UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Comparação do Desempenho de Infraestruturas Virtualizadas de Elevada Disponibilidade Usando Hypervisors Nativos ou Containers: Microsoft Hyper-V Versus Docker

(Versão Definitiva Após Defesa Pública)

André Filipe Castro Louro

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º ciclo de estudos)

Orientador: Prof. Doutor Mário Marques Freire

Covilhã, Julho de 2018

Dissertação elaborada no Instituto de Telecomunicações, Delegação da Covilhã, por André Filipe Castro Louro, Licenciado em Engenharia Informática pela Universidade da Beira Interior, sob orientação do Doutor Mário Marques Freire, Investigador Sénior do Instituto de Telecomunicações e Professor Catedrático do Departamento de Informática da Universidade da Beira Interior, e submetida à Universidade da Beira Interior para obtenção do grau de mestre em Engenharia Informática.

Dedicatória

Primeiramente dedico esta dissertação aos meu pais por todo o apoio que me facultaram durante todo o meu percurso como estudante universitário. Aos excelentes professores do Departamento de Informática que me facultaram todo o conhecimento valioso para a minha futura vida profissional, em especial ao meu professor e orientador Mário Marques Freire, que me orientou, enriquecendo o meu conhecimento durante o meu percurso como aluno de mestrado.

Agradecimentos

Gostaria de agradecer à Universidade da Beira Interior, ao Departamento de Informática e ao Instituto de Telecomunicações por todo o incentivo à investigação e pelo apoio ao programa de mestrado; e agradecer ao Doutor Professor Mário Marques Freire pela excelente orientação durante todo o processo de elaboração da dissertação.

Resumo

A virtualização nativa ao nível do *hardware* é atualmente usada em *datacenters* para suporte à virtualização de infraestruturas de computação de elevada disponibilidade. Os *hypervisors* nativos têm sido a principal escolha para a construção de infraestruturas virtualizadas baseadas em *failover clusters* usando o *Failover Clustering* no caso do Microsoft Windows Hyper-V. Contudo, o recente surgimento da tecnologia Docker, a tecnologia de *containers* de *software* mais usada a nível mundial, tem tornado popular a virtualização ao nível do sistema operativo, permitindo criar, através do modo *Swarm*, infraestruturas virtualizadas de elevada disponibilidade. A virtualização ao nível do *hardware* apresenta vantagens em relação à virtualização ao nível do sistema operativo, de entre as quais se destacam o isolamento e a flexibilidade de aplicações, mas apresenta desvantagens tais como uma maior complexidade de implementação e lentidão na inicialização e no armazenamento de imagens de máquinas virtuais. Por outro lado, a virtualização nativa ao nível do *hardware* tem evoluído no sentido de melhorar o respetivo desempenho, em que as arquiteturas de *hypervisors* baseadas em virtualização completa com tradução binária estão a ser substituídas por arquiteturas de *hypervisors* com paravirtualização com suporte por *hardware*. Esta dissertação pretende avaliar e comparar o desempenho de duas infraestruturas virtualizadas de elevada disponibilidade, uma baseada na virtualização nativa ao nível do *hardware* usando o Microsoft Hyper-V com *Failover Clustering* e a outra baseada na virtualização ao nível do sistema operativo usando o Docker em modo *Swarm*. As infraestruturas virtualizadas implementadas consistiram num *failover cluster* com dois nós físicos e um servidor de armazenamento. Foi avaliado e comparado o desempenho destas infraestruturas usando quatro diferentes *benchmarks*, o IOzone cujo alvo de avaliação é o sistema de ficheiros, o RAMspeed que permite a medição do desempenho da memória do sistema, o iPerf3 para medições ativas da largura de banda na comunicação entre plataformas e o Geekbench que é um *benchmark* de processadores multiplataforma. Os resultados obtidos com estes quatro *benchmarks* mostram, de forma geral, que o desempenho do Docker em modo *Swarm* é superior ao desempenho do Microsoft Hyper-V com *Failover Clustering*. Foram também analisados os tempos de migração de máquinas virtuais e de *containers* naquelas duas infraestruturas virtualizadas, tendo-se observado tempos de migração da ordem de 18,4s para máquinas virtuais com sistema operativo Windows sobre o Hyper-V, tempos de migração da ordem de 15,35s para máquinas virtuais com sistema operativo Linux sobre o Hyper-V e tempos de migração de Docker *containers* em modo *Swarm* na ordem de 5,94s e tempos de migração de *Nested* Docker *containers* na ordem de 6,79s. Estes resultados mostram que o Docker apresenta um desempenho superior em relação ao Hyper-V em termos de tempos de migração de máquinas virtuais ou de *containers* de um nó para o outro do *failover cluster*.

Palavras-chave

Virtualização, máquina virtual, *container*, *hypervisor*, Docker, Hyper-V, desempenho e *benchmarking*.

Abstract

Native virtualization at the hardware level is currently used in datacenters to support virtualization of high availability computing infrastructures. Native hypervisors have been the main choice for building virtualized infrastructures based on failover clusters using Failover Clustering in the case of Microsoft Windows Hyper-V. However, the recent emergence of Docker technology, the most widely used software container technology worldwide, has made virtualization popular at the operating system level, allowing the creation, through the Swarm mode, of virtualized infrastructures of high availability. Virtualization at the hardware level has advantages over virtualization at the operating system level, among which application isolation and flexibility stand out, but it has drawbacks such as greater implementation complexity and slow boot and storage of images of virtual machines. On the other hand, native hardware virtualization has evolved to improve performance, where hypervisor architectures based on full binary-based virtualization are being replaced by hardware-supported paravirtualization hypervisor architectures. This dissertation aims to evaluate and compare the performance of two highly available virtualized infrastructures, one based on native virtualization at the hardware level using Microsoft Hyper-V with Failover Clustering and the other based on operating system virtualization using Docker in Swarm mode. The virtualized infrastructures implemented consisted of a failover cluster with two physical nodes and a storage server. The performance of these infrastructures was evaluated and compared using four different benchmarks, the IOzone whose evaluation target is the file system, the RAMspeed that allows the measurement of system memory performance, iPerf3 for active measurements of bandwidth in communication between platforms and the Geekbench which is a benchmark of multiplatform processors. The results obtained with these four benchmarks generally show that Docker performance in Swarm mode is higher than Microsoft Hyper-V performance with Failover Clustering. The migration times of virtual machines and containers in these two virtualized infrastructures were also analyzed, with migration times of the order of 18.4s for virtual machines with Windows operating system on Hyper-V, migration times of the order of 15.35s for virtual machines with Linux operating system over Hyper-V and Docker containers migration times in Swarm mode in the order of 5.94s and migration times of Nested Docker containers in the order of 6.79s. These results show that Docker delivers superior performance over Hyper-V in terms of migration times from virtual machines or containers from one node to another from the failover cluster.

Keywords

Virtualization, virtual machine, container, hypervisor, Docker, Hyper-V, performance and benchmarking.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AES	Advanced Encryption Standard
AFP	Apple Filling Protocol
ALU	Arithmetic Logic Unit
API	Application Programming Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
FFT	Fast Fourier Transform
FPU	Floating-point Unit
FTP	File Transfer Protocol
GEMM	General Matrix Multiplication
HTML	HyperText Markup Language
I/O	Input/Output
IP	Internet Protocol
IPC	Inter-process Communication
iSCSI	Internet Small Computer Systems Interface
JPEG	Joint Photographic Experts Group
KVM	Kernel-based Virtual Machine
LLVM	Low Level Virtual Machine
LZMA	Lempel-Ziv-Markov chain algorithm
MAC	Mandatory Access Control
MMU	Memory Management Unit
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
MMX	Matrix Math Extensions
NAS	Network Attached Storage
NAT	Network Address Translation
NFS	Network File System
NIC	Network Interface Controller
PDF	Portable Document Format
QEMU	Quick Emulator
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SGDT	Store Global Descriptor Table
SMB	Server Message Block

SMSW	Store Machine Status Word
SSE	Streaming SIMD Extensions
TCP	Transmission Control Protocol
TLB	Translation Lookaside Buffer
UDP	User Datagram Protocol
VM	Virtual Machine
VMCB	Virtual Machine Control Block
VMM	Virtual Machine Monitor
ZFS	Zettabyte File System

Capítulo 1

Introdução

1.1 Enquadramento da Dissertação

A mudança de paradigma da computação clássica para computação em nuvem [1], associada a um novo modelo de negócio, tem permitido oferecer às aplicações e serviços maior flexibilidade, melhor utilização de recursos, maior escalabilidade e adaptabilidade e redução de custos. Como consequência natural, a oferta deste tipo serviços por parte dos fornecedores de serviços *cloud* (*cloud services providers*) tem tido uma forte adesão das empresas a nível mundial. De acordo com o *State of the Cloud Report* da RightScale 2018 [2], numa entrevista a 997 profissionais técnicos de um conjunto amplo e transversal de organizações sobre a adoção de computação em nuvem, observou-se que 96% dos entrevistados usam computação em nuvem e que 81% dos entrevistados têm uma estratégia *multi-cloud*.

No cerne do paradigma de computação em nuvem está a virtualização, cuja ideia foi inicialmente proposta na década de 60 [3]. A virtualização pode ser implementada em várias camadas de abstração de um sistema de computador, desde o nível de arquitetura do conjunto de instruções do microprocessador até ao nível de aplicação, também designado por nível de processo, no qual a Java Virtual Machine (JVM) constitui um exemplo bem conhecido de máquinas virtuais de linguagens de alto nível. Esta dissertação foca-se na virtualização ao nível do *hardware* e na virtualização ao nível do sistema operativo.

A virtualização ao nível do *hardware*, normalmente associada à mudança de paradigma da computação em *desktops* para *datacenters*, oferece um desempenho excelente, um isolamento de aplicações muito bom e uma boa flexibilidade em termos de aplicações, à custa de uma elevada complexidade de implementação. Por outro lado, a virtualização ao nível do sistema operativo oferece um desempenho excelente, um fraco isolamento de aplicações e uma fraca flexibilidade em termos de aplicações, mas apresenta uma menor complexidade de implementação quando comparada com a virtualização ao nível do *hardware* [4]. Apesar de a virtualização ao nível do *hardware* se ter tornado na escolha preferencial para a implementação de *datacenters* de média ou grande dimensão, a virtualização ao nível do sistema operativo também pode ser interessante para a implementação de sistemas virtualizados em empresas.

Um exemplo ilustrativo sobre o recente interesse pela tecnologia de *containers* é o da Netflix [5], que em 2008 adotou uma estratégia de migração total para a nuvem e começou a migrar

toda a infraestrutura de computação *hosted* interna para a AWS (Amazon Web Services) [6, 7]. Atualmente, quase todos os serviços da Netflix são executados em máquinas virtuais (VMs) na AWS. Ao longo dos anos, a Netflix contribuiu para a criação de normas nativas na nuvem, tais como, microserviços fracamente acoplados (*loosely coupled microservices*) e infraestrutura imutável (*immutable infrastructure*), que se tornaram em melhores práticas neste setor da industria, tendo a migração total para a nuvem constituído um enorme sucesso para a Netflix. Contudo, apesar de já ter uma arquitetura nativa na nuvem, a Netflix está a investir na tecnologia de *containers*, abordando a adoção de *containers* para uma infraestrutura nativa existente na nuvem [6, 7].

Esta dissertação foca-se no estudo e comparação do desempenho de infraestruturas virtualizadas de elevada disponibilidade, baseadas em *failover clusters*, usando virtualização nativa ao nível do *hardware* ou virtualização ao nível do sistema operativo. Nesta dissertação, para a implementação da virtualização nativa ao nível do *hardware* foi considerado o *hypervisor* Microsoft Hyper-V [8] e para a implementação da virtualização ao nível do sistema operativo foi considerado o Docker [9].

1.2 Definição do Problema e Objetivos da Investigação

O problema abordado nesta dissertação consiste em avaliar e comparar o desempenho de infraestruturas virtualizadas de elevada disponibilidade usando virtualização nativa ao nível do *hardware* e virtualização ao nível do sistema operativo.

O principal objetivo da investigação consiste em implementar, configurar, testar, avaliar e comparar o desempenho de uma infraestrutura virtualizada ao nível do *hardware*, constituída por um *failover cluster* com dois nós usando como *hypervisor* nativo o Microsoft Hyper-V, e o desempenho de uma infraestrutura virtualizada ao nível do sistema operativo constituída por um *cluster* com dois nós correndo o Docker em modo Swarm.

1.3 Estratégia para a Resolução do Problema

A estratégia para a resolução do problema consiste numa abordagem experimental envolvendo a especificação e implementação em paralelo, no laboratório do grupo de investigação, de uma infraestrutura virtualizada de elevada disponibilidade com virtualização nativa ao nível do *hardware*, usando três máquinas físicas, e uma infraestrutura de elevada disponibilidade com virtualização ao nível do sistema operativo, recorrendo a outras três máquinas físicas. Após a realização de testes às duas infraestruturas implementadas de modo a verificar o correto funcionamento destas infraestruturas, será elaborado um estudo comparativo do desempenho

de ambas as infraestruturas de modo a colocar em evidencia os respetivos méritos.

1.4 Contribuições

Existem estudos de desempenho envolvendo a virtualização ao nível do *hardware* e ao nível do sistema operativo. Contudo, esses estudos focam-se em máquinas virtuais ou *containers* que não integram infraestruturas de elevada disponibilidade (*clusters*). Na opinião do autor, a principal contribuição desta dissertação consiste no estudo comparativo do desempenho de infraestruturas virtualizadas de elevada disponibilidade envolvendo virtualização ao nível do *hardware* e ao nível do sistema operativo, tendo para o efeito sido implementado um failover cluster baseado em Hyper-V e um *cluster* de Docker *Engines* (Docker em modo *Swarm*), respetivamente.

1.5 Limitações do Trabalho Desenvolvido

A quantidade de máquinas físicas disponibilizadas para a elaboração do estudo comparativo limitou a escala de implementação das infraestruturas de elevada disponibilidade ao nível de virtualização do *hardware* e do sistema operativo. Apesar de terem sido disponibilizadas seis máquinas físicas, a implementação das duas infraestruturas de elevada disponibilidade em simultâneo permitiu realizar estudos de desempenho de ambas as infraestruturas em simultâneo e a exploração da variação de parâmetros e *workloads* em experiências numa e noutra infraestrutura simultaneamente. Contudo, a implementação de ambas as infraestruturas em simultâneo conduziu a que os ambientes experimentais implementados fossem de reduzida dimensão e apenas usados para investigar o desempenho das duas infraestruturas referidas.

1.6 Organização do Documento

A dissertação encontra-se organizada em cinco capítulos principais. Os assuntos e organização dos principais capítulos desta dissertação podem ser resumidos da seguinte forma.

O Capítulo 1, relativo à Introdução, descreve o enquadramento da dissertação, o problema a resolver e os objetivos da investigação subjacentes a esta dissertação, a estratégia para a resolução do problema, as principais contribuições, as limitações do trabalho desenvolvido e a organização da dissertação.

O Capítulo 2 é dedicado ao *Background* e Estado da Arte, sendo este introduzido por uma breve introdução histórica da virtualização. Os conceito de virtualização, e de como esta é realizada

ao nível dos recursos (CPU, Memória e I/O) são apresentados, assim como os tipos de virtualização existentes. A exposição da informação acerca da virtualização baseada em *hypervisor* é realizada através da definição do conceito de *hypervisor*, da apresentação de exemplos e da explicação das arquiteturas, dos tipos e das técnicas de virtualização baseada em *hypervisors*. A exposição da informação acerca da virtualização baseada em *containers* e da virtualização *nested* é realizada através da definição dos conceitos, exemplos e da especificação das arquiteturas associadas a cada tipo de virtualização. A virtualização introduz *overheads* sendo este conceito explorado no capítulo, assim como o conceito de *benchmarking* que é crucial para a elaboração do estudo comparativo. Todos os conceitos anteriores constituem o *background* teórico necessário para a compreensão da temática da dissertação, e é realizada a exposição dos pontos principais dos trabalhos cuja temática esta relacionada com a deste documento.

No Capítulo 3 os ambientes experimentais são apresentados através da definição do *test bed* experimental, e da especificação de todos os recursos de *hardware* e *software* utilizados na componente experimental. Os procedimentos da implementação das infraestruturas de elevadas disponibilidade são expostos, através dos detalhes da implementação do armazenamento de dados e através de todas as configurações necessárias para a implementação das infraestruturas do *test bed*. As ferramentas usadas para realizar a análise de desempenho nas infraestruturas implementadas são descritas neste capítulo.

No Capítulo 4 a implementação de testes executados pelas ferramentas de *benchmark* é realizada, sendo depois elaborada a comparação experimental do desempenho das infraestruturas implementadas através de uma análise detalhada dos dados recolhidos pelas ferramentas de *benchmark*.

No capítulo 5 as conclusões sobre a temática da dissertação são elaboradas com base em toda a informação recolhida nos capítulos anteriores, determinando qual tecnologia de virtualização que possui melhor desempenho e em que cada cenário se enquadra melhor cada tecnologia. O trabalho futuro que pode resultar devido à elaboração desta dissertação é explicitado.

Capítulo 2

Background e Estado da Arte

2.1 Introdução

O conceito de virtualização teve a sua origem nos anos 60, quando a IBM investiu em soluções robustas de *time-sharing*, que consistiam no uso partilhado dos recursos de um computador entre vários utilizadores, com o objetivo de aumentar a eficiência dos utilizadores e dos recursos do computador partilhados. Este modelo representou um grande avanço das tecnologias de computadores [3].

O custo de fornecimento de capacidade de computação foi reduzido, permitindo a organizações e até mesmo entidades individuais, usarem computadores, sem a necessidade de possuir um [3]. Em 1974, Popek e Goldberg introduziram os requisitos da virtualização no artigo "Formal Requirements for Virtualizable Third Generation Architectures" [10], constituindo as *guidelines* para a virtualização de recursos de *hardware* [11].

Atualmente, os *datacenters* utilizam técnicas de virtualização para criar a abstração de *hardware* físico, criando grandes agregados de recursos lógicos, que consistem em CPUs, memória, discos, armazenamento de ficheiros, rede, aplicações. Os recursos são fornecidos aos utilizadores ou clientes em forma consolidada de máquinas virtuais ágeis e escaláveis [3].

Neste capítulo será abordado o conceito de virtualização, as diferentes arquiteturas de virtualização, as diferentes técnicas de virtualização existentes, as tecnologias de virtualização, *overheads* de virtualização, ferramentas de *benchmarking* e comparação ao nível de segurança das arquiteturas de virtualização.

