seja executada, mesmo em casos de não haver a definição prévia de redes do provedor de nuvem, de *Security Groups*, de chaves de acesso ou de *Floating* IPs disponíveis. Assim, é realizada a verificação das definições destes componentes, e em caso de não ser listado algum componente, é feito automaticamente a sua definição e integração no ambiente.

```
TASK [Information] ********************************************************************  10:16:11
ok: [172.16.16.175] => {
    "msg": [
        "The Instance node-test was created",
        "     - State: ACTIVE ",
        "     - IPs: test=192.168.222.109, 10.20.20.77",
        ""
    ]
}
```

Figura 5.4- Output do playbook de criação de uma nova instância

Em simultâneo, é possível eliminar uma instância existente apenas indicando o nome da mesma (referência ao requisito [R7]), bem como no caso de necessidade de alteração de recursos de uma máquina (por exemplo: pode ser necessário aumentar os recursos de uma instância) (referência ao requisito [R8]). É de referir que nesta execução é realizada a verificação do nome da instância inicialmente, de forma a perceber se a instância realmente existe. A Figura 5.5 demonstra o *output* da execução do *playbook* referente à eliminação de uma instância existente.

```
TASK [debug] ******************************************************************************  11:11:32
ok: [172.16.16.175] => {
    "msg": "The Instance node-test was deleted"
}
```

Figura 5.5- Output de playbook de eliminação de instância

Além disto, é importante que sejam implementados mecanismos que permitam a migração de uma instância para outro provedor de nuvem (referência ao requisito [R9]), uma vez que no caso de um provedor de nuvem estar danificado ou necessitar de ser eliminado do ambiente é importante que as máquinas que possuam tarefas ou serviços, sejam mantidos no ambiente, apesar de ser alterada a sua localização. Esta migração pode ser feita através de uma migração *live* entre duas zonas de disponibilidade do mesmo provedor de nuvem, ou entre provedores de nuvem distintos através da criação de um *snapshot* da instância. Para a criação do *snapshot*, a máquina deve ser parada, e como tal esta não é uma migração *live*. Após isto, o *snapshot* irá ser guardado num ficheiro que deverá ser transferido para o outro provedor de nuvem e posteriormente, ser feita a inicialização de uma nova máquina através deste mesmo ficheiro. Durante a execução é realizada a verificação da existência da instância com o nome especificado, de forma a tornar o processo transparente. O processo de criação de instância através do *snapshot* comporta-se de forma semelhante à criação por Sistema Operativo, na medida em que as validações de redes do provedor de nuvem, de *Security Groups*, de chaves de acesso ou de *Floating* IPs disponíveis, são realizadas inicialmente e alteradas em caso de necessidade. A execução do *playbook* para a criação da nova instância

através do *snapshot* tem o *output* representado na Figura 5.6, onde é verificado o seu estado e os seus endereços IPs.

```
TASK [Information] ************************************************************ 11:41:23
ok: [172.16.16.175] => {
    "msg": [
        "The Instance is in ACTIVE state and has the following IPs:",
        "   test=192.168.222.203, 10.20.20.237",
        ""
    ]
}
```

Figura 5.6- Output do playbook de inicialização de uma instância através de Snapshot

Com o objetivo de facilitar os processos descritos, foi criado um mecanismo que permite verificar no nome da instância através do IP da mesma. Isto porque os mecanismos implementados necessitam, grande parte, do nome da instância, e em alguns casos será mais fácil saber o IP da mesma. Na Figura 5.7 é demonstrada a execução deste mecanismo, o qual apresenta o nome da instância que tem como endereço IP o definido no *playbook*.

```
TASK [debug] ******************************************************************* 14:08:34
ok: [172.16.16.175] => {
    "msg": " name of the Instance with IP 10.20.20.20 is node01"
}
```

Figura 5.7- Output do playbook de verificação de nome da instância através do endereço IP

Além disto, e de forma a introduzir mecanismos que possam ser necessários para garantir uma maior segurança no ambiente, foi criado um *playbook* para atualização das máquinas do ambiente, assim como para atualização das ferramentas utilizadas (referência ao requisito [R23]). Assim, para a execução deste mecanismo apenas é necessário definir os *hosts* nos quais deve ser feita esta mesma atualização. Acrescendo que este mecanismo foi calendarizado para ser executado semanalmente, assim, é garantido que as máquinas possuem o *software* atualizado.

De forma a permitir a definição de políticas de segurança para as diversas máquinas criadas dentro de um domínio específico foram criados mecanismos que permitem essa mesma execução. Assim, é possível a criação de uma nova política de segurança (onde apenas é necessário definir o nome da mesma), sendo que a sua criação apenas pode ser realizada se já não existir nenhuma com o nome definido (referência ao requisito [R12]). Além disto, a criação da *Security Group* pode ser executada, independentemente de ser definida alguma descrição para a mesma ou alguma regra.

Da mesma forma é permitida a modificação de políticas já existentes, conforme demonstrado na Figura 5.8, permitindo a eliminação de uma regra existente na política definida, ou a criação de uma nova regra. Desta forma, para a modificação de políticas existentes (referência ao requisito [R13]), é necessário explicitar o nome da mesma e a regra a adicionar/eliminar. Relativamente à ação a executar, o mecanismo permite que a mesma seja definida, ou que

em caso de isso não acontecer, seja verificada se a regra declarada já existe. Em caso afirmativo a regra é eliminada, em caso negativo, a regra é adicionada.

```
TASK [Modify Security Group] ************************************************ 17:16:41
changed: [172.16.16.175]

TASK [Show Security Group] ************************************************** 17:16:47
changed: [172.16.16.175]

TASK [debug] **************************************************************** 17:16:52
ok: [172.16.16.175] => {
    "msg": [
        "+--------------------------------------+-------------+-----------+-----------+------------+-----------------------+",
        "| ID                                   | IP Protocol | Ethertype | IP Range  | Port Range | Remote Security Group |",
        "+--------------------------------------+-------------+-----------+-----------+------------+-----------------------+",
        "| 7ab35b2c-24e7-48fb-a42e-2b889df41dba | None        | IPv4      | 0.0.0.0/0 |            | None                  |",
        "| 8f1ed03d-4694-4d72-9ba1-4b3900bc5bd0 | None        | IPv4      | 0.0.0.0/0 |            | None                  |",
        "| 8f679660-fd2d-4932-a366-90f70a83dfad | None        | IPv6      | ::/0      |            | None                  |",
        "| c1f55be4-39d5-412e-afb9-f21aa03f686e | tcp         | IPv4      | 0.0.0.0/0 | 22:22      | None                  …
```

Figura 5.8- Output de playbook de alteração de Security Group

Além disto, e de forma a garantir que todos os ambientes definidos e orquestrados são confiáveis (referência ao requisito [R11]) foi feita a partilha dos certificados de acesso às *dashboards* para a máquina onde foi instalado o AWX, assim é garantido que todos os domínios existentes, são conhecidos pelo administrador, o qual deve fazer a partilha dos certificados correspondentes. Assim, de forma a validar a confiança dos domínios deve ser validado se o certificado dos mesmos é existente na diretoria "*/etc/ssl/certs*" com o nome "*cert<IP>.pem*" (ex: o domínio orquestrado na máquina 172.16.16.175, possui o certificado "*cert172.16.16.175.pem*", o qual originalmente se encontra em "*/var/snap/microstack/common/etc/ssl/cert/cert.pem*"). Em caso afirmativo, podem ser realizadas as tarefas desejadas. Em caso negativo, é apresentada uma mensagem que informa que o domínio não é confiável, e como tal as tarefas não devem ser executadas, tal como representado na Figura 5.9. O caso negativo pode falhar por não existir o certificado em questão (assume-se que o administrador não sabia da existência deste domínio e como tal não foi efetuada a partilha de certificados) ou por o certificado ser falsificado e como tal não corresponder ao certificado do domínio correspondente.

```
TASK [debug] **************************************************************** 11:15:51
ok: [172.16.16.195] => {
    "msg": "Confidence Error"
}

TASK [meta] ***************************************************************** 11:15:51

PLAY RECAP ****************************************************************** 11:15:51
172.16.16.195              : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=1
```

Figura 5.9- Output de verificação de confiança sem sucesso

No caso de existir o certificado do domínio na máquina AWX e o mesmo permitir o acesso à *dashboard* do provedor de nuvem assume-se que o certificado é confiável e como tal aparece essa mesma mensagem, conforme representado na Figura 5.10.

Figura 5.10- Output de verificação de confiança com sucesso

Além disto, é de referenciar que a máquina no qual foi implementado o provedor de nuvem deve estar definida, assim como as suas credenciais de acesso na *dashboard* AWX, o que permite que se for tentada alguma tarefa numa máquina não registada, o *playbook* não é executado devido a erro de acesso SSH.

5.2.1. Testes Funcionais

A avaliação da eficácia destes mecanismos, foi realizada através dos testes funcionais descritos na Tabela 5.2, de forma a perceber que características são tidas em conta na execução dos mecanismos implementados. Isto é importante pois o ambiente não tem sempre as mesmas características e os mecanismos devem estar aptos a resolver qualquer problema que seja necessário. Por exemplo, no caso de criação de uma nova instância, o provedor de nuvem pode não ter configuração de redes, ou não ter *Floating* IPs disponíveis, e o mecanismo deve estar atento a estas necessidades e fazer as configurações necessárias para que isto não seja um problema.

Tabela 5.2- Testes Funcionais aos Mecanismos de Automação de multi-domínios

ID	Teste	Descrição do Teste	Resultado do Teste
[TF1]	Remoção da Instalação de Microstack	A execução do *playbook* deve remover uma implementação Microstack, e caso esta ferramenta não esteja instalada deve fornecer essa mesma informação.	Bem-sucedido
[TF2]	Reinício da Instalação de Microstack	A execução do *playbook* deve reiniciar uma implementação Microstack, e caso a ferramenta não esteja instalada deve fornecer essa mesma informação.	Bem-sucedido
[TF3]	Atualização da versão de Rancher	A execução do *playbook* deve atualizar a versão da ferramenta Rancher, e caso esta não esteja instalada deve fornecer essa mesma informação.	Bem-sucedido
[TF4]	Verificação de Nome de Instância através do IP	A execução do *playbook* deve verificar o nome da instância	Bem-sucedido

		através do IP. Caso a instância não seja uma instância de Microstack ou caso não seja definido o IP a verificar deve ser fornecida essa mesma informação.	
[TF5]	Eliminação de Instância em domínio Microstack	A execução do *playbook* deve eliminar a instância do domínio Microstack. Caso a instância não seja uma instância do domínio deve fornecer essa mesma informação. E no caso de não ser definido o nome da instância ou não haja instâncias com o nome definido o *playbook* deve fornecer essa informação.	Bem-sucedido
[TF6]	Criação de *snapshot* de uma instância em domínio Microstack	A execução do *playbook* deve criar um *snapshot* da instância do domínio Microstack. Caso a instância não seja uma instância desse domínio deve fornecer essa mesma informação. Caso não seja definido o nome da instância ou não haja instâncias com o nome definido o *playbook* deve fornecer essa informação.	Bem-sucedido
[TF7]	Criação de Instância a partir de *snapshot*	A execução do *playbook* deve criar uma nova instância em domínio Microstack a partir de um *snapshot*. Caso o domínio não seja Microstack deve fornecer essa mesma informação, e o mesmo para o caso de não ser definido o nome da instância, o formato do disco ou o seu *flavor*. Além disto o *playbook* deve verificar se existem redes internas disponíveis, redes externas, *router*, *security groups*, chaves, e Floating IPs. Em caso de algum componente não estar disponível, deve ser criado pelo *playbook* de forma a permitir a criação da instância.	Bem-sucedido
[TF8]	Alteração de Security Group	A execução do *playbook* deve alterar um *security group*	Bem-sucedido

		específico definido no domínio Microstack. Caso o domínio não seja Microstack, não seja definido o nome do *security group* ou este não exista o *playbook* deve fornecer essa mesma informação. Além disto, caso não seja definida nenhuma ação (criação ou eliminação) de uma regra, o *playbook* verifica se a regra já existe no *security group*, decidindo se irá criar ou eliminar a regra.	
[TF9]	Alteração de recursos de Instâncias	A execução do *playbook* deve alterar os recursos de uma determinada instância do domínio Microstack. Caso a instância não pertença ao domínio, não seja definido o nome da instância ou não haja instâncias com o nome definido o *playbook* deve fornecer essa mesma informação. Além disto, caso não haja mais alternativas de *flavor* a utilizar é criado um *flavor* com mais recursos do que o atual.	Bem-sucedido
[TF10]	Criação de Instância	A execução do *playbook* deve criar uma nova instância em domínio Microstack. Caso o domínio não seja Microstack deve fornecer essa mesma informação, e o mesmo para o caso de não ser definido o nome da instância, imagem ou o seu *flavor* ou estes valores não existirem. No último caso, o *playbook* irá criar um *flavor* novo. Além disto o *playbook* deve verificar se existem redes internas disponíveis, redes externas, *router*, *security groups*, chaves, e Floating IPs. Em caso de algum componente não estar disponível, deve ser criado pelo *playbook* de	Bem-sucedido

		forma a permitir a criação da instância.	
[TF11]	Criação de Security Group	A execução do *playbook* deve criar um novo *security group* num domínio Microstack. Caso o domínio não seja Microstack, não seja definido o nome do *security group* ou já exista um *security group* com o mesmo nome o *playbook* deve fornecer essa informação. O *playbook* deve ser executado independentemente de ser definida a descrição ou alguma regra para criação do *security group*.	Bem-sucedido
[TF12]	Verificação de Confiança	A execução do *playbook* deve permitir validar a confiança em um determinado domínio. Isto significa que deve verificar o certificado SSL de acesso à interface *web* do provedor. Caso o domínio seja desconhecido, ou esteja a ser feito o acesso SSL através de um certificado incorreto, o *playbook* deve fornecer essa mesma informação. Caso, o domínio seja confiável, deve ser apresentada uma mensagem com a confirmação.	Bem-sucedido

É importante realçar os desafios *multi-tenancy* destes ambientes, e como tal foram implementados mecanismos que permitem diminuir os pontos de entrada de possíveis ataques. Nomeadamente, uma das principais preocupações passa pela variedade de domínios, que podem ser de diferentes provedores, geridos por diferentes entidades e em diferentes locais, e como tal, é fundamental saber que domínios são confiáveis. Sendo possível distinguir os domínios de confiança, é aumentada a dificuldade de um atacante se fazer passar por um domínio do ambiente. Outra questão que se coloca prevê-se na conexão entre domínios e entre as instâncias dos próprios domínios, uma vez que nem todo o tráfego deve ser permitido, de forma a evitar que sejam efetuadas conexões desnecessárias ou perigosas. Assim, através da definição de grupos de segurança para cada instância, é possível negar todo o tráfego à partida, e apenas permitir o tráfego que seja realmente necessário.

Outro ponto, prevê-se na utilização de versões descontinuadas ou versões que possam possuir algumas vulnerabilidades. Como tal, foram implementados mecanismos que permitem uma

maior automação e eficiência na atualização, quer seja das máquinas ou a atualização das próprias ferramentas que são utilizadas no ambiente.

Além disto, de forma a avaliar os tempos de duração de cada mecanismo e a sua variação foram realizados 5 testes a cada um dos mecanismos, isto é, foram executados os *playbooks* e registados os valores de duração 5 vezes. Na Figura 5.11 é representada a variação de alguns dos *playbooks* ao longo dos testes. Apesar de apenas estarem representados alguns *playbooks* é importante referir que a variação dos restantes é muito semelhante e que a escolha destes foi meramente aleatória. Em cada teste as variáveis utilizadas foram as mesmas, como tal, em todos os testes são efetuados o mesmo número de tarefas.

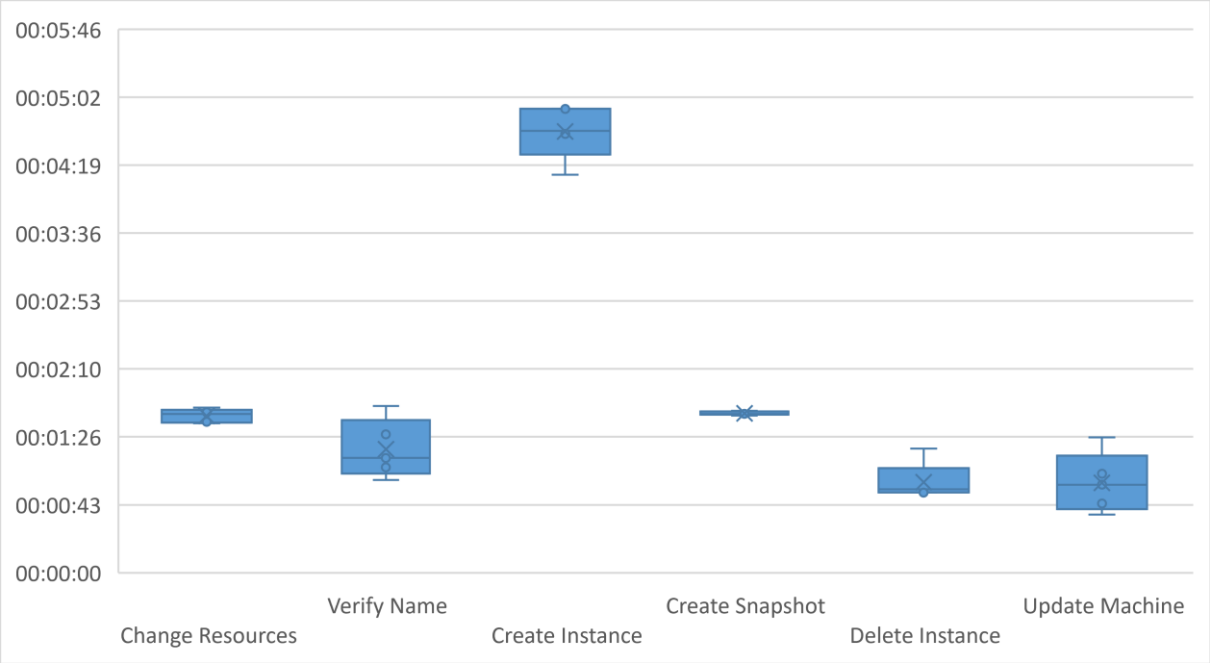

Figura 5.11- Representação da variação do tempo de execução de playbooks

Através desta visualização é possível analisar que a variação dos tempos de cada teste não é significativa, no entanto, a variação analisada deve-se ao tráfego de rede existente ou a possíveis variações na máquina destino que possam alterar o tempo de execução de comandos e a sua resposta. Relativamente à diferença de tempo entre cada *playbook*, é previsível, uma vez que os mesmos possuem diferentes números de tarefas, com diferentes complexidades. Além disto, tal como referido, os mecanismos foram realizados com as mesmas variáveis, no entanto, caso os mecanismos fossem realizados com variáveis diferentes (por exemplo: num teste de criação de instância, não haver redes previamente definidas, e num teste seguinte essas redes já estarem definidas), o tempo de execução do mesmo também seria diferente por cada teste representado. Quanto maior for o número de tarefas a executar e a sua complexidade, maior será o tempo necessário para a execução do mecanismo como um todo.

5.3 Síntese de Capítulo

Na Secção 5.1 são descritas as ferramentas e a abordagem que foi utilizada de forma a permitir a orquestração de múltiplos domínios de uma forma eficiente. Esta orquestração visa executar mecanismos que permitem automatizar as diversas atividades que possam ser necessárias no ambiente, bem como atividades CRUD em instâncias de domínios, ou atividades calendarizadas que visam manter a segurança e monitorização das instâncias.

Assim, na Secção 5.2 é feita a descrição e apresentação dos mecanismos que foram criados, os quais são executados através de *playbooks* que são armazenados num repositório GitHub privado e acedidos pela API AWX para a sua execução. A implementação destes mecanismos é efetuada através de Operadores Kubernetes que permitem a execução de forma transparente das ações a realizar e/ou verificar.

Além disto, não pode ser esquecido que ambientes multi-domínio possuem desafios relativos a segurança, nomeadamente, o facto de serem ambientes com diversos provedores que podem ser geridos por entidades distintas, em localizações distintas. E desta forma é fundamental que haja uma verificação de confiança quando são executadas ações em algum destes domínios, de forma a garantir que o domínio pertence ao ambiente e que a sua existência é conhecida. Assim, neste capítulo são também implementados mecanismos que permitem esta verificação e validação, aumentando a segurança do ambiente.

Na Secção 5.2.1 é feita uma apresentação e análise dos testes funcionais a cada mecanismo elaborado, bem como uma avaliação ao tempo de execução dos mesmos, de forma a avaliar a solução implementada e perceber de que forma poderia haver melhorias na mesma.

Assim, da perspetiva da estrutura geral do documento, o capítulo que agora finda permite a contextualização da implementação e das atividades realizadas com o objetivo de automatizar a orquestração de múltiplos domínios, de uma forma mais simplificada.

