Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo Comparativo de Performance entre Máquina Virtual e Container Docker

Maysa Meirelles Casella

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Jan Mendonça Corrêa

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo Comparativo de Performance entre Máquina Virtual e Container Docker

Maysa Meirelles Casella

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Jan Mendonça Corrêa (Orientador)
CIC/UnB

Prof. Dr. Germana Menezes da Nobrega
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
CIC/UnB

Prof. Dr. Jorge Henrique Cabral Fernandes
Coordenador do Curso de Computação — Licenciatura

Brasília, 22 de Dezembro de 2023

Dedicatória

A todos que acreditam que outra realidade é possível.

Agradecimentos

Ao meu orientador, Jan Mendonça Corrêa, meus sinceros agradecimentos pela atenção, apoio e confiança.

Aos Professores Wilson Henrique Veneziano e Germana Menezes da Nobrega pela disponibilidade em ler, avaliar e contribuir com este trabalho.

Aos professores do Departamento de Ciência da Computação da Universidade de Brasília que, cada um ao seu tempo, contribuiu de alguma forma para a minha formação como Cientista da Computação.

Aos meus pais, Lourdinha e Salvatore que, sem medir esforços, sempre foram meu alicerce. Pessoas de caráter que me ensinaram a ser a adulta que sou hoje e que sempre me deram apoio, carinho, amor e compreensão.

Aos meus irmãos, Thiago e Felipe, por todo companheirismo e por sempre torcerem por mim.

Ao meu querido Ítalo, companheiro de todas as horas, pelo apoio e encorajamento ao longo dos anos.

Aos meus familiares e amigos por fazerem parte de minha jornada e compartilharem as alegrias, angústias e inquietações.

A todas as pessoas que de alguma forma contribuíram para a realização deste trabalho.

Resumo

A possibilidade de criar máquinas virtuais e a capacidade de abstração de *hardware* permitiu que múltiplos sistemas operacionais pudessem ser executados utilizando uma mesma plataforma física. Assim surgiu a virtualização, que é o mecanismo por trás das tecnologias Máquina Virtual e Container Docker. Máquinas Virtuais são computadores de *software* emulado com a mesma funcionalidade que os computadores físicos, enquanto containers são ambientes isolados em uma máquina, que compartilham recursos de *hardware*, como CPU, RAM e transmissão de dados, via rede, entre vários *softwares*. Com o avanço dessas tecnologias e a facilidade de acesso à utilização delas, muitas empresas que buscavam agilidade e eficiência em seus sistemas encontraram aí uma possibilidade particularmente útil de ter grandes sistemas de forma mais segura e barata, com grande capacidade de adaptabilidade, confiabilidade e disponibilidade. Entretanto, para a escolha de qual tecnologia melhor se adequa a realidade de cada usuário, é preciso analisar o desempenho de cada uma delas em cenários distintos. Diante dessa necessidade de análise de desempenho, este trabalho traz uma avaliação sobre as vantagens e desvantagens das máquinas virtuais e containers Docker sob uma perspectiva de performance de CPU com análise de oito artigos científicos. Nos testes analisados, os containers Docker saíram-se melhor na maior parte das avaliações, com exceção dos cenários iniciais com menos instâncias, demonstrando assim o poder da ferramenta. Apesar dos resultados obtidos, faz-se necessário um estudo do cenário onde as tecnologias serão aplicadas, de forma a avaliar as vantagens e desvantagens da aplicação de cada uma das técnicas.

Palavras-chave: Virtualização; Máquina Virtual; Container Docker; Avaliação de performance.

Abstract

The possibility of building virtual machines and the ability to abstract hardware allowed multiple operating systems to run using the same physical platform, this brings virtualization, the mechanism behind Virtual Machine and Container Docker technologies. Virtual Machines are software emulated computers with the same functionality as physical computers, while containers are isolated environments on a machine, that share hardware resources such as CPU, RAM and data transmission, via network, between various software. With the advancement of these technologies and the easy access to use them, many companies that seek for agility and efficiency in their systems found there a particularly useful possibility of having large systems in a safer and cheaper environment, with great adaptability, reliability and availability. However, in order to choose which technology best suits the reality of each user, it is necessary to analyze the performance of each one of them in different scenarios. Faced with this need for performance analysis, this work brings an assessment of the advantages and disadvantages of virtual machines and Docker containers from a perspective of CPU, with an analysis of eight scientific articles. In the analyzed tests, Docker containers did better in most evaluations, with the exception of initial less instances scenarios, this demonstrates the power of the tool. Despite the results obtained, it is necessary to study the scenario where the technologies will be applied, in order to assess the advantages and disadvantages of applying each of the techniques.

Keywords: Virtualization; Virtual Machine; Container Docker; Performance evaluation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

A1 Artigo 1.

A2 Artigo 2.

A3 Artigo 3.

A4 Artigo 4.

A5 Artigo 5.

A6 Artigo 6.

A7 Artigo 7.

A8 Artigo 8.

AMD-V AMD-Virtualization.

CaaS Containers como Serviço.

CPU Unidade Central de Processamento.

GNU Licença Pública Geral GNU (*General Public License)* .

GOS Sistema Operacional Visitante ou *Guest Operating System*.

HD Disco Rígido.

HOS Sistema Operacional Hospedeiro ou *Host Operating System*.

HPL Biblioteca Linpack de Alta Performance ou *Hight Performance Linpack*.

IaaS Infraestrutura como Serviço.

IBM International Business Machines Corporation.

IVT Intel Virtualization Technology.

KVM Máquina Virtual baseada em Kernel ou *Kernel Virtual Machine*.

RAM Memória de Acesso Aleatório.

SO Sistema Operacional.

TSL Segurança de Camada de Transporte.

VM Máquina Virtual.

VMM Monitor de Máquina Virtual.

Capítulo 1

Introdução

A possibilidade de criar Máquinas Virtuais e a capacidade de abstração de *hardware* permitiu que múltiplos Sistemas Operacionais pudessem ser executados utilizando uma mesma plataforma física [1]. Assim surgiu a virtualização, que é o mecanismo por trás das tecnologias: Máquina Virtual e Container Docker, conceitos que serão detalhados a seguir.

Máquinas Virtuais são computadores de *software* emulados com a mesma funcionalidade que os computadores físicos. Graças a isso, elas ganharam um importante papel no mercado de Infraestrutura como Serviço (IaaS), ficando ao alcance de muitos usuários que necessitavam de maior mobilidade em seus sistemas. Essa tecnologia auxiliou muitas organizações que possuem sistemas em multiplataformas de clientes e também algumas grandes empresas de servidores de hospedagem, que passaram a poder criar uma máquina de acesso para cada cliente, podendo proporcionar ao usuário final uma experiência similar ao uso de um equipamento físico real [2].

Apesar disso, é comum que empresas definam suas arquiteturas com base em medições de uso do seu sistema considerando picos de utilização que podem raramente acontecer, resultando em aquisições e provisionamento de grandes estruturas de infraestrutura que podem ficar subutilizadas entre os períodos de maior pico de utilização de seus serviços. A fim de prevenir a aquisição de recursos sobressalentes ociosos, várias empresas passaram a utilizar serviços de virtualização e containers, que possibilitam o escalonamento de recursos conforme demanda real do mercado [3].

Posteriormente ao surgimento das Máquinas Virtuais, houve o surgimento de Containers, que é um ambiente isolado em uma máquina. A tecnologia de containers surge como uma alternativa às máquinas virtuais, fazendo com que o uso dos recursos de *hardware*, como CPU, RAM e transmissão de dados via rede pudessem ser compartilhados entre vários *softwares*. O uso dessa tecnologia estende o conceito de Infraestrutura como Serviço (IaaS) para Containers como Serviço (CaaS) [4].