2.2 Conceito de Virtualização

A virtualização é uma técnica para abstrair as características físicas de recursos computacionais de forma a que outros sistemas, aplicações, e utilizadores finais possam interagir com esses recursos. A máquina virtual é uma representação lógica de uma máquina em *software*. Através da dissociação do *hardware* físico do sistema operativo, a virtualização fornece mais flexibilidade operacional e aumenta a taxa de utilização do *hardware* físico subjacente [12].

Conforme referido em [13] por Campbell e Jeronimo, com a virtualização, os recursos computa-

cionais podem ser partilhados por vários utilizadores, as tecnologias de virtualização permitem que múltiplas máquinas virtuais, com sistemas operativos heterogéneos, corram em simultâneo de forma isolada, na mesma máquina física.

Ao emular um sistema de *hardware* completo, desde o processador à placa de rede, cada máquina virtual partilha um conjunto comum de *hardware*, sem o conhecimento que o conjunto está a ser utilizado por outras máquinas virtuais. O sistema operativo que se encontra em execução na máquina virtual apenas consegue observar um conjunto consistente e normalizado de *hardware*, independentemente do conjunto de *hardware* físico [13].

A virtualização fornece uma forma de isolamento e controlo de recursos, permite que vários *workloads* executem de forma eficiente em apenas uma máquina física e permite que servidores que tradicionalmente requeriam múltiplas máquinas, sejam consolidados em uma única máquina de custo efetivo através do uso de máquinas virtuais ou de *containers* [14].

2.3 Virtualização de Recursos

2.3.1 Virtualização da CPU

A arquitetura da CPU pode ser virtualizada se suportar a execução de instruções privilegiadas e não privilegiadas das máquinas virtuais na CPU em modo de utilizador enquanto que o *hypervisor* executa em modo *supervisor* [4]. Na figura 2.1 pode ser observado um exemplo de como é virtualizada a CPU usando a virtualização completa).

Existem múltiplos tipos de instruções [4]:

- Instruções não privilegiadas de máquinas virtuais que executam diretamente na máquina *host* para maior eficiência.
- Instruções privilegiadas que executam em modo privilegiado e são capturadas se executadas fora deste modo.
- Instruções de controlo que tentam mudar as configurações dos recursos a ser utilizados.
- Instruções sensíveis ao comportamento que têm diferentes comportamentos, dependendo da configuração dos recursos, incluindo as operações de *load* e *store* sobre a memória virtual.

Quando instruções privilegiadas, que incluem instruções de controlo e sensíveis ao comportamento de uma máquina virtual, são executadas, estas são capturadas pelo *hypervisor* [4]. Nem todas as arquiteturas de CPU podem ser virtualizadas. A arquitetura RISC CPU pode ser naturalmente virtualizada devido ao facto de todas as instruções sensíveis ao controlo e comporta-

mento serem instruções privilegiadas, enquanto que a arquitetura x86 não foi desenhada para suportar virtualização, isto porque existem dez instruções sensíveis, como a SGDT (*Store Global Descriptor Table*) e SMSW (*Store Machine Status Word*), que não são instruções privilegiadas [4].

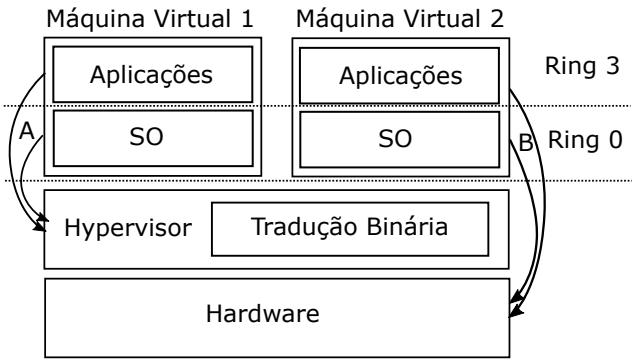

Figura 2.1. Representação esquemática da virtualização da CPU usando virtualização completa (figura adaptada de [4]). SO:Sistema Operativo, A:Instruções Privilegiadas, B:Instruções Não Privilegiadas.

A virtualização de uma CPU com múltiplos núcleos é muito mais complexa do que a virtualização com apenas um núcleo. Contudo, o facto de possuir mais do que um núcleo permite que tenha um melhor desempenho [4]. Em vez de o tempo ser compartilhado entre as tarefas num processador com um só núcleo, podem ser compartilhadas pelos vários núcleos, podendo estas tarefas conter apenas uma ou múltiplas *threads*. Através do uso deste tipo de virtualização, é possível fazer uso de todos os núcleos da CPU física. Uma CPU virtual pode possuir mais núcleos que a CPU física, contudo o número de núcleos que possuir a mais encontram-se em estado de pausa [4], como se pode observar na figura 2.2.

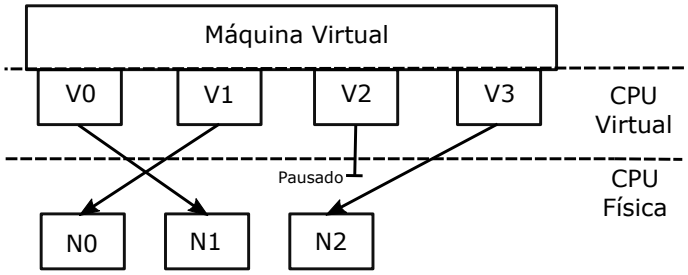

Figura 2.2. Representação esquemática da distribuição de tarefas na virtualização de *multi-core* CPU. V0, V1, V2 e V3 representam núcleos virtuais e N0, N1 e N2 representam núcleos físicos (figura adaptada de [4]).

2.3.2 Virtualização da Memória

A virtualização de memória virtual é semelhante ao suporte de memória virtual que os sistemas operativos modernos fornecem. Um sistema operativo tradicional mantém o mapeamento de memória virtual para memória máquina através do uso de tabelas de páginas (*page tables*). Todas as CPUs modernas com a arquitetura x86 incluem uma MMU (*Memory Management Unit*) e um TLB (*Translation Lookaside Buffer*) para otimizar o desempenho da memória virtual. No entanto, num ambiente de execução virtual, a virtualização da memória virtual envolve a partilha da memória máquina do sistema na RAM e alocação dinâmica da memória física das máquinas virtuais, sendo necessário um processo de mapeamento de duas etapas mantido pela máquina *guest* e pelo *hypervisor* [4].

As duas etapas do processo de mapeamento consistem em [4]:

- Mapeamento de memória virtual para memória física - A virtualização da MMU deve ser suportada, sendo transparente para o sistema operativo *guest*, que continua a controlar o mapeamento de endereços virtuais para memória física das máquinas virtuais, mas o sistema operativo *guest* não tem acesso direto à memória física. Cada tabela de páginas do sistema operativo tem no *hypervisor* uma tabela de páginas correspondente, designada por tabela de páginas sombra.

- Mapeamento de memória física para memória máquina - O *hypervisor* é responsável pelo mapeamento da memória física das máquinas *guest* para memória máquina *host*. Os endereços físicos de memória são traduzidos para memória máquina utilizando outro conjunto de tabelas de páginas definidas pelo *hypervisor*.

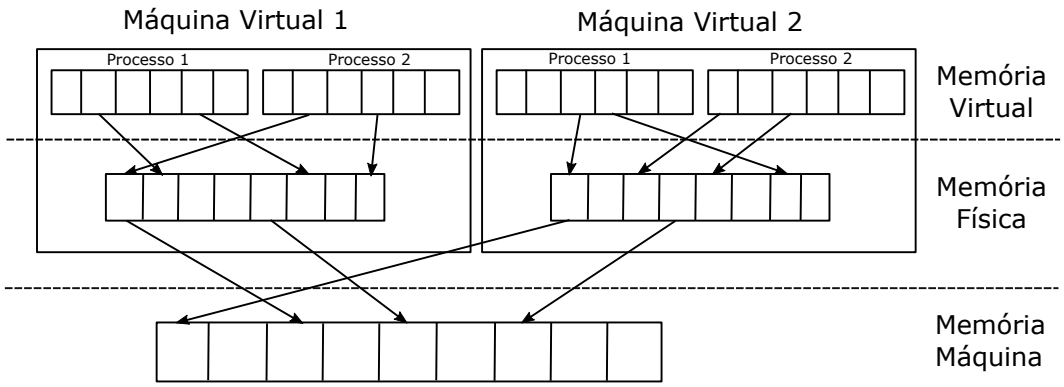

Figura 2.3. Representação esquemática da virtualização de Memória (figura adaptada de [4]).

2.3.3 Virtualização de I/O

A virtualização de *Input/Output* envolve a gestão do encaminhamento de pedidos de I/O entre os dispositivos virtuais e *Hardware* físico partilhado. Existem três técnicas de implementação da virtualização de I/O [4]:

- Emulação completa do dispositivo - A emulação do dispositivo é implementada na camada de virtualização e mapeia dispositivos reais de I/O para dispositivos virtuais para o *driver* da máquina *guest* poder utilizar.
- Paravirtualização - Também conhecida como modelo do *driver* dividido, que consiste num *frontend driver* e um *backend driver*. O *frontend driver* faz a gestão de pedidos I/O do sistema operativo *guest* e o *backend driver* é responsável pela gestão dos dispositivos de I/O reais e a multiplexagem de dados I/O das diferentes máquinas virtuais.
- I/O direto - As máquinas virtuais tem acesso direto aos dispositivos reais. Através desta técnica é possível alcançar um desempenho semelhante ao nativo sem grandes custos de CPU. No entanto, as implementações de virtualização de I/O direto têm-se focado em rede para *mainframes*.

2.4 Virtualização Baseada em *Hypervisors*

2.4.1 Definição e Tipos de *Hypervisors*

Na virtualização baseada em *hypervisors* é criada uma camada de abstração entre o *hardware* e os sistemas operativos *guest*. Este processo é feito através de um *software* chamado *hypervisor*. A principal função deste tipo de virtualização é a criação de múltiplos sistemas operativos e aplicações executáveis em paralelo no mesmo *hardware* através da criação de instâncias virtuais de *hardware* para cada máquina virtual, reduzindo custos com o aumento da utilização de *hardware* [15].

O Virtual Machine Monitor ou *hypervisor* é um *software* que faz a gestão da CPU, memória, transferência de dados I/O, interrupts e o conjunto de instruções priveligiadas e não previligiadas do ambiente virtual [11].

Um *hypervisor*, baseado no seu nível de implementação pode ser classificado como um dos dois tipos seguintes [11]:

- Tipo 1 - Encontra-se sobre o *hardware* e a comunicação entre o *hardware* e a máquina virtual é direta. O sistema operativo *host* não é necessário no tipo 1, devido ao *hypervisor*

executar diretamente sobre a máquina física. Por este motivo também é designado por *hypervisor* nativo ou *hypervisor bare metal*. Exemplos deste tipo de *hypervisor* podem ser observados na tabela 2.1.

- Tipo 2 - Encontra-se instalado sobre o sistema operativo *host* (também designado de *hosted hypervisor*) para gerir facilmente máquinas virtuais com suporte em configurações de *hardware* do sistema operativo. A camada extra entre o *hardware* e as máquinas virtuais nos *hypervisors* do tipo 2 causa ineficiência em comparação com os *hypervisors* do tipo 1. Exemplos deste tipo de *hypervisor* podem ser observados na tabela 2.1.

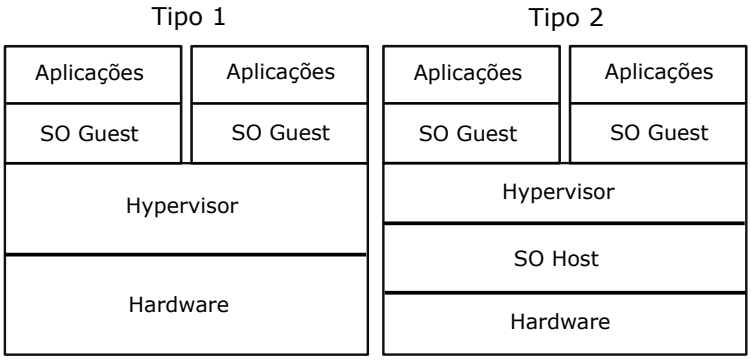

Figura 2.4. Tipos de *Hypervisor*. SO:Sistema Operativo.

Os conceitos de máquina *host* e *guest*, são usados no *hypervisor* para definir domínios diferentes. O domínio da máquina *host* contem o *hypervisor* que cria e gere as máquinas virtuais, e o domínio da máquina *guest* contem todas as máquinas virtuais, hospedadas sobre a máquina *host*, num ambiente seguro e isolado. Com estes dois domínios, o *hypervisor* é capaz de impor fronteiras de recursos para múltiplas máquinas virtuais. As máquinas virtuais são referidas tipicamente como entidades encapsuladas que incluem o sistema operativo e as aplicações em execução [11]. Através da implementação deste tipo de virtualização, surgiram várias técnicas virtualização baseadas em *hypervisor*, sendo as principais: a virtualização completa, a paravirtualização e a virtualização assistida por *hardware*.

Hypervisor Tipo 1		*Hypervisor* Tipo 2	
Designação	*Stable Release*	Designação	*Stable Release*
Oracle VM Server [16]	31/5/2016	PCem [17]	18/2/2017
Hyper-V [18]	17/10/2017	DOSBox [19]	10/5/2010
KVM [20]	5/9/2012	QEMU [21]	2/10/2017
z/VM [22]	11/11/2016	Oracle VirtualBox [23]	18/10/2017
Proxmox VE [24]	5/7/2017	VMware Fusion [25]	26/9/2017
VMware-ESXi [26]	5/10/2017	PikeOS [27]	-/4/2017
Xen [28]	10/4/2017	VMware Workstation [29]	26/9/2017

Tabela 2.1. Exemplos de *hypervisors* organizados por tipo.

2.4.2 Arquiteturas de *Hypervisors*

A forma como o *hypervisor* do tipo 1 aloca recursos e como faz a gestão do uso de *drivers* depende se o *hypervisor* é do tipo monólito ou *micro-kernel*. Um *hypervisor* monólito tal como o VMware ESXi, faz a gestão de todos os acessos ao *hardware* para cada máquina virtual, e todos os *drivers* de *hardware* encontram-se hospedados no *hypervisor*. As máquinas virtuais requerem acesso aos *drivers* através do *hypervisor*. A maior vantagem deste *design* é que não necessita de um sistema operativo *host*. O *hypervisor* funciona como uma plataforma operacional que suporta todos os sistemas operativos das máquinas virtuais. As desvantagens deste *design* são o suporte limitado fornecido pelo *hardware* e a instabilidade dos *drivers* de dispositivos que são diretamente incorporados nas camadas de funcionalidade, o que significa que caso um driver seja atualizado, e possua um *bug* ou uma vulnerabilidade de segurança, as arquiteturas virtuais e físicas poderão ser comprometidas [30].

O design *Micro-kernel* (Hyper-V e Xen) em contraste não requer que os *drivers* de dispositivos estejam incorporados no *hypervisor*, ou seja em vez de serem incorporados, os *drivers* do *hardware* físico são todos instalados num sistema operativo, que se encontra em execução na partição pai (máquina virtual também chamada domínio 0) do *hypervisor*. O que significa que não existe a necessidade de instalar *drivers* de dispositivos de suporte a *hardware* físico para cada sistema operativo das máquinas *guest*, que executam como partições filhas. O acesso a recursos de *hardware* físico pelas máquinas *guest* (partições filhas) é todo realizado através de uma comunicação com a partição pai que é o intermediário entre o *hardware* físico e as máquinas *guest*. Esta comunicação é feita através de um *bus* rápido baseado em memória no caso de a partição filha ser paravirtualizada, ou através do uso de dispositivos emulados fornecidos pela partição pai em caso de virtualização completa. A vantagem deste *design* é a inexistência da necessidade de incorporação de *drivers* no *kernel* do *hypervisor*. A desvantagem é a necessidade de instalação de um sistema operativo na partição pai antes que a execução do *hypervisor* possa ocorrer. Caso o sistema operativo falhe por alguma razão, todas a máquinas virtuais também irão falhar [30].

2.4.3 Virtualização Completa

A virtualização completa é uma técnica de virtualização que tem como requerimento que as instruções privilegiadas de CPU do sistema operativo *guest* sejam capturadas pelo *hypervisor* e só depois emuladas. Esta técnica de captura e emulação é também chamada de virtualização clássica, prevalecendo desde a sua criação. Certas instruções, no entanto, não podem ser virtualizadas, como por exemplo, algumas da arquitetura x86 que foi virtualizada de forma clássica até terem sido adicionadas extensões de virtualização de *hardware*. A virtualização completa depende de tradução binária para detetar e capturar instruções que não podem ser

virtualizadas [14].

Os *hypervisors* que implementem a virtualização completa conseguem executar sem que sejam feitas modificações, tornando-a numa técnica de virtualização bastante prática. O sistema operativo *guest* executa em modo não privilegiado, e cada instrução privilegiada é capturada e emulada pelo *hypervisor*, pois este executa em modo privilegiado. Os estados de privilégio do *hardware* físico e os estados de privilégio da máquina virtual são diferentes. Estruturas de dados de CPU, como ponteiros para tabelas de páginas são mantidos como uma cópia sombra dos registos do *guests* que são acedidos pelo *hypervisor* quando as instruções forem emuladas. A consistência das estruturas de dados, como as tabelas de páginas (que são armazenadas em memória física), é mantida pelo *hypervisor* através de mecanismos de proteção da MMU para detetar vestígio de acesso à memória e fazer a atualização das estruturas sombra [14]. A representação esquemática da execução da tradução binária e da execução direta na virtualização completa encontra-se na figura 2.5.

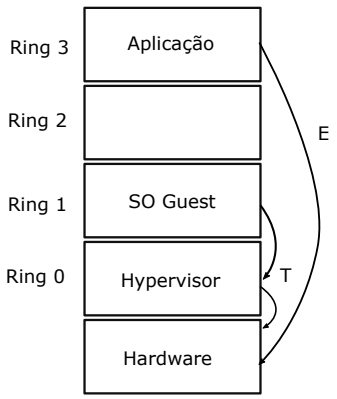

Figura 2.5. Representação esquemática da execução da tradução binária e da execução direta na virtualização completa (figura adaptada de [15]). SO:Sistema Operativo, T:Tradução binária, E:Execução Direta.