Capítulo 6 Orquestração de Clusters

Neste capítulo é abordada a temática de orquestração de clusters, tendo como referência os requisitos da implementação e a arquitetura proposta como solução para os mesmos. Assim, através da orquestração de clusters pretende-se implementar e estudar as possíveis implementações em ambientes multi-domínios.

Na Secção 6.1 é apresentada a ferramenta a ser utilizada e de que forma é que a mesma irá interagir com os diversos clusters e os respetivos componentes. Além disto, é feita a implementação de clusters Kubernetes através desta mesma ferramenta e realizado o seu estudo, de forma a perceber qual a implementação mais segura a ser executada. Assim, na Secção 6.2 são apresentadas as implementações realizadas sem integração de domínios em nuvem e com a integração dos mesmos.

Na Secção 6.3 é realizada a integração de Prometheus e Grafana nos Clusters Kubernetes implementados e a sua monitorização, quer a nível de recursos, como a nível de tráfego de rede e de tempo de *deployment* de microserviços.

A descrição de processos de automação de tarefas, através de Operadores K8s é feita na Secção 6.4, onde são implementados mecanismos que permitem a migração de microserviços dentro de um mesmo Cluster, ou entre Clusters distintos. Por último, na Secção 6.5 são apresentadas algumas abordagens que visam resolver a problemática de orquestração de multi-clusters, uma vez que estes ambientes acarretam novos desafios. Para finalizar, é realizada uma síntese, de forma a concluir os resultados obtidos no decorrer do capítulo.

6.1 Abordagem

A utilização de *containers* tem sido cada vez mais exponencial, uma vez que o seu uso traz diversos benefícios ao processo de implementação e gestão de aplicações, no entanto, com o uso de *containers* surge o aumento da sua utilização, provocando o aumento do número dos mesmos. Isto acarreta problemas na sua gestão, uma vez que é necessário estar atento a todos os componentes e a possíveis necessidades do ambiente. Desta forma, surge a necessidade de orquestração dos mesmos, de uma forma eficiente e inteligente, de forma a diminuir a necessidade de intervenção humana, e a ter sempre em consideração o requisito [R_NF5].

De forma a demonstrar os recursos de orquestração de uma implementação nativa em nuvem, e considerando a problemática de vários domínios, o Rancher será utilizado de forma a permitir a orquestração de múltiplos clusters, desde o seu processo de criação e/ou importação, análise dos seus componentes e a sua constante monitorização. Esta ferramenta possui suporte para vários provedores de nuvem, o que facilita a intermediação de orquestração de diferentes domínios, uma vez que o ambiente deste trabalho possui múltiplos provedores de nuvem que devem ser integrados e onde devem ser implementados clusters Kubernetes.

Tal como referido, a plataforma Rancher permite a implementação e orquestração de múltiplos clusters de uma forma sincronizada, quer os mesmos possuam provedores de nuvem distintos ou sejam implementados através de provedores de Kubernetes diferentes.

Rancher permite a criação de clusters Kubernetes, os quais podem ser criados através de RKE ou K3s. A primeira opção, RKE (Rancher Kubernetes Engine) é uma distribuição Kubernetes com certificação CNCF que visa simplificar a instalação e operação do Kubernetes e automatizar este mesmo processo. Além disto, esta distribuição permite que as operações realizadas sejam independentes do Sistema Operativo e da plataforma onde está a ser executado. RKE disponibiliza duas versões, RKE1 e RKE2, onde a principal diferença prática que os distingue revê-se no facto de RKE1 usar o Docker para implementação e gestão dos componentes do Plano de Controle e do Container Runtime. Enquanto, a versão mais recente, inicia os componentes do Plano de Controle como *pods* estáticos que são geridos pelo *kubelet*, e o Container Runtime como *containerd*. Como tal, na prática, ao ser utilizado RKE1 deve ser verificado inicialmente que as máquinas onde o mesmo será executado, têm o Docker previamente instalado e inicializado. Outro fator de distinção passa pela terminologia da sua preparação, ou seja, enquanto no provisionamento de RKE1 são utilizados "*Node Templates*", no RKE2 é possível configurar *pools* de nós.

Relativamente aos clusters K3s, a sua implementação é efetuada da mesma forma que RKE2 e como tal na prática, os utilizadores não são apresentados a grandes diferenças. A sua maior diferença, e que muitas vezes pode influenciar a escolha do provedor passa por K3s ser uma distribuição bastante mais leve, sem implicações significativas, mas que pode ser implementado em dispositivos com baixos recursos.

De forma a orquestrar os diversos clusters, independentemente de provedores, o Servidor Rancher implementa, de forma automática, um cluster *Controller* por cada cluster orquestrado pela ferramenta. Enquanto isto, é realizada também a instalação de um cluster *Agent* em cada cluster, de forma a permitir a constante comunicação entre os diversos ambientes, bem como a sua monitorização e orquestração. Esta comunicação é realizada através de quatro processos entre os componentes da arquitetura, representação presente na Figura 6.1.

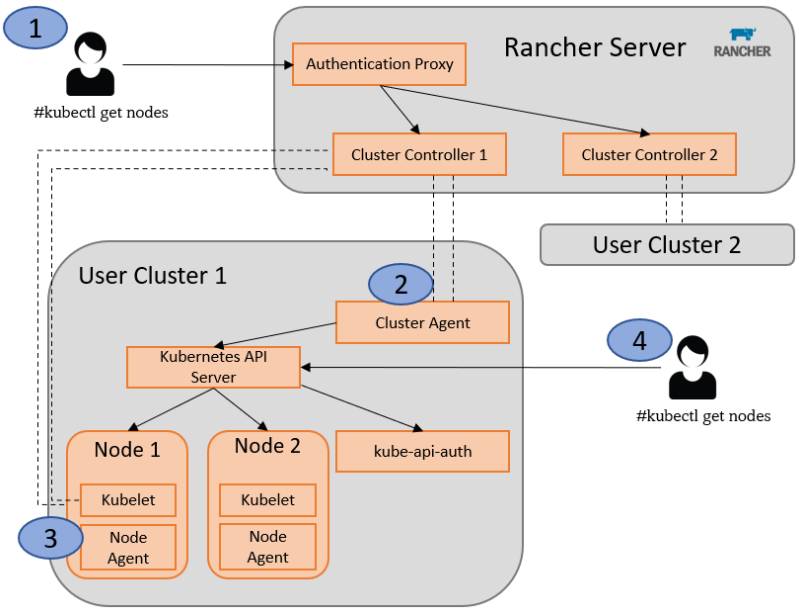

Figura 6.1- Comunicação entre Servidor Rancher e clusters

Sempre que um utilizador requisitar alguma informação de um cluster, este pedido pode ser efetuado de duas formas distintas. A primeira alternativa é quando o pedido é efetuado através do Servidor Rancher, onde a sua autenticação é realizada através do Proxy de autenticação, responsável pelo encaminhamento das chamadas de API para o cluster correspondente. De seguida, a comunicação passa a ser realizada através de uma *service account* que fornece uma identidade para os processos executados em *pods*. Posteriormente, e uma vez que cada cluster possui um cluster *Agent*, o mesmo é responsável por abrir um túnel com o cluster *Controller* respetivo. No entanto é de referir que em caso deste componente, cluster *Agent*, não se encontrar disponível, a comunicação não é negada, mas passa a ser feita diretamente com o Node Agent, através da criação de um túnel. Assim, é possível enviar esta solicitação para o Servidor Kubernetes API que irá responder à mesma.

Em caso do pedido de informação de um cluster ser efetuado por um utilizador autorizado (e com isto quer se dizer que o utilizador faz o pedido diretamente de um dos nós do cluster) o pedido é diretamente encaminhado para o Servidor Kubernetes API que irá comunicar com o *kube-api-auth*, um microserviço que fornece as funcionalidades de autenticação necessárias. Isto pode ser efetuado quando o Rancher se encontra *down* ou para reduzir a latência em caso das máquinas se encontrarem a uma longa distância.

É importante referir que o cluster *Agent* efetua a comunicação através do *hostname* do Rancher, e não do endereço IP, como tal, este agente deve ser capaz de fazer a tradução do mesmo. Isto acarreta a necessidade de criação de DNS Record com o *hostname* implementado para o Rancher, e que o mesmo seja conhecido por todas as máquinas do ambiente. O agente que é criado automaticamente irá correr num *pod* como tal, apenas o conteúdo do ficheiro "*resolv.conf*" é transmitido da máquina para o *pod*. Isto significa que não basta ser feita a tradução no ficheiro de *hosts*. Outra alternativa verificada foi o uso de nip.io, *wildcard* DNS que permite o mapeamento de qualquer endereço IP para um nome de *host* sem a necessidade de alteração de ficheiros ou de criação de DNS Record. Assim, permite que o nome "rancher-<IP>.nip.io" seja mapeado para o devido endereço IP, no entanto esta tradução não está habilitada no interior dos *pods*, como tal o seu uso não é reconhecido.

Após a implementação do Rancher, o qual foi implementado num cluster K3s (Lightweight Kubernetes) de forma a não fazer um uso excessivo de recursos e de forma a simplificar o processo de instalação, pode ser visualizada a *dashboard* desta ferramenta. Esta *dashboard* permite visualizar os vários clusters orquestrados (tanto no caso de serem importados, como no caso de serem criados de raiz por esta ferramenta), bem como permite definir políticas de segurança e associar clusters a domínios em nuvem, tal como será demonstrado na Secção 6.2. Desta forma, um dos clusters existentes é designado por "local", sendo o cluster referente à instalação K3s e no qual foi implementado o Servidor Rancher e todos os componentes necessários.

Além disto, Rancher inclui uma variedade de ferramentas integradas para auxiliar nas operações de DevOps, integrando serviços externos para ajudar os clusters a serem executados com mais eficiência, em categorias como *logging*, monitorização, alertas, notificações e *scans*.

A monitorização contínua é muitas vezes útil para minimização do tempo de resposta a incidentes e na garantia que tanto as aplicações, como a própria infraestrutura se comportam conforme previsto. Assim, permite rastrear recursos de cluster, como memória, CPU,

armazenamento e largura de banda, facilitando o processo de orquestração de ambientes nativos em nuvem e a possível prevenção de danificação do ambiente.

Relativamente à categoria de monitorização, a ferramenta integrada é o Prometheus, uma ferramenta de monitorização e de alerta de eventos. Desta forma, permite a recolha de métricas, a avaliação dos limites configurados e o acionamento de alertas em caso de alguma condição ser verdadeira. Este processo é efetuado através da exportação de métricas e o envio das mesmas para um servidor central, designado por Servidor Prometheus, através de HTTP. Neste sentido, numa implementação do Kubernetes, o servidor central pode fazer uso da API do Kubernetes para a extração das métricas. Por outro lado, de forma a permitir a visualização das métricas recolhidas pelo Prometheus, o Grafana é uma ferramenta bastante utilizada como complemento.

6.2 Deployment de Clusters Kubernetes com Rancher

Nesta secção será analisado e cumprido os objetivos da Tabela 6.1, os quais visam a implementação de mecanismos que permitam a implementação de clusters Kubernetes através de uma ferramenta de orquestração de clusters, e do teste da mesma, de forma a averiguar as possibilidades disponibilizadas pela mesma para a orquestração de múltiplos clusters. Tendo sempre em consideração o requisito [R_NF3].

Tabela 6.1- Descrição dos Requisitos Cumpridos na Secção 6.2

ID	Requisito	Descrição
[R4]	Implementação do ambiente em *containers* Kubernetes, representando clusters, através de uma ferramenta de orquestração de *clusters*	Implementação de um cluster Kubernetes através da ferramenta Rancher
[R5]	Implementação de uma plataforma que permita a orquestração de diversos *clusters* Kubernetes de forma unificada	Estudo e teste das possibilidades de orquestração de múltiplos clusters com a ferramenta Rancher
[R14]	Implementação de mecanismos que permitam a inicialização de *containers* num cluster específico	A ferramenta Rancher possibilita a criação de qualquer componente num cluster registado, através de linha da *shell kubectl* ou da criação de um ficheiro YAML
[R15]	Implementação de mecanismos que permitam a paragem e eliminação de *containers* num cluster específico	A ferramenta Rancher possibilita a eliminação ou a paragem forçada de qualquer componente de um cluster registado
[R16]	Implementação de mecanismos que permitam a atualização de *containers* num cluster específico	A ferramenta Rancher possibilita o mecanismo de "*Redeploy*" ou "*Rollout*", possibilitando atualizações de *containers* e a possível recuperação de versões anteriores

[R20]	Implementação de políticas de segurança que permitam limitar o acesso de cada *container* a informações e/ou serviços	A ferramenta Rancher permite definir e aplicar políticas de segurança aos diversos *containers* dos clusters implementados e/ou registados na ferramenta
[R21]	Implementação de políticas de segurança que permitam limitar o acesso de cada *container* a recursos (como por exemplo, o uso de memória)	A ferramenta Rancher permite a definição de limites de recursos para cada *container*, nomeadamente limites de reserva mínima e limites de uso de memória ou CPU
[R22]	Implementação de políticas de segurança que permitam proteger o acesso a cada nó do cluster e à API	A ferramenta Rancher permite a criação e partilha de certificados aquando da criação de um cluster e respetivos nós
[R30]	Implementação e validação de integração de clusters Kubernetes em domínios em nuvem	A ferramenta Rancher possibilita a criação de clusters Kubernetes em domínios em nuvem, existindo a integração com múltiplos provedores de nuvem, nomeadamente, o OpenStack

Desta forma, com o objetivo de testar de uma forma inicial a ferramenta Rancher e a sua possibilidade de orquestração de clusters, primeiramente foi criado um cluster sem qualquer integração com domínios em nuvem (referência ao requisito [R4] e [R5]). Esta criação foi realizada através da implementação de RKE2, uma vez que estes dispositivos possuíam os recursos necessários para a mesma. Como tal, foi criado um cluster de raiz, o qual é designado por "k8srke2", e que pode ser observado na Figura 6.2. Este cluster Kubernetes possui dois nós (um deles possui os *roles* "Control Plane", "etcd" e "worker", e o outro apenas possui a função de *worker*).

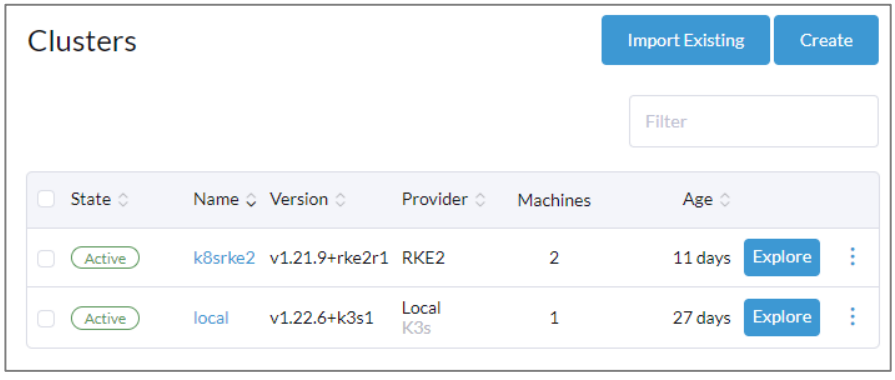

Figura 6.2- Clusters orquestrados pelo Rancher

Através da plataforma, é possível ter acesso direto à linha de comandos do Cluster, que permite a inicialização de *containers* e a gestão dos recursos do ambiente Kubernetes (referência ao requisito [R14]). Assim, neste cluster, foi realizada a implementação e monitorização de aplicações distintas baseadas em microserviços. Aplicações que diferem, tanto na sua topologia como no seu objetivo primordial. A primeira aplicação (*2-tier*) possui o objetivo de gerar tráfego de rede efetivo e realista e gerar usos excessivos de recursos, através

da utilização de *iperf3* (*software* utilizado para testes de *bandwidth*, uma vez que efetua a injeção de pacotes para a medição do desempenho de rede) e stress-ng (projeto para medição de desempenho em caso de testes de stress). A segunda aplicação (*3-tier*) serve o propósito de fornecer uma arquitetura padrão simples, mas realista em simultâneo, como ponto de partida para fins de demonstração, onde são implementados serviços de *backend*, base de dados e servidor *web*. A última aplicação (*12-tier*) consiste numa aplicação de comércio eletrônico baseado em *web*, a qual possui a topologia representada na Figura 6.3.

Estas diferentes aplicações foram escolhidas como cenários de referência para aplicações XR, compostas por vários microserviços e diferentes topologias, que podem ser efetivamente monitorizadas num ambiente nativo em nuvem, suportando mecanismos de previsão, agendamento e orquestração inteligente.

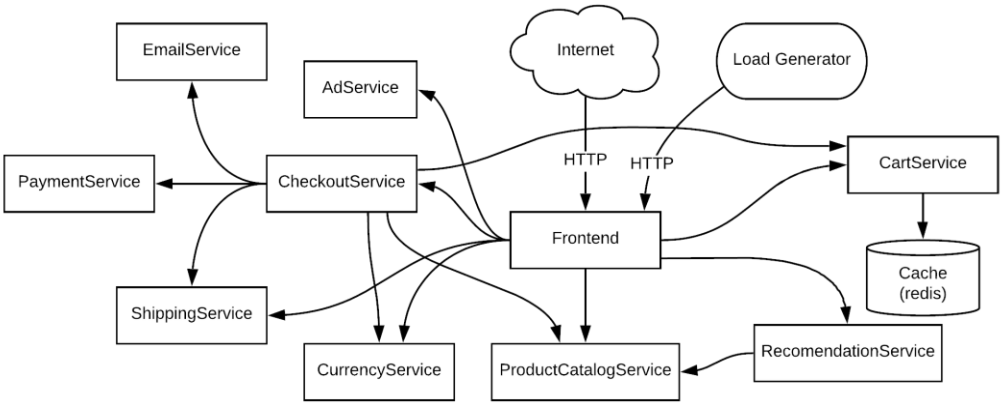

Figura 6.3- Topologia da aplicação 12-tier [91]

Na plataforma Rancher, ao ser realizada a exploração dos clusters existentes é possível visualizar os nós que fazem parte dos mesmos, o seu Sistema Operativo, Role ("control plane", "etcd" e/ou "worker") e estado. Além disto, é também possível visualizar o número total de recursos, o seu estado e detalhes, de forma a ser possível a monitorização de todos os componentes. Da mesma forma, é possível eliminar, atualizar e/ou recuperar uma determinada versão de um recurso do cluster Kubernetes (referência ao requisito [R15] e [R16]).

Dentro de cada cluster, é fundamental que sejam incorporados mecanismos de forma a limitar o uso de recursos de cada *container*, de forma que o cluster não seja prejudicado pelo uso excessivo de recursos de um *deployment* (referência ao requisito [R21]). Assim, através do Rancher é possível limitar este mesmo uso, através da definição de valores como CPU *Reservation*, *Memory Reservation*, CPU *Limit* e *Memory Limit*. O valor *Reservation* permite definir o valor mínimo para atribuição a um *deployment*, assim mesmo no caso de outros partilharem memória e/ou CPU é garantido que há uma quantidade mínima desses valores que está sempre garantida a esse *deployment*. Isto permite evitar que um *deployment* prejudique o uso de recursos dos restantes, uma vez que cada um possui uma reserva de recursos definida. Por outro lado, o valor limite permite estabelecer um valor máximo de uso de CPU e/ou memória, de forma a evitar usos exagerados dos recursos. A definição destes valores permite o controle do uso de CPU e memória de cada *deployment*, e consequentemente, o controle dos recursos de cada cluster, de forma a garantir que o mesmo permanece saudável e que a sua disponibilidade não é prejudicada.

Assim, de forma a garantir que os valores aplicados não causam qualquer consequência negativa e que são aplicados corretamente, os *deployments* foram inicializados sem qualquer

definição de limites e *Reservation*, e só após a sua inicialização e monitorização dos valores normais de uso destes recursos é que foi efetuada a sua definição.

Com o objetivo de testar esta definição de valores, foi utilizada a aplicação 2-*tier*, a qual faz uso de stress-ng (plataforma de stress de CPU, que provoca picos de CPU exagerados). Foi definido o valor de limite de CPU como 1500m e o valor de limite de memória como 900Mi e registado o *output* da ultrapassagem deste valor limite. Conforme a Figura 6.4 e Figura 6.5 que presentam gráficos disponibilizados pelo Prometheus e Grafana (abordados em mais detalhe na Secção 6.3), o *pod* é inicializado normalmente, no entanto, quando chega ao pico de recursos definidos, os valores de CPU e memória decrescem rapidamente uma vez que o *pod* é finalizado, não ultrapassando os valores definidos.