Com o avanço dessas tecnologias e a facilidade de acesso à utilização delas, muitas empresas que buscavam agilidade e eficiência em seus sistemas encontraram aí uma possibilidade particularmente útil de ter grandes sistemas de forma mais segura e barata, com grande capacidade de adaptabilidade, confiabilidade e disponibilidade [5].

Entretanto, para a escolha de qual tecnologia melhor se adequa a realidade de cada usuário, é preciso analisar o desempenho de cada uma delas em cenários distintos.

Diante dessa necessidade de análise de desempenho, justifica-se esta pesquisa, portanto, por sua contribuição à área, uma vez que traz uma reflexão sobre as vantagens e desvantagens das Máquinas Virtuais e Containers Docker sob uma perspectiva de performance de CPU.

1.1 Objetivos

Neste item serão apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é realizar um estudo comparativo de performance entre as tecnologias Máquina Virtual e Container Docker.

1.1.2 Objetivos Específicos

Para atingir o objetivo exposto anteriormente, temos os seguintes objetivos específicos que irão nortear o trabalho:

- Explicar a arquitetura e funcionamento de cada uma das tecnologias;
- Reunir pesquisas anteriores que tenham aplicado estudos comparativos entre as duas tecnologias;
- Avaliar e discutir a diferença de performance entre as tecnologias, considerando a perspectiva de desempenho e consumo na utilização de CPU.

1.2 Metodologia Científica

De acordo com Oliveira [6], metodologia se refere ao "processo que se inicia desde a disposição inicial de se escolher um determinado tema para pesquisar, até a análise dos dados com as recomendações para minimização ou solução do problema pesquisado". Ou seja, podemos entender metodologia como um processo que engloba um conjunto de métodos

e técnicas para que seja possível analisar e conhecer a realidade que nos cerca e, assim, produzir novos conhecimentos.

Sendo assim, a fim de delimitar a metodologia a ser utilizada neste trabalho, utilizaremos o método e o instrumento listado a seguir.

1.2.1 Método

Quanto ao método, esse trabalho tem como base a pesquisa bibliográfica descritiva [7], com levantamento e comparação de dados reais já mensurados, produzidos e publicados anteriormente por outros autores, de forma que esse estudo tenha o embasamento teórico necessário e para que seja baseado em dados já fundamentados no meio acadêmico, trazendo maior diversidade de cenários para as análises de performance de CPU.

1.2.2 Instrumento de Pesquisa

Neste estudo, o instrumento utilizado para a coleta de dados foi a análise de oito artigos científicos publicados em periódicos e repositórios acadêmicos, dentro do período de sete anos, que tenham realizado pesquisas comparativas entre as tecnologias Máquina Virtual e Container Docker sob a ótica de desempenho de CPU.

1.3 Organização do Trabalho

De forma a facilitar a leitura, esse trabalho está dividido da seguinte forma:

- **Capítulo 1: Introdução**

  Capítulo onde são listados os objetivos do trabalho, bem como a metodologia utilizada e os resultados esperados.

- **Capítulo 2: Referencial Teórico**

  Capítulo onde apresentamos um breve histórico a respeito da evolução da tecnologia referente às Máquinas Virtuais e Containers Docker, bem como a fundamentação teórica a respeito da arquitetura e funcionamento delas.

- **Capítulo 3: Estudos Comparativos**

  Capítulo em que trazemos estudos comparativos que relacionam as duas tecnologias e os resultados encontrados em cada um deles.

- **Capítulo 4: Conclusão**

Capítulo em que apontamos as conclusões a que chegamos com esse trabalho, bem como sugestões de trabalhos futuros que podem ser realizados com Máquinas Virtuais e Containers Docker.

Capítulo 2

Referêncial Teórico

Neste capítulo apresentamos um breve histórico a respeito da evolução da tecnologia referente às Máquinas Virtuais e Containers Docker, bem como a fundamentação teórica a respeito delas.

2.1 Evolução da Tecnologia

Os primeiros computadores inventados eram grandes e muito caros [8]. Porém, sua agilidade na execução de algumas funções, como cálculos, fez com que seu uso se tornasse cada vez mais indispensável. Foi criado então, no final dos anos 1960, o *time-sharing*, que permitiu que vários usuários utilizassem simultaneamente, de forma transparente, um mesmo computador [9]. Entretanto, embora este tenha sido um grande marco na computação, ele também gerou um novo problema de como administrar diferentes aplicações suscetíveis a falhas em um único computador.

Como forma de resolver este problema, à época, foi proposto a utilização de vários computadores distintos, o que garantia o isolamento entre as aplicações, além de aumentar significativamente seu desempenho, porém apresentando um altíssimo custo e grande desperdício de recursos, uma vez que os computadores permaneciam ociosos por grandes períodos [10].

A fim de sanar a questão de custo e utilização de recursos, a IBM desenvolveu a primeira Máquina Virtual, que permitia que um mesmo computador fosse subdividido em vários outros, com isolamento entre as aplicações. Este sistema de virtualização foi chamado de CP-67 e disponibilizava ao usuário do *mainframe* IBM 360/67 um sistema virtual [11].

O segundo lançamento da IBM foi o VM/370, um *Virtual Machine Monitor* (VMM) ou Monitor de Máquina Virtual, que contava com arquitetura estendida, ou seja, que permitia o uso de algumas instruções extras visando a virtualização.

Para entendermos melhor a virtualização, é importante definirmos que "virtualização é a simulação de um *hardware/software* que roda sobre outro *software*. Este conceito de ambiente simulado é chamado de máquina virtual (VM – Virtual Machine)" [12].

Segundo Motyczka *et. al.* [13], após essas primeiras tentativas de virtualização, a arquitetura que se estabeleceu como a mais utilizada é a x86 (IA-32), que foi adotada pelos computadores pessoais. Apesar de ser a mais comum, essa arquitetura não foi projetada visando a virtualização, uma vez que algumas instruções não necessitam de modo privilegiado para serem executadas, como é o caso de programas que acessam diretamente o *hardware*, o que pode acabar prejudicando a estabilidade do sistema.

De forma a suprir as dificuldades de virtualização da arquitetura x86, algumas técnicas e projetos foram desenvolvidas, tal como o *VMWare, Xen, Virtual PC, Hyper-V, VirtualBox*, entre outros.

Apesar das primeiras dificuldades encontradas para virtualização, essa é uma tecnologia que até os dias de hoje é amplamente utilizada, uma vez que ela não se atém mais somente ao fato de permitir o uso de um mesmo sistema por vários usuários, mas também por oferecer diversas vantagens, como: segurança, custo, adaptabilidade, confiabilidade e disponibilidade, balanceamento de carga e suporte a aplicações legadas [5].

2.2 Virtualização (Máquina Virtual)

Quando falamos de virtualização, é importante citarmos que VMM também é conhecido como *Hypervisor*, podendo ser de dois tipos [14] :

- Tipo 1 (*bare-metal* ou virtualização total): aqueles que são executados diretamente no *hardware* do hospedeiro; ou
- Tipo 2 (*hosted* ou virtualização parcial): aqueles que são executados como uma camada do Sistema Operacional (SO).

Servidores são frequentemente virtualizados no modo *bare-metal*. Já o *hosted* é mais utilizado em soluções voltadas para uso em computadores pessoais.

A Figura 2.1 exemplifica os dois tipos de *Hypervisor* citados anteriormente.

Ainda sobre *Hypervisor*, ele é o responsável pela virtualização e controle dos recursos compartilhados pelas Máquinas Virtuais, além de ser o responsável pelo escalonamento entre elas, decidindo sobre a sequência de execução das instruções a cada momento. Em outras palavras, ele funciona como uma camada de abstração entre o *hardware* e o SO que está sendo executado, servindo como um isolamento entre as VM's e o *host* [14].