2.4.4 Paravirtualização

A paravirtualização é uma técnica de virtualização que requer a modificação dos sistemas operativos *guest*. A modificação é efetuada para que seja possível efetuar chamadas ao *hypervisor* (*Hypercalls*) em vez da captura e emulação de instruções privilegiadas por parte do *hypervisor*. Esta técnica é particularmente eficaz para arquiteturas que não possam ser virtualizadas através da virtualização completa, como por exemplo a clássica arquitetura x86, para obter um elevado desempenho de virtualização. A paravirtualização não requer a modificação nas aplicações que se encontram a executar no sistema operativo modificado. Esta técnica popularizou-se devido ao *hypervisor* Xen que foi desenvolvido primeiramente sobre a arquitetura x86, para evitar a tradução binária e para virtualizar de forma eficiente a MMU. Na paravirtualização à semelhança

da virtualização completa as instruções não privilegiadas, são executadas diretamente [14].

A paravirtualização, com a cooperação das modificações do *kernel* dos sistemas operativos *guest*, oferece uma melhoria ao nível do desempenho da CPU, memória e virtualização de I/O, em comparação à virtualização completa. A virtualização da memória na paravirtualização é feita pelo *hypervisor* que faz o registo das tabelas de páginas dos sistemas operativos *guest* diretamente na MMU com acesso apenas para leitura para evitar *overheads* e complexidade no que diz respeito à atualização de tabelas de páginas sombra [11]. A representação esquemática da execução de *hypercalls* e da execução direta na paravirtualização encontra-se na figura 2.6.

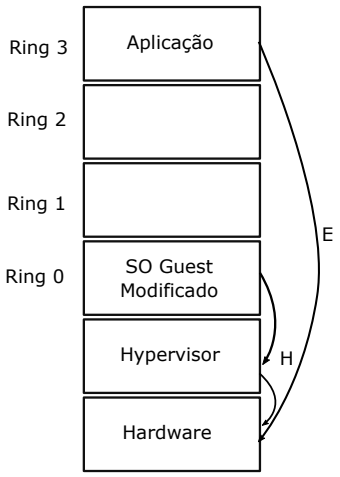

Figura 2.6. Representação esquemática da execução de *hypercalls* e da execução direta na paravirtualização (figura adaptada de [15]). SO:Sistema Operativo, H:Hypercalls, E:Execução Direta.

2.4.5 Virtualização Assistida por *Hardware*

Na virtualização assistida por *hardware*, a virtualização de captura e emulação é suportada diretamente por *hardware*. Esta técnica tornou-se possível na arquitetura X86 através da implementação de extensões da CPU que dão suporte à virtualização [14].

As extensões de virtualização em CPUs da Intel introduziram uma nova estrutura de dados na memória chamada de *virtual machine control block* (VMCB) que mantem informação sobre os estados das CPUs virtuais da máquina virtual. Estas extensões também introduziram um novo modo de execução, o modo *guest*, que suporta a execução de instruções privilegiadas e não privilegiadas. Uma nova instrução também foi criada, chamada chamada vmrun, a qual transfere o controlo do modo *host* para o modo *guest*, carregando os estados guardados no VMCB, retomando a execução da máquina *guest*. A execução continua no *guest* até que alguma condição expressada no bit de controlo do VMCB force à paragem da máquina virtual. Caso aconteça a saída, o *hardware* guarda o estado da máquina *guest* no VMCB, carrega o estado dado pelo

hypervisor e retoma a sua execução em modo *host*. O VMCB também fornece parâmetros de diagnostico de forma a dar a conhecer ao *hypervisor* o porquê de uma máquina virtual ter parado a execução [14]. A representação esquemática da execução de captura de pedidos do sistema operativo e da execução direta de pedidos na virtualização assistida por *hardware* encontra-se na figura 2.7.

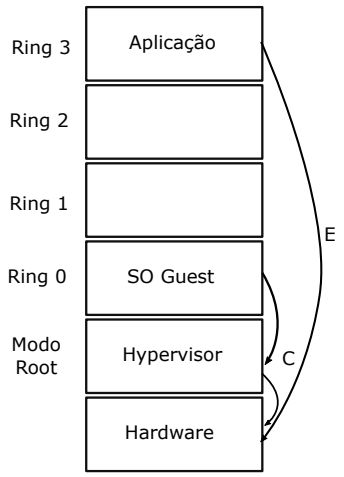

Figura 2.7. Representação esquemática da execução de captura de pedidos do sistema operativo e da execução direta de pedidos na virtualização assistida por *hardware* (figura adaptada de [15]). SO:Sistema Operativo, C:Captura de pedidos do SO, E:Execução Direta.

2.4.6 Exemplos de *Hypervisors*

O *Kernel Virtual Machine* (KVM) é um *software* existente no Linux, que permite ao sistema operativo atuar como um *hypervisor* do tipo 1. O KVM utiliza funcionalidades de virtualização de *hardware* em processadores recentes para reduzir a complexidade e overheads. Suporta a emulação de I/O através do QEMU e a paravirtualização de I/O através de dispositivos virtio. A combinação de aceleração de virtualização e paravirtualização de I/O permite uma redução no *overhead* de virtualização para níveis baixos. O KVM suporta migração em tempo real, permitindo que o conteúdo dos servidores físicos de todo o *datacenter* seja transferido para outros servidores físicos (para a realização da manutenção) sem afetar o estado das máquinas *guest*. A utilização do KVM é realizada através de ferramentas de gestão como a libvirt [20].

O Microsoft Hyper-V é uma tecnologia baseada em virtualização ao nível do *hardware*, existem duas variantes, a versão *standalone* chamada Hyper-V *Server* e o componente que se pode instalar no sistema operativo Windows *Server*. O Hyper-V tem como requisito um processador que possua a capacidade de virtualização assistida por *hardware*, permitindo um código base de virtualização mais compacto e melhorias associadas ao desempenho. A arquitetura do Hyper-V é

baseada em *hypervisors micro-kernel*, em que o sistema operativo *host*, é referido como partição pai, e fornece funcionalidades de gestão e *drivers* de *hardware*. As aplicações encontram-se nas máquinas *guest*, que também podem ser designadas por partições filhas [18].

2.5 Virtualização Baseada em *Containers*

2.5.1 Virtualização ao Nível do Sistema Operativo

A virtualização ao nível do sistema operativo é usada para fornecer gestão e isolamento dos recursos, primariamente em ambientes Linux. A virtualização baseada em *containers* deriva do conceito de os contentores serem a unidade padrão de transporte. No contexto de aplicações é referido como uma forma ágil de colocar aplicações num ambiente isolado de execução. O isolamento é criado através de três componentes principais: *chroot, cgroups e namespaces* do *kernel*. O *chroot* é um comando em Linux que permite a um processo mudar a diretoria *root* para criar sistemas de ficheiros específicos de *containers*. O *cgroups* é um subsistema do *kernel* através do qual é possível atribuir aos processos quotas de recursos. Os *namespaces* do *kernel* permitem a todos os *containers* receberem as suas próprias configurações de rede e comunicação entre processos, IPC e *namespaces* [15]. A figura 2.8 é uma representação esquemática da virtualização baseada em *containers*.

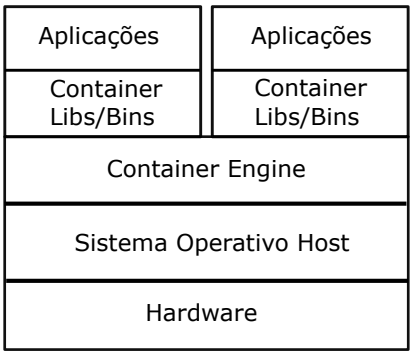

Figura 2.8. Representação esquemática da virtualização baseada em *containers* (figura adaptada de [15]).

A *Containerization* permite criar múltiplas instâncias de espaços de utilizadores, em que cada uma é isolada das outrasoutras. Estes segmentos de instâncias são designados por *containers*. No Linux, em que várias distribuições possuem o mesmo *kernel*, é possível que cada *container* tenha uma distribuição diferente. Contudo terá de ter sempre o mesmo sistema operativo (*kernel*) que o *host*. As aplicações que correm nos *containers* As aplicações que executam nos *containers* podem ter as suas próprias bibliotecas, sendo estes adaptáveis a um tipo especifico de aplicação [15].

A partilha de recursos do *kernel* faz com que os *containers* sejam muito mais leves na execução

do que as máquinas virtuais. O espaço dentro do *container* é apenas o necessário para executar cada aplicação, mantendo o mínimo de espaço, os *drivers* são mantidos dentro do sistema operativo *host* partilhado como parte do *kernel* (figura 2.8). Esta tecnologia de virtualização permite então que múltiplas aplicações executem de forma isolada, umas das outras, num único sistema operativo partilhado. Para executar múltiplas aplicações isoladas na mesma máquina *host* na virtualização de servidores tradicional seria necessário criar uma máquina virtual por aplicação [15]. As tecnologias de *containers* mais relevantes são referidas na tabela 2.2:

Containers	
Designação	*Stable Release*
LXC [31]	23/11/2016
Docker [15]	26/9/2017
OpenVZ [32]	21/11/2017
Lmctfy [33]	28/3/2014
FreeBSD *jail* [34]	26/7/2017
Kubernetes [35]	15/12/2017
Systemd-nspawn [36]	6/10/2017
Solaris *Containers* [37]	15/11/2010

Tabela 2.2. Tecnologias de *containers* mais relevantes.

2.5.2 Docker

O Docker é uma plataforma *open-source* que permite a implementação de aplicações dentro de *containers* de *software*. As dependências das aplicações estão incluídas nos *containers*, ou seja, qualquer código em tempo de execução, ferramentas de sistema, bibliotecas do sistema, ou seja, tudo o que é necessário para a execução da aplicação. O Docker faz uso de imagens para iniciar novos *containers* e são usados *templates* para a criação de *containers* em que cada um é uma instância de uma imagem em particular. As imagens são criadas a partir de sistemas de ficheiros em camadas, que permitem a partilha de ficheiros comuns, reduzindo assim o espaço de armazenamento do disco e acelerando a velocidade de *upload* e *download* das imagens [15]. O *Swarm* é um grupo de máquinas que executam o Docker e estão agregados a um *cluster*. Todos os comandos continuam a ser possíveis mas são executados pela máquina *swarm manager*. As máquinas no *Swarm* podem ser físicas ou virtuais, quando uma máquina se agrega ao *Swarm* passa a ser referida como *node*. O *swarm manager* é a única máquina com capacidade de autorizar a agregação de um *node* [38].

O Kubernetes é um sistema *open-source* que permite implementar de forma automática, escalar e gerir aplicações *containerized*. É possível agrupar Docker *containers* que constituem uma aplicação em unidades lógicas para uma gestão mais flexível [35].

2.6 Virtualização Nested

A virtualização *nested* é uma extensão da virtualização baseada em *hypervisors*, que permite aos *hypervisors* e às suas máquinas virtuais, que executem dentro de outro *hypervisor* (*hypervisor host* e *hypervisor guest* representados na figura 2.9). Esta técnica de virtualização tornou-se importante em sistemas operativos como o Windows e Linux que são utilizados normalmente como sistemas operativos *guest*, ganhando estes a funcionalidade de *hypervisor*. O *hardware* não suporta virtualização *nested* na arquitetura x86. Contudo foram propostas várias medidas com o objetivo de otimizar a implementação deste tipo de virtualização [14].

Na implementação do projeto *Turtles* [39], o *hypervisor* na camada inferior inspeciona a paragem de máquinas virtuais e redireciona-as para o *hypervisor* na camada superior para serem emuladas. A virtualização da memória é feita através de paginação multidimensional que condensa as várias tabelas de páginas, em uma ou duas tabelas de páginas baseadas em MMU. Na virtualização I/O, um dispositivo multi-nível é utilizado para que seja possível realizar *bypass* na comunicação entre as múltiplas camadas [14].

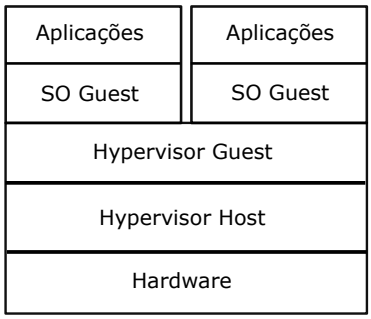

Figura 2.9. Representação esquemática da arquitetura da virtualização *nested* (figura adaptada de [39]). SO:Sistema Operativo.

2.7 *Overheads* de Virtualização

A virtualização impõe *overheads* ao nível do desempenho da CPU, memória, rede e disco. O *hyperisor* possui *overheads* da CPU, sendo estes originados no escalonamento duplo, equidade no escalonamento, velocidades assimétricas da CPU, devido a múltiplas CPUs virtuais correrem sobre a CPU física partilhada e gestão de *interrupts*. Também existem *overheads* ao nível da memória, devido a recuperação e duplicação de memória. O *overhead* na recuperação de memória é causado pela limitação da informação fornecida pelo *hypervisor* sobre quais são as melhores páginas do sistema operativo *guest* candidatas a expulsão, o que pode fazer o *hypervisor* e o sistema operativo *guest* tomar más decisões de paginação. O *overhead* da duplicação de memó-

ria é originado pelo facto de cada máquina virtual ter uma cópia completa do sistema operativo devido aos requisitos de isolamento. A rede possui *overheads* devido ao processamento de pacotes e instabilidade da rede. O desempenho do disco é afetado por *overheads* provenientes do escalonamento de I/O [14].

Muitos *overheads* são amplificados porque o sistema operativo *guest* não tem a noção que se encontra a executar num ambiente virtualizado, ou não foi otimizado para trabalhar com custo de acesso do I/O virtualizado que é muito maior que a do *hardware* físico [14].

Os *containers* não têm o mesmo tipo de *overhead* ao nível da CPU porque todas as aplicações executam no mesmo *kernel* da máquina *host*. Os *containers* são afetados por problemas de duplicação de memória. Contudo alguns executáveis e bibliotecas de baixo nível podem ser partilhadas entre *containers*. Ao nível de desempenho da rede, existem *overheads* com origem na pilha da rede do sistema operativo *host*, porque as *NICs* não podem ser virtualizadas e na tradução de endereços de rede (NAT). A implementação de *containers* como o Docker também possui *overheads* de disco devido aos sistemas de ficheiros em camadas que possuem a semântica *copy-on-write* por omissão [14].

Existem múltiplas fontes de *overhead*, sendo as apresentadas na tabela 2.3 as seguintes [14]:

- Escalonamento Duplo - Em virtualização baseada em *hypervisor*, estão presentes dois níveis de escalonamento: o processo de escalonamento do sistema operativo *guest* para as CPUs virtuais que são implementadas no *hypervisor* como *threads* e o escalonamento das *threads* das CPUs virtuais para a CPU física realizado pelo *hypervisor*. No processo de escalonamento de *hypervisors* atuais é desconhecida a existência de um segundo nível de escalonamento no sistema operativo *guest*, que pode causar uma degradação significativa do desempenho das aplicações que se encontram a executar de forma paralela nas máquinas virtuais.

- Equidade de escalonamento - O escalonamento duplo no *hypervisor* e no sistema operativo *guest* torna difícil atingir equidade no escalonamento para máquinas virtuais multiprocessadoras simétricas, em que cada uma possui duas ou mais CPUs virtuais. Uma máquina virtual pode ter todas as suas CPUs virtuais a executar na CPU física, mas outra máquina virtual pode ter todas as suas CPUs multiplexadas no *hardware* físico.

- CPUs assimétricas - O algoritmo utilizado no escalonamento tenta realizar uma distribuição por todas as CPUs para otimizar a eficiência do sistema, através da exploração da informação acerca da CPU para fazer previsões com base no desempenho de *workloads* prévios. No entanto, num ambiente virtual em que as CPUs virtuais são multiplexadas na CPU física, a suposição que cada componente é igual deixa de existir, o que conduz a que o escalonador do sistema operativo *guest* distribua trabalho pelas CPUs virtuais de forma sub-ótima o que conduz à subutilização ou sobre utilização das CPUs.

- *Interrupts* - Os *interrupts* podem ser tratados de forma quase imediata pelo mediador de *interrupts* que se encontra na CPU da máquina física. Nas máquinas virtuais, o mediador de *interrupts* não executará até que o *hypervisor* faça o escalonamento do CPU virtual para execução, o que provoca *delays* significativos. Em máquinas físicas é possível redirecionar um *interrupt* para múltiplas CPUs, tornando a entrega de *interrupts* mais flexíveis. No entanto, em alguns *hypervisors* a entrega de *interrupts* é limitada apenas a uma CPU específica.

- Recuperação de Memória - Quando o limite da memória física é ultrapassado, o *hypervisor* necessita de recuperar memória a partir das máquinas virtuais e desalojar páginas, através do envio destas para o armazenamento físico. No entanto, o *hypervisor* não possui muita informação acerca de quais são as páginas *guest* mais qualificadas à expulsão da memória, porque estas páginas são geridas pelos sistemas operativos *guest*.

- Duplicação de Memória - Os sistemas operativos *guests* são constituídos por uma imagem do *kernel*, bibliotecas partilhadas e serviços do sistema operativo, que causam *overhead* na pegada da memória. As máquinas virtuais encontram se em ambiente isolados uns dos outros, cada máquina possui uma cópia de sistema operativo. Duas máquinas virtuais podem possuir a mesma versão do sistema operativo, existindo assim uma duplicação na memória acerca da informação dos dois sistemas operativos iguais.

- Processamento de pacotes - O desempenho ao nível do I/O de rede do *hypervisor* é focado em fazer o output de TCP/IP o mais rápido possível sem se preocupar com a latência e o *overhead* de cada pacote, sendo problemático na execução de aplicações que envolvam grandes transferências de pacotes.

- NAT - O protocolo NAT introduz um *overhead* porque a tradução de endereços consome ciclos da CPU, que faz aumentar a latência na transmissão de pacotes e o número de conexões necessárias.

- Instabilidade da rede - A partilha das CPUs pode causar um output instável de TCP e UDP. A variância anormal causa no output de pacotes um *delay* que afeta de forma significativa o desempenho da rede.

- Escalonamento do I/O de disco - Na abordagem de camadas do *hypervisor*, as máquinas virtuais partilham o mesmo disco físico e existem múltiplas políticas paralelas de escalonamento do I/O de disco em execução: uma ao nível do *hypervisor* ou do sistema operativo *host* e outra ao nível do sistema operativo *guest* de cada máquina virtual. Os escalonadores de I/O de disco podem ser incompatíveis uns com os outros, de forma a que os pedidos de I/O tenham de ser reordenados de tal forma que o desempenho pode ser reduzido.