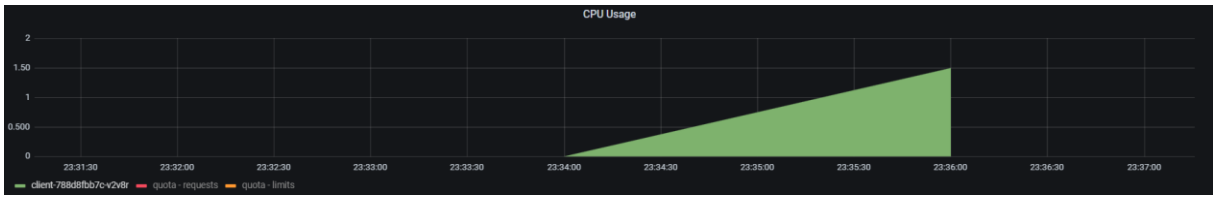

Figura 6.4- Registo do aumento de utilização de CPU

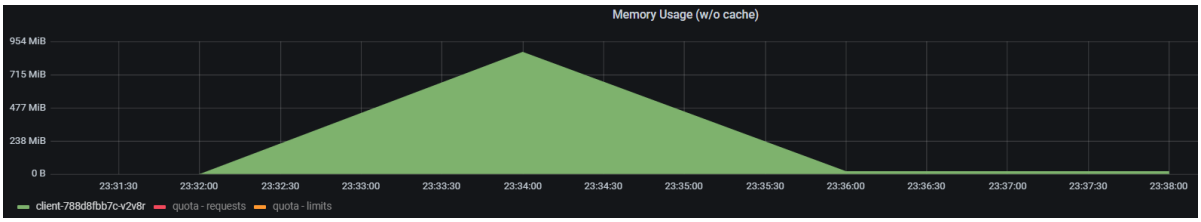

Figura 6.5- Registo do aumento de utilização de memória

Da mesma forma, é importante definir políticas de segurança para cada container (referência ao requisito [R20]), as quais podem ser aplicadas na *dashboard* da ferramenta Rancher. Por padrão, existem três grupos de políticas configurados, nomeadamente o "*restricted*" (restringe ações de utilizador e não permite o escalonamento de privilégios), "*restricted-noroot*" (restringe todas as ações de utilizador, não permite o escalonamento de privilégios e exige que os *containers* sejam executados sem privilégios de *root*) e "*unrestricted*" (política bastante permissiva e equivalente a executar o Kubernetes sem as políticas de segurança habilitadas). A definição destas políticas permite definir características, como:

- Escalonamento de Privilégios;
- Leitura de *Root Filesystems*;
- Utilizador e Grupo que o container usa para ser executado;
- Permitir/Bloquer a especificação de opções SELinux;
- Listar *host paths* permitidos;
- Permitir/Bloquear capacidades, como acesso ao relógio do sistema ou alarmes.

Desta forma, os clusters implementados foram registados com as políticas de segurança de *pod* que podem ser observadas na Tabela 6.2. As quais devem ser analisadas, uma vez que algumas opções podem influenciar o desenvolvimento de algumas aplicações.

Tabela 6.2- Políticas de segurança implementadas ao nível de Pod

Política	Valor	Detalhes
Escalonamento de Privilégios	Negação	A recusa de escalonamento de privilégios nega a possibilidade de o utilizador alterar o seu ID.
Priviledged	Negação	Esta opção deverá ser habilitada em casos de necessidade de acesso a todos os dispositivos do *host*, ou para executar recursos do Sistema Operativo. Nos restantes casos, como o atual, esta opção deve ser desabilitada, de forma a não permitir que os *pods* tenham acesso direto ao *host*.
Read Only Root Filesystem	Negação	Se esta opção não for negada, o sistema de arquivos de raiz apenas tem a permissão de leitura, não podendo ser gravável.
Host namespaces	Host IPC: Não Host Network: Não Host PID: Não	Estas opções devem ser desabilitadas de forma a não permitir que o *container* partilhe a rede e o *namespace* do ID do processo do *host*. Se estas opções não forem desabilitadas torna-se possível o acesso do *pod* a serviços no próprio *host* e para registar a atividade de rede de outros *pods* ou para o escalonamento de privilégios através do ID do processo.
Capacidades	Capacidades Recusadas (CHOWN, KILL, MAC_ADMIN, MAC_OVERRIDE, NET_ADMIN, NET_BIND_SERVICE, NET_BROADCAST, SETUID, SETGID, SYSLOG, SYS_ADMIN, SYS_BOOT, SYS_CHROOT) Capacidades Permitidas (AUDIT_CONTROL, AUDIT_WRITE, BLOCK_SUSPEND, DAC_OVERRIDE, DAC_READ_SEARCH, FOWNER, FSETID, IPC_LOCK, IPC_OWNER, LEASE, LINUX_IMMUTABLE, MKNOD, NET_RAW, SETFCAP, SETPCAP, SYS_MODULE, SYS_NICE, SYS_PACCT, SYS_PTRACE,	No sentido de evitar o escalonamento de privilégios através de algum recurso, foi feito o *deny* de algumas capacidades (por exemplo: o comando "*chown*" permite alterar o dono de um determinado ficheiro, o que pode ser muitas vezes utilizado para escalonar os privilégios de diretorias ou ficheiros, o comando "*kill*" permite parar imediatamente qualquer processo que esteja a correr. Estes comandos devem ser desabilitados de forma a não permitir a execução destas ações).

	SYS_RAWIO, SYS_RESOURCE, SYS_TIME, SYS_TTY_CONFIG, WAKE_ALARM)	
Volumes	emptyDir, secret, persistentVolumeClaim, downwardAPI, configMap, projected	Definição do tipo de volumes que podem ser criados, nomeadamente volumes para injeção de dados (*configMap*) ou transmissão de dados sensíveis (*secret*), volumes inicializados aquando da criação de *pods* (*emptyDir*), volumes que permitem mapear múltiplos volumes na mesma diretoria (*projected*) e volumes que permitem disponibilizar dados para aplicações (*downwardAPI*). Alguns tipos de volumes que podem ser habilitados, encontram-se descontinuados ou possuem possíveis entradas de ataques, como tal, não foram permitidos.
Host Path	"/home"	Especificação do caminho permitido para a criação de volumes *hostPath*. De forma a não serem criados volumes na diretoria principal "/", foi definido o caminho "/home" para que todos os volumes sejam implementados nesta diretoria.
FS Group	MustRunAs	Os volumes dos *pods* são alocados por o FS *Group* especificado. Neste caso, o intervalo definido será principalmente com o objetivo de não incluir o grupo *root* (de ID 0).
Host Ports	0-7000	Por defeito se não for definido este parâmetro não é permitido o uso de nenhuma porta, como tal foi definido o intervalo especificado para permitir o uso de portas, sem limitações.
Run As User	MustRunAsNonRoot	O *pod* tem de ser submetido por um utilizador diferente de *root* (ID=0). Isto previne que sejam executados *pods* com o máximo de privilégios.
Run As Group	MayRunAs	O *pod* pode ser inicializado por um utilizador de qualquer grupo primário, no entanto, se forem definidos grupos, o utilizador tem de pertencer a um dos grupos definidos para o *pod* ser implementado.
Grupos Suplementares	MayRunAs	O *pod* pode ser inicializado por um utilizador de qualquer grupo suplementar, no entanto, se forem definidos grupos, o utilizador tem

		de pertencer a um dos grupos definidos para o *pod* ser implementado.

De forma a validar as opções selecionadas, pode ser visualizada a Figura 6.6 onde foi procedida à tentativa de implementação de um *pod* como privilegiado. Como esta opção foi recusada nas políticas de segurança, o *pod* foi impossibilitado de ser criado.

```
Error from server (Forbidden): error when creating "https://git.io/fNhJX": pods
"privileged" is forbidden: PodSecurityPolicy: unable to admit pod: [spec.contain
ers[0].securityContext.privileged: Invalid value: true: Privileged containers ar
e not allowed]
```

Figura 6.6- Output de criação de pod como privilegiado

Em semelhança, a Figura 6.7 mostra a tentativa de criação de um *pod* a correr como *root* (*user* id=0) e com o grupo *root* (*group* id=0). Tentativa esta que se revelou negada, devido às permissões referidas anteriormente.

```
Error from server (Forbidden): error when creating "two.yaml": pods "privileged"
 is forbidden: PodSecurityPolicy: unable to admit pod: [spec.containers[0].secur
ityContext.runAsUser: Invalid value: 0: running with the root UID is forbidden s
pec.containers[0].securityContext: Invalid value: []int64{0}: group 0 must be in
 the ranges: [{1 6}]]
```

Figura 6.7- Output de criação de pod a correr como root

A Figura 6.8 representada a tentativa de criação de *pod* com permissão de escalonamento de privilégios, que como foi negada inicialmente, bloqueia a possibilidade de criação de *pods* com esta característica.

```
Error from server (Forbidden): error when creating "three.yaml": pods "privilege
d" is forbidden: PodSecurityPolicy: unable to admit pod: [spec.containers[0].sec
urityContext.allowPrivilegeEscalation: Invalid value: true: Allowing privilege e
scalation for containers is not allowed]
```

Figura 6.8- Output de criação de pod com permissão para Escalonamento de Privilégios

Além disto, e de forma a maximizar o controle de acesso a cada nó do cluster, e em simultâneo à API (referência ao requisito [R22]) devem ser tidas algumas configurações em atenção, como o uso de TLS (as comunicações dentro do Plano de Controle, entre este componente e *Kubelet*, entre API Server e *Kubelet* e entre Servidor API e *etcd*, bem como as comunicações de acesso direto ao Servidor API devem ser realizadas através de conexões TLS). Neste sentido, Rancher permite a configuração e partilha de certificados de forma automática aquando da criação de clusters e respetivos nós, não havendo a preocupação extra dos utilizadores serem responsáveis por essa mesma partilha. Além disto, em caso de ser detetada alguma intrusão ou alguma anomalia, podem ser gerados novos certificados, de forma que os anteriores sejam revogados e não voltem a ser utilizados.

No entanto, se não fosse utilizada uma ferramenta de orquestração de clusters, este requisito podia se tornar difícil de alcançar, uma vez que com a habilidade de escalonamento do Kubernetes, torna-se complexo configurar continuamente a gestão de certificados TLS. Assim, para facilitar este desafio o Kubernetes introduz características de TLS bootstrapping, de forma a simplificar e automatizar as etapas necessárias para o estabelecimento de conexões TLS. Neste caso seria necessário configurar algumas opções no *kubelet*, nomeadamente o ficheiro de configurações e o ficheiro de configurações de *bootstrap* (o qual deve ser gerado),

configurar opções no *kube-apiserver*, nomeadamente o modo de autorização e o ficheiro de autenticação *token* (que deve ser gerado previamente). Também é necessário configurar opções no *kube-controller-manager*, nomeadamente o ficheiro da chave e do certificado do cluster. Quando são analisadas as *ClusterRoles* do cluster, é possível visualizar a *ClusterRole* "*system:node-bootstraper*" que é criada automaticamente e que será responsável por gerir os pedidos de acesso ao cluster. No entanto, é necessária a criação de mais três *ClusterRoles*, uma responsável por gerir o *bootstrap* ao nível do *kubelet*, outra responsável por aprovar os nós, e outra que permite a renovação dos certificados dos nós. Após isto, apenas quando é observado os CSR do cluster, o *output* mostra os certificados que foram aprovados automaticamente pelo *kubelet-bootstrap*, e os certificados pendentes que necessitam de aprovação. De forma a aprovar os certificados é necessário introduzir o comando "*kubectl certificate aprove <NOME-CERTIFICADO>*" e após isto, os nós serão integrados no cluster.

Tendo estes aspetos em consideração foi realizada a implementação de clusters Kubernetes em domínios em nuvem com a utilização da plataforma Rancher (referência ao requisito [R30]). Esta implementação pode ser feita de várias maneiras, nomeadamente através da incorporação de um cluster por cada domínio, tal como representado na Figura 6.9. Todas as instâncias implementadas dentro de um domínio são nós de um cluster, ou seja, o número de clusters que serão implementados será o mesmo número que a quantidade de domínios de nuvem que o ambiente possui.

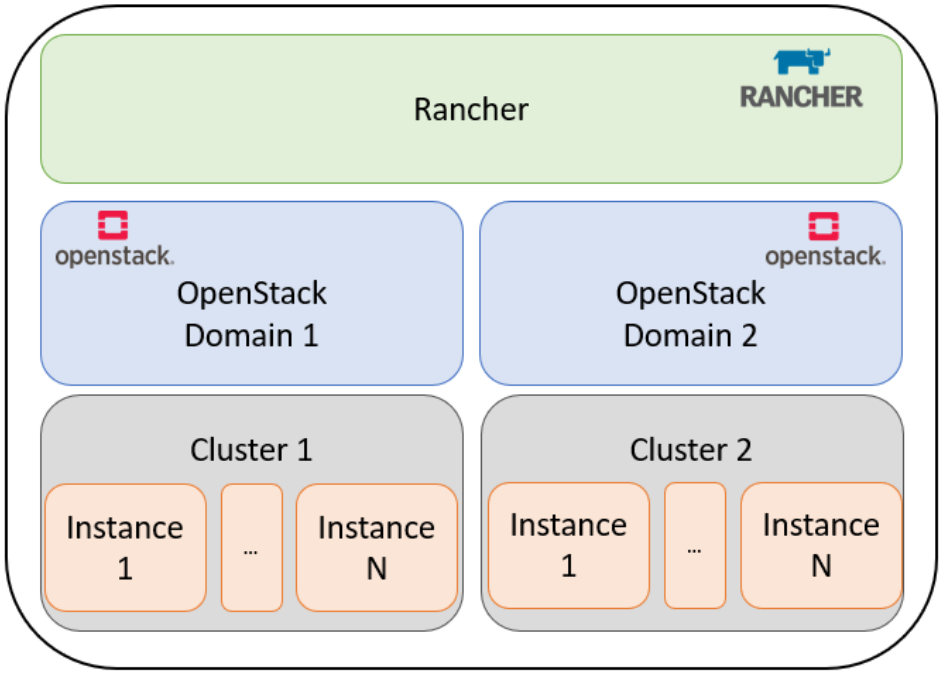

Figura 6.9- Representação da Arquitetura de Orquestração de clusters no caso de cada cluster estar implementado num único domínio

Outra possível integração de clusters em domínios em nuvem passa pela dispersão dos vários clusters, ou seja, cada cluster pode ter um nó em múltiplos domínios em simultâneo. Isto significa que um domínio pode possuir máquinas em 2 clusters distintos, como representado na Figura 6.10, ou até possuir as máquinas todas como componentes do mesmo cluster.

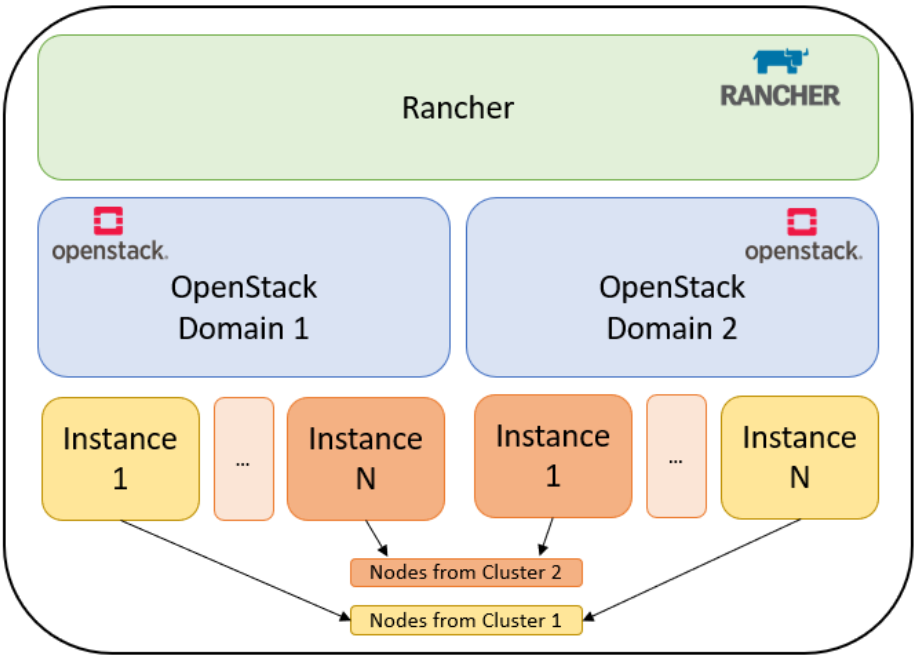

Figura 6.10- Representação da Arquitetura de Orquestração de clusters no caso de cada cluster possuir nós de múltiplos domínios

Ambas as abordagens possuem as suas vantagens, como o facto de no primeiro caso, ser possível um maior isolamento do domínio, e do próprio cluster, uma vez que a comunicação com as portas de comunicação entre os vários nós, apenas é permitida dentro da mesma rede, enquanto no segundo caso, esta comunicação tem de ser autorizada para acesso ao exterior, o que acresce uma possível entrada de ataques. No entanto, o segundo caso tem a vantagem de possuir uma maior tolerância a falhas, no caso de falha de um provedor de nuvem, o cluster não fica indisponível, uma vez que possui componentes em outros domínios que possam estar disponíveis, e assim manter a disponibilidade dos serviços oferecidos por esse cluster. Esta situação não acontece no caso de cada cluster ser incorporado num único domínio, uma vez que em caso de falha de um domínio, todos os nós do cluster se tornam indisponíveis, e consequentemente os seus serviços. No entanto, é importante haver alternativas que visem resolver estes casos de indisponibilidade dos serviços, quer seja, pelo caso de o cluster apenas ter componentes num único domínio e o mesmo falhar, ou por haver algum problema de indisponibilidade no cluster.

Para esta implementação foram realizados alguns testes, que provaram que a integração de clusters RKE2 era bastante dificultada uma vez que devido aos recursos dos domínios de nuvem, havia níveis de latência superiores aos previsíveis e como tal, os componentes do Kubernetes tinham bastante dificuldade na comunicação entre si. O mesmo acontece, mas de forma menos significativa, no caso de ser instalado um Servidor K3s numa instância de domínio em nuvem uma vez que o Servidor irá disponibilizar o *Control Plane* e *etcd* do cluster.

Assim, a implementação de clusters em domínios em nuvem não se revelou uma integração muito simples, na medida em que a necessidade de maior número de recursos se mostrou ser bastante significativa. A tentativa de inicialização de um cluster RKE2, bem como a

inicialização de cluster K3s inteiramente num único domínio de nuvem apresentava erros de latência bastante significativos, os quais impossibilitavam a comunicação entre componentes Kubernetes como representado pelo output do Rancher na Figura 6.11, onde demonstra o reinício constante dos componentes, de forma a estabelecerem comunicação entre eles. O mesmo acontecia, no caso de as máquinas apenas serem *workers* de clusters RKE2, os erros de latência impossibilitavam a comunicação com o nó *Master*.

```
[INFO ] waiting for at least one bootstrap node
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting for agent to check in and apply initial plan
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, etcd, kube-apiserver, kube-controller-manager, kube-scheduler,
kubelet
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, etcd, kube-apiserver, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, etcd, kube-apiserver, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver, kube-controller-manager, kube-scheduler
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver, kube-controller-manager
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-controller-manager
[INFO ] provisioning bootstrap node(s) custom-6e31ff32aee7: waiting on probes: calico, kube-apiserver, kube-controller-manager
```

Figura 6.11- Output do Rancher como resultado de erros de latência

Assim, e uma vez que a latência é um fator crucial a ter em atenção na implementação deste ambiente, e de forma a não correr riscos de indisponibilidade no cluster, a arquitetura implementada passa pela representada na Figura 6.12. Onde foi realizada a implementação de um Servidor K3s fora do domínio em nuvem, ou seja, numa máquina externa, e a posterior implementação da *role worker* em instâncias do domínio em nuvem. Desta forma, não ocorrem problemas de latência, bem como se disponibiliza uma máquina fora do domínio que poderá ser bastante útil no caso de o domínio falhar ou ter algum problema de segurança, assim têm se sempre o *backup* do cluster. No entanto, esta não é uma integração automatizada, uma vez que esta implementação é realizada através da criação manual de máquinas do OpenStack e posterior implementação do cluster. Implementação esta que é executada da mesma forma que uma máquina fora do domínio em nuvem, e não de forma automática como se previa.