Podemos citar como exemplos de recursos compartilhados pelas VM's:

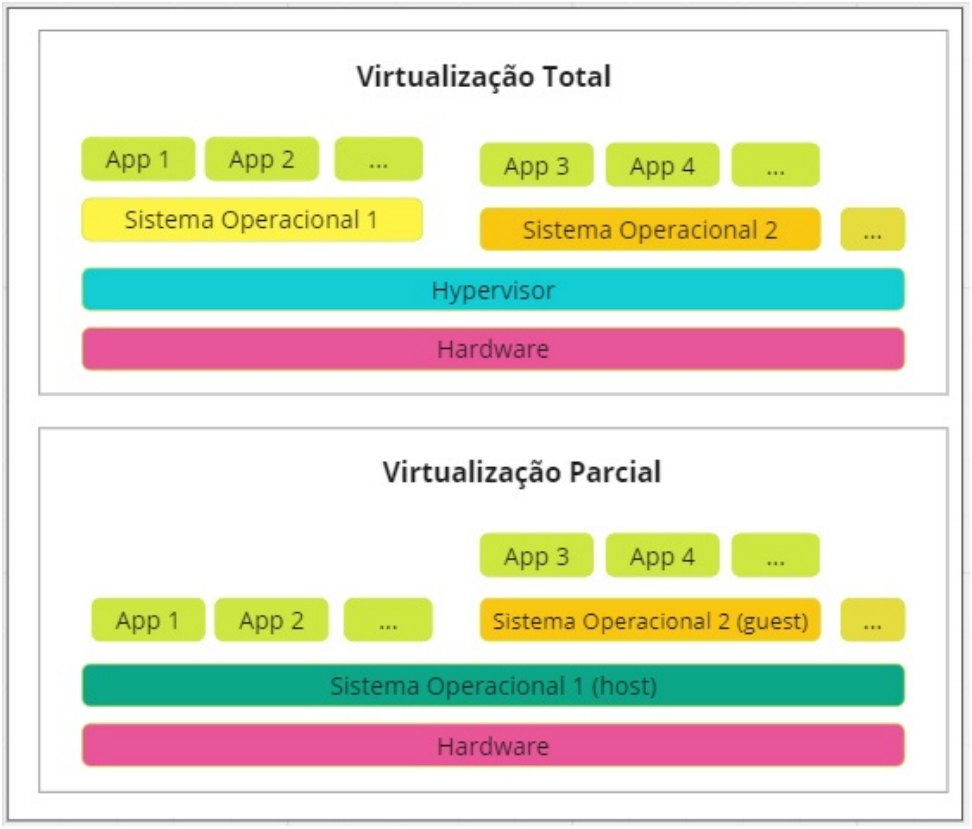

Figura 2.1: Ilustração do *hypervisor* tipo *hosted* e *bare-metal*.
Fonte: Elaborado pela autora, baseado em: [14]

- Memória (RAM);
- Processamento (CPU);
- Dispositivos de entrada e saída (*i/o acess*); e
- Armazenamento (HD).

As Máquinas Virtuais são operacionalizadas pelo *Hypervisor* e, com a instalação de SO's distintos, é possível executar várias VM's em um mesmo Servidor [14].

Segundo [4], as VM's são amplamente utilizadas como um serviço de arquitetura IaaS, ou seja, provém um serviço de infraestrutura como serviço, de forma a facilitar a oferta de recursos físicos para seus clientes. Esse tipo de serviço oferece a customização e ajuste dos componentes de *hardware* a depender dos requisitos das aplicações. Diversas empresas oferecem o serviço de virtualização em nuvens customizadas, como Amazon (*AWS*), Google (*Cloud*) e Microsoft (*Azure*).

Sobre virtualização, é importante definirmos alguns conceitos relacionados a essa tecnologia, como o de instruções privilegiadas e não privilegiadas. De acordo com Takeuti [15], as instruções não privilegiadas "são aquelas que não modificam a alocação ou o estado

de recursos compartilhados por vários processos simultâneos", ou seja, memória principal, processadores e alguns registradores especiais. Já as instruções privilegiadas são aquelas "que podem alterar o estado e alocação desses recursos." Para isso, os computadores podem funcionar de duas formas: uma com acesso administrador e acesso total a todos os recursos (modo supervisor) e outra, sem esse acesso privilegiado e com algumas limitações (modo usuário).

No modo usuário, também chamado de espaço de aplicação, é onde as aplicações são normalmente executadas e onde não é possível executar instruções privilegiadas, que são restritas ao modo supervisor [5].

Já no modo supervisor, é onde é possível executar todas as instruções do conjunto de instruções do processador, tanto as privilegiadas como as não-privilegiadas, uma vez que esse modo tem o controle total da CPU [5].

Apesar do VMM ser executado em modo supervisor, as VM's são executadas em modo usuário, então, quando uma VM tenta executar uma instrução privilegiada, é gerada uma interrupção e o VMM faz a emulação da execução dessa instrução [16].

Vale citarmos que na arquitetura x86 existem quatro níveis de privilégio, os chamados *rings*. Os *rings* recebem números crescentes de 0 a 3 como classificação, sendo 0 o de maior privilégio de execução de instruções e 3 o de menor privilégio. Por isso, os SO's são executados no nível 0 [5].

De acordo com Damasceno [17], um conceito importante que deve ser definido é o de Sistema Operacional Hospedeiro (HOS) e Sistema Operacional Visitante (GOS). O primeiro se refere ao SO que é executado sobre o *hardware*, sendo o SO nativo da máquina onde ocorrerá a virtualização. Já o segundo, trata do SO que será executado sobre o *hardware* virtualizado, ou seja, o que será executado na máquina virtual. Para máquinas hospedeiras só é permitida a execução de um SO hospedeiro por vez, enquanto podem ser executados simultaneamente diversos SO's visitantes.

A seguir, detalharemos as duas formas de implementação dos monitores de máquina virtual (VMM), que são a virtualização total e a virtualização parcial.

2.2.1 Virtualização Total

Com a virtualização total o objetivo é fornecer ao SO visitante uma réplica do *hardware* subjacente, de forma que o sistema operacional visitante seja executado sem modificações sobre o VMM, ou seja, o *hypervisor* simula todo o *hardware* da máquina física, criando uma réplica total do *hardware*, fazendo com que as VMs sejam executadas de forma isolada, agindo como se não estivessem em um ambiente virtual. [14].

A vantagem desse tipo de virtualização é a larga aceitação que existe por parte dos diversos tipos de SO's disponíveis. Porém, ela traz algumas desvantagens: [5]

- O número de dispositivos suportado pelo VMM é extremamente elevado e, de forma a resolver isso, as implementações de virtualização utilizam dispositivos genéricos, que apesar de terem um funcionamento adequado na maior parte dos dispositivos disponíveis no mercado, não garantem que possam atingir sua capacidade total.
- Como o SO visitante não tem conhecimento de estar sendo executado sobre uma VMM, as instruções executadas pelo SO devem primeiro ser testadas pelo VMM para somente depois serem executadas no *hardware*, ou então executadas pelo VMM através de uma simulação para o sistema visitante.
- Por fim, também é necessário contornar alguns problemas gerados pelos SO's, uma vez que eles foram implementados para serem utilizados de forma única em uma máquina física, e não compartilhando recursos com demais SO's. Um exemplo desse problema, é a queda no desempenho devido a paginação da memória virtual. [18]

Embora a virtualização total apresente alguns problemas de desempenho, os processadores *Intel* e *AMD* favorecem esse tipo de virtualização, sendo que a tecnologia de virtualização da *Intel* é a IVT (*Intel Virtualization Technology*), codinome *Vanderpool*, e a da AMD é a AMD-V (*AMD-Virtualization*), codinome Pacífica. As duas foram desenvolvidas de forma independente, o que faz com que possa haver alguns problemas na portabilidade dessas máquinas de uma arquitetura para outra [18].