- Sistemas de ficheiros em camadas - Nas tecnologias de *containers* estes permitem ao utilizador empilhar sistemas de ficheiros uns no topo de outros e reutilizar as camadas para

reduzir o espaço utilizado para armazenamento e simplificar a gestão de sistemas de ficheiros de *containers*. No entanto, a utilização de sistemas de ficheiros em camadas introduz um *overhead* significativo no armazenamento porque as operações destes sistemas têm de movimentar-se entre múltiplas camadas.

Componente	Fonte de *Overhead*	Tecnologias de Virtualização	
		Hypervisor	*Container*
CPU	Escalonamento Duplo	✓	-
	Equidade no Escalonamento	✓	-
	Assimetria de CPU	✓	-
	Interrupts	✓	✓
Memória	Recuperação de Memória	✓	-
	Duplicação de Memória	✓	✓
Rede	Processamento de Pacotes	✓	-
	NAT	-	-
	Instabilidade de Rede	-	✓
Disco	Escalonamento de I/O	✓	-
	Sistemas de Ficheiros em Camadas	-	✓

Tabela 2.3. *Overheads* presentes na virtualização. O símbolo ✓ indica a existência de *overhead* (tabela adaptada de [14]).

2.8 *Benchmarking*

Um teste computorizado que realiza a medição das propriedades de uma tecnologia em particular é designado por *benchmarking*. As propriedades em avaliação podem incluir, velocidade, eficiência, taxa de transferência, etc. A utilização de *benchmarks* é importante, pois auxilia na seleção do equipamento antes da compra, que deve ser testado antes num ambiente similar em *workloads* a uma situação real de trabalho. A análise do cenário menos favorável revelasse necessária, contudo nem sempre é possível devido à indisponibilidade da replicação do ambiente circundante [40].

A execução de testes às características da tecnologia com propósitos académicos ou de investigação, faz com que exista a dificuldade em fornecer a configuração de um sistema real. Nestas situações a utilização de *benchmarks* serve de propósito para fornecer sistemas de aplicações próximas da realidade para a obtenção de resultados aproximados do cenário real [40].

A implementação de um sistema deve ter em consideração que aquando da sua duração, deve ser feita uma análise ao desempenho e ao custo do sistema e através dos resultados de ferramentas de *benchmark* é possível retirar conclusões. Muitos fatores devem ser considerados quando é realizado *benchmarking* para estabelecer uma comparação entre diferentes produtos de vendedores, sendo os resultados que fornecem uma correspondência razoável para as

dimensões das aplicações e sistema, aqueles que devem ser considerados [40].

Os *benchmarks* têm como foco o desempenho ou o custo. O *benchmarking* focado no desempenho tem como objetivo a mais alta eficiência, independentemente do custo do sistema, enquanto que o focado em custos tem como objetivo obter o custo mais baixo independentemente do desempenho do sistema [40].

A tabela 2.4 apresenta exemplos de *benchmarks*, organizados em torno do componente do sistema em que realizam testes.

Componente	*Benchmark*
CPU	Passmark [41] SPECvirt-sc2013 [41] Geekbench [42] Linpack [41] SysBench [41] UnixBench [41] Novabench [41]
Memória	Memtest86 [41] Memtest [41] Memtach [41] RAMspeed [43]
I/O do Disco	Postmark [40] Crystal DiskMark [41] DiskBench [41] IOzone *Filesystem benchmark* [44] DataMark [41] ATTO benchmark [41]
Rede	Netperf [41] Uperf [41] iPerf [45] Netmeter [41] PRTG [41] Sockperf [41] LANBench [41]

Tabela 2.4. Exemplos de ferramentas de *benchmark*.

2.9 Comparação em Termos de Segurança

Para uma comparação ao nível de segurança dos dois tipos de arquitetura de virtualização, é necessário analisar os comportamentos das arquiteturas quando estão sujeitas a ataques, como por exemplo negação de serviço (DoS - *denial of service*) e acesso não autorizado a dados. Os ataques de *denial of service* podem ser de dois tipos: Causar falhas no *hardware* ou exaustão de recursos [46].

Ao nível de indução de falhas no *hardware* da arquitetura baseada em *hypervisors*, a falha

do *kernel* corre apenas dentro do ambiente dedicado à máquina virtual, não afetando outras máquinas virtuais. No entanto, pode causar um efeito nas outras máquinas virtuais, dependendo apenas do tipo de ataque de DoS. Um ataque à rede, que tem como objetivo explorar uma vulnerabilidade numa camada de rede superior (ex. UDP), só conseguirá afetar a máquina alvo. Mas um ataque a componentes das camadas inferiores do *hypervisor* (ex: driver da interface de rede), que se encontra em execução na máquina *host*, pode induzir falhas nas máquinas *host* e *guest* ao mesmo tempo. Outras vulnerabilidades também podem ser introduzidas através do próprio *hypervisor*. Na arquitetura baseada em *containers*, um atacante pode causar falhas no *kernel* do sistema operativo (através de uma vulnerabilidade no driver da interface de rede ou um bug na pilha de rede), o que provoca uma falha geral na interface de rede, desabilitando o serviço para todos os *containers*. Os efeitos dos ataques DoS deste tipo afetam mais esta arquitetura do que a baseada em *hypervisors* [46].

Os ataques de DoS que visam a exaustão de recursos afetam as duas arquiteturas através de sobrecarga de pedidos às múltiplas interfaces de *hardware*, quer seja CPU, memória, armazenamento ou rede. À arquitetura baseada em *hypervisor* podem ser impostos limites no *hardware* disponível para cada máquina *guest*, na tentativa de minimizar danos na máquina *host*. O impacto de ataques de exaustão de recursos à arquitetura baseada em *containers* depende da tecnologia utilizada e as vulnerabilidades que cada uma possui. Através do uso de SE Linux e ambiente *chroot* é possível restringir o uso do CPU e o consumo de memória por parte de um *container*. As tecnologias mais avançadas de *containers* permitem impor restrições aos recursos de forma análoga a *hypervisors* [46]. Os ataques à arquitetura baseada em *hypervisor* ao nível de acesso não autorizado resultam da exploração de vulnerabilidades em *hypervisors*, que permitem que o atacante passe de um acesso à máquina *guest* para um acesso à máquina *host*. Estes ataques são possíveis através da exploração de uma vulnerabilidade do serviço *filer*, através de código que se encontra no espaço do *kernel* em que não é possível restringir os privilégios com mecanismos de MAC (*Mandatory Access Control*), no sentido em que a vulnerabilidade existente no código poderá conduzir à execução de código arbitrário em modo *kernel*. Um atacante que possua acesso ao *ring* 0 tem a capacidade de desativar o SE Linux [46].

O ambiente *chroot* presente na arquitetura baseada em *containers* não previne efetivamente ataques de acesso não autorizado, ou seja, quando existe um ataque a um *container* existe a possibilidade de o atacante escapar desse ambiente para a máquina *host*. As tecnologias de *containers* mais recentes já integram uma funcionalidade do Linux conhecida como *cgroups* que oferece um melhor isolamento, no entanto estas tecnologias são muito propicias a vulnerabilidades e são apenas capazes de oferecer isolamento limitado [46].

A arquitetura de virtualização baseada em *hypervisor* devido a possuir mais camadas seria esperado que fosse mais segura que a arquitetura baseada em *containers*. No entanto, existe a possibilidade do acesso aos dados dos clientes devido à existência de vulnerabilidades na interface de rede da máquina *host*. Devido ao isolamento existente na arquitetura baseada em

hypervisor, a probabilidade de um ataque a outras máquinas virtuais é muito baixa em comparação com a arquitetura baseada em *containers* [46].

2.10 Trabalho Relacionado

Há relatos na literatura da especialidade de vários trabalhos de investigação envolvendo virtualização nativa ao nível de *hardware* ou ao nível do sistema operativo. Esta secção descreve os principais trabalhos relacionados com a temática da dissertação, focando os principais aspetos de cada um desses trabalhos.

Em [46], Kurmus et al. apresentam um estudo comparativo sobre as duas arquiteturas de virtualização existentes, a baseada em *hardware* (*hypervisor*) e a baseada em sistema operativo (*containers*), em que são utilizadas ferramentas de *benchmark* para avaliar ao nível de segurança qual das arquiteturas possui um melhor desempenho. o estudo revela que ambas as arquiteturas são viáveis ao nível de segurança, no entanto a arquitetura baseada em *hypervisors* possui um *overhead* significativo em comparação à arquitetura baseada em *containers*, devido a possuir camadas adicionais isolantes.

Na dissertação de Graziano [47] é relatada uma investigação que se foca no estudo de duas tecnologias de virtualização (Xen e KVM), em que são conduzidas avaliações de desempenho ao nível geral, taxa de transferência e eficiência de máquinas virtuais ao nível de isolamento e escalabilidade para uso como *host* para o *Xen World Project*. Os resultados das avaliações permitiram concluir que o Xen possui um desempenho superior ao do KVM em cenários relacionados com o I/O de disco, e que o KVM demonstra um desempenho superior na execução de operações computacionalmente intensivas.

A dissertação de Sridharan [48] aborda a comparação de forma quantitativa e qualitativa, com o objetivo de comparar o desempenho de três *hypervisors*, o VMware ESXi, o Zen e o KVM, que usam a ferramenta standard de *benchamark* SPECvirt_sc2010v1.01 para avaliar o seu desempenho sobre diferentes *workloads* que simulam situações reais. Através da análise de resultados obteve-se a conclusão que o *hypervisor* VMware ESXi possui um melhor desempenho global, sendo este seguido pelo Zen e por ultimo o KVM.

Em [14], Enberg aborda os overheads das duas principais técnicas de virtualização, *hypervisor* e *container*, sendo as tecnologias de virtualização KVM e Docker. O uso de múltiplas ferramentas de *benchmark* (Netperf, Memcached e Mutilate) permitiu chegar a uma conclusão que a tecnologia de virtualização Docker possui um desempenho mais aproximado ao nativo, sendo superior ao desempenho das tecnologias de *hypervisor*.

Na dissertação de Kavita Agarwal [49] é reportada a realização de um estudo comparativo entre o *hypervisor* KVM e a tecnologia de *containers* Docker. Os principais parâmetros da análise

comparativa são a latência, a densidade e a pegada deixada na memória. Os métodos de redução da pegada na memória também são referidos. É concluído que a arquitetura de máquinas virtuais possui uma pegada na memória maior que a da arquitetura de *containers*.

No artigo de Voras et al. [50] é apresentada uma comparação entre várias tecnologias de virtualização *open-source* e comerciais presentes no mercado. É feito o uso de múltiplas ferramentas de *benchmark* para análise completa dos sistemas. O objetivo deste estudo é uma avaliação igualitária de todas tecnologias para uma tomada de decisão para fins de investigação. Os resultados do estudo revelam que o desempenho das tecnologias de *hypervisors* se aproxima do desempenho nativo quando os *workloads* executados são *single-threaded*. Contudo as diferenças mais acentuadas no desempenho estão associadas à execução de *workloads multi-threaded*.

Em [20], Felter et al. exploram o desempenho de máquinas virtuais, em contraste com *containers* de Linux. Através do uso de *workloads* gerados por *benchmarks* foi elaborada uma análise de desempenho a todos os componentes do sistema. As tecnologias de virtualização analisadas foram o KVM, que representa a tecnologia de *hypervisor*, e o Docker que representa a tecnologia de *containers*. Os resultados da análise demonstram que a tecnologia de *containers* possui um desempenho igual ou superior à tecnologia de *hypervisor*.

Na dissertação da autoria de Vishrutha Adla [41] é feita uma análise comparativa dos *hypervisors* Hyper-V e VMware ESXi, com o objetivo de verificar qual possui melhor desempenho geral. O isolamento da rede e de *hardware* é importante para a redução de *overheads* de virtualização. Os resultados da análise revelam que o VMware ESXi possui um melhor desempenho em geral em comparação com o Hyper-V.

De forma semelhante, Naveed Yaqub [40] é autor de uma dissertação que teve como objetivo deduzir através da comparação entre dois *hypervisors*, KVM e VMware ESXi, recorrendo a ferramentas de *benchmark* (IOzone, RAMspeed e UnixBench), qual possui melhor desempenho e a menor quantidade de *overheads* em cada componente do sistema.Como resultado verificou-se que o VMware ESXi possui um melhor desempenho que o KVM ao nível de I/O de disco e da CPU. O KVM possui um melhor desempenho ao nível de memória.

Em [51], Jianga et al. apresentaram uma comparação energética de cinco tecnologias de virtualização instaladas sobre seis tipos de *hardware* diferente. Das tecnologias de virtualização usadas, quatro delas são *hypervisors* (VMware ESXi, Microsoft Hyper-V, KVM e XenServer) e a restante é uma tecnologia baseada em *containers* (Docker). Uma das conclusões do estudo realizado foi que o consumo de energia das tecnologias de virtualização depende do *hardware* sobre o qual se encontram a executar.

A VMware [52] realizou um teste comparativo entre o seu *hypervisor* VMware ESX Server 3.0.1 e o *hypervisor* da concorrência Xen 3.0.3. Através do uso de ferramentas de *benchmark* (ferramentas SPEC, Netperf e Passmark) é feita uma análise quantitativa e qualitativa de cada *hypervisor*.

O resultado da análise permitiu concluir que o VMware ESX Server possui um desempenho superior ao nível da escalabilidade e prontidão para produção, necessário para uma implementação eficiente de um *datacenter*.

Em [53], Reddy e Rajamani avaliaram o desempenho de três *hypervisors*, VMware ESXi, XenServer e KVM, tendo usado o *framework* SIGAR para obter informações do sistema e a ferramenta de *benchmark* Passmark para a criação de *workloads* em ambientes de *cloud* privada. A conclusão retirada da avaliação foi que o *hypervisor* VMware ESXi possui um melhor desempenho do que os outros dois ao nível da CPU e da rede, enquanto que o XenServer possui uma eficiência superior ao nível de memória e de I/O de disco relativamente aos outros *hypervisors* em estudo.

Na tabela 2.5 é possível estabelecer uma comparação entre os múltiplos documentos relacionados com a temática da dissertação e esta dissertação.

Documento de:	Avaliação de tecnologias de *containers*	Avaliação de tecnologias de *hypervisors*	Implementação de infraestruturas de elevada disponibilidade
Esta dissertação	✓	✓	✓
A. Kurmus et al. [46]	✓	✓	–
C. D. Graziano [47]	–	✓	–
S. Sridharan [48]	–	✓	–
P. Enberg [14]	✓	✓	–
K. Agarwal [49]	✓	✓	–
I. Voras et al. [50]	–	✓	✓
W. Felter et al. [20]	✓	✓	–
V. Adla [41]	–	✓	–
N. Yaqub [40]	–	✓	–
C. Jianga et al. [51]	✓	✓	–
Vmware [52]	–	✓	✓
V. V. Reddy e L. Rajamani [53]	–	✓	✓

Tabela 2.5. Comparação entre os múltiplos documentos relacionados com a temática da dissertação e esta dissertação.

2.11 Conclusão

Neste capítulo foram abordados os tipos, técnicas, tecnologias e *overheads* de virtualização, assim como também os problemas de segurança associados a cada tipo de virtualização. Todo este conhecimento em conjunto com a análise de trabalhos realizados na temática da dissertação facultam a informação necessária para a implementação dos ambiente experimentais do capítulo 3 e posteriormente para a análise de resultados do capítulo 4. Ao analisar as vulnerabilidade de segurança da virtualização ao nível de *hardware* e da virtualização ao nível do sistema operativo, foi obtida a conclusão que ambos tipos de virtualização podem ser alvo de

ataques de negação de serviço (DoS - *denial of service*) e ataques que permitem o acesso não autorizado a dados. A realização de *benchmarking* é a melhor metodologia para a realização do estudo comparativo entre infraestruturas de elevada disponibilidade, porque permite a recolha de resultados ao nível do desempenho.

A comparação do estudo comparativo realizado nesta dissertação com os estudos relacionados com a temática da dissertação permitiu obter a conclusão, que nem todos os estudos relacionados abordam em simultâneo as tecnologias de *hypervisors* e de *containers*, e os que abordam, não possuem infraestruturas de elevada disponibilidade implementadas.

Capítulo 3

Implementação do Ambiente Experimental

3.1 Introdução

Neste capítulo são apresentados os ambientes experimentais, os recursos de *hardware* e *software* utilizados no *test bed* experimental, os métodos de implementação dos ambientes experimentais e as ferramentas de avaliação de desempenho (benchmarks) que realizam os testes de desempenho aos ambientes, de acordo com as métricas definidas por estes. As infraestruturas implementadas sobre o *test bed* são, o *Failover Cluster* sobre Hyper-V e o Docker *Swarm*. As ferramentas de *benchmark* usadas na avaliação das máquinas virtuais e containers das infraestruturas anteriormente referidas são o Geekbench, o IOzone, o RAMspeed e o Iperf.

3.2 *Test Bed* Experimental

3.2.1 Caracterização do *Test Bed* Experimental

O *test bed* é constituído por três máquinas físicas (representadas na figura 3.1), duas das quais apresentam especificações técnicas idênticas e foram utilizadas para construir ambientes onde foram instalados os nós dos *clusters*, a terceira máquina possui especificações diferentes, com a finalidade de ser utilizada como servidor de armazenamento de dados.

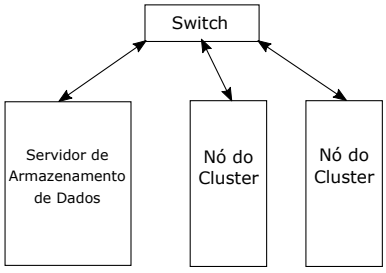

Figura 3.1. Representação esquemática da arquitetura básica do *test bed* (as especificações dos nós do *cluster* encontram-se na tabela 3.1 e as especificações do servidor de armazenamento de dados encontram-se na tabela 3.2).

Todas as máquinas físicas estão conectadas na mesma rede (10.0.5.0/24) através de um *switch* de nível 2 com 8 portas em 10BASE-T/100BASE-TX/1000BASE-T para que exista comunicação entre elas. Como alvo de avaliação nesta dissertação encontram-se dois ambientes experimentais diferentes, correspondendo cada ambiente à implementação de uma tecnologia de virtualização diferente (*Mycrosoft* Hyper-V e Docker *Swarm*).