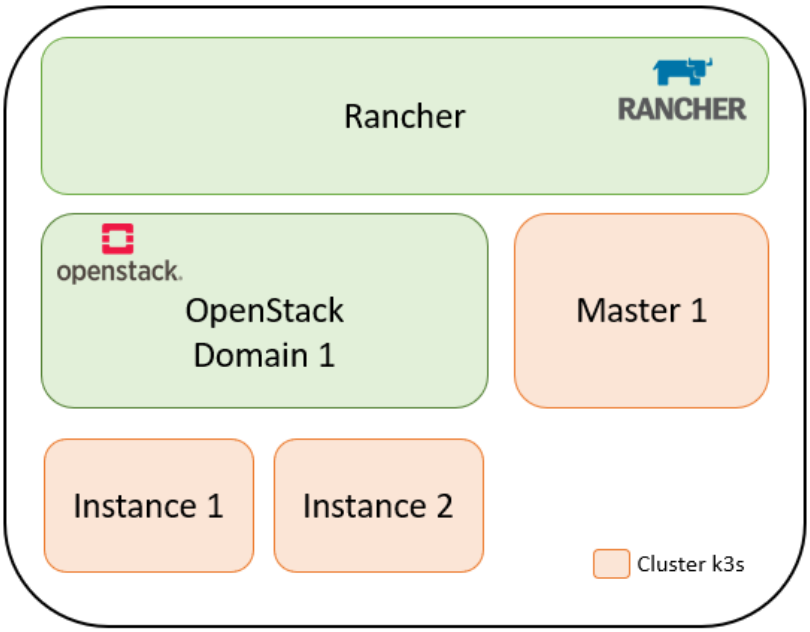

Figura 6.12- Representação da Arquitetura de Orquestração de clusters

Olhando para a disponibilidade da arquitetura implementada, é de referir que se o OpenStack possuir alguma falha de disponibilidade, o cluster continua operacional. Se a falha de disponibilidade for referente à interface *web* do OpenStack, os nós *workers* continuam disponíveis pois o seu estado continua ativo, no entanto, se a falha tiver origem na máquina OpenStack ou no próprio serviço, as máquinas *workers* ficam indisponíveis, ficando apenas ativo o nó *Master*.

6.3 Integração de Prometheus e Grafana para monitorização dos Clusters Kubernetes

Nesta secção será analisado e cumprido os objetivos da Tabela 6.3, os quais visam a implementação de mecanismos que permitam a monitorização de recursos dos diversos clusters, de forma a avaliar possíveis situações anómalas e a tornar a sua mitigação mais rápida.

Tabela 6.3- Descrição dos Requisitos Cumpridos na Secção 6.3

ID	Requisito	Descrição
[R27]	Implementação de mecanismos que permitam monitorizar o cluster, os seus nós e serviços relativamente a recursos físicos	A utilização do Prometheus, juntamente com Grafana permite a recolha destas métricas, bem como a visualização dos dados
[R28]	Implementação de mecanismos que permitam monitorizar o cluster, os seus nós e serviços relativamente a tráfego de rede	A utilização do Prometheus, juntamente com Grafana permite a recolha destas métricas, bem como a visualização dos dados
[R29]	Implementação de mecanismos que permitam monitorizar os serviços do cluster relativamente ao seu *deployment time*	Foi criada uma *dashboard*, que através dos dados recolhidos pelo Prometheus, permite a visualização do *deployment time*

Tal como referido, e de forma a efetuar uma monitorização mais abrangente e analisar diferentes características dos diversos clusters, foi instalado o Prometheus e Grafana, já integrados no Rancher, em cada um dos clusters. Isto fornece uma forma simples e eficiente de visualizar várias métricas específicas de *pod* e cluster com suporte nativo. Ao utilizar esta abordagem de monitorização, em particular a arquitetura Prometheus, podem ser obtidas diferentes métricas específicas de XR, utilizando bibliotecas adicionais e exportadores Prometheus para expor o tipo de métricas que são desejadas.

Uma das métricas mais relevantes é a latência, um fator crítico num ambiente nativo em nuvem com suporte a serviços XR, uma vez que os atrasos podem afetar negativamente a experiência geral do utilizador. Neste sentido, é importante a inclusão de um mecanismo de medição do tempo de implementação de uma aplicação (o tempo entre a criação até ao seu correto funcionamento), de forma a avaliar e regular adequadamente o desempenho da mesma.

Além disto, normalmente, o tempo de implementação não leva em conta a disponibilidade do serviço, ou seja, medimos o tempo até que o *pod* esteja em execução, e não o tempo que o serviço demora a estar acessível e totalmente operacional (por exemplo, no caso de um servidor *web* é necessário tempo para este se encontrar acessível, assim como no caso de uma base de dados, é necessário tempo para a migração dos dados). Assim, com o objetivo de avaliar o tempo de implementação das aplicações mencionadas, foi desenvolvido um componente que permite calcular o tempo de implementação, com base nos eventos registados do Kubernetes. Isto porque as métricas disponíveis para uso relatadas pelo Kubernetes (*kube-state-metrics*) não fornecem esse mecanismo (nomeadamente, as métricas disponíveis não têm em consideração o tempo de *pull* da imagem do *container*).

De forma a ser percetível a discrepância no tempo de implementação de diferentes serviços e aplicações, é importante que sejam entendidos os vários estágios deste processo. Este processo inclui o agendamento dos *pods*, a extração das imagens, e por fim, a criação e inicialização do mesmo.

A Figura 6.13 representa os serviços de uma das aplicações (aplicação *3-tier*), e os seus respetivos tempos de implementação, de uma forma comparada entre o caso de ser feito o *pull* da imagem do container no processo de implementação e o caso de não ser realizado o *pull*. Tal como é percetível, o tempo de implementação é bastante diferente nos dois casos, uma vez que o *pull* da imagem é a etapa que leva mais tempo, mesmo tendo em consideração que este valor pode depender do tamanho da imagem. De forma a neutralizar este valor, os *pods* podem ser configurados de forma que o *pull* da imagem só seja executado se a imagem não estiver presente localmente no nó vinculado ao *pod*. Assim, na sua primeira implementação irão ser extraídas, uma vez que as imagens não se encontram armazenadas em *cache*, no entanto, as seguintes implementações já não irão realizar este processo.

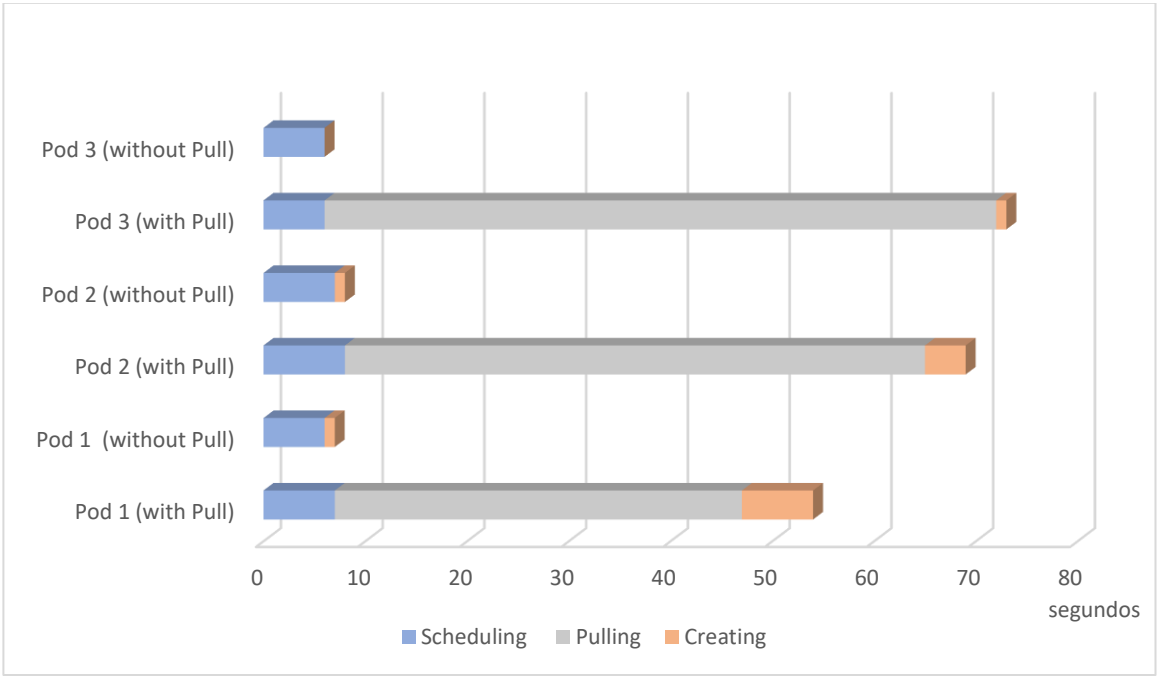

Figura 6.13- Etapas incluídas no tempo de implantação

Apesar disto, é importante que seja feita uma análise de como a abordagem de eliminação do tempo de *pull* se reflete em ambientes de ritmo acelerado e desenvolvimento rápido, uma vez que levanta inúmeras questões sobre configuração de nós e orquestração de cluster. A

manutenção das imagens locais em nós torna-se mais difícil de lidar com ambientes nativos em nuvem altamente complexos e com diversos nós. Como tal, a priorização do tempo de implementação em detrimento da complexidade é uma questão que exige uma avaliação adequada, e o entendimento do seu impacto, de forma a adequar os ambientes nativos em nuvem a casos específicos.

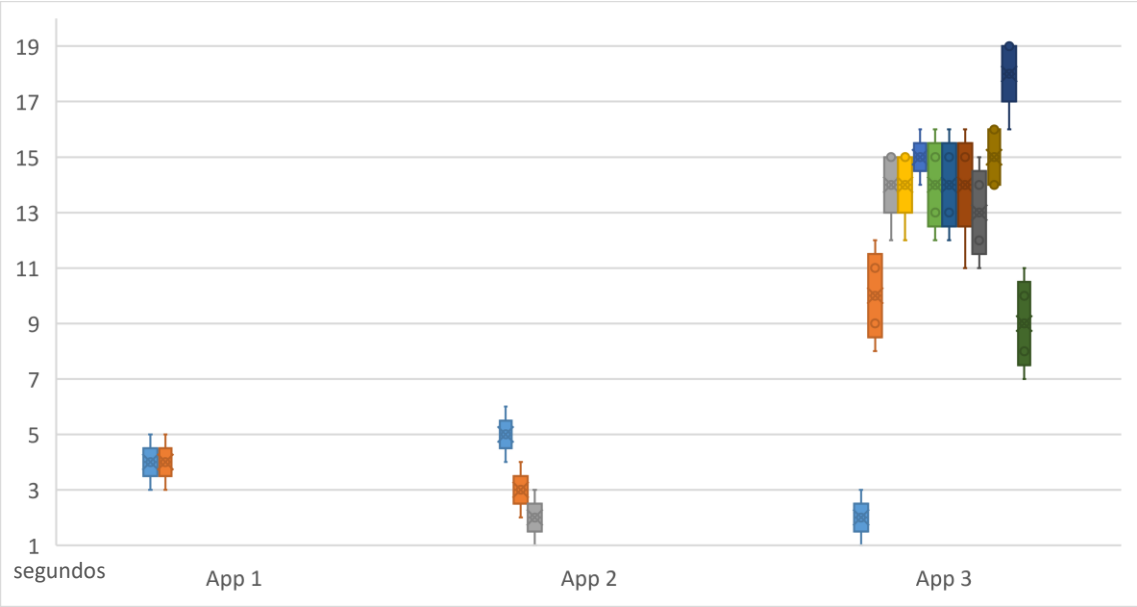

Figura 6.14- Tempo médio de implantação dos pods para diferentes aplicações

Os valores observados na Figura 6.14 representam os valores dos vários testes realizados, de forma a ser demonstrada a diferença dos tempos de *deployment* entre aplicações. Estes valores podem ser explicados pela complexidade crescente de cada aplicação, quanto mais serviços a aplicação englobar, mais tempo demora para a sua implementação. De forma a obter o valor de demora para a aplicação (como um todo) se encontrar totalmente implementada, deve ser visualizado o tempo máximo de implementação dos seus *pods*, isto é, o *pod* que demorou mais tempo para ser implementado (por exemplo, a aplicação 3 demora, em média, 18 segundos para implementar, uma vez que o *pod* com mais tempo de implementação precisa de 18 segundos, em média).

Neste caso, as imagens dos diversos containers já tinham sido armazenadas, e como tal, estes tempos de implementação não englobam o tempo de *pull* das mesmas. Assim, a justificação para a divergência destes valores, neste caso não poderá estar no tamanho das imagens, mas sim na dependência entre serviços, isto é, por exemplo, a aplicação 3 possui o maior número de serviços, e consequentemente, o maior número de dependências entre si, enquanto a aplicação 1 apenas possui dois serviços, os quais não possuem dependências entre si, e como tal, os seus valores de implementação são bastante semelhantes. Estas dependências entre serviços em aplicações nativas em nuvem podem ter impacto em várias operações (por exemplo, migração de serviços ou dimensionamento), e o mesmo para aplicações XR, uma vez que as suas topologias são complexas e possuem inúmeras restrições dinâmicas. O que se traduz na importância da sua orquestração em tempo real.

Tal como referido anteriormente, o Kubernetes não reporta métricas que permitam fazer a análise do tempo de implementação, como tal, numa segunda fase foi criada uma nova *dashboard* no Grafana que permite analisar e avaliar estas métricas através dos *Health Checks*

disponibilizados pelo Kubernetes (referência ao requisito [R29]). O Kubernetes possui três *Health Checks*, nomeadamente:

- Liveness Probe - Verificação do estado do *container*, analisando se o mesmo se encontra a correr ou não. Em caso negativo, o mesmo efetua a política de reinício que estiver definida.
- Readiness Probe - Verificação do estado da aplicação, isto é, se a mesma se encontra preparada para receber e responder a pedidos. Em caso negativo, o IP do *pod* é removido da lista de *endpoints* do serviço, uma vez que ainda não se encontrará disponível.

Estes *probes* podem ser utilizados de forma a perceber o estado e progressão da implementação de aplicações. Para a definição dos mesmos é importante perceber qual o teste que fará mais sentido para a sua verificação, nomeadamente, para servidores *web* o mais utilizado é a execução de uma solicitação HTTP GET (onde deve ser especificado o caminho e a porta a utilizar), para servidores de base de dados pode ser efetuada a execução de um comando (ex: um pedido de acesso à base de dados), ou pode ser utilizado *socket* TCP (neste caso, deve ser definido a porta a utilizar).

Através da definição e configuração dos *probes* em cada aplicação que seja implementada, podem ser utilizadas métricas do Kubernetes que permitem analisar o número de vezes que um determinado *probe* deu sucesso, ou por outro lado, que falhou. As visualizações dos resultados desta métrica podem ser visualizadas na Figura 6.15. Desta forma, é possível calcular o tempo em que o *probe* deu sucesso pela primeira vez, e posteriormente, calcular o tempo que demorou até esse mesmo instante.

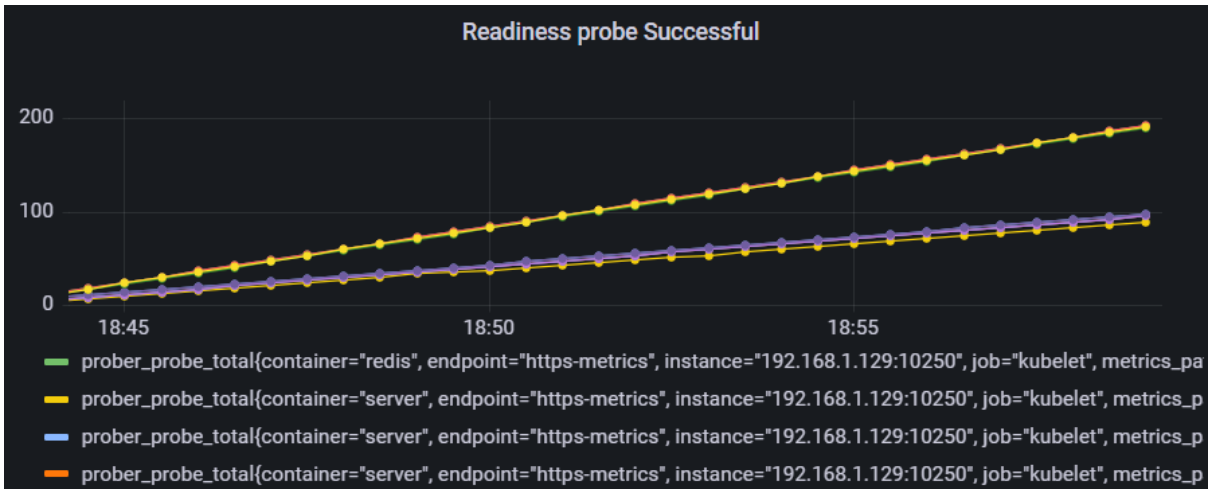

Figura 6.15- Número de vezes que o teste de Readiness probes da aplicação 12-tier foram bem-sucedidos

Além disto, também é importante avaliar e analisar mecanismos que permitam a visualização do uso de CPU e memória (referência ao requisito [R27]), a fim de entender se alguma aplicação está a fazer um uso excessivo destes recursos. Assim, a sua monitorização permite minimizar o esgotamento e o uso indevido de recursos, e desta forma, manter o ambiente o

mais saudável possível. Neste sentido, o Grafana disponibiliza *dashboards* para a visualização destas métricas, tal como representado na Figura 6.16 e Figura 6.17.

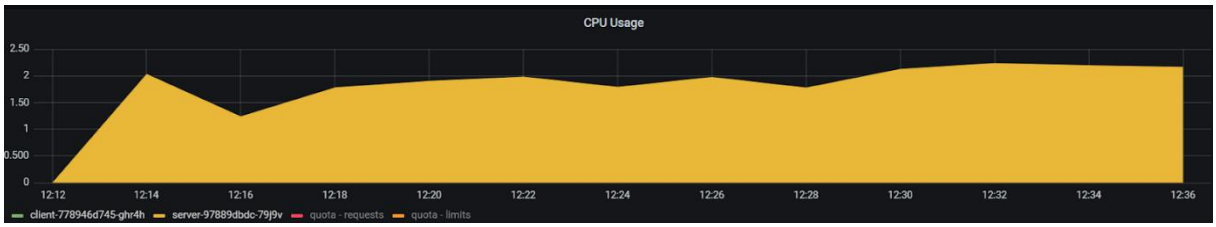

Figura 6.16- Uso de CPU pela aplicação 2-tier

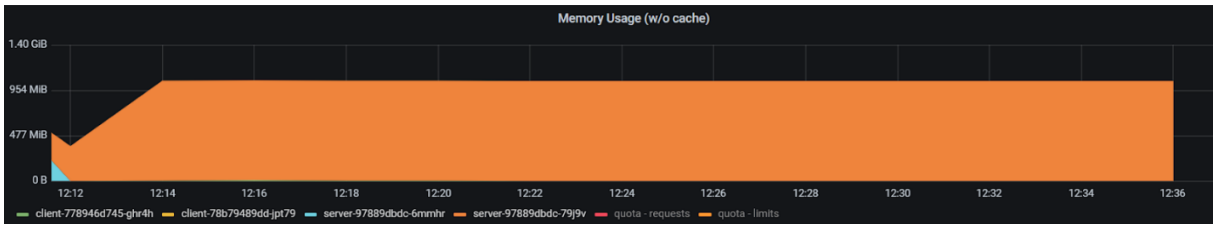

Figura 6.17- Uso de memória pela aplicação 2-tier

Os valores apresentados devem-se ao facto de esta aplicação estar a utilizar *stress-ng*, um serviço que permite o teste de recursos de um ambiente, o qual provoca um uso excessivo dos mesmos.

Da mesma forma, foi criada uma *dashboard* que permita analisar os recursos utilizados por cada *container* em cada *pod* de um *namespace* específico, de forma a perceber, nomeadamente, se os valores definidos como limites de memória e CPU são ultrapassados em algum instante. A Figura 6.18 e Figura 6.19 representam um exemplo da utilização de CPU e memória, respetivamente, do *container* "server" dos *pods* da aplicação 12-tier.

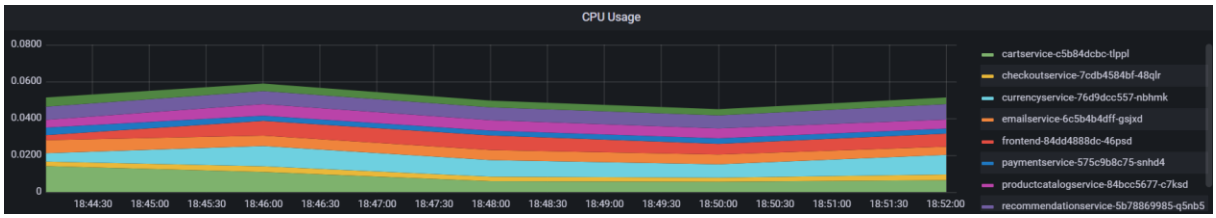

Figura 6.18- Uso de CPU por container "server" da aplicação 12-tier

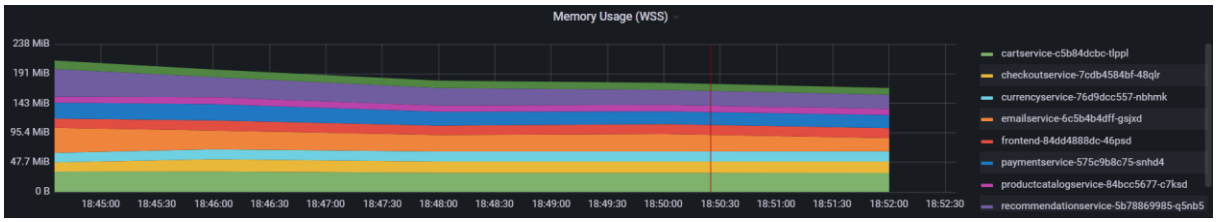

Figura 6.19- Uso de memória por container "server" da aplicação 12-tier

Outra métrica bastante importante a ter em conta é o tráfego de rede (referência ao requisito [R28]), isto porque, ao ser analisado podem ser verificadas conexões excessivas (como por exemplo, picos de tráfego de rede que não são recorrentes). Desta forma, a Figura 6.20 demonstra um dos gráficos disponíveis pelo Grafana que permite visualizar picos de tráfego de rede na aplicação 2-tier. Estas variações devem-se ao facto de esta aplicação estar a utilizar

iperf3, uma vez que são gerados tráfegos de rede com uma determinada duração, de forma a testar esta mesma métrica.