2.2.2 Virtualização Parcial

A virtualização parcial, ou para-virtualização, costuma ser uma alternativa à virtualização total. No caso de virtualização parcial, o SO é modificado de forma a chamar o VMM sempre que uma instrução a ser executada possa alterar o estado do sistema, ou seja, ser uma instrução privilegiada. Com isso, o desempenho aumenta, já que não há a necessidade de se testar todas as instruções previamente pelo VMM [18].

Outra vantagem é que não há a necessidade de uso de *drivers* genéricos, pois os dispositivos de *hardware* são acessados por *drivers* em sua própria máquina virtual, aumentando assim o seu desempenho.

A decisão da melhor técnica de virtualização é intimamente ligada ao processador da máquina hospedeira, bem como se ela possui ou não em seu processador um conjunto de instruções que suportem a virtualização.

Visto ambas definições, a decisão de usar um *hypervisor bare-metal* ou *hosted* deve ir além da questão de se ter ou não um SO no *host*. A primeira opção, por exemplo, consegue prover um número maior de opções de acesso de entrada e saída (*I/O access*) uma vez que está situado diretamente sobre o *hardware*, disponibilizando mais desempenho para os que optam por essa arquitetura.

Já a segunda opção, consegue prover maior compatibilidade de *hardware*, o que permite executar *softwares* de virtualização com uma gama mais ampla de configurações de *hardware*, diferentemente do modo *bare-metal* [14].

2.2.3 Aplicações

A virtualização é uma técnica que pode ser utilizada em diversos setores, tais como desenvolvimento de *software*, laboratórios de ensino, serviços de infra-estrutura, entre outros.

Em relação às aplicações para desenvolvimento de *software*, a virtualização permite que o desenvolvedor realize testes em suas implementações em ambiente de produção sem efetivamente ter que fazer suas publicações nesse ambiente. Dessa forma, pode-se testar as dependências e verificar comportamentos do sistema em outros ambientes, dentre outras opções.

Já em relação ao uso da virtualização em laboratórios de ensino, a virtualização permite que o estudante trabalhe em uma máquina isolada da máquina física, podendo fazer diversos testes e implementações sem se preocupar com falhas após o uso. Dessa forma, a virtualização reduz custos com manutenção, além de aumentar a flexibilidade e a segurança das máquinas.

Relacionado a infra-estrutura, cabe à virtualização o papel de diminuir o gasto com recursos e também com espaço para alocação de dispositivos, pois várias máquinas de servidores podem ser convertidas em apenas um servidor composto por várias máquinas virtuais com isolamento de serviços entre si.

2.2.4 Vantagens

Apresentamos abaixo as principais vantagens da virtualização, segundo [16]:

- Segurança: Através do uso de máquinas virtuais, pode-se definir qual o melhor ambiente para executar cada serviço, testando-se diversos requisitos de segurança, de ferramentas e de sistemas operacionais;
- Confiança e disponibilidade: como cada VM é isolada dos demais sistemas, vulnerabilidades ou falhas em um dos serviços ou sistemas não prejudicam os demais;
- Custo: Redução do número de servidores físicos através da virtualização dos menores em máquinas mais poderosas, o que reduz os custos com a obtenção e manutenção das máquinas;
- Adaptação: As variações de carga de serviço podem ser tratadas através de ferramentas que realocam os recursos entre as VM's de forma autônoma;

- Balanceamento de carga: todas as VM's são encapsuladas pelo *hypervisor*, de maneira que é fácil trocar a máquina de plataforma a fim de aumentar o seu desempenho;
- Suporte a aplicações legadas: Para facilitar períodos de transição e migração de sistemas em empresas, a antiga aplicação pode ser mantida em uma VM;
- *Hardware* legado: Também é possível emular *hardwares* legados, que estejam suscetíveis a falhas e possam possuir um alto custo de manutenção. Dessa forma, as máquinas que fazem uso desse tipo de funcionalidade podem rodar com *hardwares* mais novos, com menor custo de manutenção e maior confiabilidade.

2.2.5 Desvantagens

Apresentamos a seguir as principais desvantagens da virtualização [18]:

- Segurança: Atualmente as VM's são menos seguras que as máquinas físicas justamente devido ao VMM. Esse ponto é justificado pois se o sistema hospedeiro tiver alguma vulnerabilidade, todos os demais sistemas hospedados nessa máquina também estarão vulneráveis. Como o VMM é uma camada de *software*, ele também está sujeito a vulnerabilidades;
- Gerenciamento: Como os ambientes virtuais precisam ser instanciados, monitorados, configurados e salvos, o gerenciamento dessas ações podem ser complexas. Apesar deste item contar com grandes investimentos da área de virtualização, alguns VMM's podem apresentar falhas, comprometendo a segurança e desempenho das VM's;
- Desempenho: A introdução de uma camada extra de *software* entre o SO e o *hardware* pode aumentar a demanda de processamento em comparação a acessos realizados de forma direta, ou seja, sem o VMM ou *hypervisor*. É também muito complexo mensurar quantas VM's podem ser executadas por processador sem que a qualidade do serviço de virtualização seja impactada.

2.3 Containers Docker

Conforme a Red Hat Inc.[19], a fim de sanar a necessidade de particionar sistemas em vários outros subsistemas, Jacques Gélinas, em 2001, deu início a implementação de ambientes isolados utilizando Linux, através do projeto *VServer*. Pouco tempo depois, mais duas tecnologias foram combinadas a ele para tornar a abordagem de sistemas isolados uma realidade, foram eles: o *cgroup* e o *systemd*.

"Os grupos de controle (*cgroups*) são uma funcionalidade do *kernel* que controla e limita o uso de recursos por um processo ou grupo de processos." Enquanto o *systemd* é um "sistema de inicialização que configura o espaço do usuário e gerencia processos" [19].

O *sysstemd* é usado por *cgroups* para dar mais controle aos processos isolados. Ambas as tecnologias, além de adicionarem um controle geral ao Linux, serviram como abordagem para a separação eficaz de ambientes, trazendo assim os primeiros conceitos de container.

Um container é um conjunto de um ou mais processos organizados isoladamente do sistema, ou seja, são processos em que todos os arquivos necessários para executá-los estão organizados por imagens individuais [19]. Em termos mais específicos, um container é uma instância isolada e executável de uma imagem [20].

A tecnologia de containers é uma alternativa ao uso das VM's, uma vez que ela possibilita o uso dos mesmos recursos de *hardware*, além de permitir o compartilhamento de consumo de CPU e memória RAM entre eles. Dessa forma, os mesmos recursos de *hardware* puderam ser utilizados por várias aplicações de *software*, o que proporcionou a extensão dos serviços de Infraestrutura como Serviço (IaaS) para Containers como Serviço (CaaS). Segundo [4], a tecnologia de containers possibilitou rodar mais aplicações de *software* no mesmo *hardware* do que quando utilizado uma VM.

A partir desse conceito, em 2008, surge o Docker, um *software* de código aberto que utiliza o *kernel* do SO para automatizar a implantação de aplicações através de containers que sejam altamente portáteis e auto suficientes e que independam de *hardware*, linguagem ou *framework* de desenvolvimento, separando assim, as dependências entre aplicativos e infraestrutura [21].

Apesar do Docker se basear em containers, ele não é o mesmo que um container tradicional, uma vez que ele, entre outras habilidades, executa, cria e constrói containers [22].

A Figura 2.2 mostra uma abstração das principais diferenças entre containers tradicionais e Docker.