Para criar um *cluster* de elevada disponibilidade para a tecnologia de virtualização Hyper-V são necessários pelo menos dois nós do *cluster*. A *live* e *quick migration* só é possível entre os dois nós, com recurso a uma máquina de armazenamento onde toda as máquinas virtuais são armazenadas. Cada nó do *cluster* encontra-se instalado numa máquina física que contem o sistema operativo Windows *Server* 2016. Ambos os nó do cluster possuem duas máquinas virtuais, uma com o sistema operativo Ubuntu 16.04 e outra com Microsoft Windows *Server* 2016. Existem duas redes nesta implementação, uma rede física (10.0.5.0/24) que permite aos nós do *cluster* comuniquem entre si e uma rede virtual (192.168.127.0/24) que permite que as máquinas virtuais possam comunicar entre si e a Internet através do uso do protocolo NAT (*Network Address Translation*). Os endereços IP desta rede são atribuídos por DHCP (*Dynamic Host Configuration Protocol*). Este ambiente foi implementado de acordo com o esquema representado figura 3.2.

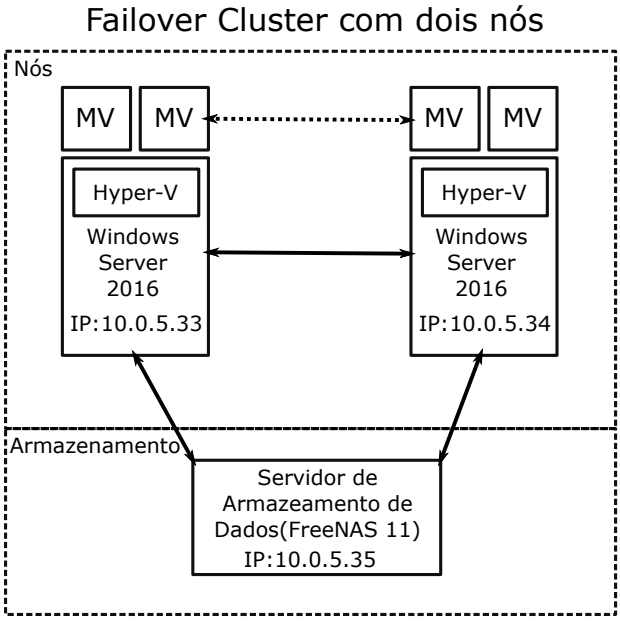

Figura 3.2. Representação esquemática do ambiente experimental do *failover cluster* em Hyper-V. MV: Máquina Virtual.

O modo *Swarm* da tecnologia Docker permite-nos criar um *cluster* básico com apenas três nós. O primeiro nó é o *manager* que corresponde a uma máquina física, que faz a gestão de todo o *Swarm*, e os outros dois nós do *Swarm* são designados por *workers*. A implementação realizada no âmbito desta dissertação possui dois *workers* que se encontram sobre a mesma máquina física. No entanto um *worker* (*node* 1) encontra-se sobre o *host* e outro (*node* 2) sobre uma máquina virtual criada e gerida pelo *hypervisor* KVM. Existem três redes definidas neste ambiente

experimental, uma rede física (10.0.5.0/24) e duas redes virtuais. Uma das redes virtuais é a criada por padrão pelo KVM (192.168.122.0/24) para permitir a comunicação entre as máquinas *host* e *guest*, mas também entre *guest* e a Internet através de NAT, a outra rede virtual é uma rede *overlay* criada pelo Docker *Swarm* que possui o endereço IP 10.0.0.0/24 e permite a comunicação entre os *containers* e a Internet através de NAT, os endereços de IP são atribuídos por DHCP. Todos os nós do *cluster* possuem na configuração mínima um *container* em execução. O *storage* é todo realizado localmente pelos nós, devido à pequena dimensão deste ambiente experimental, não tendo sido adicionado um sistema de armazenamento de *containers*. O ambiente foi implementado de acordo com o esquema da figura 3.3 exceto a parte do armazenamento. A figura representa um cenário mais escalável e industrial.

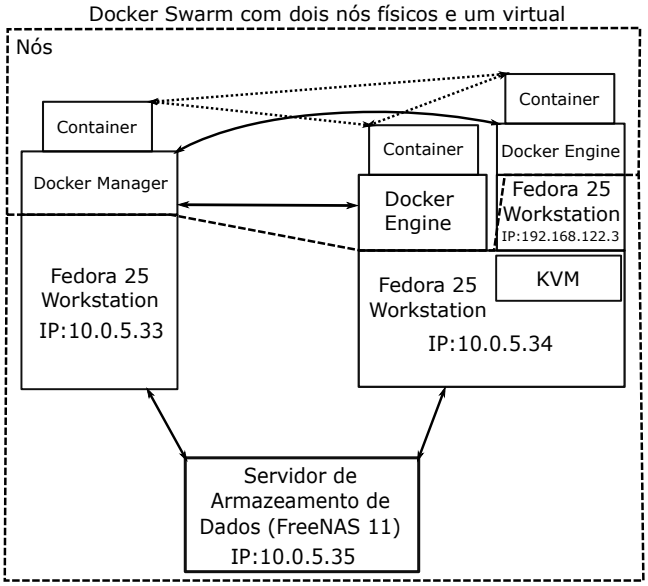

Figura 3.3. Representação esquemática do ambiente experimental do Docker *Swarm*.

3.2.2 Especificações de *Hardware* e *Software* do *Test Bed*

Nós do *Cluster*	
CPU	Intel Core i7-7700 3.60 GHz
Memória	Kingston DDR4 2133MHz 8192MB (x2)
Disco	SATA ST1000DM010-2EP102 1000.2 GB
Motherboard	ASUS PRIME B250-PRO
Sistemas Operativos	Linux Fedora 25 *Workstation* Microsoft Windows *Server* 2016

Tabela 3.1. Especificações de *hardware* e *software* das máquinas nós do *cluster*.

Servidor de armazenamento de dados	
CPU	Intel Core i7-7700 3.60 GHz
Memória	Kingston DDR4 2133MHz 8192MB (x2)
Discos	SATA ST1000DM010-2EP102 1000.2GB (x2) TOSHIBA HDWD105 500.1 GB
Motherboard	ASUS 170 PRO GAMING
Sistema Operativo	FreeNAS 11

Tabela 3.2. Especificações de *hardware* e *software* do servidor de armazenamento de dados.

3.3 Implementação das Infraestruturas Virtualizadas de Elevada Disponibilidade

3.3.1 Armazenamento de Dados

O armazenamento nos ambientes experimentais descritos na secção anterior, é realizado com recurso a um servidor de armazenamento, sendo este uma máquina que possui um sistema operativo que possibilita as operações de armazenamento. O sistema operativo foi instalado sobre o disco de 500GB da máquina de armazenamento. No servidor de armazenamento considerado nesta dissertação foi instalado o FreeNAS 11 [54] é um *Network Attached Storage* (NAS), gratuito e de código aberto que permite a partilha de dados sobre uma rede, usando protocolos de partilha baseados em ficheiros, incluindo o SMB (*Server Message Block*, para utilizadores de Windows), NFS (*Network File System*, para sistemas baseados em Unix), AFP (*Apple Filling Protocol*, para utilizadores de Mac OS), FTP (*File Transfer Protocol*) e iSCSI (*Internet Small Computer System Interface*, para partilha por blocos). O FreeNAS 11 usa o sistema de ficheiros ZFS (*Zettabyte File System*) para armazenamento, gestão, e proteção de dados. Este sistema fornece funcionalidades mais avançadas como *snapshots* para o armazenamento de versões antigas dos ficheiros, backups incrementais remotos para manter os dados seguros noutro dispositivo sem grandes transferências de dados, compressão inteligente que permite a redução rápida do tamanho dos ficheiros, o que possibilita a transferência rápida dos mesmos [54, 55]. A proteção dos dados armazenados é realizada pelo sistema de ficheiros ZFS através da utilização do *software* RAID-Z, que permite o uso de proteção de paridade até RAID 5. No *test bed* experimental o servidor de armazenamento de dados possui apenas dois discos para armazenamento, o que permite no máximo realizar RAID 0, no caso de um dos discos falhar, existe um disco de backup [55].

O protocolo utilizado nos ambientes experimentais é o iSCSI, que transporta dados de armazenamento ”baseados em blocos” com elevado desempenho sobre uma rede IP. O transporte é

realizado entre o iSCSI *initiator* que se encontra nos nós e o iSCSI *target* que se encontra no servidor de armazenamento de dados [56].

No protocolo iSCSI os comandos SCSI são encapsulados, e é feito o encapsulamento dos pacotes da camada de rede, que contêm os dados. Os pacotes são enviados através da rede usando uma conexão *point-to-point* e, quando são recebidos os pacotes, são traduzidos pelo protocolo para SCSI [56].

Quando um *initiator* estabelece uma conexão com o target, o sistema operativo vê o armazenamento como um dispositivo SCSI local que pode ser formatado. Este processo é transparente para as aplicações, sistemas de ficheiros e sistemas operativos [56].

Para qualquer cenário experimental, é necessário realizar um conjunto de procedimentos para que o armazenamento por iSCSI esteja disponível. Esse conjunto de procedimentos de forma ordenada é o seguinte [56]:

1. Criar uma portal;
2. Criar um iSCSI *initiator*;
3. Criar um iSCSI *target*;
4. Criar um *extent*;
5. Atribuir um *extent* a um *target*.

3.3.2 *Cluster* de Elevada Disponibilidade Usando Hyper-V

Existem alguns pré-requisitos na criação de um *failover cluster* em Hyper-V, tanto ao nível de *hardware* como de *software*. No que diz respeito à construção de um *failover cluster* precisamos, no mínimo, de dois nós e de um servidor de armazenamento de dados, podendo estas ser máquinas físicas ou virtuais. Contudo, as máquinas devem possuir a capacidade de comunicar entre elas através de uma rede. Ao nível de *software*, as máquinas nós do *cluster* têm instalado o sistema operativo Windows *Server* 2016, e deve-se proceder à instalação do role do Hyper-V e da funcionalidade *Failover Clustering*, que se encontra no *software* do sistema operativo designado por *Server Manager*. O Hyper-V e o *failover cluster* possuem restrições, no *cluster* não podem existir mais do que 64 nós e o limite de virtualização de computadores por *cluster* é de 8000 [57]. Para implementar um *failover cluster* em Hyper-V é necessário seguir os passos do anexo A.1 [58].

3.3.3 Docker Swarm

Um *cluster* que usa a tecnologia Docker *Swarm* necessita minimamente de três máquinas, as quais devem possuir um sistema operativo que possibilite a instalação do Docker *Engine*. O sistema operativo utilizado nos nós do ambiente experimental é o Linux Fedora 25 *Workstation*. A máquina *manager* também possui o mesmo sistema operativo com o Docker *Engine* instalado, mas devido a esta máquina ser o *manager* do *cluster*, que gere os nós e permite a intercomunicação entre eles, é necessária a instalação da API da Docker *Machine*. Para a utilização do modo *Swarm*, as máquinas devem ter instalada a versão do Docker *Engine* 1.12 ou superior e as portas de cada máquina devem estar abertas para as outras máquinas do *cluster* [59]. A versão do *Engine* usada nesta implementação do *Swarm* é a 1.17 *Community Edition*, que possui a API 1.25 do Docker *Engine*, sendo esta a mais recente. Com base no tutorial [59] procedeu-se à execução dos passos presentes no anexo A.2 para implementar o Docker *Swarm* no ambiente experimental.

3.4 *Benchmark* de Sistemas de Ficheiros

O IOzone [44] é uma ferramenta de *benchmark* cujo o alvo de avaliação é o sistema de ficheiros. O *benchmark* gera e mede uma variedade de operações em ficheiros. São realizados as seguintes operações para testar o desempenho de I/ de ficheiros: *read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, aio_write*. A versão do IOzone usada na elaboração dos testes nos ambientes experimentais é a 3.478. O *software* encontra-se disponível para Windows e Linux [44]. Os seguintes testes disponibilizados pelo *benchmark* são implementados nos ambientes experimentais, sendo a métrica utilizada o *throughput* (MByte/s) [60]:

1. *Read* – Indica o desempenho de leitura de um ficheiro existente no sistema de ficheiros.
2. *Write* – Indica o desempenho de escrita de um ficheiro novo no sistema de ficheiros.
3. *Re-read* – Depois da leitura, indica o desempenho de uma segunda leitura do ficheiro.
4. *Re-write* – Indica o desempenho de escrita num ficheiro já existente.
5. *Random Read* – Indica o desempenho de leitura ao ler uma informação aleatória de um ficheiro. Leitura não sequencial.
6. *Random Write* – Indica o desempenho de escrita de forma aleatória em diferentes locais de um ficheiro. Escrita não sequencial.
7. *Backward Read* - Indica o desempenho da leitura de um ficheiro ao contrário.

8. *Stride Read* - Indica o desempenho de leitura não sequencial seguindo um padrão definido por um espaçamento entre os dados.
9. *Mixed Workload* - Indica o desempenho da realização das operações de leitura e escrita em disco de forma simultânea.

Os motivos pelos quais este *benchmark* foi escolhido para ser implementado no testbed devem-se ao facto de ser uma ferramenta gratuita e *cross-platform* e permitir uma análise completa ao sistema de ficheiros, podendo esta análise ser aleatória ou sequencial.

O seguinte comando foi executado na linha de comandos para realizar os testes do *benchmark*:

```
iozone -t 2
```

3.5 *Benchmark* de Memória

O RAMspeed [43] é um *benchmark gratuito* e *open-source*, que permite a medição do desempenho da memória do sistema. O *benchmark* encontra-se disponível com versões para os seguintes sistemas operativos: Linux, Windows, FreeBSD e NetBSD. Este *benchmark* é constituído por dois sub-conjuntos de *benchmarks* que se descreve a seguir [43].

O primeiro sub-conjunto é constituído pelos benchmarks INTmark, FLOATmark, MMXmark e SSEmark, que operam com fluxos de dados lineares (sequenciais) que passam respetivamente pelas unidades ALU (*Arithmetic Logic Unit*), FPU(*Floating-point Unit*), MMX (*Matrix Math Extensions*) e SSE (*Streaming SIMD Extensions*). Estes *benchmarks* alocam um espaço especifico da memória, onde começam a escrever ou ler blocos contínuos dimensionados na potência de 2, a partir de 1Kb até ao limite do *array*. Ao fazerem uso de algoritmos simples permitem demonstrar o quão rápida é a cache e os subsistemas de memória [43].

O segundo sub-conjunto é constituído pelos *benchmarks* INTmem, FLOATmem, MMXmem e SSEmem, que permitem ilustrar o quão rápido é o desempenho real da leitura/escrita na memória. Cada um deles inclui quatro sub-testes designados por: *Copy, Scale, Add e Triad*. Os testes são simulações sintéticas, mas correlacionadas com aplicações do mundo real [43]. Os testes realizados em ambos os ambientes experimentais considerados nesta dissertação são os que pertencem a este sub-conjunto.

A versão do *benchmark* utilizada no ambiente experimental do Hyper-V é a 1.1.1 e a versão implementada no ambiente do Docker *Swarm* foi a 3.5.0. Este *benchmark* começou a ser desenvolvido mais tarde para Windows, contudo as versões para os diferentes sistemas operativos aqui considerados possuem as mesmas funcionalidades.

Os principais motivos pelos quais este *benchmark* foi escolhido para ser implementado no test-

bed devem-se ao facto de: Ser uma ferramenta gratuita e *cross-platform* e à capacidade de realizar uma análise completa às múltiplas áreas da memória e simulações sintéticas, mas correlacionadas com aplicações do mundo real [43]. A métrica usada por este *benchmark* é o *throughput* (MByte/s), sendo esta utilizada para medir o desempenho das operações realizadas à memória. Os seguintes comandos foram executados na linha de comandos para a realizar os testes do *benchmark*:

```
ramspeed -b 3
```

e

```
ramspeed -b 6
```

3.6 Benchmark de Rede

O iPerf3 [45] é uma ferramenta de *benchmark* para medições ativas da largura de banda máxima alcançável em redes IP. Suporta o ajuste de vários parâmetros relacionados a temporização, buffers e protocolos (TCP, UDP, SCTP com IPv4 e IPv6). Para cada teste, a informação dada é a largura de banda, a perda de pacotes e outros parâmetros. O iPerf foi originalmente desenvolvido pelo *National Laboratory for Applied Network Research* e pela *Distributed Applications Support Team*. O iPerf3 é desenvolvido principalmente pelo Laboratório Nacional ESnet/Lawrence Berkeley. O *benchmark* encontra-se disponível com versões para os seguintes sistemas operativos: Windows, Linux, Android, MacOS X, FreeBSD, OpenBSD, NetBSD, VxWorks, Solaris, etc. A versão de *software* utilizada na elaboração de testes nos ambientes experimentais aqui considerados é a 3.5 [45]. As funcionalidades do iPerf [45] incluem:

- TCP e SCTP
  - Medição da largura de banda.
  - Relato do tamanho do MSS/MTU (*Maximum Segment Size*/*Maximum Transmission Unit*) e dos tamanhos de leitura observados.
  - Suporte para o tamanho da janela de TCP via *buffers* de *socket* (opção -w).
- UDP
  - Criação de fluxos de UDP de largura de banda especifica pelo cliente (opção -u).
  - Medição da perda de pacotes.
  - Medição do *delay jitter*.
  - Capacidade de *multicast*.

- Possibilidade de múltiplas conexões entre cliente e servidor.
- O servidor lida com várias conexões, em vez de terminar a sua execução depois de um único teste.
- Pode ser executado por tempo especificado (opção -t), em vez de uma quantidade definida de dados a serem transferidos (opção -n ou -k).
- Impressão de relatórios periódicos, intermediários de largura de banda, *jitter* e perda em intervalos especificados (opção -i).

Os principais motivos pelos quais este *benchmark* foi escolhido para ser implementado no test-bed devem-se ao facto de ser uma ferramenta gratuita e *cross-platform* e possuir uma grande variedade de funcionalidades (referidas anteriormente). Estas qualidades fazem desta ferramenta uma das melhores entre as que se encontram disponíveis actualmente.