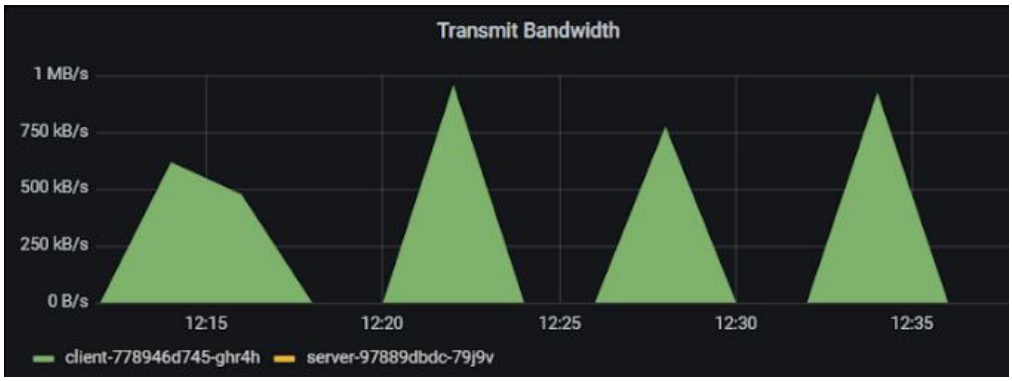

Figura 6.20- Bandwidth transmitida pela aplicação 2-tier

Uma aplicação XR utilizada por múltiplos utilizadores pode se comportar de forma diferente tendo em conta múltiplos fatores, como, número de instâncias, número de utilizadores, o próprio ambiente ou até as configurações distintas de cada utilizador. Como tal, a monitorização do uso de recursos no nível do *pod* (ou do nível de aplicação) é bastante útil para detetar situações não saudáveis ou para prever comportamentos de serviço individuais.

6.4 Migração automatizada de microserviços através de Operadores Ansible

O processo de monitorização deve ser realizado de forma constante, com o objetivo de avaliar continuamente o estado do ambiente e prever situações que possam provocar a indisponibilidade dos serviços. É importante que ao serem identificadas situações anómalas, como a utilização excessiva de CPU e memória, sejam realizadas tarefas que mantenham o funcionamento do ambiente. Assim, nesta secção será analisado e cumprido os objetivos da Tabela 6.4, os quais visam a implementação de mecanismos que permitam a automação de tarefas em processos de monitorização, como forma de evitar a indisponibilidade dos serviços, conforme o requisito [R_NF1] e [R_NF6].

Tabela 6.4- Descrição dos Requisitos Cumpridos na Secção 6.4

ID	Requisito	Descrição
[R17]	Implementação de mecanismos que permitam a migração de *containers* entre nós de um mesmo cluster	Implementação de Operador Kubernetes que permite a monitorização de cada nó e a posterior migração de serviços em caso de valores de CPU e memória muito elevados no mesmo
[R18]	Implementação de mecanismos que permitam a migração de *containers* entre clusters	Implementação de Operador Kubernetes que permite a monitorização de cada cluster e a posterior migração de serviços em caso de valores de CPU e memória muito elevados no mesmo

A solução implementada para este fim pode ser representada por o conceito de *closed-loop*, o qual deve fazer uma monitorização constante do ambiente e recolha de métricas que permitam analisar o estado do ambiente. A verificação do estado do ambiente deve ser realizada com base em algumas métricas, como a capacidade das máquinas/cluster, o tráfego de rede em cada cluster e a disponibilidade de cada elemento. Após a análise dos dados recolhidos, os mesmos devem ser avaliados de forma a perceber se é necessário a intervenção e em caso afirmativo, deve haver o planeamento e a execução das ações necessárias. Desta forma, o ciclo deve possuir a inteligência necessária para perceber o que poderão ser comportamentos anómalos, e como reagir a cada um dos possíveis casos.

Com o objetivo de automatizar estas tarefas, podem ser utilizados operadores K8s, operadores que permitem estender o comportamento do cluster sem a modificação de código do próprio Kubernetes, vinculando controladores a um ou mais recursos personalizados. Assim, estes operadores actuam como controladores de um recurso personalizado, nomeadamente a possibilidade de migrações de serviços. Os controladores implementados são responsáveis por analisar e monitorizar as caracterísitcas de rede e de recursos (por exemplo, uso de disco em máquinas e uso de CPU), e em caso de valores diferentes dos definidos podem executar as tarefas necessárias para contrariar este fator. É importante referir que apesar de estes valores serem definidos manualmente, os mesmos podem ser alterados, e inclusive podem ser aplicados dinamicamente através de um processo de inteligência artificial, que permita decidir quando é que estas operações devem ser executadas. Além disto, o mecanismo de inteligência artificial permite que possam ser utilizados outros valores para esta decisão, além dos valores de CPU e disco.

Estes operadores podem ser criados através de múltiplas ferramentas, nomeadamente através de Operator SDK que permite a criação de operadores baseados em *playbooks* Ansible. A implementação realizada foi efetuada em Ansible, uma vez que a escrita deste tipo de *playbooks* foi constante ao longo do trabalho e que a sua execução mostrou-se bastante eficiente.

Assim, de forma a permitir a migração de serviços entre nós de um mesmo cluster foi implementado um Operador K8s (referência ao requisito [R17]) em cada nó do cluster, conforme representado na Figura 6.21. No entanto, pode ser realizado apenas em um nó do cluster. Nesse caso, o operador terá que executar *playbooks* e não apenas uma *role*, uma vez que terá de efetuar tarefas em múltiplas máquinas em simultâneo e como tal, terá de ter acesso a todas as máquinas do cluster e às credenciais das mesmas, de forma a ser possível efetuar conexões SSH, a fim de monitorizar os valores necessários.

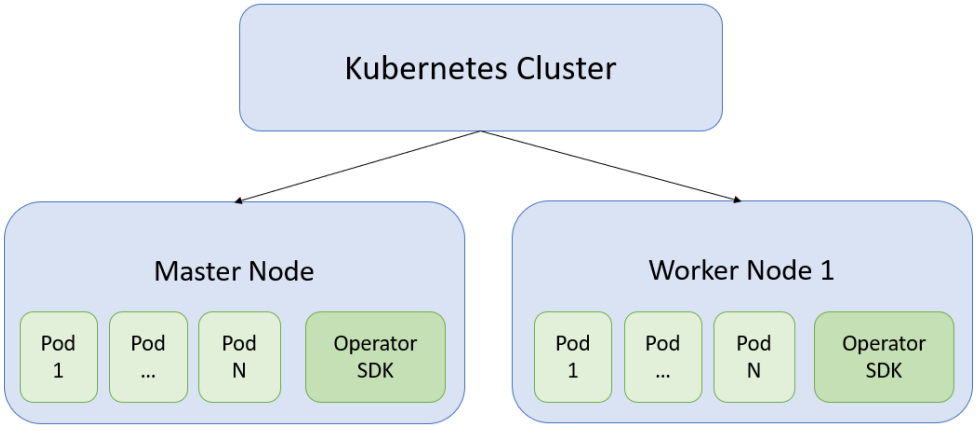

Figura 6.21- Representação da Implementação de Operador K8s para migração entre máquinas

É importante referir que o operador deve possuir volumes implementados, de forma a ter acesso à pasta local onde se encontram os ficheiros YAML para o *deployment* das aplicações. Assim, é necessário garantir que todos os ficheiros se encontram nessa mesma diretoria, uma vez que se assume que todos os componentes implementados no respetivo nó se encontram definidos em ficheiros YAML na diretoria "*/mr/zk*".

O operador irá efetuar os passos representados na Figura 6.22, onde inicialmente irá efetuar a constante monitorização do nó em que se encontra (passo 1), avaliando os valores de memória, disco e CPU. Neste caso, foi definido que uma situação de necessidade de migração de serviços ocorre quando o uso do disco é superior a 75%, o uso da CPU é superior a 75%, ou caso a memória disponível seja inferior a 200 MB. Quando estes valores são apresentados, é realizada uma verificação aos diversos nós do Cluster (passo 2), de forma a validar se existem mais nós disponíveis e se os mesmos possuem mais recursos disponíveis do que o nó original, através do comando "*kubectl top node*". Este comando permite verificar não só os diversos nós do cluster e o seu estado, como também a utilização de CPU e memória de cada um. Caso algum nó do cluster se encontre indisponível por algum motivo, o seu estado aparece como "*<unknown>*" e como tal não é contabilizado para uma possível migração. Em caso de disponibilidade de outros nós, é escolhido o nó que possui menor utilização de recursos no momento, para ser feito o *deployment* dos microserviços. A migração de serviços para outro nó é efetuada através da anotação "*nodeName*" que permite definir o nó para a implementação de um *pod*. No entanto, é de referir que se esta anotação já estiver definida no ficheiro YAML deve ser eliminada, de forma a não causar conflitos. Após a alteração do valor desta anotação, o processo de *deployment* dos microserviços (passo 3) é realizado através da utilização do comando "*kubectl replace*", uma vez que este comando procede à remoção automática dos *pods* no cluster original e posterior implementação, que será realizada no nó de destino, devido à seleção de nó efetuada anteriormente.

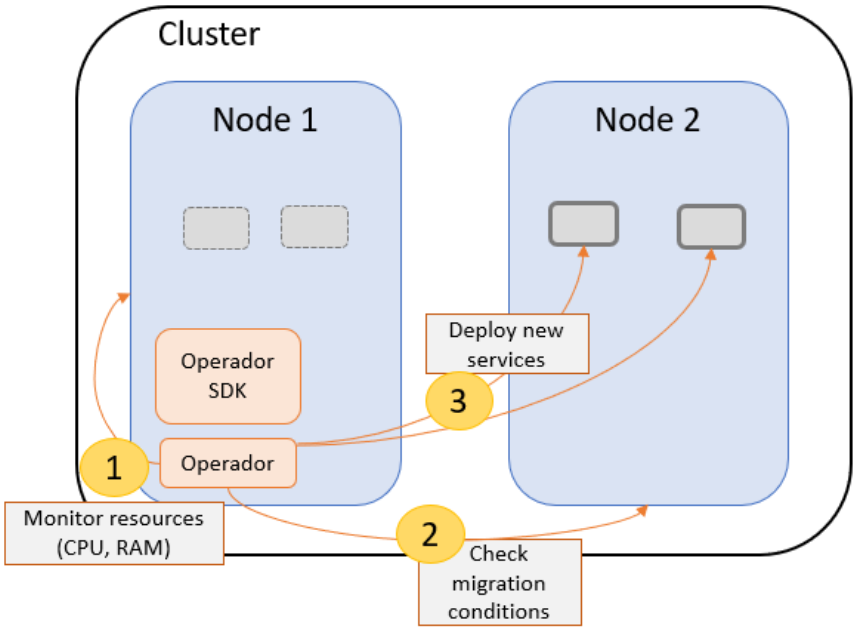

Figura 6.22- Representação dos passos para execução do operador de migração de microserviços entre nós do mesmo Cluster

Desta forma, quando o operador é inicializado o primeiro *output* a ser apresentado nos *logs* são os valores de memória, disco e CPU monitorizados, tal como mostra a Figura 6.23. Após

esta monitorização e a apresentação destes dados é avaliada a necessidade de migração dos serviços. Para ser realizada a migração os microserviços, todas as condições devem ser verdadeiras.

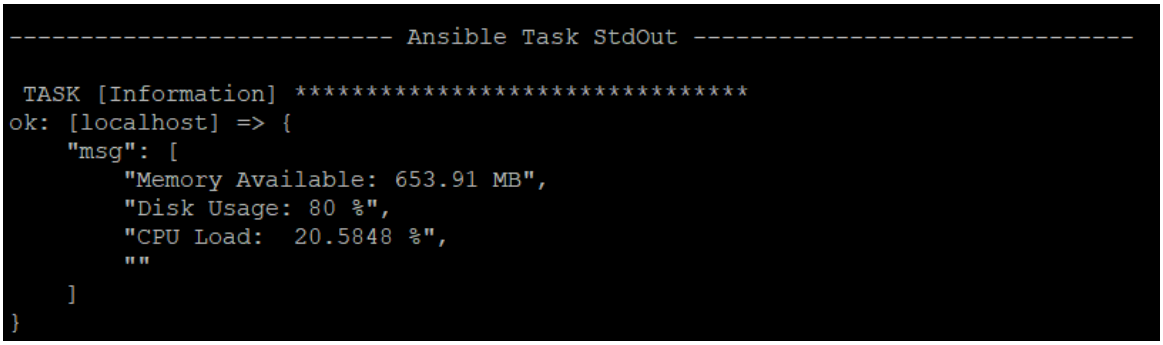

Figura 6.23- Apresentação de Valores de Monitorização de Recursos

A migração apenas será efetuada caso exista outra máquina disponível para tal e caso essa mesma máquina possua mais recursos disponíveis do que a original. Estas verificações são efetuadas através do comando "*kubectl top node*", que permite verificar os nós existentes num respetivo cluster bem como a sua memória e utilização de CPU. Quando um nó, por algum motivo, não se encontra disponível, estes valores aparecem como "*<unknown>*". Assim, no caso de não existir mais nós no cluster, ou de os existentes possuirem menos recursos disponiveis do que o original é apresentada uma mensagem nesse sentido e não é efetuada a migração de serviços, tal como representado na Figura 6.24.

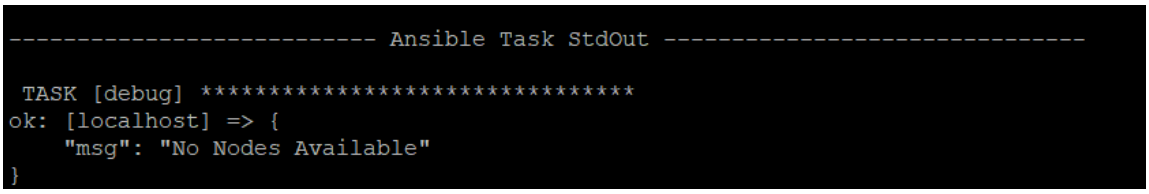

Figura 6.24- Apresentação de mensagem de não disponibilidade de migração

Por outro lado, se houver disponibilidade de migração de serviços para outro nó do mesmo cluster, essa migração é efetuada para o nó que apresentar mais recursos disponíveis. Esta migração é efetuada através da atribuição dos *pods* ao nó em questão através do campo "*nodeName*" que pode ser definido no ficheiro YAML do respetivo *pod*. Após isto pode ser feito o reposicionamento do *pod*, o qual será apagado no nó original e imediatamente aplicado no novo nó.

De forma semelhante, foi criado um Operador K8s que permite a monitorização e migração de serviços entre clusters, em caso de necessidade (referência ao requisito [R18]) em apenas um nó de cada cluster, conforme representado na Figura 6.25. Neste caso é necessário implementar um ambiente multi-cluster tal como será referido na Secção 6.5, de forma a possibilitar uma monitorização e implementação mais direta através da definição de contextos. Se este ambiente não for implementado é necessário efetuar as respetivas ações em múltiplos clusters em simultâneo através de SSH, o que acarreta um acréscimo de tempo

na implementação e a definição prévia dos endereços IPs das máquinas do cluster, e das suas respetivas credenciais de acesso.

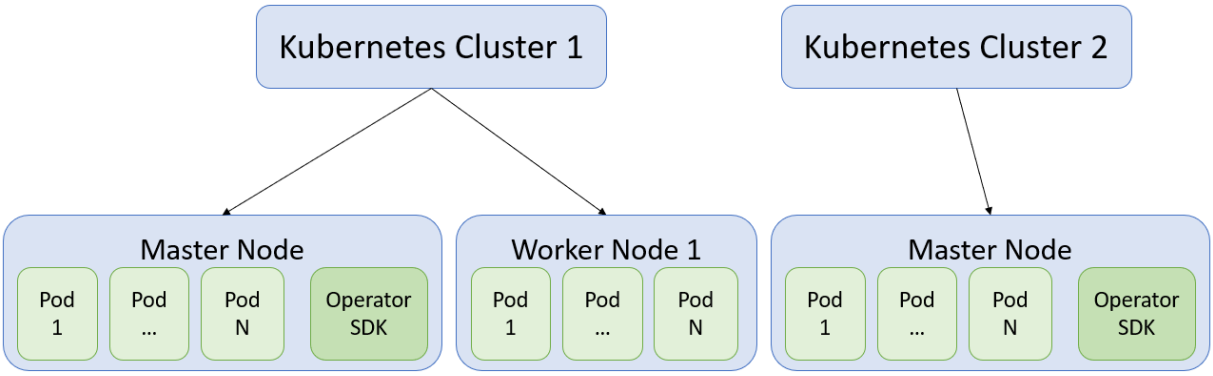

Figura 6.25- Representação da Implementação de Operador K8s para migração entre clusters

O operador irá efetuar os passos representados na Figura 6.26. Assim, inicialmente o operador irá efetuar a constante monitorização do cluster em que se encontra (passo 1), avaliando os valores de memória e CPU, através do comando "*kubectl top node*". Este comando permite recolher os valores para cada nó do cluster, como tal, torna-se possível efetuar a média desta utilização por cluster. Posteriormente é validada a necessidade de migração de microserviços, a qual se assume necessária caso o uso médio do disco no cluster seja superior a 75% e o uso médio da CPU seja superior a 75%. Caso estes valores se verifiquem, é realizada uma verificação das condições de migração para outro cluster, nomeadamente, se existe efetivamente outro cluster e se o mesmo possui nós disponíveis. Apenas caso exista esta disponibilidade é que é feito o procedimento de *deployment* dos componentes, e a posterior eliminação dos mesmos no Cluster corrente. É importante referir que a comunicação com outros clusters é realizada através de contextos, como tal, é fundamental que sejam aplicados volumes para partilha de ficheiros na diretoria "*/mr/zk*" e que nesta exista não só os ficheiros YAML como o ficheiro "*config-demo*" que possua a definição de todos os clusters existentes no ambiente, para possibilitar esta ação. A migração apenas será efetuada para os componentes que se encontrem definidos em ficheiros YAML existentes nesta diretoria, assim como apenas serão avaliados outros clusters se existir o ficheiro "*config-demo*" na diretoria.