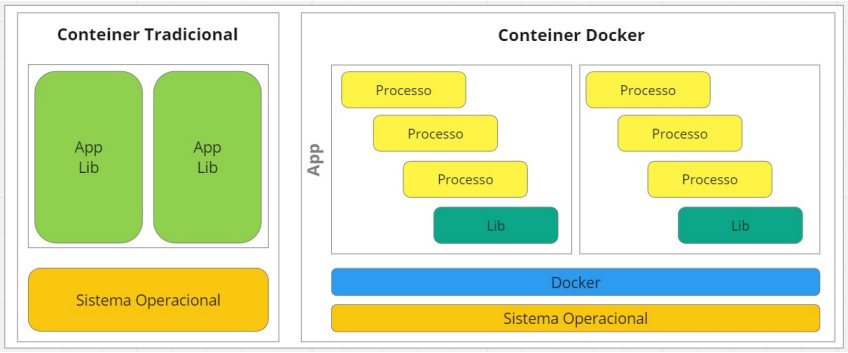

Figura 2.2: Container tradicional x Docker.
Fonte: Elaborado pela autora, baseado em: [19]

Cabe salientarmos que, no ambiente de TI, a palavra “Docker” pode vir acompanhada de três definições: a do projeto, a da ferramenta e a da empresa. O fato da empresa e das tecnologias terem o mesmo nome pode causar alguma confusão, portanto, seguem definições mais detalhadas: [22]

- *Software* "Docker": uma tecnologia para criação e uso de containers a partir do empacotamento de todos os componentes necessários para seu funcionamento.
- Comunidade *Open Source* Docker: trabalha gratuitamente na ferramenta para melhorar a tecnologia para todos os usuários.
- Empresa Docker *Inc*: se baseia no trabalho realizado pela comunidade *Open Source* do Docker, ajudando a torná-lo mais seguro, além de compartilhar os avanços encontrados com a comunidade em geral. Enquanto empresa, ela também oferece aos seus clientes empresariais todo o suporte necessário para a utilização da tecnologia.

Frisamos ainda que, neste trabalho, sempre que utilizarmos o termo “Docker”, estaremos nos referindo ao *Software*, não abrangendo os demais conceitos do termo.

2.3.1 Arquitetura e Componentes

O funcionamento do Docker se dá através da utilização do *kernel* do Linux e funcionalidades do *kernel*, como *cgroups* e *namespaces*, para segregar processos, possibilitando que eles sejam executados de maneira independente, ou seja, eles têm a habilidade de executar diversos processos e aplicações separadamente, utilizando melhor a infraestrutura e, ao mesmo tempo, mantendo a segurança entre sistemas apartados [22].

As ferramentas de container, incluindo o Docker, possuem um modelo de implantação com base em Imagem, o que facilita o compartilhamento das aplicações ou conjunto de serviços, incluindo o compartilhamento de todas as dependências deles em vários ambientes. O Docker também automatiza a implementação das aplicações dentro do ambiente de containers, de forma ágil, com controle de versões e de distribuição [22].

Docker é escrito utilizando *GO*, linguagem de programação criada pela Google, e utiliza funcionalidades nativas do *kernel* do Linux para criação de containers de sistema operacional. Isso faz com que os servidores Docker precisem ser instalados em máquinas virtuais físicas ou sobre algum SO [23].

Conforme já citado, Docker utiliza *namespaces* para oferecer espaços de serviço isolados, ou seja, containers. Sempre que um container é executado, o Docker cria um conjunto de *namespaces* para esse container, o que provém a camada de isolamento necessária para a sua execução. Cada funcionalidade de um container roda em um *namespace* isolado e tem seu acesso limitado àquele *namespace* [20].

A arquitetura do Docker foi desenvolvida com base na arquitetura cliente-servidor (Figura 2.3), onde o cliente pode ser executado de forma local ou remota, no servidor Docker.

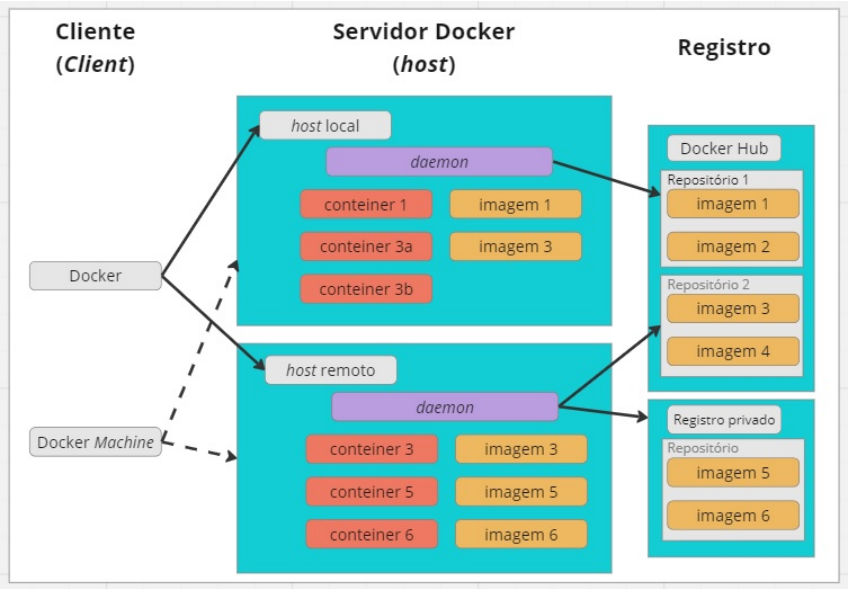

Figura 2.3: Arquitetura Docker.
Fonte: Elaborado pela autora, baseado em: [24]

Para melhor entendimento sobre o funcionamento do Docker, é preciso definirmos os principais termos e componentes dessa tecnologia [20]. São eles:

- Docker *daemon* (*dockerd*): é o centro do Docker. Ele é o responsável por receber as solicitações de API e administrar os objetos do Docker, como imagens, redes, volumes e containers. Um *daemon* também é capaz de se comunicar com outros *daemons* para administrar os serviços Docker. O *daemon* também é o responsável por persistir todos os dados do container em um único diretório, rastreando tudo que envolva o Docker.
- Docker *Client* (*docker*): Interface primária de interação entre usuário e Docker. Sempre que se usa comandos Docker, o *client* é o responsável por enviar isso ao *daemon*, que o carrega adiante. Esse tipo de comando utiliza API Docker e é capaz de se comunicar com vários *daemons* distintos e com o *daemon* do servidor Docker.
- Registros Docker: São repositórios de armazenamento das imagens Docker, podendo ser internos ou externos à organização. O Docker *Hub* é um serviço de repositório SaaS acessível pela internet para qualquer pessoa, onde os fabricantes de *software* disponibilizam suas imagens oficiais Docker para *download*, permitindo que, após

o *download*, as imagens possam ser modificadas e então salvas novamente no repositório. Para ambientes internos, é possível também executar registros em modo privado. Quando se executa os comandos *docker pull* ou *docker run*, a imagem desejada é baixada a partir do registro que estiver configurado. Quando se executa o comando *docker push*, essa imagem sobe para o registro configurado novamente.

- Imagens Docker: Cada imagem Docker é um template em modo leitura criado a partir de uma imagem de SO, suas bibliotecas e binários das aplicações desejadas. Os usuários podem baixar imagens prontas, modificar suas configurações, instalar novas aplicações e salvá-las como novas imagens.
- Containers Docker: Containers encapsulam todos os recursos necessários para a execução das aplicações e são executados a partir do SO de seu *host*. Podem ser iniciados, executados, parados, movidos e apagados. Através do uso de containers as aplicações são executadas em processos e áreas de memória isoladas.
- Docker *Machine*: Ferramenta por linha de comando para auxiliar na instalação do Docker na plataforma adequada [25].
- Docker *files*: Arquivos com instruções de como gerar uma nova imagem Docker. Cada uma dessas instruções descrevem um conjunto de passos para a criação da nova imagem, o que inclui a execução de comandos, a inclusão de novos arquivos ou diretórios, a criação de variáveis de ambiente e a definição de quais processos devem ser executados quando um novo container for iniciado por meio da nova imagem criada [26].