Os testes realizados por este *benchmark* são executados por linha de comando, funcionando uma máquina como cliente e outra como servidor. As métricas utilizadas nos testes do *benchmarks* são a largura de banda (Mbits/s) e o *jitter* (ms). Os seguintes comandos foram executados nos ambientes experimentais:

- Executar o cliente em modo TCP:

  iperf3 -c <IP-servidor> -w 20K

- Executar o cliente em modo UDP:

  iperf3 -u -c <IP-servidor>

- Executar o servidor:

  iperf3 -s

3.7 Benchmark de CPU

O Geekbench é um *benchmark* de processador *cross-platform*, com um sistema de pontuação que faz a separação entre desempenho *single-core* e *multi-core*, através do uso de *workloads* que simulam cenários reais. A versão implementada nos ambientes experimentais, Geekbench 4, realiza a sua avaliação através da comparação de uma pontuação base de 4000, que representa o desempenho do processador Intel Core i7-6600U a 2.60 GHz. O *software* encontra-se disponível para macOS, Windows, Linux, Android e iOS. A versão mais recente do *software* também avalia o desempenho da GPU em áreas como processamento de imagens e visão computacional [42, 61, 62, 63]. Os *workloads* são separados em quarto subsecções diferentes [63]:

- *Crypto* - Este tipo de *workload* mede o desempenho das instruções *crypto* do computador através da execução de tarefas criptográficas que fazem um elevado uso de instruções *crypto*.
- *Integer* - É feita a medição do desempenho das intruções *integer* através da realização de tarefas intensivas ao processador, que fazem um grande uso de instruções integer. Todos os softwares fazem uso de instruções *integer*, o que significa que uma boa pontuação de *integer* indica uma boa eficiência em geral.
- *Floating Point* - É feita a medição do desempenho de *floating points* através da realização de variadas tarefas intensivas ao processador, que fazem um grande uso de operações *floating point*.
- *Memory* - A latência e largura de banda da memória são alvos de estudo do desempenho. O *software* que faz uso de grandes estruturas de dados ou de estruturas de dados referenciais depende de uma boa eficiência da memória para manterem o processador num ritmo constante de execução de instruções.

Na tabela 3.3 encontram-se as quarto subsecções de *workloads* e as métricas respetivas de cada *workload*.

Tipo de *Workload*	*Workload*	Métrica
Cryptography Workloads	AES	GB/sec
Integer Workloads	LZMA Compression	MB/sec
	JPEG Compression	Mpixels/sec
	Canny	Mpixels/sec
	Dijkstra	MB/sec
	SQLite	MTE/sec
	LLVM	Krows/sec
	HTML 5 Parse	MB/sec
	HTML 5 DOM	MElements/sec
	PDF	Mpixels/sec
	Lua	MB/sec
	Camera	images/sec
Floating Point Workloads	GEMM	Gflops
	FFT	Gflops
	Gaussian Blur	Mpixels/sec
	Ray Trace	Kpixels/sec
	N-Body Physics	Mpairs/sec
	Rigid Body Physics	FPS
	Speech Recognition	Words/sec
Memory Workloads	Memory Copy	GB/sec
	Memory Latency	ns
	Memory Bandwidth	GB/sec

Tabela 3.3. Subsecções de *workload* do Geekbench de acordo com [42].

Os *workloads* anteriormente referidos são executados todos por um teste geral que faculta todos os dados necessários para a análise de desempenho de CPU.

Um dos motivos pelos quais este *benchmark* foi escolhido para ser implementado no *test bed* reside no facto de ser uma ferramenta gratuita e cross-platform. O outro motivo é a grande quantidade e diversidade de *workloads* disponibilizados pelo *benchmark*.

3.8 Conclusão

Neste capítulo foi descrito o ambiente experimental que faz uso da virtualização ao nível de *hardware*, sendo apresentada toda a informação sobre a infraestrutura *failover cluster* sobre Hyper-V e todos os passos para a sua implementação.

O ambiente experimental que faz uso da virtualização ao nível do sistema operativo também é apresentado através da descrição da tecnologia de virtualização Docker e através da descrição dos passos para a implementação do modo *Swarm* para a construção de um *failover cluster* ou também designado de *swarm*.

As ferramentas de *benchmark*, Geekbench, RAMspeed, IOzone e iPerf são descritas e apresentados todos os testes realizados por cada ferramenta, preparando assim a base para a análise de resultados realizada no capítulo 4.

Capítulo 4

Análise dos Resultados Experimentais

4.1 Introdução

Neste capítulo são apresentados os resultados experimentais, obtidos através das ferramentas de *benchmarks* referidas no capítulo 3, organizados por tipo de componente. Através da análise dos resultados são obtidas múltiplas conclusões acerca do desempenho de cada infraestrutura de elevada disponibilidade. As conclusões obtidas dos resultados das ferramentas de *benchmark* Geekbench, IOzone, RAMspeed e iperf permitiram determinar qual das implementações das infraestruturas virtualizadas de elevada disponibilidade possui o melhor desempenho (*Failover Cluster* sobre Hyper-V ou Docker *Swarm*).

4.2 Implementação dos Testes

Foram realizados testes sobre as duas infraestruturas de elevada disponibilidade implementadas no Capítulo 3, ao nível da CPU, memória, disco e rede, tendo sido realizados três testes por infraestrutura para cada *benchmark*, um ao nível nativo e dois realizados sobre elementos virtualizados que pertencem à infraestrutura. No *cluster* que usa a tecnologia Hyper-V foi submetido a testes um dos nós que possui o sistema operativo nativo, e duas máquinas virtuais que se encontram nos nós, uma das máquinas virtuais possui como sistema operativo o Linux Ubuntu 16.04 e a outra possui o Windows *Server* 2016.

Na infraestrutura Docker Swarm foram recolhidos dados ao nível nativo e virtual, foram recolhidos dados de um *container* que se encontra em execução sobre um nó nativo e de outro *container* que se encontra em execução numa máquina virtual, estando este *container* num ambiente de virtualização *nested*. O sistema operativo usado nos nós do *Swarm* é o Linux Fedora 25 *Workstation* e a imagem do sistema operativo usada nos *containers* é a do Linux Ubuntu 16.04.

Os processadores das máquinas físicas e virtuais foram analisados pelo *benchmark* Geekbench 4 que nos permitiu fazer uma análise comparativa do desempenho da CPU em diferentes tipos de tecnologia de virtualização (Hyper-V e Docker Swarm). O *benchmark* RAMspeed permitiu executar testes de desempenho da memória dos diferentes sistemas presentes nas máquinas virtuais ou *containers* das infraestruturas de elevada disponibilidade, sendo obtidos resultados utilizados para uma análise comparativa. Os sistemas de ficheiros das máquinas nativas e das

que se encontram sobre as infraestruturas virtualizadas são avaliados pela ferramenta de análise de desempenho IOzone.

Com vista a analisar o desempenho *cross-platforming* das rede físicas e virtualizadas é usada a ferramenta de *benchmark* iPerf, que realiza testes de latência, largura de banda e perdas entre máquinas que possuem o mesmo sistema operativo e máquinas com diferentes sistemas operativos.

A realização de dois testes de migração, um sobre uma máquina virtual Windows e o outro sobre uma máquina virtual Linux, na infraestrutura *Failover Cluster* sobre Hyper-V permite a obtenção de dados através dos logs, nomeadamente o tempo de migração das máquinas virtuais. De forma semelhante, foram realizados testes de migração sobre um *container* que se encontra sobre Docker nativo e outro *container* que se encontra num ambiente *nested* do Docker.

4.3 Comparação Experimental do Desempenho das Infraestruturas Implementadas

4.3.1 Resultados Experimentais Obtidos com o IOzone

A ferramenta IOzone permitiu realizar múltiplos testes. As máquinas nativas e virtuais do *failover cluster* em Hyper-V e os *containers* do Docker Swarm foram alvo de estudo, sendo executados os comandos da ferramenta IOzone descritos na secção 3.4, permitindo obter os resultados que são apresentados no gráfico da figura 4.1.

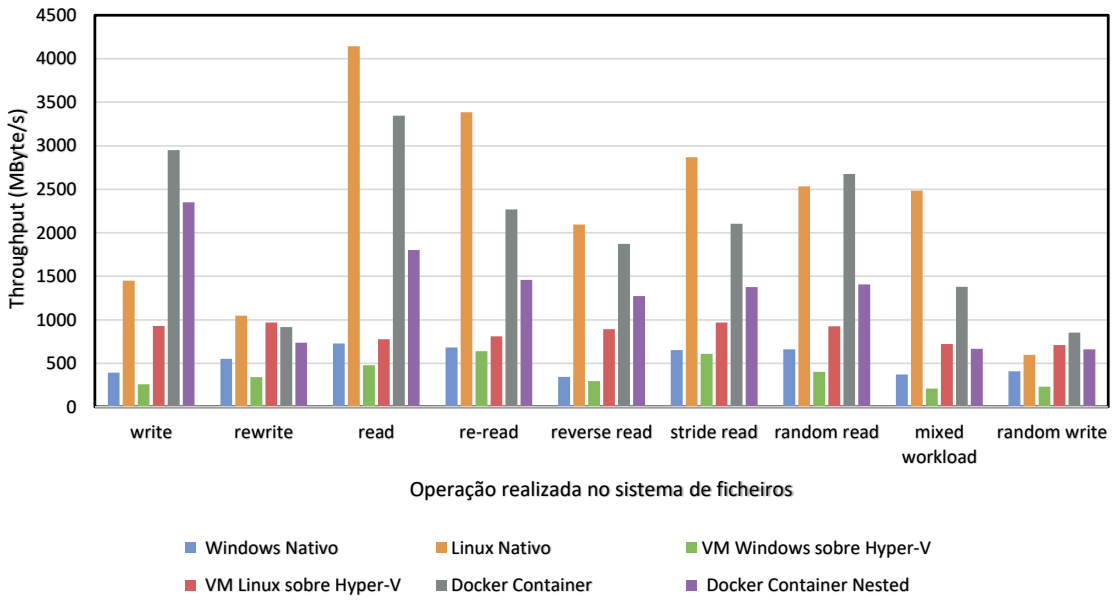

Figura 4.1. Resultados experimentais obtidos com a ferramenta IOzone.

Com base no gráfico anterior, podemos verificar que a tecnologia de virtualização Docker possui um desempenho superior em comparação ao Hyper-V ao nível de operações sobre o sistema de ficheiros. Através da análise do gráfico podemos concluir que o sistema de ficheiros num sistema operativo do tipo Linux possui uma eficiência superior ao do Windows, porque as máquinas virtuais sobre Hyper-V com sistema operativo Linux apresentam melhor desempenho em todas as operações realizadas ao sistema de ficheiros em comparação às que possuem o sistema operativo Windows.

Verificou-se também a presença de *overheads* de virtualização nas duas infraestruturas nos sistemas de ficheiros pois existe uma degradação de desempenho em comparação com os sistemas operativos nativos, sendo a degradação menor nas máquinas virtuais do hyper-v com Linux em comparação com as máquinas do hyper-v com Window, pois o sistema de ficheiros é mais eficiente.

4.3.2 Resultados Experimentais Obtidos com o RAMspeed

As máquinas nativas e virtuais do *failover cluster* em Hyper-V e os *containers* do Docker *Swarm* foram alvo de estudo, sendo executados os comandos da ferramenta RAMspeed descritos na secção 3.5, para a obtenção de resultados. Com base nos dados recolhidos foi elaborado um gráfico que permite realizar uma analise comparativa. O gráfico da figura 4.2 dispõem o *throughput* da memória ao nível de inteiros e de floats:

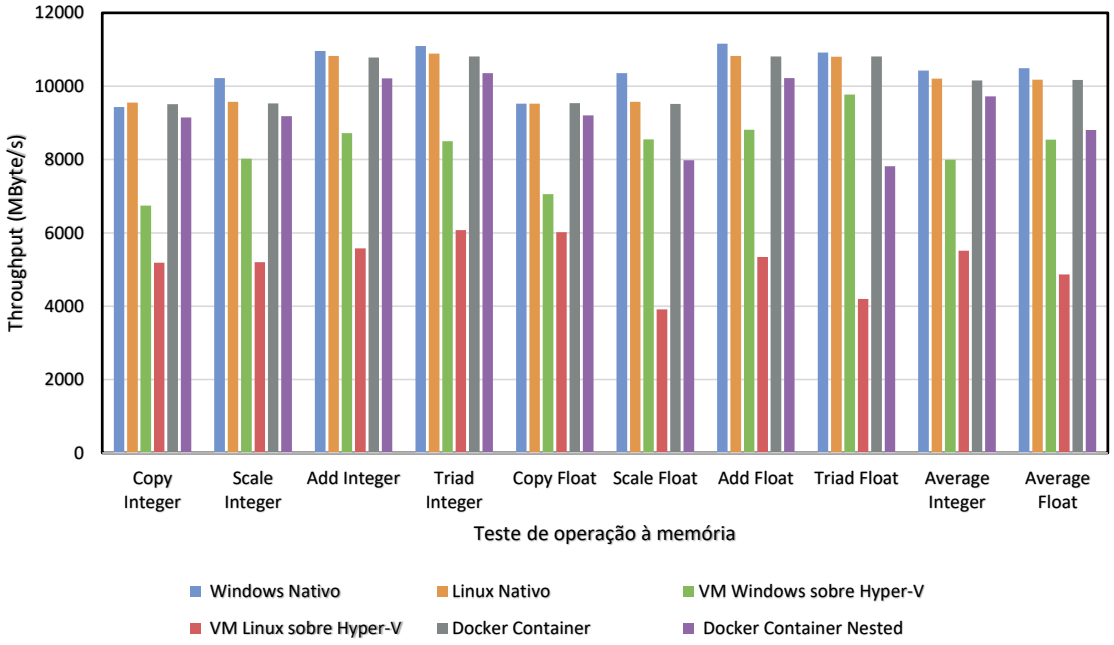

Figura 4.2. Resultados experimentais obtidos com a ferramenta RAMspeed.

O desempenho ao nível da memória na infraestrutura do Docker *Swarm* é superior à do Hyper-v, sendo visível no grafico em todos os testes de operações à memória. Ao comparar de forma

isolada cada infraestrutura, podemos verificar através dos testes de operações à memória que, na infraestrutura do Hyper-V uma máquina virtual com Windows revela um desempenho superior a uma máquina virtual com Linux. No ambiente Docker *Swarm*, verificamos que o Docker sobre um *host* nativo possui melhor desempenho do que o Docker *nested*, devido à existência de *overheads* de memória, abordados no capítulo 2. Ao nível nativo, o Docker introduz um overhead de memória muito reduzido conforme pode ser observado no gráfico da figura 4.2.

4.3.3 Resultados Experimentais Obtidos com o Geekbench

Ao nível da CPU foram realizados testes sobre as duas infraestruturas, sendo realizado 3 testes por infraestrutura, um ao nível nativo e dois realizados sobre elementos virtualizados que pertencem à infraestrutura. Os testes consistem na execução da ferramenta Geekbench sobre as máquinas nativas e virtuais do *failover cluster* em Hyper-V e sobre os *containers* do Docker *Swarm*, sendo obtidos os resultados que se encontram apresentados na tabela 4.1.

Workload	Windows Nativo	Linux Nativo	Windows sobre Hyper-V	Linux sobre Hyper-V	Docker *Container*	Docker *Container Nested*
AES (GB/s)	1,62	1,84	2,60	1,76	2,36	2,29
LZMA Compression (MB/s)	4,48	4,87	4,72	3,06	6,85	6,63
JPEG Compression (MP/s)	25,2	38,1	31,1	22,6	41,2	39,4
Canny (MP/s)	43,8	56,1	39,6	34,2	67,6	66,2
Dijkstra (MB/s)	1,59	2,37	2,53	2,23	3,79	3,59
SQLite (MTE/s)	55,4	136,4	53,8	121,7	148,1	143,4
LLVM (Krows/s)	476,2	514,8	467,6	344,6	628,4	605,3
HTML 5 Parse (MB/s)	12,0	23,5	7,79	12,2	25,3	24,5
HTML 5 DOM (MElements/s)	2,34	3,66	2,59	3,25	4,12	3,99
PDF (MP/s)	80,6	123,1	104,6	131,8	158,6	153,8
Lua (MB/s)	2,93	5,17	2,70	2,69	5,73	5,56
Camera (images/s)	6,99	12,4	7,52	8,50	14,9	13,5
GEMM (Gflops)	92,6	87,9	68,6	29,2	105,7	104,6
FFT (Gflops)	10,8	12,6	4,25	12,4	13,0	13,0
Gaussian Blur (MP/s)	76,3	48,1	53,6	48,4	92,5	89,5
Ray Trace (KP/s)	604,9	410,2	541,6	380,5	430,7	417,7
N-Body Physics (Mpairs/s)	3,16	3,49	2,06	2,44	3,65	3,55
Rigid Body Physics (FPS)	11308,0	13006,5	9429,4	5417,8	14287,7	13819,7
Speech Recognition (Words/s)	27,2	30,1	39,3	38,8	46,7	45,6
Memory Copy (GB/s)	8,76	5,37	9,07	7,19	9,48	8,26
Memory Latency (ns)	93,7	117,9	73,5	81,8	111,6	102,6
Memory Bandwidth (GB/s)	11,7	7,96	13,7	11,2	12,6	13,8
Score	3265	3665	3190	3015	4677	4577

Tabela 4.1. Resultados dos testes realizados com o Geekbench 4.

Com base nos resultados da Tabela 4.1, podemos verificar que a virtualização baseada em *containers* possui um melhor desempenho do que a baseada em *hypervisors*, sendo o score atribuído

pelo Geekbench ao Docker de 4677 e o score atribuído ao Hyper-V Windows de 3190. Quanto à infraestrutura do Hyper-V, com base nos resultados obtidos, conclui-se que uma máquina virtual com sistema operativo Windows possui melhor desempenho do que com Linux, apesar de a diferença ser diminuta. Contudo, a máquina virtual Linux possui melhor desempenho que a máquina virtual com Windows para certos *workloads* (SQLite, HTML 5 Parse, HTML 5 DOM, PDF, Camera, FFT e N-Body Physics) pelo facto de possuir bibliotecas mais otimizadas. Sendo realizado um teste no nativo, verifica-se a existência de *overheads* na virtualização pois os scores das máquinas virtuais são inferior ao valor 3265 que é a pontuação do Windows *Server* 2016 nativo. Na infraestrutura do Docker *Swarm* os *scores* foram superiores ao sistema operativo nativo, pois os *containers* são mais especializados, possuindo as suas próprias bibliotecas e binários, não sendo necessário recorrer às do sistema operativo *host*. O nó que se encontra sobre o *hypervisor* KVM apresenta uma degradação de desempenho em comparação ao Docker nativo pois existem *overheads* de virtualização. Contudo este continua a possuir um melhor desempenho em comparação com o Linux nativo.