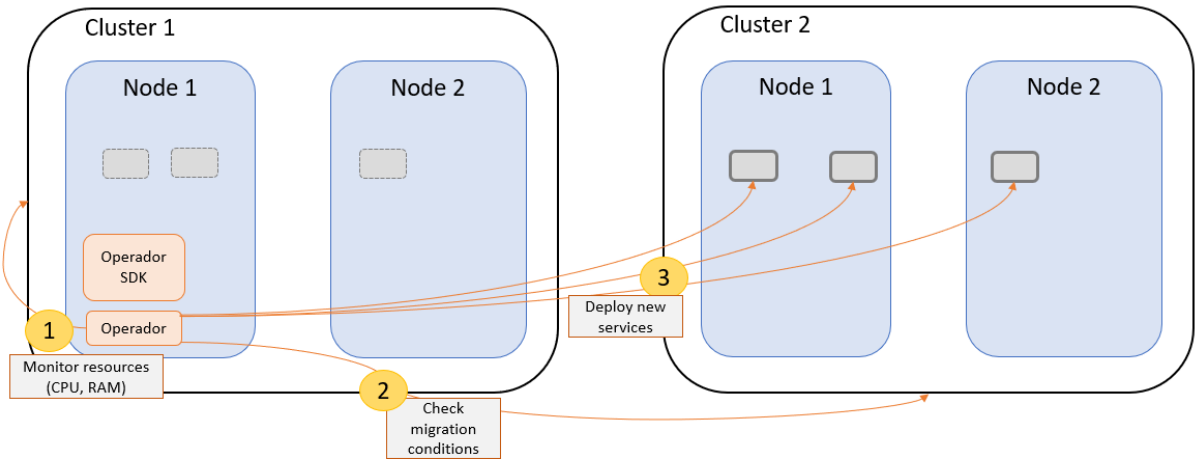

Figura 6.26- Representação dos passos para execução do operador de migração de microserviços entre Clusters

Neste caso, o operador permite a recolha autônoma dos valores solicitados e, de acordo com o valor retornado, aparecerá uma mensagem a informar se é necessário migrar os serviços

para outro cluster. Em caso de necessidade de migração dos serviços, são avaliados os contextos definidos e a sua disponibilidade, uma vez que pode só haver um contexto definido ou o mesmo não se encontrar disponível. Se este for o caso, aparecerá uma mensagem tal como representada na Figura 6.27 e não será efetuada nenhuma migração de serviços.

```
--------------------------- Ansible Task StdOut -------------------------------

 TASK [debug] ********************************
ok: [localhost] => {
    "msg": "New Context is not available"
}
```

Figura 6.27- Apresentação de mensagem de indisponibilidade de migração de serviços para outro contexto

Por outro lado, se houver disponibilidade de outro contexto para a implementação de serviços, o procedimento irá continuar, apresentando o contexto atual e o contexto para onde será efetuada a migração de serviços, tal como representado na Figura 6.28.

```
--------------------------- Ansible Task StdOut -------------------------------

 TASK [debug] ********************************
ok: [localhost] => {
    "msg": "Current context context-cluster1"
}

------------------------------------------------------------------------------
{"level":"info","ts":1654612291.415873,"logger":"logging_event_handler","msg":"[playbo
.com/v1, Kind=Verify","event_type":"playbook_on_task_start","job":"9146162617133990142

--------------------------- Ansible Task StdOut -------------------------------

TASK [Get Contexts] ************************************************************
task path: /opt/ansible/playbook.yml:55

------------------------------------------------------------------------------
{"level":"info","ts":1654612292.218476,"logger":"logging_event_handler","msg":"[playbo
d=Verify","event_type":"runner_on_ok","job":"8776926351659079364","EventData.TaskArgs"

--------------------------- Ansible Task StdOut -------------------------------

 TASK [debug] ********************************
ok: [localhost] => {
    "msg": "New Context context-cluster2"
}
```

Figura 6.28- Apresentação do contexto atual e do contexto para onde será efetuada a migração

Após a migração ser finalizada, os serviços são eliminados do contexto atual e pode ser comprovada a sua migração através da visualização dos recursos do novo contexto. Para o caso, foi efetuada a migração de um *pod* "*nginx-test*", tal como representado na Figura 6.29.

```
root@k3s-master:~/operator-verify# kubectl --context context-cluster1 get pods -n test
No resources found in test namespace.
root@k3s-master:~/operator-verify# kubectl --context context-cluster2 get pods -n test
NAME         READY   STATUS    RESTARTS   AGE
nginx-test   1/1     Running   0          91s
```

Figura 6.29- Validação da migração de serviços entre clusters

6.5 Integração e Deployment de cenários Multi-Cluster

Nesta secção será analisado e cumprido os objetivos da Tabela 6.5, os quais visam a implementação de mecanismos que permitam a abstração e gestão de aplicações em múltiplos clusters. Assim, será introduzida a temática de multi-clusters.

Tabela 6.5- Descrição dos Requisitos Cumpridos na Secção 6.5

ID	Requisito	Descrição
[R19]	Implementação de mecanismos que permitam a abstração de gestão de *containers* integrados em múltiplos clusters em simultâneo	A ferramenta Rancher permite a criação e orquestração de aplicações multi-cluster através de Fleet

Cada vez mais, os clusters Kubernetes têm crescido, tanto em número quanto em tamanho. Isso se deve a fatores de escalabilidade, restrições geográficas ou estratégias de provedores (por exemplo, vários clusters de diferentes provedores). Esta realidade é também existente no presente ambiente, uma vez que os clusters podem estar dispersos ao nível de tipo de provedor de nuvem utilizado e/ou ao nível de localização geográfica.

Assim, e de forma a maximizar a eficácia do ambiente, surgem abordagens multi-cluster (referência ao requisito [R19]) que apesar de agregarem maior complexidade ao ambiente, também introduzem novos potenciais que visam facilitar a orquestração de aplicações nestes ambientes. Em particular, estas abordagens permitem migrar aplicações entre clusters de forma transparente, o que é muito útil em casos de falha de algum nó e/ou cluster.

No entanto, estas abordagens apresentam dois desafios essenciais:

- Sincronização entre os *Control Plane* para permitir a orquestração de todos os clusters.
- Interligação que permite o acesso aos serviços dos diferentes clusters.

6.5.1. Desafios de abordagens Multi-Cluster

Com o objetivo de sincronização de *Control Plane* entre múltiplos clusters são analisadas algumas abordagens como Servidor de API dedicado, GitOps e kubelet-virtual-based.

Um **servidor de API dedicado** permite a configuração de vários clusters através de um conjunto de APIs no *Hosting* cluster. Assim, quando é necessário realizar uma implementação específica, essa seleção é realizada estendendo as APIs tradicionais do Kubernetes com uma nova semântica que permite expressar essa mesma seleção. Um exemplo de ferramenta que utiliza essa abordagem é o KubeFed (Kubernetes Cluster Federation). Usando esta ferramenta, um cluster deve ser definido como um cluster *Host*, que se comunicará com os demais através do "*kubefedctl*". Este *host* possui os tipos de configurações de cada cluster, nomeadamente a definição de *templates* e o posicionamento dos recursos. No entanto, esta ferramenta não permite a orquestração automática de todos os recursos, portanto, quando um novo recurso é adicionado a um cluster, e se destina a ser gerido pelo KubeFed, é necessário fazer a mesma especificação no *Host* cluster.

Outra abordagem é o **GitOps**, que faz uso de um repositório Git para implementação de aplicações e atualização de recursos correspondentes no cluster. Com o GitOps, sempre que for necessário executar uma nova implementação, é necessário criar um GitRepo no cluster

Management. Depois disso, e de acordo com a configuração implementada, o GitRepo será propagado para os clusters necessários. A ferramenta Fleet é um exemplo de uso desta abordagem, integrada na ferramenta Rancher, é a estratégia recomendada pela ferramenta para a utilização de aplicações multi-cluster, uma vez que permite gerir implementações Kubernetes YAML ou *helm charts*.

Outra abordagem possível para o desafio de sincronização de clusters do *Control Plane* de controle é o **Virtual Kubelet**, uma implementação de *kubelet* do Kubernetes que permite a conexão entre o Kubernetes e outras APIs. Esta abordagem tem a representação de dois clusters, um cluster Superior e um cluster Inferior. Um provedor VK, *Upper* cluster, possibilita mapear um cluster remoto para um nó de cluster local. Assim, o cluster *Upper* é responsável por todo o escalonamento de recursos nos pods restantes, por meio dos nós virtuais.

Comparando as abordagens mencionadas, deve-se destacar o fato de a primeira abordagem (Servidor API dedicado) ter uma gestão centralizada. Isto pode causar problemas de segurança e disponibilidade, pois se o cluster *Host* falhar por algum motivo, torna-se impossível a orquestração do restante ambiente. Além disso, deve-se notar que a localização dos pods é apenas dinâmica ao usar o Virtual-Kubelet e, como tal, somente neste caso o agendamento e migração dos mesmos são possíveis.

Tendo em conta o segundo desafio, que se refere à comunicação entre serviços, é importante considerar abordagens que envolvam interoperabilidade com diferentes configurações de cluster e compatibilidade entre diferentes parâmetros de rede, uma vez que o ambiente pode ser bastante heterogêneo, e essas preocupações devem ser levadas em conta. Nesse sentido, são apresentadas abordagens como interconexão CNI-provedor, interconexão CNI-Agnostic e *Service Mesh*.

Abordagens de interconexão **CNI-provedor** visam ampliar a capacidade de CNI ("*Container Network Interface*") para gestão de múltiplas instâncias em diferentes clusters, como tal, depende necessariamente da respetiva implementação de CNI. Por exemplo, CiliumMesh é um exemplo desta abordagem, que estende a capacidade do CNI Cilium, que deve ser implementado em cada cluster. Somente após a sua implementação e exposição, para se tornar acessível externamente, é possível realizar comunicações entre os serviços, embora, neste caso, devam ser criadas chaves TLS para cada comunicação.

A fim de contrariar a necessidade de uma CNI específica, podem ser utilizadas abordagens como a **CNI-Agnostic**. Exemplos dessa abordagem são o Submariner e o Skupper. O Submariner, por exemplo, possui uma arquitetura centralizada baseada em um *broker* que recolhe informações sobre as configurações do cluster e devolve os parâmetros para uso. Skupper é um serviço de camada 7 (camada da aplicação) que interconecta vários clusters, permitindo a comunicação numa rede virtual.

Com Submariner, a rota de tráfego é obtida pelo *Route Agent* que roteia o tráfego entre clusters para o *Gateway Engine* ativo. Cada cluster possui um *Gateway Engine* (que pode ser implementado em mais de um nó para tolerância a falhas), responsável por estabelecer e gerir os túneis seguros para os outros clusters. Para facilitar a troca de dados entre os *Gateway Engines*, permitindo que eles se descubram, o componente *Broker* deve ser acessível por todos os clusters. Com esta ferramenta, é possível a monitorização constante dos clusters conectados por meio do *Gateway*, pois periodicamente *pinga* cada cluster e recolhe métricas (por exemplo, latência média). Caso alguma comunicação falhe, o seu estado de conexão é marcado como falhado. Nestes casos, é possível habilitar o modo *LoadBalancer*, para poder

estabelecer comunicação mesmo em caso de ocorrência de *failover*, atualizando para o novo *Gateway* ativo disponível.

As abordagens de ***Service Mesh*** também podem ser usadas em ambientes multi-cluster, uma vez que essas abordagens introduzem um *container sidecar* como *proxy* para fornecer vários recursos, nomeadamente permitir o roteamento de tráfego de malha entre clusters distintos. O Istio é um exemplo desta abordagem, que possibilita, também, a implementação de instâncias do *Control Plane* em mais do que um cluster primário, a fim de aumentar a disponibilidade do ambiente.

A utilização do Istio é realizada com a implementação de clusters Primários, que possuem um *Control Plane*, responsável por configurar todas as comunicações entre instâncias dentro da malha. Clusters sem *Control Plane* são designados por clusters Remotos, que devem aceder ao *Control Plane* para obter as configurações necessárias. O ambiente pode ser composto por mais do que um cluster Primário, neste caso, cada um deles deve receber a configuração das implementações através do Servidor API. No entanto, essa duplicação de configurações, caso seja necessário implementar alterações, exige que todas sejam alteradas, processo que não é automatizado. O Istio permite o isolamento entre clusters, zonas ou regiões, através da utilização de mais do que um *Control Plane*, desta forma, mesmo tendo um ambiente multi-cluster, é possível ter isolamento entre diferentes partes e maior disponibilidade do sistema. Neste contexto, esta ferramenta permite que, em caso de falha em algum cluster, o tráfego possa ser encaminhado temporariamente para outro cluster.

De acordo com a análise a estas abordagens, vale ressaltar que a maior discrepância se verifica no facto de as abordagens CNI-Provedor, além de dependerem de CNI, apresentam desvantagens no facto de não suportarem IPs sobrepostos, nem suporte a tecnologia *Secure Tunnel*.

É importante mencionar que apesar das vantagens de se utilizar uma solução genérica, a escolha deve ser influenciada pela necessidade de suporte a conexões UDP, bem como por mecanismos que não aumentem significativamente a latência, pois o ambiente deve possuir suporte a serviços XR. De acordo com esses requisitos deve-se notar que a ferramenta Istio suporta o *proxy* de qualquer tráfego TCP, mas o mesmo não acontece com as comunicações UDP. Este protocolo funcionará normalmente, mas não pode ser usado em componentes somente de *proxy* (por exemplo, *gateways* de entrada ou saída), pois eles funcionarão sem qualquer intercetação de *proxy*. No entanto, para superar esse problema, diferentes abordagens podem ser utilizadas, como a combinação do CNI Cilium com a arquitetura Istio. Dessa forma, o tráfego que não for observado pelo Istio será observado pelo Cilium, nomeadamente, tráfego UDP e IPv6. Além disto, essa abordagem traz outros benefícios em termos de segurança, como proteção em caso de *sidecars* comprometidos, pois desta forma o Cilium funciona como uma segunda verificação ao tráfego que circula na malha.

6.5.2. Comunicação entre serviços de diferentes Clusters

Inicialmente, e de forma a permitir uma comunicação eficiente entre serviços de múltiplos clusters é importante que os nós dos diferentes clusters possuam conectividade entre si, e que as redes dos diferentes clusters permitam a comunicação inter-cluster. Além disto, e de forma a não causar conflitos, deve ser garantido que todos os clusters e respetivos nós possuem intervalos de PodCIDR distintos, e endereços IP únicos.

De forma a permitir a comunicação entre diferentes serviços e uma maior facilidade de acesso entre os mesmos, esse mesmo acesso pode ser configurado através de ficheiros de configuração do Kubernetes, que permitem configurar clusters, utilizadores e contextos distintos para diferentes clusters. A configuração de cada cluster pode ser efetuada individualmente (um ficheiro de configuração por cada cluster), ou em simultâneo no mesmo ficheiro. Desta forma primeiramente é possível definir os vários clusters através da definição do seu servidor, onde deve ser indicado o endereço do Servidor Kubernetes, e a definição da autoridade de certificados. Após isto, cada utilizador deve ser definido através do certificado e da sua respetiva chave. Estes valores já são automaticamente definidos quando são inicializados os clusters, como tal não é preciso a sua criação, apenas a sua reutilização do ficheiro de configuração original.

Por último, é necessário definir os diversos contextos, cada contexto pode representar acesso ao mesmo cluster, apenas diferenciando o utilizador ou o *namespace*. Ou por outro lado, cada contexto pode diferenciar o cluster a utilizar e o utilizador correspondente. Assim, quando for utilizado um comando *kubectl* num determinado contexto, o mesmo será executado no cluster e *namespace* definido, utilizando as credenciais do utilizador definido.