Quanto a sua arquitetura, Docker utiliza a arquitetura cliente-servidor, sendo essa comunicação realizada com API REST, através do uso de *sockets* ou interfaces de rede. O Docker *client* aciona o *daemon*, que constrói, executa e distribui os containers Docker. O Docker client pode ser executado no mesmo sistema do *daemon* ou pode estar conectado a um *daemon* remoto. [20]

As alterações feitas em cada diretório dentro de um container são isoladas e não podem ser vistas fora dele. Para isolar os processos entre os containers, o Docker utiliza mecanismos de *namespaces*, e para monitorar ou limitar o uso de recursos de *hardware*, é feito uso dos *cgroups*. O sistema de arquivos utilizado é o UFS, que faz a implementação do sistema de arquivos em múltiplas camadas. Toda a comunicação entre o servidor Docker e os repositórios de registro devem ser feitas por meio de protocolo TSL (https) e uso de chaves de criptografia, de forma a garantir a integridade e confidencialidade dos dados [5].

2.3.2 Execução de Containers

Para que um novo container seja criado, é necessário que seja enviado um comando *docker run* para o *daemon*, informando o nome da imagem e o comando que deve ser executado no container após o término de sua inicialização. Recebendo esse comando, o Docker executa os seguintes passos: [20]

- *Download* da imagem solicitada no servidor, sendo que, caso ela ainda não exista, é feito seu *download* a partir do repositório configurado.
- Criação do novo container.
- Alocação de um arquivo de sistema tipo "leitura-escrita" e criação da camada de leitura e escrita para uso das aplicações. Isso permite que o container crie ou modifique diretórios ou arquivos em seu arquivo local.
- Criação de uma interface de rede, permitindo assim a conexão do container com a rede padrão. Nesse momento é atribuído também um endereço IP ao container.
- Execução de processos ou aplicações definidas pelo usuário. Captura e criação de arquivos de *log* para as aplicações, permitindo que o usuário possa verificar como as aplicações estão rodando.

2.3.3 Modelos de Operação

Os containers Docker podem ser executados tanto em máquinas físicas quanto em máquinas virtuais e o grupo delas é chamado de *Cluster*. [27]

Nas máquinas físicas, conforme monstrado na Figura 2.4, os containers adicionam uma camada de isolamento entre o *host* e as aplicações, de forma a aumentar a segurança. Como os containers normalmente são menores que uma VM, é possível executar uma grande quantidade deles em um mesmo servidor físico.

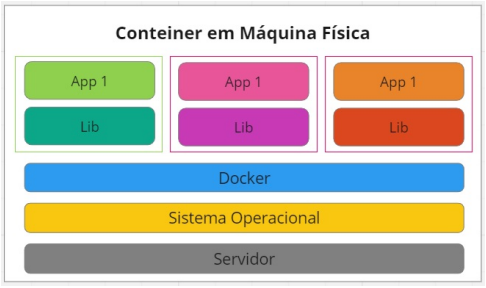

Figura 2.4: Containers Docker em Máquina Física.
Fonte: Elaborado pela autora, baseado em: [28]

Quanto ao uso de containers em Máquinas Virtuais, o isolamento da aplicação é feito no mesmo nível da VM pelo *Hypervisor*, conforme mostrado na Figura 2.5. Dessa forma, a execução do container Docker sobre uma máquina virtual aumenta ainda mais a sua segurança pois provê duas camadas de isolamento entre as aplicações. Nesse contexto, o Docker fica responsável pelo isolamento dos processos dentro da máquina virtual e pelo controle dos recursos de *hardware* das aplicações; enquanto que o *Hypervisor* fica responsável pelo isolamento através da VM [28].

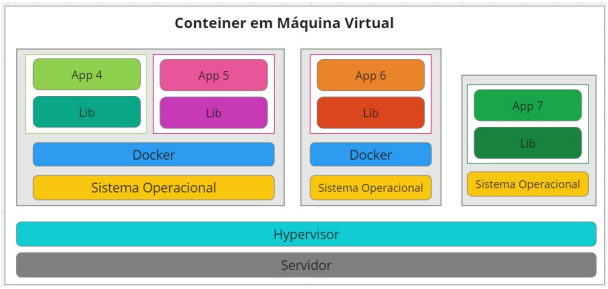

Figura 2.5: Containers Docker em Máquina Virtual.
Fonte: Elaborado pela autora, baseado em: [28]

2.3.4 Vantagens

Apresentamos abaixo as principais vantagens do uso de Docker: [22]

- Modularidade: habilidade de desativar um pedaço da aplicação para reparo ou atualização, sem a necessidade de interrompê-la totalmente;
- Camadas e controle de versão de imagens: cada arquivo de imagem do Docker é composto por uma série de camadas combinadas. Cada vez que uma imagem é alterada, uma nova camada é criada. Durante a execução do Docker, ele reutiliza essas camadas sempre que precisa criar novos containers, o que trás agilidade para o processo;
- Reversão: peculiaridade da criação de camadas. Ela dá a possibilidade de reversão de uma imagem que teve uma iteração mal sucedida em seu estado anterior;
- Implantação rápida: a execução, provisionamento e disponibilização de um novo *hardware* pode exigir dias, porém, com containers Docker, todo esse processo leva apenas alguns segundos;

2.3.5 Desvantagens

Apresentamos abaixo as principais desvantagens do uso de Docker: [22]

- Gerenciamento: Apesar de possuir a habilidade de gerenciar containers, utilizando apenas o Docker pode se tornar difícil gerenciar e orquestrar sistemas com muitos containers. Eventualmente poderá ser necessário que o usuário precise organizá-los manualmente, fornecendo serviços de rede, segurança, telemetria, entre outros, para todos eles. Atualmente já existem sistemas paralelos que auxiliam nessa orquestração de containers, como é o caso do *Kubernetes*, um sistema *open source* que automatiza a implantação, o dimensionamento e a gestão de aplicações em containers, porém, ele não será objeto de estudo deste trabalho.
- Segurança: o uso do *daemon* também pode representar vulnerabilidades à segurança, uma vez que, para a sua execução, ele requer privilégios de administrador, deixando brechas de segurança quanto a modificação de arquivos restritos. Nesse contexto, é necessário escolher com cautela os usuários que terão acesso a esse processo e o local onde ele será armazenado. Salienta-se que um *daemon* local terá menos chances de sofrer um ataque do que um *daemon* em um local mais público, como o caso de um servidor *web*.

2.4 Comparação Máquinas Virtuais e Containers

Conforme previamente citado, Máquinas Virtuais e Containers possuem muitos pontos em comum, como ambientes isolados e benefícios de alocações. Porém, são bem diferentes, uma vez que containers virtualizam o SO, e a VM virtualiza o *hardware*.

A Tabela 2.1 apresenta as principais semelhanças e diferenças entre as duas tecnologias:

Tabela 2.1: Semelhanças e Diferenças entre VM e Container.

Recurso	**Máquina Virtual**	**Container**
Isolamento	Isolamento completo entre a VM, o SO *host* e as demais VM's. Esse recurso é útil para sistemas onde o limite de segurança é um item crítico.	Isolamento leve entre o container, o *host* e os demais containers. Não fornece um limite de segurança tão grande quanto o de uma VM. Quando é necessário um nível de segurança maior, recomenda-se isolar o container dentro de uma VM.
Sistema Operacional	Executa um SO completo, incluindo-se o *kernel*, o que exige mais recursos do sistema (CPU, memória e armazenamento).	Executa apenas a parte do modo usuário de um SO e pode ser adaptado para conter apenas os serviços necessários para seu aplicativo, consumindo menos recursos do sistema.
Compatibilidade do convidado	Executa qualquer SO dentro da VM.	Só pode ser executado na mesma versão do SO *host*. Quando é necessário executar outra versão, pode-se isolar o container dentro de uma VM.
Implantação	Necessário utilização de uma VMM para gerenciamento. Processo mais lento.	Necessário utilização do Docker, via linha de comando. Para orquestrar vários containers juntos, recomenda-se utilização de um serviço extra, como o orquestrador *Kubernetes*.