4.3.4 Resultados Experimentais Obtidos com o iPerf

Os dados obtidos através dos dois testes (largura de banda do TCP e jitter do UDP) realizados de acordo com as instruções da ferramenta iPerf (secção 3.6) permitiram a elaboração de dois gráficos comparativos ao nível do desempenho da rede. Os testes foram realizados sempre com recurso a um par de máquinas virtuais ou *containers*, funcionando uma como servidor e outra como cliente. O gráfico da figura 4.3 apresenta um estudo comparativo *cross-platform* da largura de banda das redes virtuais de cada infraestrutura.

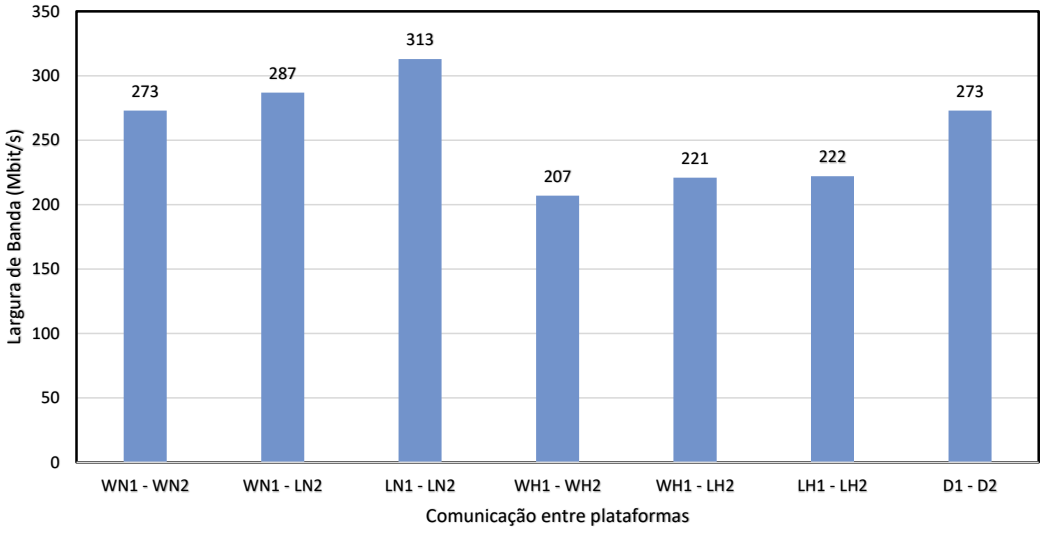

Figura 4.3. Resultados experimentais da largura de banda de TCP obtidos com a ferramenta iPerf3. WN1 e WN2: diferentes nós do *cluster* com Windows nativo, LN1 e LN2: diferentes nós do *cluster* com Linux nativo, WH1 e WH2: máquinas virtuais Windows sobre o Hyper-V em diferentes nós do *cluster*, LH1 e LH2: máquinas virtuais Linux sobre o Hyper-V em diferentes nós do *cluster*, D1 e D2: *containers* Docker em diferentes nós do *Swarm*.

O gráfico da figura 4.4 apresenta o *delay* existente nas redes físicas e virtuais associado ao protocolo UDP.

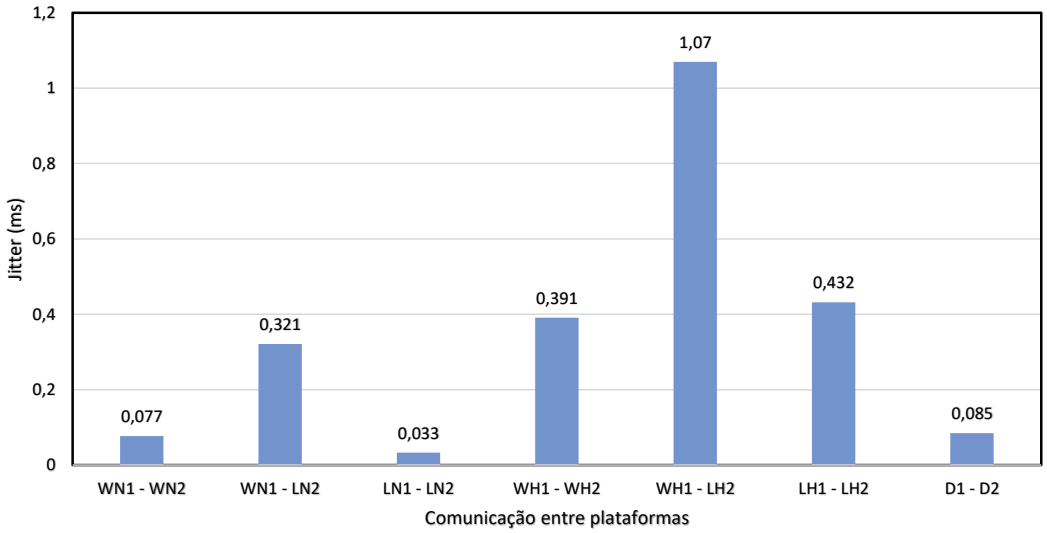

Figura 4.4. Resultados experimentais do *jitter* de UDP obtidos com a ferramenta iPerf3. WN1 e WN2: diferentes nós do *cluster* com Windows nativo, LN1 e LN2: diferentes nós do *cluster* com Linux nativo, WH1 e WH2: máquinas virtuais Windows sobre o Hyper-V em diferentes nós do *cluster*, LH1 e LH2: máquinas virtuais Linux sobre o Hyper-V em diferentes nós do *cluster*, D1 e D2: *containers* Docker em diferentes nós do *Swarm*.

Através dos gráficos representados nas figuras 4.3 e 4.4 podemos concluir que a largura de banda é superior na infraestrutura do Docker Swarm (273 Mbits/s) em comparação com a do *Failover Cluster* sobre Hyper-V (222 Mbits/s).

Na infraestrutura do *Failover Cluster* sobre Hyper-V podemos verificar com base nos três testes feitos que duas máquinas virtuais Linux comunicam mais rápido do que duas máquinas virtuais Windows, sendo a largura de banda entre uma máquina Linux e uma máquina Windows muito semelhante à largura de banda entre duas máquinas Linux. Os *overheads* de virtualização encontram-se presentes também nas redes virtuais, tendo estas uma eficiência mais reduzida do que as redes nativas.

Também foram realizados testes de largura de banda usando o protocolo UDP, mas o resultado foi o mesmo para todas as redes (1,05 Mbits/s). O jitter é a métrica a ser utilizada pois os resultados da largura de banda não permitem obter uma conclusão. A análise do jitter permitiu obter a conclusão que este é inferior na rede da infraestrutura Docker *Swarm* em comparação com a rede do *Failover Cluster* Hyper-V. Nesta rede o *jitter* é elevado quando as máquinas virtuais possuem sistemas operativos diferentes, e é reduzido quando os sistemas operativos são iguais. Neste gráfico também é possível observar a existência de *overheads* na virtualização da rede pois o *jitter* nas redes das infraestruturas virtuais é superior ao das redes nativas.

4.3.5 Desempenho de Migração de Máquinas Virtuais versus *Containers*

As infraestruturas implementadas são de elevada disponibilidade o que permite a realização de migração de máquinas virtuais (infraestrutura *Failover Cluster* Hyper-V) ou de *containers* (infraestrutura Docker *Swarm*). Como o objetivo da dissertação consiste na comparação das duas infraestruturas de elevada disponibilidade, foi também incluído o desempenho da migração de máquinas virtuais e *containers*. Os logs facultados pelas aplicações (Failover Cluster Manager da Microsoft e a API 1.25 da Docker) que fazem a gestão das infraestruturas permitem obter os tempos de migração, os quais são apresentados no gráfico da figura 4.5.

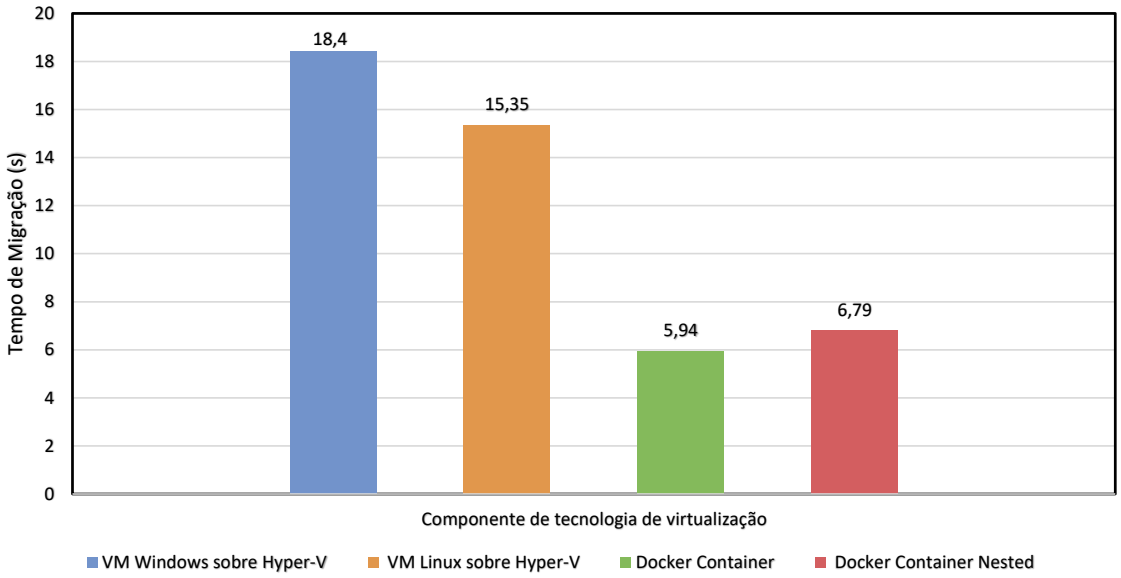

Figura 4.5. Tempos de migração das máquinas virtuais e de *containers*.

Os dados do gráfico da figura 4.5 permitem a obtenção de múltiplas conclusões, sendo a primeira que os tempos de migração da infraestrutura com virtualização ao nível de sistema operativo (o tempo de migração do Docker *container* é de 5,94 segundos e o tempo de migração do *nested* Docker *container* é de 6,79 segundos) são inferiores ao da infraestrutura com virtualização ao nível de *hardware* (o tempo de migração da máquina virtual com Windows é de 18,40 segundos e o tempo de migração da máquina virtual com Linux é de 15,35 segundos), revelando assim a existência de um melhor desempenho de migração na virtualização ao nível do sistema operativo. Este facto pode resultar também do sistema operativo que se encontra em execução nas máquinas virtuais, o que permite chegar à segunda conclusão, uma máquina virtual com um sistema operativo Linux, demora menos tempo na migração que uma máquina virtual que possua Windows.

Há outros fatores que também podem influenciar estes resultados, nomeadamente, os *overheads* ao nível do sistema de ficheiros pois na infraestrutura que faz uso do *hypervisor* é usado um sistema de armazenamento de dados e na infraestrutura que usa o Docker *Engine* o armazena-

mento é feito localmente, podendo afetar os tempos de migração.

A terceira conclusão obtida é que o tempo de migração no ambiente Docker *nested* (6,79 segundos) é superior ao tempo de migração no ambiente Docker sobre a máquina nativa (5,94 segundos), devido à presença de *overheads* associados à camada de virtualização relativa ao *hypervisor* (KVM).

4.4 Conclusão

Neste capítulo é descrita a implementação dos testes realizados pelas ferramentas de *benchmark* (Geekbench, IOzone, Ramspeed e Iperf), e a implementação dos testes de migração realizados às máquinas virtuais e *containers*.

Os resultados obtidos através da elaboração dos testes das ferramentas de *benchmark* foram alvo de análise e permitiram a obtenção de uma conclusão, sendo esta que a infraestrutura que faz uso da virtualização ao nível do sistema operativo (Docker *Swarm*) ao nível da CPU, memória, sistema de ficheiros e rede possui um melhor desempenho que a infraestrutura baseada em virtualização ao nível de *hardware* (*Failover Cluster* sobre Hyper-V).

Os testes de migração realizados às máquinas virtuais e *containers* fornecem resultados que permitem concluir que um *container* tem um menor tempo de migração em comparação com uma máquina virtual. As máquinas virtuais com o sistema operativo Linux têm um menor tempo de migração em comparação com as máquinas virtuais com o sistema operativo Windows.

Capítulo 5

Conclusão e Trabalho Futuro

5.1 Principais Conclusões

As tecnologias de virtualização (Hyper-V e Docker) são implementadas em infraestruturas de elevada disponibilidade (*Failover Cluster* sobre Hyper-V e Docker *Swarm*) que permitem a elaboração de múltiplos testes, através dos quais é possível obter conclusões para o estudo comparativo entre as duas tecnologias.

Ao analisar a tecnologia de virtualização Hyper-V foi possível concluir que esta permite a implementação de uma infraestrutura de elevada disponibilidade cujas máquinas virtuais podem possuir uma grande diversidade de sistemas operativos em ambientes isolados, sendo esta infraestrutura uma das mais flexíveis e com um elevado grau de isolamento. Este tipo de infraestrutura é uma das mais implementadas em empresas e *datacenters* de média e grande dimensão, contudo possui desvantagens como a elevada complexidade de implementação e lentidão na inicialização e no armazenamento de imagens de máquinas virtuais.

O estudo da tecnologia de virtualização Docker permitiu concluir que uma infraestrutura de elevada disponibilidade que implemente este tipo de tecnologia é mais simples, dinâmica e eficiente em comparação com infraestruturas de elevada disponibilidade que façam uso de virtualização ao nível do *hardware*, pois os *containers* possuem bibliotecas e binários específicos para a sua funcionalidade (ex:servidor web e base de dados), não sendo necessário a partilha de bibliotecas e binários do sistema operativo entre *containers*.

Contundo existem desvantagens que se encontram presentes na tecnologia de *containers*, sendo estas: O facto dos *containers* apenas poderem possuir serviços que executem sobre sistemas operativos da mesma família (o mesmo *kernel*) que a máquina *host* e o facto de possuirem um grau de isolamento de aplicação inferior às máquinas virtuais.

Ao nível de implementação é possível concluir que é muito mais complexo implementar a infraestrutura de *Failover Cluster* usando Hyper-V em comparação com a infraestrutura Docker *Swarm*, conforme explicado na secção 3.3.

O estudo dos resultados recolhidos pelas ferramentas de *benchmark* (Geekbench, IOzone, RAMspeed e iPerf) permite concluir que a infraestrutura baseada em *containers* Docker possui um melhor desempenho ao nível da CPU, memória RAM, sistema de ficheiros, rede e migração de *containers* em comparação com a infraestrutura baseada no *hypervisor* Hyper-V, aumentando o

interesse das empresas pelas tecnologias de *containers*.

Por outro lado, também se observou que as máquinas virtuais e *containers* demonstram um desempenho inferior às máquinas nativas devido à presença dos *overheads* de virtualização.

5.2 Sugestões de Trabalho futuro

As infraestruturas implementadas nesta dissertação representam um ambiente laboratorial de pequena dimensão, que pode ser expandido através da incorporação de mais nós ou de mais *hypervisors* de variados tipos e tecnologias de *containers*. As infraestruturas possuem uma escala que permite a elaboração de testes, contundo seria também interessante obter resultados experimentais de infraestruturas de elevada disponibilidade virtualizadas em ambientes reais. O baixo grau de isolamento de *containers*, faz com que a sua segurança possa ser comprometida, por esta razão um estudo futuro deveria ser realizado com o objetivo de melhorar a segurança em ambientes de *containers*.

Bibliografia

[1] M. Armbrust *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. 1

[2] RightScale. (2018) State of the cloud report. Ultimo acesso: 6/6/2018. [Online]. Available: https://www.rightscale.com/lp/state-of-the-cloud 1

[3] Oracle. (2011) Introduction to virtualization. Ultimo acesso: 15/6/2018. [Online]. Available: https://docs.oracle.com/cd/E20065_01/doc.30/e18549/intro.htm 1, 5

[4] K. Hwang, G. C.Fox, and J. J.Dongarra, *Distributed and Cloud Computing From Parallel Processing to the Internet of Things*. Elsevier, Outubro 2011, ch. 3. xv, 1, 6, 7, 8, 9

[5] Netflix. Ultimo acesso: 18/6/2018. [Online]. Available: https://www.netflix.com/pt/ 1

[6] A. Leung, A. Spyker, and T. Bozarth, "Titus: Introducing containers to the netflix cloud," *Communications of the ACM*, vol. 61, no. 2, pp. 38–45, Fevereiro 2018. 2

[7] ——, "Titus: Introducing containers to the netflix cloud," *ACM Queue*, vol. 15, no. 5, p. 30, Novembro 2017. 2

[8] Microsoft. Hyper-v. Ultimo acesso: 15/6/2018. [Online]. Available: https://msdn.microsoft.com/pt-pt/library/mt169373(v=ws.11).aspx 2

[9] Docker. Docker - build, ship, and run any app, anywhere. Ultimo acesso: 18/6/2018. [Online]. Available: https://www.docker.com 2

[10] Popek and Goldberg, "Formal requirements for virtualizable third generation architectures." Julho 1974, ultimo acesso: 19/6/2018. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4815&rep=rep1&type=pdf 5

[11] H. Lee, "Virtualization basics: Understanding techniques and fundamentals." 2014, ultimo acesso: 19/6/2018. [Online]. Available: http://dsc.soic.indiana.edu/publications/virtualization.pdf 5, 9, 10, 13

[12] IBM, "Virtualization in education." Outubro 2007, ultimo acesso: 19/6/2018. [Online]. Available: http://www-07.ibm.com/solutions/in/education/download/Virtualization%20in%20Education.pdf 5

[13] S. Campbell and M. Jeronimo, "An introduction to virtualization." *Applied Virtualization*, pp. 1–15, 2006, intel. 5, 6

[14] P. Enberg, "A performance evaluation of hypervisor, unikernel, and container network i/o virtualization." Maio 2016, ultimo acesso: 19/6/2018. [Online]. Available: https:

//pdfs.semanticscholar.org/6c18/39128cc8451e7cda5d16f1159d7847c4fb4b.pdf xvii, 6, 12, 13, 14, 17, 18, 20, 23, 25

[15] A. Aspernäs and M. Nensén, "Container hosts as virtual machines." 2016, ultimo acesso: 19/6/2018. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1014984/FULLTEXT01.pdf xv, 9, 12, 13, 14, 15, 16