Desta forma, a partir dos clusters implementados na Secção 6.2, foi criado o ficheiro de configuração que permite o acesso entre os mesmos, tal como demonstrado na Figura 6.30.

```
root@template-debian11:/# kubectl --kubeconfig=/mr/zk/config-demo config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://172.16.16.157:6443
  name: cluster-k3s
- cluster:
    insecure-skip-tls-verify: true
    server: https://172.16.16.154:6443
  name: cluster-rke2
contexts:
- context:
    cluster: cluster-k3s
    namespace: default
    user: user-cluster-k3s
  name: context-cluster-k3s
- context:
    cluster: cluster-rke2
    namespace: default
    user: user-cluster-rke2
  name: context-cluster-rke2
current-context: context-cluster-k3s
kind: Config
preferences: {}
users:
- name: user-cluster-k3s
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: user-cluster-rke2
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

Figura 6.30- Output do comando "kubectl config view"

Isto permite que através de um nó de um respetivo cluster, tendo o ficheiro de configuração definido como anteriormente representado, seja possível executar tarefas e analisar recursos no cluster Remoto.

No entanto, esta abordagem apenas permite o conhecimento e a gestão de clusters, não permitindo a comunicação entre serviços de múltiplos clusters. Neste sentido, uma possível solução passa pela implementação de Cilium, em conjunto com Istio, de forma a permitir ter uma solução adaptada às necessidades do ambiente, nomeadamente, o *proxy* de comunicações UDP, uma vez que o Istio apenas possui o suporte para comunicações TCP. Desta forma quando o tráfego não passa pelo *sidecar* do Istio (isto significa que não são aplicadas quaisquer políticas de segurança pelo Istio), irá obrigatoriamente passar pelo Cilium, o qual irá aplicar as políticas de segurança pré-definidas, independentemente do protocolo. O Cilium suporta tráfego TCP, UDP, ICMP e IPv6, no entanto, se por algum motivo o protocolo utilizado na comunicação não for suportado, o pacote é descartado.

Assim, a implementação destas ferramentas em conjunto pode ser realizada conforme representado na Figura 6.31, onde o Cilium permite o roteamento e a comunicação de/para serviços do cluster, e o Istio permite a comunicação entre clusters distintos. Desta forma, a comunicação TCP entre serviços passa sempre pelo *sidecar* do Istio, e pelo Cilium, possuindo duas fontes de aplicação de políticas, enquanto o tráfego UDP, por exemplo, apenas passará pelo Cilium. É de referir que nesta arquitetura foi utilizada a implementação de um cluster Primário com um cluster Remoto, o que significa que o Cluster2 (cluster Remoto) irá utilizar o *Control Plane* do Cluster1. O componente *istiod* irá observar os Servidores API de ambos os clusters em procura por *endpoints*, de forma a permitir a descoberta de serviços. Os serviços comunicam através do Gateway implementado no cluster Primário. Outra possível abordagem seria a implementação de dois clusters Primários, onde cada um teria um componente *istiod* que iria comunicar diretamente com o Servidor API de ambos os clusters.

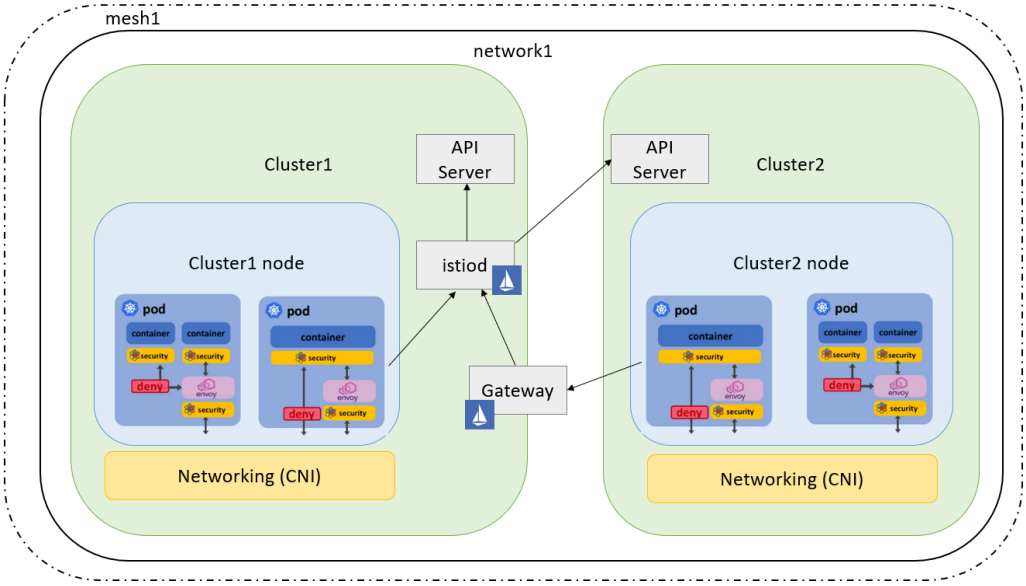

Figura 6.31- Representação da Arquitetura de Istio+Cilium em ambiente Multi-cluster

Tendo esta implementação em consideração, podem ser destacadas algumas vantagens deste uso, tais como:

- Segurança de protocolos diferentes de TCP e IPv4: Apesar de Istio apenas possuir suporte para aplicação de políticas a protocolos TCP e IPv4, o Cilium permite tornar este suporte mais abrangente, possuindo suporte a protocolos UDP, ICMP e IPv6.
- Proteção contra *sidecars* maliciosos: O Istio não aplica regras de segurança ao próprio *sidecar*, como tal, se o mesmo estiver comprometido tem acesso à rede e pode inclusive manipular as suas regras definidas. Neste sentido, o Cilium permite definir políticas de segurança de forma a validar estes casos.
- Segurança entre containers do mesmo *pod*: O Cilium permite definir políticas de forma a obrigar todo o tráfego a passar pelo *proxy sidecar*.
- Controle de dados encriptados: Istio não consegue inspecionar cabeçalhos HTTP de conexões encriptadas por TLS, no entanto, o Cilium fornece visibilidade e controle sobre estes dados.

A plataforma Rancher permite esta mesma integração, no entanto, de forma individual. Ou seja, permite a escolha de um CNI aquando da criação de clusters, à exceção de clusters K3s, no entanto, o CNI pode ser implementado posteriormente. Relativamente ao Istio, a plataforma também permite a instalação desta ferramenta em cada cluster integrado. Apesar disto, a plataforma não permite automatizar esta instalação em caso de ser um ambiente multi-cluster, como tal, a instalação destes componentes deve ser realizada de forma externa à plataforma, de forma a não criar conflito com instalações individuais.

6.5.3. Estratégias para deployment de aplicações em cenários multi-Cluster

Com o objetivo de aumentar a disponibilidade e escalabilidade dos serviços, podem ser implementadas estratégias como a implementação de aplicações multi-Cluster, onde aplicações podem ser implementadas em vários clusters em simultâneo. Além disto, esta estratégia também possui vantagens a nível de conformidade com regulamentações, ou seja, cada cluster poderá ser adaptado para atender às regulamentações específicas da sua localização geográfica, e assim abranger o maior número de localizações que possam ser necessárias. Desta forma podem ser enumeradas algumas vantagens da sua utilização e implementação, tal como:

- Aumento da disponibilidade da aplicação, uma vez que a sua implementação é realizada em múltiplos clusters em simultâneo, podendo ser implementados em diferentes localizações geográficas;
- Conformidade com a regulamentação de múltiplas localizações;
- Maior facilidade de implementação da aplicação, visto que esta implementação não necessita de ser efetuada individualmente em cada cluster, podendo ser facilitada, minimizando o tempo necessário e a interação humana;
- Permite a integração de múltiplos fornecedores Kubernetes em simultâneo, não sendo necessário que todos os clusters possuam as mesmas características.

No entanto, é importante referir que esta estratégia, embora possa ser implementada com a ajuda de ferramentas que permitam esta característica, requer uma maior gestão dos diversos clusters, da aplicação e das suas necessidades.

Através do Rancher é possível implementar aplicações multi-cluster de forma simplificada, uma vez que a mesma utiliza a ferramenta Fleet. A sua arquitetura está representada na Figura 6.32, onde o Fleet *Controler* é implementado e inicializado automaticamente no cluster onde foi instalado o Servidor Rancher, e posteriormente em cada cluster que é implementado através do Rancher, é implementado um Fleet *Agent*. Quando é implementada uma aplicação multi-cluster, podem ser escolhidos os clusters onde se deseja fazer a implementação, uma vez que todos os clusters já possuem o componente Fleet incorporado.

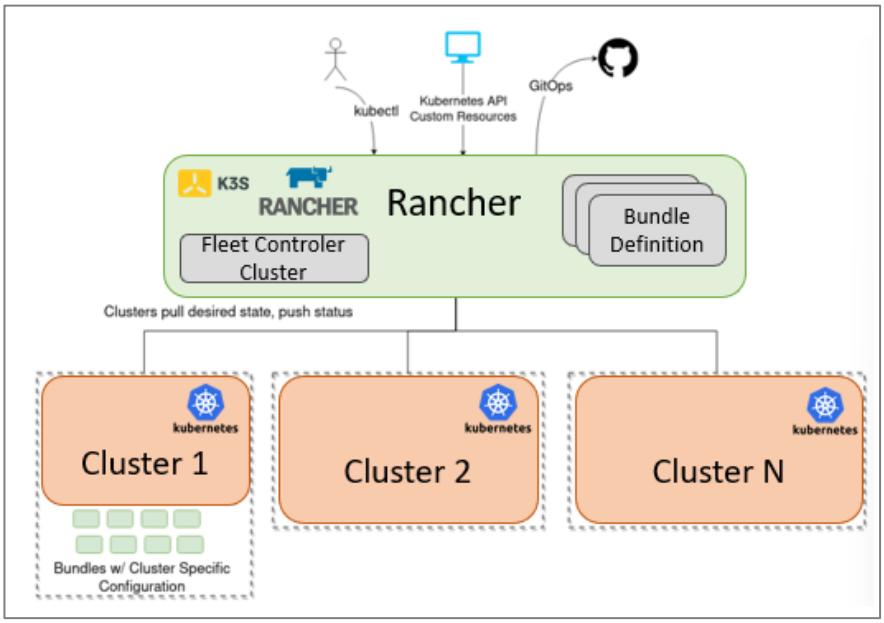

Figura 6.32- Arquitetura da utilização de aplicações multi-cluster, adaptado de [92]

Assim, através da definição de um GitRepo é possível implementar uma aplicação em múltiplos clusters em simultâneo, onde se torna possível diferenciar valores para cada cluster, como por exemplo, o número de réplicas ou o tipo de serviço. Além disto, permite que sejam especificados os clusters para o seu desenvolvimento, ou que sejam utilizados seletores que permitam agrupar clusters, por exemplo, no caso de haver mais do que um cluster de testes, não se torna necessário identificar o nome dos clusters explicitamente, basta dizer que pertencem ao ambiente “teste”.

Esta solução prevê-se bastante vantajosa uma vez que permite simplificar a implementação de aplicações em clusters diferentes, uma vez que cada cluster pode ter necessidades e recursos distintos, e desta forma pode ser feita a definição de valores específicos para cada um dos mesmos. Além disto, a utilização de GitRepos pode ser utilizada para cluster individual, ou seja, através da definição de um GitRepo, pode ser realizada uma implementação única, em apenas um cluster, tal como demonstrado na Figura 6.33.

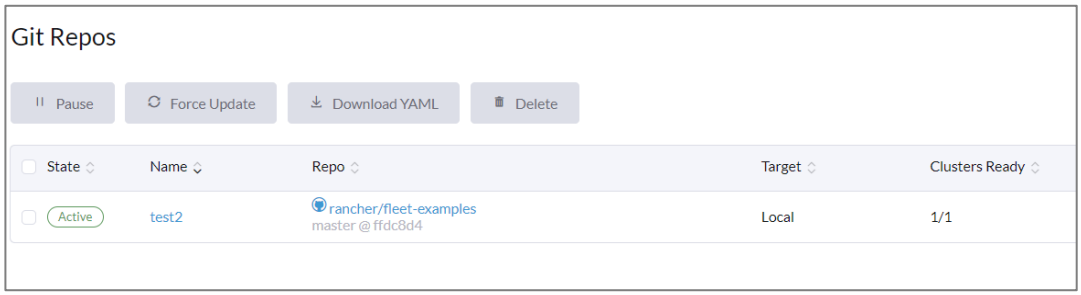

Figura 6.33- Representação da implementação de GitRepo em apenas um cluster

6.6 Testes Funcionais

De forma a integrar os mecanismos implementados na arquitetura representada na Figura 6.34. A implementação dos clusters foi realizada através da ferramenta Rancher, a qual possibilita a criação de clusters RKE2 ou K3s. Foi criado um cluster RKE2, o qual possui duas máquinas que não se encontram no domínio OpenStack e um cluster K3s, que possui o *Master* externo ao domínio OpenStack, e duas máquinas *workers* integradas no domínio em nuvem.

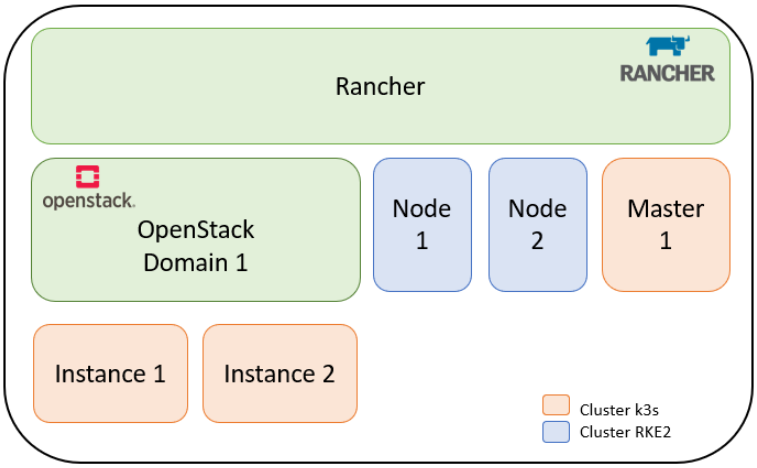

Figura 6.34- Representação da Arquitetura Implementada

Desta forma foi elaborada uma comparação entre os diferentes clusters, de forma a perceber se os problemas de latência mencionados na Secção 6.2 foram evitados ou se a solução implementada ainda possuía latências elevadas. Assim, foi realizada a implementação de três *containers* nos dois domínios e registados o seu tempo de *deployment*. Primeiramente, foi inicializada a implementação dos *containers* cuja imagem ainda não existia nas máquinas, onde foi registada uma diferença bastante significativa entre os tempos dos dois clusters. Enquanto o cluster RKE2 demorou 60 segundos a realizar o *pull* de uma imagem "nginx", o cluster K3s demorou 250 segundos (4x mais), e o mesmo se registou nos restantes *pulls*. Depois disto foi realizada a implementação dos mesmos serviços nos dois clusters, desta vez, como já tinha sido feito o *pull* inicial das imagens, o tempo não engloba esse processo. No entanto, também neste caso foi registada uma diferença significativa do tempo de implementação (desde o agendamento do *pod* até ao *Readiness Probe* dar o primeiro sucesso). Enquanto o cluster RKE2 demorou apenas 9 segundos, em média, na implementação de um servidor *web*, o cluster K3s demorou 49 segundos (cerca de 5x mais). A diferença do tempo referida pode ser observada na Figura 6.35, onde são apresentadas as médias de cada um dos tempos registados em 10 tentativas para cada caso.

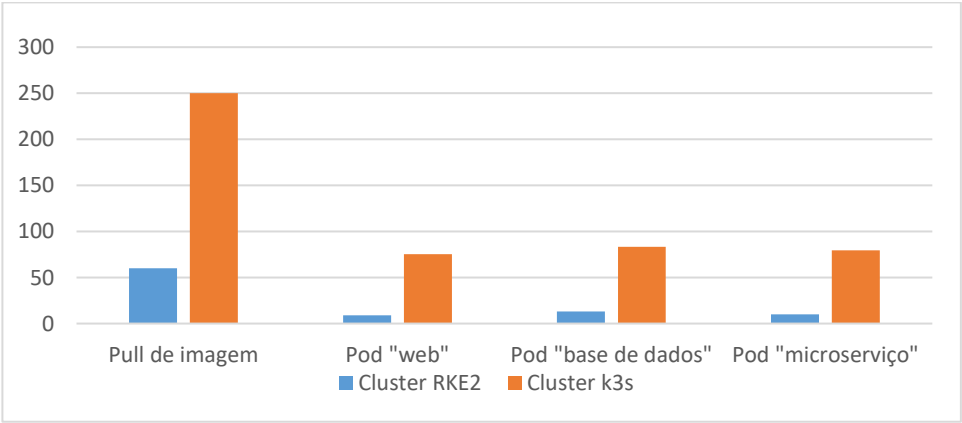

Figura 6.35- Representação dos Tempos de implementação entre cada cluster

No caso de ser feito o *pull* das imagens apenas se as mesmas não se encontrarem localmente é necessário ter em atenção que acarreta uma maior complexidade na manutenção das mesmas, devido ao número de nós a gerir e uma vez que estamos a falar de ambientes em nuvem. Neste sentido, é necessário ter em atenção que priorizar o tempo de implementação em detrimento da complexidade pode nem sempre ser uma opção adequada, e como tal deve ser avaliada conforme o ambiente e as suas necessidades. Para contornar este problema, uma das possibilidades é o uso de técnicas de *cache* inteligente, que permite a pré-busca das imagens necessárias dos serviços e armazenamento das mesmas nos nós ou nas proximidades dos mesmos.

Neste sentido, algumas abordagens foram analisadas, uma vez que este aparenta ser um problema recorrente em ambientes em nuvem. Como tal, neste momento já existem muitas possibilidades de implementação de DaemonSet (componente que garante que todos os nós, ou os nós especificados corram uma cópia de um determinado *pod*) ou operadores K8s que permitam efetuar o *pull* das imagens de forma prévia e transmitir as imagens para todos os nós do cluster. No caso, foi utilizado um operador K8s, designado por kube-fledged[1]. Este operador permite criar e gerir um armazenamento de imagens diretamente em cada um dos nós do cluster. Assim, devem ser definidas as imagens necessárias (que podem ser armazenadas em todos os nós ou em nós específicos), e atualizar o armazenamento, após isto as imagens serão *pulled* em cada um dos nós.

Por exemplo, no cluster K3s foi criada a *cache* que pode ser visualizada na Figura 6.36, onde são definidas as imagens que devem ser recolhidas, nomeadamente a imagem “nginx”, “mysql” e “microsvc” para todos os nós do cluster, uma vez que não é feita a especificação de nenhum nó em concreto. Por outro lado, são definidas as imagens que devem ser *pulled* no nó que possui a *label* “*dedicated: master*”, neste caso apenas os nós especificados irão possuir estas imagens.

```
"spec": {
    "cacheSpec": [
        {
            "images": [
                "nginx:latest",
                "mysql:latest",
                "prateeksingh1590/microsvc:1.1"
            ]
        },
        {
            "images": [
                "claudiatavares/operator-verify:v0.5.0",
                "claudiatavares/operator-verifymachine:v0.3.0"
            ],
            "nodeSelector": {
                "dedicates": "master"
            }
        }
    ],
```

Figura 6.36- Conteúdo do ficheiro de imagecaches implementado no cluster K3s

O *pull* das imagens é realizado de forma praticamente instantânea, assim que este ficheiro seja alterado é logo procedido à atualização das imagens armazenadas em todos os nós. Desta forma, o *pull* das imagens pode ser realizado antes da implementação dos *pods*, de forma que estes quando forem executados, não necessitem de efetuar esta ação, e que o seu tempo de *deployment* seja menor.

[1] https://github.com/senthilrch/kube-fledged

Além disto, foram implementados os Operadores K8s referidos na Secção 6.4, no entanto, e de forma a evitar os problemas de latência mencionados anteriormente, os testes realizados foram os representados na Figura 6.37 para a experimentação do Operador de migração entre nós do mesmo cluster, onde foi realizada a migração de *pods* do nó "Node1" do cluster RKE2 para o *worker* disponível.

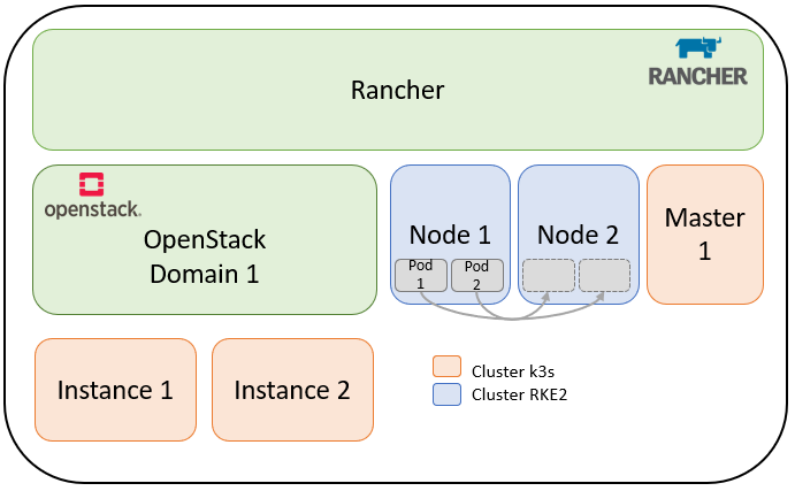

Figura 6.37- Representação da Migração de Pods entre nós do mesmo cluster

Da mesma forma, foi realizado o teste do Operador K8s para migração de componentes entre clusters, onde foi efetuada a migração de serviços do cluster RKE2 para o cluster K3s conforme representado na Figura 6.38. Como não é feita nenhuma especificação do nó onde deverá ser efetuada a implementação, o cluster de destino é que fará o agendamento dos *pods*. Esta migração não só abrange a implementação de *pods* como de qualquer componente do Kubernetes que esteja referido no ficheiro YAML, inclusive volumes.

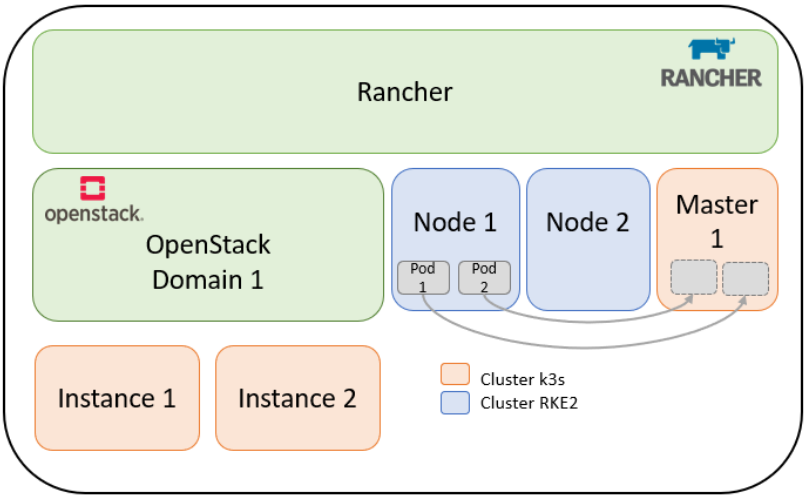

Figura 6.38- Representação da Migração de Pods entre clusters

É de realçar que os resultados obtidos com a migração de *pods* entre nós do mesmo cluster, e entre clusters distintos foram de sucesso, uma vez que nos dois casos foi possível fazer a migração corretamente. O tempo de migração, no caso de migração dentro do mesmo cluster é de 25 segundos, no caso de migração entre clusters é de 30 segundos, um tempo bastante aceitável e satisfatório. Na Figura 6.39 é possível observar a mudança de nó, ou seja, o *pod*

"webserver" começa por ser inicializado no nó *Master*, o qual depois é terminado e inicializado no nó *worker*, uma vez que foi necessário efetuar a migração do mesmo.

```
webserver   0/1     Terminating   0          33s   10.42.198.20   master-n
ode   <none>           <none>
webserver   0/1     Pending       0          0s    <none>         worker01
-node   <none>           <none>
webserver   0/1     Pending       0          1s    <none>         worker01
-node   <none>           <none>
webserver   0/1     ContainerCreating   0          1s    <none>         wo
rker01-node   <none>           <none>
webserver   0/1     ContainerCreating   0          1s    <none>         wo
rker01-node   <none>           <none>
webserver   1/1     Running             0          4s    10.42.105.17   wo
rker01-node   <none>           <none>
```

Figura 6.39- Observação da migração de pod "webserver" entre nós do mesmo cluster

Da mesma forma, na Figura 6.40 é possível observar a migração do *pod* "webserver" entre diferentes clusters, o primeiro terminal demonstra que o *pod* é terminado no cluster currente, neste caso, o RKE2. E posteriormente, é feita a inicialização deste mesmo *pod* no cluster K3s, uma vez que foi necessário proceder à migração do mesmo.