Continua na próxima página

Tabela 2.1 – Continuação da tabela

Recurso	**Máquina Virtual**	**Container**
Atualizações e *upgrades* do Sistema Operacional	Necessário baixar e instalar as atualizações individualmente em cada uma das VM's. Processo mais lento.	Necessário atualizar o arquivo que gera o container para que ele aponte para a nova versão do SO. Necessário também recompilar e reimplantar o container. Processo mais ágil.
Balanceamento de carga	O balanceamento de carga da VM move as VM's em execução para outro servidor em *cluster*. Para que o serviço não fique fora do ar, pode ser necessária a utilização de um servidor redundante.	Os próprios containers não se movem. Ao invés disso, é necessário apenas iniciar ou parar outros containers. Essa ação pode ser feita automaticamente quando se faz o uso de um orquestrador.
Tolerância a falhas	As VMs podem se mover para outro servidor redundante (quando há), com o SO da VM reiniciando no novo servidor.	Se um nó de *cluster* falhar, todos os containers em execução nele deverão ser criados em outro nó do *cluster*. Essa ação pode ser feita automaticamente quando se faz o uso de um orquestrador.
Rede	Usa adaptadores de rede virtuais.	Usa uma exibição isolada de um adaptador de rede virtual. O *firewall* do *host* é compartilhado com os containers, consumindo assim menos recursos.

2.5 Virtualização e Computação em Nuvem

Apesar de serem termos parecidos, a virtualização e a computação em nuvem não são a mesma coisa, uma vez que a virtualização é a base para a computação em nuvem.

A diferença principal entre elas é apenas o local em que a virtualização de servidores acontecerá e como será gerida. Servidores virtuais podem ser criados internamente, dentro de uma organização, sem a necessidade de terceiros. Enquanto isso, a computação em nuvem exige que haja a contratação de uma empresa especializada para prover e gerir o Servidor *Cloud*. O acesso a esse servidor ocorre de maneira remota, via internet.

Pode-se dizer então que a computação em nuvem depende da virtualização de servidores, mas não o contrário. A virtualização é a base das duas soluções e pode ser feita em servidores físicos ou remotos, como o caso da nuvem [29].

Apresentados os principais conceitos sobre Máquinas Virtuais e Containers Docker, vamos no próximo capítulo analisar pesquisas comparativas realizadas entre as duas tecnologias.

Capítulo 3

Estudos Comparativos

Este capítulo apresenta pesquisas comparativas entre as tecnologias Máquina Virtual e Container Docker, na perspectiva de performance de CPU.

Para todos os cenários abaixo descritos, delimitaremos o termo "instância"como sendo o número de Máquinas Virtuais ou de Containers Docker utilizados para cada um dos testes realizados.

3.1 Artigo 1 - "*Performance evaluation between Docker Container and Virtual Machines in Cloud Computing Architectures*", 2017.

No Artigo 1 (A1) [4], os autores realizam as comparações utilizando máquinas de servidor *HP Proliant* com um único processador *Intel Xeon* E5-2407 @ 2.20 GHz com 16 GB RAM, 1 TB de HD. Como sistema operacional foi utilizado o Linux Debian 8 na versão 64-bit.

Os autores realizam também testes de performances de disco, memória RAM e de rede, porém esses critérios não serão abordados no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o *VirtualBox* versão 5.1.6. como a ferramenta de VM, devido a sua licença GNU que permite distribuição gratuita e por possuir uma interface gráfica amigável. A versão do Docker utilizada foi a 1.12 em um ambiente de containers.

Os testes de performance de CPU são apresentados na Figura 3.1, e foram realizados em um ambiente configurado com base no *Top500 HPL Calculator*, que é uma biblioteca utilizada para classificar supercomputadores mundialmente, sendo baseada em cálculos de álgebra linear com o objetivo de otimizar a performance através de cálculos que maximizem os recursos disponíveis entre computadores com memória distribuída.

Para os cálculos de HPL, são utilizadas variáveis que se baseiam no total de memória disponível, no número de nós utilizados, de processadores e de núcleos disponíveis, além da porcentagem de memória utilizada. Para esse artigo foi considerado o uso de 60% de memória RAM em 4, 6 e 16 instâncias, onde o Docker obteve um melhor resultado nos 3 cenários, com uma diferença aproximada de um *Gigaflop* melhor que a VM, ou seja, em um mesmo intervalo de tempo, houve uma diferença aproximada de execução de 1 bilhão de operações a mais por segundo no Docker que na VM, o que representa um percentual de desempenho aproximado de 4% de diferença entre os cenários.

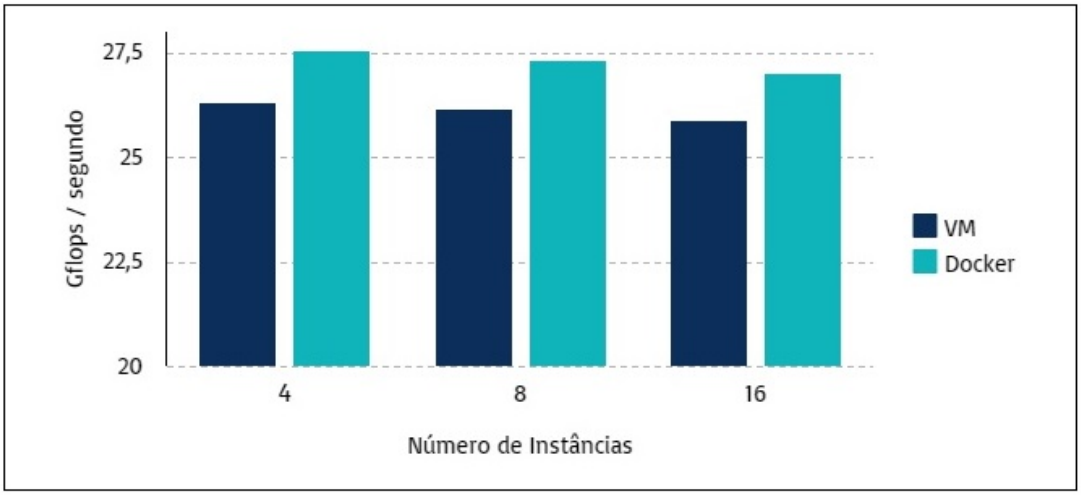

Figura 3.1: Resultado dos testes de desempenho de CPU para 4, 8 e 16 instâncias.
Fonte: Elaborado pela autora, baseado em dados extraídos de: [4]

Como resultado final, os autores concluem que, para a pesquisa em questão, o Docker possui mais vantagens de uso que o VMM no que tange a utilização de CPU, uma vez que é mais eficiente que a VM. Os autores relacionam essa vantagem com o fato de o Docker contar com maior disponibilidade de recursos de *hardware* em comparação a VM.

3.2 Artigo 2 - "*Comparison between common virtualization solutions: Vmware workstation, Hyper-v and Docker*", 2021.

No Artigo 2 (A2) [30], os autores realizam as comparações utilizando máquinas de servidor com processador *Intel* i5-9300H @ 2.40 GHz; 8GB RAM. Como sistema operacional foi utilizado o Linux Ubuntu 20.04.