[16] Oracle. Oracle vm server for x86. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.oracle.com/virtualization/vm-server-for-x86/index.html 10

[17] S. Walker. Pcem. Ultimo acesso: 15/6/2018. [Online]. Available: https://pcem-emulator.co.uk/ 10

[18] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, "Benchmarking the performance of microsoft hyper-v server, vmware esxi and xen hypervisors," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 4, pp: 922-933, no. 12, Dezembro 2013. 10, 15

[19] DOSBox. Dosbox, an x86 emulator with dos. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.dosbox.com/ 10

[20] W. Felter *et al.*, "An updated performance comparison of virtual machines and linux containers." *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172, Março 2015. 10, 14, 24, 25

[21] Q. team. Qemu. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.qemu.org/ 10

[22] IBM. z/vm virtualization technology. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.ibm.com/it-infrastructure/z/zvm 10

[23] Oracle. Oracle vm virtualbox. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.virtualbox.org/ 10

[24] Proxmox. Proxmox ve. Ultimo acesso: 15/6/2018. [Online]. Available: https://pve.proxmox.com/wiki/Main_Page 10

[25] VMware. Fusion. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.vmware.com/products/fusion.html 10

[26] ——. Esxi. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.vmware.com/products/esxi-and-esx.html 10

[27] SYSGO. Pikeos hypervisor - sysgo - embedding innovations. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.sysgo.com/products/pikeos-hypervisor/ 10

[28] L. Foundation. The xen project, the powerful open source industry standard for virtualization. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.xenproject.org/ 10

[29] VMware. Workstation pro. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.vmware.com/products/workstation-pro.html 10

[30] J. Shropshire, "Analysis of monolithic and microkernel architectures: Towards secure hypervisor design." *2014 47th Hawaii International Conference on System Sciences*, pp. 5008–5017, Janeiro 2014. 11

[31] C. Commons. Linux containers - lxc - introduction. Ultimo acesso: 15/6/2018. [Online]. Available: https://linuxcontainers.org/lxc/introduction/ 16

[32] Parallels. Openvz virtuozzo containers wiki. Ultimo acesso: 15/6/2018. [Online]. Available: https://openvz.org/Main_Page 16

[33] Google. lmctfy - let me contain that for you. Ultimo acesso: 15/6/2018. [Online]. Available: https://github.com/google/lmctfy 16

[34] FreeBSD. Freebsd handbook chapter 14. jails. Ultimo acesso: 15/6/2018. [Online]. Available: https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html 16

[35] T. K. Authors. Production-grade container orchestration. Ultimo acesso: 14/4/2018. [Online]. Available: https://kubernetes.io/ 16

[36] A. Yemelianov. (2017, Fevereiro) Systemd and containers: An introduction to systemd-nspawn. Ultimo acesso: 15/6/2018. [Online]. Available: https://blog.selectel.com/systemd-containers-introduction-systemd-nspawn/ 16

[37] Oracle. Oracle solaris containers. Ultimo acesso: 15/6/2018. [Online]. Available: http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html 16

[38] Docker. Docker concepts. Ultimo acesso: 14/4/2018. [Online]. Available: https://docs.docker.com/get-started/#docker-concepts 16

[39] M. Ben-Yehuda *et al.*, "The turtles project: Design and implementation of nested virtualization." 2010, ultimo acesso: 19/6/2018. [Online]. Available: https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Ben-Yehuda.pdf xv, 17

[40] N. Yaqub, "Comparison of virtualization performance: Vmware and kvm." Maio 2012, ultimo acesso: 19/6/2018. [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/34900/Yaqub.pdf 20, 21, 24, 25

[41] V. Adla, "Comparing performance of hyper-v and vmware considering network isolation in virtual machines." Setembro 2013, ultimo acesso: 19/6/2018. [Online]. Available: http://trap.ncirl.ie/907/1/vadla.pdf 21, 24, 25

[42] P. Labs. Geekbench 4 cpu workloads. Ultimo acesso: 18/4/2018. [Online]. Available: https://www.geekbench.com/doc/geekbench4-cpu-workloads.pdf xvii, 21, 35, 36

[43] R. M. Hollander and P. V. Bolotoff. Ramspeed, a cache and memory benchmarking tool. Ultimo acesso: 18/4/2018. [Online]. Available: http://alasir.com/software/ramspeed/ 21, 33, 34

[44] D. Capps and W. Norcott. Iozone filesystem benchmark. Ultimo acesso: 18/4/2018. [Online]. Available: http://www.iozone.org/ 21, 32

[45] Iperf.fr. iperf - the ultimate speed test tool for tcp, udp and sctp. Ultimo acesso: 14/4/2018. [Online]. Available: https://iperf.fr/ 21, 34

[46] A. Kurmus *et al.*, "A comparison of secure multi-tenancy architectures for filesystem storage clouds." *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*, pp. 471–490, Dezembro 2011. 21, 22, 23, 25

[47] C. D. Graziano, "A performance analysis of xen and kvm hypervisors for hosting the xen worlds project." *Graduate Theses and Dissertations*, no. 12215, p. 49, 2011, ultimo acesso: 19/6/2018. [Online]. Available: https://lib.dr.iastate.edu/etd/12215 23, 25

[48] S. Sridharan, "A performance comparison of hypervisors for cloud computing." Agosto 2012, ultimo acesso: 19/6/2018. [Online]. Available: https://digitalcommons.unf.edu/cgi/viewcontent.cgi?article=1275&context=etd 23, 25

[49] K. Agarwal, "A study of virtualization overheads." Agosto 2015, ultimo acesso: 19/6/2018. [Online]. Available: http://www.oscar.cs.stonybrook.edu/papers/files/KavitaAgarwalMSThesisSubmission.pdf 23, 25

[50] I. Voras, M. Orlić, and B. Mihaljević, "An early comparison of commercial and open-source cloud platforms for scientific environments." *KES-AMSTA 2012: Agent and Multi-Agent Systems. Technologies and Applications*, pp. 164–173, 2012. 24, 25

[51] C. Jianga *et al.*, "Energy efficiency comparison of hypervisors." *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8, Novembro 2016. 24, 25

[52] VMware, "A performance comparison of hypervisors." Janeiro 2007, ultimo acesso: 19/6/2018. [Online]. Available: https://www.vmware.com/pdf/hypervisor_performance.pdf 24, 25

[53] V. V. Reddy and L. Rajamani, "Performance evaluation of hypervisors in the private cloud based on system information using sigar framework and for system workloads using passmark." *International Journal of Advanced Science and Technology*, vol. 70, pp. 17–32, Setembro 2014. 25

[54] iXsystems. Freenas storage operating system. Ultimo acesso: 24/3/2018. [Online]. Available: http://www.freenas.org 30

[55] ——. Features-freenas-open source storage operating system. Ultimo acesso: 24/3/2018. [Online]. Available: http://www.freenas.org/about/features/ 30

[56] M. T. Corporation. iscsi basics: A practical introduction. Ultimo acesso: 19/6/2018. [Online]. Available: http://www.mosaictec.com/pdf-docs/whitepapers/iSCSI_Guide.pdf 31

[57] Microsoft. (2017, Abril) Deploy a hyper-v cluster. Ultimo acesso: 7/4/2018. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/jj863389(v=ws.11) 31

[58] M. S. Pro. (2017, Maio) Implementing failover clustering with windows server 2016 hyper-v. Ultimo acesso: 7/4/2018. [Online]. Available: http://www.msserverpro.com/implementing-failover-clustering-windows-server-2016-hyper-v/ 31

[59] Docker. Getting started with swarm mode. Ultimo acesso: 14/4/2018. [Online]. Available: https://docs.docker.com/engine/swarm/swarm-tutorial/ 32

[60] L. Blog. Iozone i/o operations. Ultimo acesso: 18/4/2018. [Online]. Available: http://kongll.github.io/2015/06/19/iozone/ 32

[61] P. Labs. Geekbench 4 - cross-platform benchmark. Ultimo acesso: 18/4/2018. [Online]. Available: https://www.geekbench.com 35

[62] ——. Downloading geekbench 4. Ultimo acesso: 18/4/2018. [Online]. Available: https://www.geekbench.com/download/ 35

[63] ——. Interpreting geekbench 4 scores. Ultimo acesso: 18/4/2018. [Online]. Available: http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-4-scores 35

Apêndice A

Anexos

A.1 Passos da Implementação do *Failover Cluster* sobre Hyper-V

- Conectar todas máquinas físicas ou virtuais em rede para que possam comunicar (conectar os nós ao *storage* e entre si).

    1. Abrir o *Server Manager*, clicar no menu Tools e de seguida clicar em iSCI *Initiator*.
    2. Na janela iSCSI *Initiator Properties* clicar na secção *Target*, inserir o endereço de IP do servidor de armazenamento de dados e clicar *Quick Connect*.
    3. Clicar na secção *Volumes and Devices*, clicar em Auto Configure e de seguida clicar em OK.
    4. No *Server Manager* clicar no menu Tools e de seguida clicar em *Computer Management*.
    5. Clicar em *Disk Management*.
    6. Clicar com o botão direito do rato nos discos não alocados, clicar em *Online*.
    7. Clicar com o botão direito do rato nos discos não alocados, clicar em *Initialize Disk* e clicar OK.
    8. Clicar com o botão direito do rato nos discos não alocados, clicar em *New Simple Volume*.
    9. Na página *Welcome to the New Simple Volume Wizard* clicar *Next*.
    10. Na página *Specify Volume Size* do *wizard* clicar *Next*.
    11. Na página *Assign Drive Letter or Path* do *wizard* clicar *Next*.
    12. Na página *Format Partition do wizard*, na caixa *Volume label* escrever o nome do disco, clicar *Next*.
    13. Na página *Completing the New Simple Volume Wizard* clicar *Finish*.
    14. Repetir o mesmo processo para o outro nó do cluster.

- Instalar nos nós o Hyper-V e a funcionalidade de *Failover Clustering*.

    1. Abrir o *Server Manager*, clicar no menu *Manage* e clicar em *Add Roles and Features*.
    2. Na página *Before you begin do Add Roles and Features*, clicar *Next*.

3. Na página *Select installation type*, selecionar *Role-based or feature-based installation*, clicar *Next*.
4. Na página *Select destination server*, assegurar a seleção do servidor, clicar *Next*.
5. Na página *Select server roles*, selecionar o Hyper-V,na caixa da dialogo *Add Roles and Features Wizard* clicar *Add Features*,clicar *Next*.
6. Na página *Select features*,selecionar o *Failover Clustering*, na caixa da dialogo *Add Roles and Features Wizard* clicar *Add Features*, clicar *Next*.
7. Na página Hyper-V, clicar *Next*.
8. Na página *Create Virtual Switches*, garantir que nada é selecionado, clicar *Next*.
9. Na página *Virtual Machine Migration*, clicar *Next*.
10. Na página *Virtual Default Stores*, clicar *Next*.
11. Na página *Confirm installation selections*, selecionar *Restart the destination server automatically if required*,na caixa de dialogo *Add Roles and Features Wizard*, clicar *Yes*, clicar *Install*.
12. Realizar o mesmo processo para a outra máquina nó.

- Configurar os *Virtual Switches* nos nós.

1. Abrir o *Server Manager*, clicar no menu *Tools*, clicar Hyper-V *manager*.
2. No Hyper-V *Manager*, no menu *Action*, clicar em *Virtual Switch Manager*.
3. Por baixo de *Create virtual switch*, selecionar *External*, clicar em *Create Virtual Switch*.
4. Na pagina *New Virtual Switch*, escrever o nome na caixa de texto (tem de ser igual para os dois nós).Em *Connection type*, clicar *External* e selecionar o adaptador de rede, clicar *Apply*.
5. Na caixa de dialogo *Apply Networking Changes*, clicar *Yes*.
6. Repetir o mesmo processo no segundo nó.

- Validar a configuração do *cluster*.

1. No *Server Manager*, clicar no menu *Tools*, clicar *Failover Cluster Manager*.
2. Em *Failover Cluster Manager*, no painel central, por baixo de *Management*, clicar *Validate Configuration*.
3. Na pagina *Before You Begin do Validate a Configuration Wizard*, clicar *Next*.
4. Na pagina *Select Servers or a Cluster*, inserir os nomes dos servidores na caixa de texto, clicar *Add*.
5. Verificar se os servidor se encontram em *Selected server*, clicar *Next*.
6. Na pagina *Testing Options*, selecionar a opção *Run only tests I selected*, clicar *Next*.

7. Na pagina *Test Selection*, clicar *Next*.
8. Na pagina *Confirmation*, clicar *Next*.
9. São realizados os testes de validação do *cluster*.
10. Na pagina *Summary*, clicar *Finish*.

- Criar o *cluster*.

1. Em *Failover Cluster Manager*, no painel central, por baixo de *Management*, clicar *Create Cluster*.
2. Na pagina *Before You Begin* do *Create Cluster Wizard*,clicar *Next*.
3. Na pagina *Select Servers or a Cluster*, inserir os nomes dos servidores na caixa de texto, clicar *Add*.
4. Verificar se os servidor se encontram em *Selected server*, clicar *Next*.
5. Na pagina *Access Point* for *Administering the Cluster*, na caixa de texto *Cluster Name* escrever o nome do *cluster* e na caixa de texto *Address* escrever o endereço de IP do *cluster* (10.0.5.43), clicar *Next*.
6. Na pagina *Confirmation*, selecionar a opção *Add all eligible storage to the cluster*, clicar *Next*.
7. Quando terminar a criação do *cluster*, clicar *Finish*.

- Criar um Cluster Shared Volume.

1. Na consola *Failover Cluster Manager*, clicar nome do *cluster* para expandir as opções.
2. Clicar em *Storage* para expandir mais opções, Clicar em *Disk*.
3. Clicar com o botão direito do rato no disco que diz *Available Store*, clicar na opção *Add to Cluster Shared Volumes*.

- Criar uma máquina virtual de elevada disponibilidade no *cluster*.

1. Em *Failover Cluster Manager*, clicar com o botão direito do rato em *Roles*, selecionar *Virtual Machines...*, clicar *New Virtual Machine...*
2. Selecionar o nó do *cluster*, clicar OK.
3. Em *New Virtual Machine Wizard*, na página *Before You Begin*, clicar *Next*.
4. Na página *Specify Name and Location*, escrever o nome do *cluster* na caixa de texto *Name*, selecionar a opção *Store the virtual machine in a different location*, clicar *Browse*, selecionar a localização. C:\ClusterStorage\Volume1 e clicar *Select Folder*, clicar *Next*.
5. Na pagina *Specify Generation*, selecionar a opção mais indicada para o sistema operativo que vai ser instalado na máquina, clicar *Next*.

6. Na pagina *Assign Memory*, clicar *Next*.
7. Na pagina *Configure Networking*, selecionar o virtual *switch* usado no *cluster*, clicar *Next*.
8. Na pagina *Connect Virtual Hard Disk*, selecionar a opção *Create a virtual hard disk*, escolher o tamanho do disco e clicar *Next*.
9. Na pagina *Installation Options*, selecionar a opção *Install an operating system from a bootable image file*, clicar *Browse*, selecionar o ficheiro iso, clicar *Next*.
10. Na pagina *Completing the New Virtual Machine Wizard*, clicar *Finish*.

- Realizar um teste de *failover* planeado e não planeado.

1. Na consola do *Failover Cluster Manager*, clicar em *Nodes* para expandir.
2. Clicar com o botão direito do rato no nó que queremos realizar o teste, clicar em *More Actions*, clicar *Stop Cluster Service*.
3. Na consola do *Failover Cluster Manager*, clicar em *Roles*, verificar que as máquinas virtuais se moveram para o outro nó.

A.2 Passos da Implementação do Docker *Swarm*

- Criar o *Swarm* no *manager*

1. Abrir um terminal ssh para a máquina que vai executar como nó *manager*. Usando a Docker *Machine*, a conexão via ssh é possivel através do uso do seguinte comando:

   docker-machine ssh 10.0.5.33

2. Executar o seguinte comando para a criação do *Swarm*:

   docker swarm init --advertise-addr 10.0.5.33

3. Verificar o estado do *Swarm* através do comando

   docker info

- Adicionar os dois nós ao *Swarm*

1. Abrir um terminal ssh para a máquina que vai executar como nó *worker* (10.0.5.34 ou 192.168.122.3).
2. Executar o comando produzido pelo *output* do comando docker swarm init para criar e adicionar o nó ao *Swarm*.
3. Repetir os dois passos anteriores para o segundo nó *worker*.
4. Verificar o estado do *Swarm*, através da execução do comando no nó *manager*:

docker node ls

- Implementar um serviço no *Swarm*

  1. Abrir um terminal ssh para a máquina que vai executar como nó *manager*. Usando a Docker *Machine*, a conexão via ssh é possivel através do uso do seguinte comando:

     docker-machine ssh 10.0.5.33

  2. Criar uma rede *overlay* que permite a comunicação entre os *containers* de todos os nós através do comando:

     docker network create -d overlay hello_net

  3. Implementar um serviço através da execução do comando:

     docker service create --name mv --replicas 1 -t --network hello_net --endpoint-mode dnsrr --limit-memory 1024m ubuntu:16.04

     - O comando docker service create cria o serviço.
     - A *flag* name nomeia o serviço.
     - A *flag* replica específica o numero de instâncias a executar.

  4. Verificar se o serviço se encontra a executar, através do uso do comando:

     docker service ls

  5. Para executar um *container* num dos nós é utilizado o seguinte comando:

     docker \textit{container} exec -ti <ID do \textit{container}> /bin/bash

- Inspecionar o serviço a executar no *Swarm*

  1. Inspecionar o estado do serviço através do comando:

     docker service inspect --pretty ms

  2. Verificar quais nós executam o serviço através da execução do comando:

     docker service ps mv

- Escalar o serviço no *Swarm*

  1. Para escalar o serviço por todo o *Swarm* executar o comando para criar um *container* em cada nó:

     docker service scale mv=3

  2. Verificar o numero de execuções do serviço nos nós através da execução do comando:

     docker service ps mv

- Drenar um nó do *cluster*

  1. Executar o comando docker node ls para verificar os nós que se encontram ativos.

2. Drenar um nó através da execução do comando:

   docker node update --availability drain node-1

3. Verificar o estado do nó drenado através do comando:

   docker node inspect --pretty node-1

4. Verificar que os serviços que se encontravam no nó drenado foram retomados por outro nó, através do uso do comando:

   docker service ps mv

5. Executar o comando docker node update –availability active node-1 para retomar o estado do nó drenado para ativo.