```
webserver   0/1     Terminating         0          18s
^Croot@master-node:~#
```

172.16.16.157 - PuTTY

```
root@template-debian11:~# kubectl get pods -n test --watch
NAME        READY   STATUS    RESTARTS   AGE
webserver   0/1     Pending   0          0s
webserver   0/1     Pending   0          0s
webserver   0/1     ContainerCreating   0          0s
webserver   1/1     Running             0          12s
root@template-debian11:~#
```

Figura 6.40- Observação da migração de pod "webserver" entre clusters

Assim, neste capítulo foi dado um especial enfâse ao fator de disponibilidade e latência do ambiente, isto significa que, em todos os mecanismos implementados foi sempre tido em consideração estes aspetos. A latência á um fator importante uma vez que estamos a falar de um ambiente de serviços XR, e como tal a latência deve ser o mínimo possível de forma a não prejudicar a experiência do utilizador. Da mesma forma, a disponibilidade dos microserviços e dos próprios clusters deve ser uma constante, uma vez que se torna fundamental que não haja problemas de indisponibilidade no ambiente. Tendo isto em consideração, o ambiente implementado prevê a falha de domínios em nuvem, ou de algum cluster, e em caso de isto acontecer, efetua as ações necessárias para evitar a indisponibilidade dos microserviços. Da mesma forma, é realizada uma monitorização constante do ambiente, de forma que em caso de alguma atividade anómala, possam ser tomadas as ações necessárias para evitar problemas de indisponibilidade ou de propagação de algum tipo de atividade maliciosa. Esta monitorização pode se tornar mais automatizada através de mecanismos de inteligência artificial que permitam a decisão das ações e dos passos de mitigação a serem realizados.

Em forma de conclusão, as integrações realizadas neste capítulo foram efetuadas com sucesso. A ferramenta utilizada permite a criação de clusters, quer sejam constituídos por nós de domínios em nuvem ou não, e permite também a sua monitorização através da disponibilização de instalação de ferramentas como Prometheus e Grafana. No entanto estas ferramentas não permitem monitorizar alguns parâmetros importantes, como o *deployment*

time. Apesar disto, tendo em consideração as capacidades e os componentes do Kubernetes este problema é solucionável a partir da utilização de *probes* que permitem verificar o estado dos serviços implementados. Além disto, e uma vez que a ferramenta não disponibiliza a possibilidade de efetuar migrações entre clusters, foi analisada a funcionalidade de Operadores K8s, que permitem efetuar ações automatizadas conforme os valores obtidos, por exemplo sobre a utilização de CPU ou de memória dos mesmos. Por fim, foi avaliada a possibilidade de implementação Multi-cluster a qual se vê bastante útil em ambientes em nuvem. Assim, de uma forma geral, a solução implementada foi de encontro às necessidades, no entanto, com a ressalva de que a latência em ambientes em nuvem é maior do que a esperada, e como tal, foram tomadas considerações de forma a contornar a situação, nomeadamente através do pré-*pull* das imagens a utilizar, tal como referido neste capítulo.

6.7 Síntese de Capítulo

Na Secção 6.1 são descritas as ferramentas que são utilizadas nas secções seguintes e que permitem a orquestração de clusters de uma forma eficiente, bem como é feita a integração entre as ferramentas utilizadas.

Posteriormente, na Secção 6.2 foi realizada esta mesma orquestração, onde foi realizada a implementação e orquestração de um cluster com múltiplos nós, os quais não estão inseridos em nenhum domínio em nuvem. E posteriormente foi realizada a mesma tarefa, no entanto, usando nós em domínios de nuvem, ou seja, foi realizada a integração com os domínios OpenStack.

De seguida, na Secção 6.3 foi feita a descrição da monitorização realizada aos diversos clusters, com base nas ferramentas e nas análises que são possibilitadas através do Rancher, uma vez que esta plataforma possui integração com ferramentas de monitorização que permitem facilitar este processo e obter métricas bastante úteis e diversificadas. No mesmo ponto, foram também criadas novas *dashboards* com base em métricas que se revelam importantes no estudo.

Na Secção 6.4 foi realizada a análise de mecanismos que permitam automatizar processos após a observação de métricas anómalas, através da implementação de Operadores K8s que permitem a monitorização dos valores de recursos do Cluster e respetivos nós e a posterior migração de microserviços em caso de necessidade.

Na Secção 6.5 foi feita a exploração de implementações Multi-cluster, quais os desafios destas implementações e de que forma podem ser contornáveis, assim como a especificação e possíveis soluções a implementar.

Por último, na Secção 6.6 foi detalhada a integração dos diversos mecanismos implementados num único ambiente e realizada uma avaliação ao desempenho dos mesmos. O qual demonstrou ser um resultado positivo, apesar dos valores de latência verificados nos nós inseridos em domínios em nuvem. Neste caso, mostrou-se importante integrar mecanismos de pré-*pull* de imagens, que permitem diminuir esta latência.

Assim, da perspetiva da estrutura geral do documento, o capítulo que agora finda permite a contextualização da implementação e das atividades realizadas com o objetivo de orquestração de clusters.

Capítulo 7 Conclusões e Trabalho Futuro

O presente capítulo sintetiza as conclusões retiradas da investigação e das implementações realizadas. Além disto, é apresentado o trabalho futuro, como forma de seguimento ao trabalho apresentado neste documento.

7.1 Conclusões

Cada vez mais, o conceito de orquestração de *containers* tem ganho uma elevada escala e o seu uso tem sido crescente. Isto deve-se ao facto de permitir a orquestração de infraestruturas constituídas por um ou mais ambientes distintos de uma forma mais simplificada. A utilização de orquestradores permite que aplicações baseadas em microserviços sejam implementadas de forma automatizada, geridas de forma unificada e que sejam escalados conforme as suas próprias necessidades e do ambiente.

No âmbito do projeto europeu CHARITY surge a necessidade de criação de uma relação entre infraestruturas de baixa latência de forma a facilitar as necessidades de aplicações emergentes, através do alcance dos benefícios da inteligência e orquestração automatizada em nuvem, *edge* e recursos de rede. Esta inteligência deverá ser o mais autónoma possível, e como tal, surgem conceitos e abordagens como Zero-touch e *closed-loops* que são tidos em consideração. No entanto, não pode ser esquecido que a orquestração inteligente de múltiplos domínios acarreta a necessidade de implementação de mecanismos e APIs que permitam a interação com os diversos, independentemente do seu tipo e da sua localização.

O contributo do trabalho aqui documentado passa pela análise e investigação de possíveis abordagens que possam ser utilizadas, não só para a implementação e orquestração de múltiplos domínios distintos, como a integração dos mesmos com mecanismos de orquestração inteligente.

Assim, o trabalho apresenta e documenta os mecanismos criados e implementados para a melhoria do ambiente, de forma que o mesmo se torne mais automatizado e que possa reagir a situações anómalas. Desta forma, foram realizados processos de automatização de tarefas que visam evitar a necessidade de interação humana constante em múltiplos domínios de nuvem. Objetivo este que se considera completado na íntegra, na medida em que todos os mecanismos referidos como requisitos foram implementados e testados. Com este tópico, foram explorados conceitos de nuvem, orquestração e a implementação de *playbooks* Ansible. No entanto, é de referir que apesar da ferramenta Rancher possibilitar a integração com domínios OpenStack e de ter sido implementado um cluster com instâncias OpenStack, o *deployment* do cluster foi feito de forma manual e não de forma automática através deste mesmo provedor. Isto deveu-se a problemas de latência apresentados com frequência que não foram solucionáveis no tempo em questão.

Posteriormente, foram explorados ambientes de cluster de *containers*, nomeadamente a sua orquestração e monitorização, de forma a serem implementados mecanismos autónomos

para mitigação de possíveis ataques que possam levar à falha de disponibilidade do sistema. Este objetivo considera-se cumprido, nomeadamente melhorado através da solução implementada para o pré-*pull* de imagens.

Em suma, o trabalho correu conforme o expectável apesar de alguns entraves que atrasaram algumas implementações, nomeadamente, problemas de acesso à infraestrutura. Além disto, é também de referir que o trabalho foi desenvolvido no âmbito de um projeto europeu, como tal, algumas implementações poderiam ser mais exploradas e melhoradas, no entanto, a existência de *deadlines* e a necessidade de implementações em conjunto com outros parceiros obriga ao foco em determinados contextos.

7.2 Trabalho Futuro

No seguimento do trabalho documentado, sugere-se como trabalho futuro a integração de mecanismos que permitam a comunicação entre serviços de clusters distintos de forma a permitir que aplicações possam ser orquestradas em múltiplos clusters. Além disto, esta vertente é também importante pois possibilita que diferentes equipas trabalhem na mesma aplicação sem a necessidade de ser no mesmo ambiente.

Em relação à integração de vários clusters, com nós distribuídos, nomeadamente entre domínios de nuvem, é possível que a latência seja acrescida, como tal, e de forma a diminuir este fator incentiva-se a exploração de técnicas de *cache* inteligentes, que permitam o pré-pull das imagens necessárias e que as mesmas sejam mantidas em locais próximos dos nós. Esta exploração foi efetuada e implementada, no entanto existem diversas soluções que possibilitam diminuir o tempo necessário para o *pull* das imagens, diminuindo assim o tempo de implementação de cada aplicação. Assim, seria interessante que fossem implementadas mais soluções e que houvesse uma comparação entre as mesmas, de forma a perceber qual será a melhor abordagem para o ambiente.

Os conceitos aqui apresentados são aplicáveis no contexto da área de serviços XR, uma vez que as implementações demonstradas apresentam uma fase de estudo e exploração para ser aplicada em ambiente de serviços XR, no âmbito do projeto CHARITY, como tal, prevê-se que o trabalho seja uma abordagem inicial e um ponto inicial para o desenvolvimento final do projeto.

Referências

[1] “OneSource,” [Online]. Available: https://www.onesource.pt/. [Acedido em 2021].

[2] CNCF, “CNCF,” [Online]. Available: https://www.cncf.io/about/who-we-are/.

[3] “CHARITY,” [Online]. Available: https://www.charity-project.eu/en. [Acedido em 2021].

[4] “5G-EPICENTRE,” [Online]. Available: https://www.5gepicentre.eu/. [Acedido em 2021].

[5] “FUDGE-5G,” [Online]. Available: https://fudge-5g.eu/en. [Acedido em 2021].

[6] K. T. a. A. S. Jaybie A. de Guzman, “Security and Privacy Approaches in Mixed Reality: A Literature Survey,” *ACM Comput. Surv. 52, 6, Article 110,* vol. 52, nº 6, p. 37, 2019.

[7] J. P. D. C. I. M. M. U. M. S. Marcelo Amaral, “Performance Evaluation of Microservices Architectures using Containers,” IEEE 14th International Symposium on Network Computing and Applications, 2015.

[8] N. Kratzke, “About Microservices, Containers and their Underestimated Impact on Network Performance,” 2017.

[9] P. J. O. Z. Claus Pahl, “Microservices and Containers,” 2020.

[10] A. Gandini, “Stateful vs. Stateless: the good, the bad and the ugly.,” 30 dezembto 2019. [Online]. Available: https://www.proud2becloud.com/stateful-vs-stateless-the-good-the-bad-and-the-ugly/. [Acedido em 2022].

[11] L. A. Vayghan, M. A. Saied, M. Toeroe e F. Khendek, “Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes,” em *IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, 2019.

[12] p. B. F. B. L. F. G. M. R. M. A. P. Isam Mashhour Al Jawarneh, “Container Orchestration Engines: A Thorough Functional and Performance Comparison,” em *IEEE International Conference on Communications (ICC)*, 2019.

[13] Kubernetes, “Kubernetes,” 21 agosto 2021. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/. [Acedido em 6 setembro 2021].

[14] M. Kazla, “Etcd In Kubernetes,” the cloud report, [Online]. Available: https://the-report.cloud/etcd-in-kubernetes. [Acedido em 2022].

[15] Red Hat, “Red Hat,” [Online]. Available: https://www.redhat.com/pt-br/topics/microservices/what-is-a-service-mesh. [Acedido em 3 setembro 2021].

[16] K. Balaji, “MuleSoft,” 6 fevereiro 2020. [Online]. Available: https://blogs.mulesoft.com/dev-guides/microservices/what-is-a-service-mesh/. [Acedido em 3 setembro 2021].

[17] Y. L. J. G. Z. Z. Y. H. Wubin Li, “Service Mesh: Challenges, State of the Art, and Future Research Opportunities,” em *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2010.

[18] F. Moyer, “Comprehensive Container-Based Service Monitoring with Kubernetes and Istio,” em *SREcon18*, 2018.

[19] Cilium, “Getting Started Using Istio,” 2020. [Online]. Available: https://docs.cilium.io/en/v1.9/gettingstarted/istio/. [Acedido em 2022].

[20] Apache Kafka, “Kafka,” [Online]. Available: https://kafka.apache.org/intro. [Acedido em 2022].

[21] A. Berwaldt, “Apache Flume,” 26 outubro 2018. [Online]. Available: https://medium.com/apache-flume/introdu%C3%A7%C3%A3o-b5c0c97b5634. [Acedido em 2022].

[22] S. R. H. J. M. K. R. V. S. Victor Heorhiadi, “Gremlin: Systematic Resilience Testing of Microservices,” em *IEEE 36th International Conference on Distributed Computing Systems*, Nara, Japan, 2016.

[23] EndRun Technologies, “Network Time Synchronization,” EndRun Technologies, [Online]. Available: https://endruntechnologies.com/products/ntp-time-servers/network-time-synchronization. [Acedido em 2022].

[24] infostrech, “infostrech,” [Online]. Available: https://www.infostretch.com/resources/white-papers/chaos-engineering/. [Acedido em 27 setembro 2021].

[25] E. B. Nitin Sukhija, “Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus,” em *IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, Leicester, UK, 2019.

[26] K. G. Y. W. Hirokuni Kitahara, “Highly-Scalable Container Integrity Monitoring for Large-Scale Kubernetes Cluster,” em *IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, 2020.

[27] A. Abdelrazik, “Docker vs. Kubernetes vs. Apache Mesos: Why What You Think You Know is Probably Wrong,” 31 julho 2017. [Online]. Available: https://d2iq.com/blog/docker-vs-kubernetes-vs-apache-mesos. [Acedido em 2022].

[28] Microsoft, “What is Cloud Native?,” 11 outubro 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition. [Acedido em 2021].

[29] L. M. C. I. V. R. S. J. M. X. L. F. P. A. S. B. M. L. V. G. L. D. A. A. M. Carlos J. Bernardos, "Internet-Draft," *Multi-domain Network Virtualization,* p. 36, 12 setembro 2019.

[30] T. A. P. Rosado, "Implementação de uma infraestrutura de Cloud privada baseada em OpenStack," Coimbra, 2016.

[31] O. M. A. a. M. E. Sefraoui, "OpenStack: toward an open-source solution for cloud computing," vol. 55, nº 3, 2012.

[32] A. Rawal, "Introduction to OpenFlow Protocol," 7 fevereiro 2021. [Online]. Available: https://www.section.io/engineering-education/openflow-sdn/. [Acedido em 2022].

[33] Red Hat, "Red Hat," [Online]. Available: https://www.redhat.com/pt-br/topics/openstack. [Acedido em 4 outubro 2021].

[34] K. Pepple, Deploying OpenStack, Estados Unidos: O'REILLY, 2011.

[35] M. A. M. E. Omar Sefraqui, "International Journal of Computer Applications," *OpenStack: toward an open-source solution for cloud computing,* vol. 55, nº 3, 2012.

[36] A. V. D. G. Demétrius Roveda, "Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação (REABTIC)," *Understanding, Discussing and Analyzing the OpenNebula and OpenStack's IaaS Management Layers,* vol. 1., nº 15. , 2015.

[37] Apache CloudStack, "Apache CloudStack," [Online]. Available: https://cloudstack.apache.org/. [Acedido em 2022].

[38] E. D. K. V. G. M. B. P. B. B. B. P. M. Gideon Juve, "Data Sharing Options for Scientific workflow applications on Amazon EC2," 2009.

[39] T. S. E. N. Guohui Wang, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," 2010.

[40] "Amazon EC2," [Online]. Available: https://aws.amazon.com/pt/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc. [Acedido em 2021].

[41] Amazon, "Amazon," [Online]. Available: https://aws.amazon.com/pt/ec2/features/. [Acedido em 4 outubro 2021].

[42] N. N. A. E. Kostas Katsalis, "Multi-Domain Orchestration for NFV: Challenges and Research Directions," 15th International Conference on Ubiquitous Computing and Communications and 2016 8th International Symposium on Cyberspace and Security, 2016.

[43] L. Suomalainen, "Automatic Discovery of Host Machines in Cloudify-powered Cluster," Helsinki, 2019.

[44] Cloudify, "Cloudify," [Online]. Available: https://docs.cloudify.co/4.1.0/blueprints/overview/. [Acedido em 6 outubro 2021].

[45] A. P. F. d. A. Leonardo Rebouc¸as de Carvalho, “Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators,” em *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, 2020.

[46] P. B. S. Z. Desislava Ivanova, “Development of PaaS using AWS and Terraform for medical imaging analytics,” em *AIP Conference Proceedings 2048*, 2018.

[47] N. Naik, “Cloud-Agnostic and Lightweight Big Data Processing Platform in Multiple Clouds Using Docker Swarm and Terraform,” 2021.

[48] C. T. D. M. A. G. T. Luis Tomas Bolivar, “On the Deployment of an Open-Source, 5G-Aware Evaluation Testbed,” 2018.

[49] H.-H. C. B. T. O. S. W. K. B. S. K. Seokho Son, “Cloud SLA relationships in multi-cloud environment: models and practices,” Canberra, Australia, 2017.

[50] C. T. D. M. A. G. T. L. T. Bolivar, “2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud),” 2018.

[51] S. T. H. C. H. N. Nishant Kumar Singh, “Automated provisioning of application in IAAS cloud using Ansible configuration management,” Dehradun, India, 2015.

[52] M. S. J. K. K. Z. J. P. M. K. P. Masek, “Unleashing Full Potential of Ansible Framework: University Labs Administration,” 2018.

[53] Z. T. G. Y. W. Yiran, “Design and implementation of continuous integration scheme based on Jenkins and Ansible,” 2018.

[54] R. B. Thiyagarajan, “Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible,” Dublin, 2020.

[55] M. Rajaprabha, “LibCloud: uma nuvem privada de recursos de biblioteca digital,” vol. 5, nº 2, 2013.

[56] Apache Libcloud, “Libcloud,” [Online]. Available: https://libcloud.readthedocs.io/en/stable/getting_started.html. [Acedido em 6 outubro 2021].

[57] G. G. Q. L. Y. G. X. Z. Xiaolong Wen, “Comparison of Open-Source Cloud Management Platforms: OpenStack and OpenNebula,” 2012.

[58] “Mist.io,” [Online]. Available: https://docs.mist.io/article/16-overview. [Acedido em 2021].

[59] Puppet, “The Puppet platform,” [Online]. Available: https://puppet.com/docs/puppet/7/platform_components.html. [Acedido em 2022].

[60] T. Kajinami, “puppet-openstack-integration,” [Online]. Available: https://opendev.org/openstack/puppet-openstack-integration#all-in-one.

[61] M. Ray, “Chef for OpenStack,” 27 junho 2012. [Online]. Available: https://www.chef.io/blog/chef-for-openstack. [Acedido em 2022].

[62] Juju, “JUJU,” [Online]. Available: https://juju.is/. [Acedido em 2022].

[63] N. Kracser, “Multi-cluster testing with kind and MetalLB,” 1 outubro 2020. [Online]. Available: https://banzaicloud.com/blog/multi-cluster-testing/. [Acedido em 12 novembro 2021].

[64] L. Marino, “Dynamic Application Placement in a Kubernetes,” Torino, 2020.

[65] H. G. C. K. E. E. Lars Larsson, “Decentralized Kubernetes Federation Control Plane,” 2020.

[66] G. P. J. T. E. E. M. A. Tamiru, “Instability in Geo-Distributed Kubernetes Federation: Causes and Mitigation,” 2020.

[67] “Kubernetes Cluster Federation,” [Online]. Available: https://github.com/kubernetes-sigs/kubefed. [Acedido em 2021].

[68] A. M. Luque, “Developing and Deploying NFV solutions with OpenStack, Kubernetes and Docker,” Catalunya, 2019.

[69] D. P. J. K. R. S. Pekka Pääkkönen, “Architecture for Enabling Edge Inference via Model Transfer from Cloud Domain in a Kubernetes Environment,” vol. 13, nº 1, 2020.

[70] “Rancher,” [Online]. Available: https://rancher.com/. [Acedido em 2021].

[71] E. P. B. S. J. A. T. M. A. AlZain, “Cloud Computing Security: From Single to Multi-clouds,” Hawaii, 2012.

[72] A. A. P. G. P. T. L. L. J. X. Hussain AlJahdali, “Multi-Tenancy in Cloud Computing,” em *IEEE 8th International Symposium on Service Oriented System Engineering*, 2014.

[73] W. S. Y. H. Z. H. W. B. G. Chang Jie Guo, “A Framework for Native Multi-Tenancy Application Development and Management,” em *9th IEEE International Conference on E-Commerce*, 2007.

[74] N. G. M. J. L. L. I. N. M. J. Bohli, “Security and Privacy-Enhancing Multicloud Architectures,” 2013.

[75] Intel, “Intel® Software Guard Extensions (Intel® SGX),” [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html.

[76] Arm, “TRUSTZONE FOR CORTEX-A,” [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-a.

[77] AMD, “AMD Secure Encrypted Virtualization (SEV),” [Online]. Available: https://developer.amd.com/sev/.

[78] O. B. S. M. S. C. Scott Rose, “Zero Trust Architecture,” [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf. [Acedido em 2022].

[79] D. L. P. C. E. R. P. H. G. Nathan F. Saraiva de Sousa, “End-to-End Service Monitoring for Zero-Touch Networks,” *Journal of ICT Standardization,* vol. 9_2, nº 10.13052/jicts2245-800X.923, pp. 91-112, 26 maio 2021.

[80] ETSI, “ETSI GS ZSM 002,” *Zero-touch network and Service Management (ZSM),* vol. 1.1.1, 8 setembro 2019.

[81] “ETSI GS ENI 005 V2.1.1,” *Experiential Networked Intelligence (ENI); System Architecture,* p. 174, dezembro 2021.

[82] C. B. a. T. Taleb, “AI-Driven Zero Touch Network and Service Management in 5G and Beyond: Challenges and Research Directions,” vol. 34, nº 2, 2020.

[83] “5GEx,” [Online]. Available: https://5g-ppp.eu/5gex/.

[84] “5G-TRANSFORMER,” [Online]. Available: http://5g-transformer.eu/.

[85] “ANASTACIA,” [Online]. Available: http://www.anastacia-h2020.eu/.

[86] “MiCADOscale,” [Online]. Available: https://micado-scale.eu/successful-completion-project-cola/.

[87] “INSPIRE-5Gplus,” [Online]. Available: https://www.inspire-5gplus.eu/. [Acedido em 2021].

[88] “MonB5G,” [Online]. Available: https://www.monb5g.eu/. [Acedido em 2021].

[89] “5GZORRO,” [Online]. Available: https://www.5gzorro.eu/. [Acedido em 2021].

[90] A. M. Nim Jayawardena, “microservices-demo,” 2022. [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo. [Acedido em 2022].

[91] Rancher, “Fleet,” 18 março 2022. [Online]. Available: https://github.com/rancher/fleet/tree/v0.3.9. [Acedido em 2022].