Os autores realizam também testes de performances de disco e memória RAM, porém esses critérios não serão abordados no escopo deste trabalho, uma vez que há apenas

o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores utilizaram mais de um VMM nesse estudo, sendo assim, para este trabalho, utilizaremos apenas os resultados encontrados utilizando o *VMWare* como a ferramenta de VMM, e o Docker para o ambiente de containers.

Os testes de performance de CPU são apresentados na Figura 3.2, e foram realizados com a CPU buscando por números primos. Para essa análise foi utilizada a ferramenta *Sysbench*, onde a CPU deveria procurar números primos entre números inteiros até que o total de números primos fosse igual a um valor pré-definido, no caso, uma lista de 0 a 20.000.

Nesse teste, devido ao uso direto dos recursos de *hardware* da máquina hospedeira e sua menor quantidade de camadas de abstração, o Docker obteve um melhor resultado de performance de desempenho com uma diferença de 0,13% entre os cenários analisados.

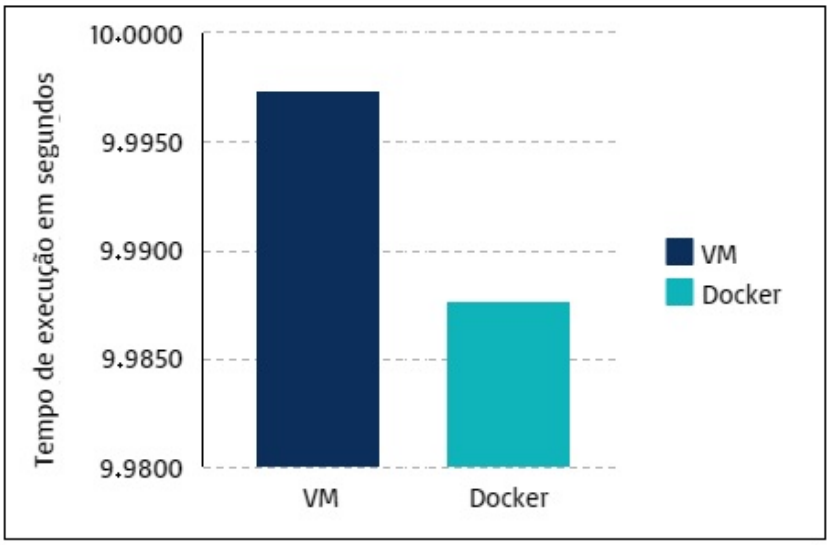

Figura 3.2: Resultado dos testes de performance de CPU para 20.000 números.
Fonte: Elaborado pela autora, baseado em dados extraídos de: [30]

Como resultado final, os autores concluem que, para a pesquisa em questão, o Docker teve o melhor desempenho devido a sua arquitetura leve e flexível, porém, enquanto tecnologia de containers, o Docker possui a desvantagem de não poder funcionar como um computador, sendo apenas um ambiente composto de um espaço de execução e alguns programas. Os autores frisam ainda que diferentes tecnologias não competem entre si, uma vez que a melhor escolha da tecnologia a ser utilizada dependerá das necessidades específicas de cada usuário.

3.3 Artigo 3 - "*Performance evaluation of docker container and virtual machine*", 2020.

No Artigo 3 (A3) [31], os autores realizam as comparações utilizando máquinas de servidor HP com dois processadores *Intel Xeon* E5-2620 v3 @ 2.40 GHz com 12 núcleos e 64 GB RAM. Como sistema operacional foi utilizado o Linux Ubuntu 16.04 de 64 bits e *kernel* 3.10.0.

Os autores realizam também testes de disco, memória RAM, estresse, velocidade de operação e análise T, porém esses critérios não serão abordados no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o Docker para o ambiente de containers e, como ferramenta de VM, o KVM, *Kernel Virtual Machine*, que é uma tecnologia de virtualização nativa do Linux, responsável por criar Máquinas Virtuais baseadas em *Kernel*.

Os testes de performance de CPU são apresentados na Figura 3.3, e foram realizados com a CPU buscando por números primos. Para essa análise foi utilizada a ferramenta *Sysbench*, onde a CPU deveria procurar por números primos em uma lista de 0 a 50.000 por até 60 segundos. Nesse teste, o Docker obteve um melhor resultado de performance de desempenho com uma diferença de aproximadamente 47% entre os cenários analisados. Essa vantagem é atribuída à presença do *Hypervisor* na VM.

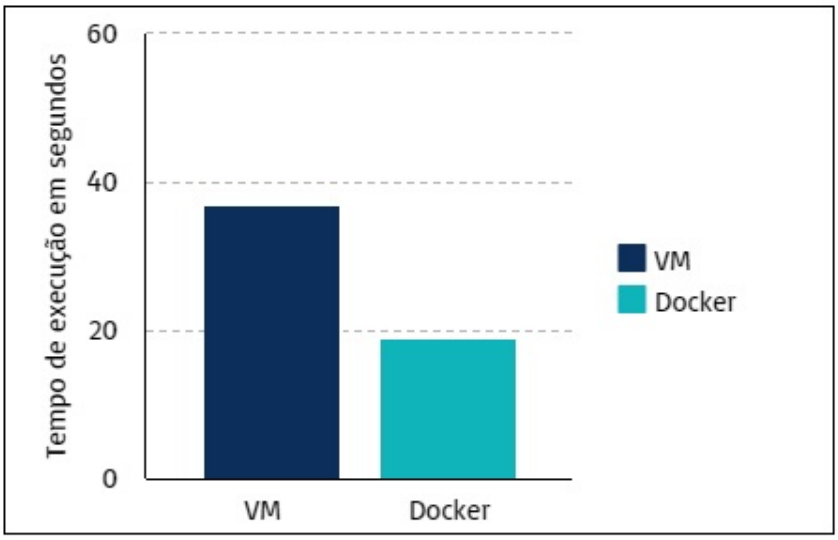

Figura 3.3: Resultado dos testes de performance de CPU para 50.000 números
Fonte: Elaborado pela autora, baseado em dados extraídos de: [31]

Como resultado final, os autores concluíram que, para a pesquisa em questão, o Docker obteve melhor desempenho em todos os testes realizados devido a presença da camada

de virtualização existente na VM, o que a torna menos eficiente que containers quando falamos em aspectos de performance de CPU.

3.4 Artigo 4 - "*Performance comparison between Virtual Machines and Docker Containers*", 2018.

No Artigo 4 (A4) [1],os autores realizam as comparações de performance utilizando como máquina um servidor *Intel* i3 com 4 GB RAM. Como sistema operacional foi utilizado o Linux Ubuntu 16.04 LTS.

Os autores realizam também testes de memória RAM, porém esse critério não será abordado no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o *VMWare* como a ferramenta de VM e Docker para o ambiente de containers.

Os testes de performance de CPU são apresentados na Figura 3.4, e foram realizados com a CPU buscando por números primos. Para essa análise foi utilizada a ferramenta *Sysbench*, onde a CPU deveria procurar números primos em um cenário de testes com 30 amostras e 5 níveis de carga de trabalho, detalhados a seguir:

- I - intervalo numérico de 1 até 20.000;
- II - intervalo numérico de 1 até 30.000;
- III - intervalo numérico de 1 até 40.000;
- IV - intervalo numérico de 1 até 50.000;
- V - intervalo numérico de 1 até 60.000.

Para os cinco cenários, o Docker obteve um melhor resultado de performance, porém, com apenas alguns décimos de segundos a menos, apontando uma diferença na performance de desempenho de aproximadamente 0,7% entre os cenários analisados. Apesar de pequena, essa diferença, em grande escala, pode representar por uma grande diferença de performance entre os dois cenários.

Como resultado final, os autores concluem que, para a pesquisa em questão, o Docker possui mais vantagens de uso que o VMM quando considerada a análise de processamento, uma vez que possui maior eficiência em vários pontos, por haver maior disponibilidade de recursos de *hardware* para os containers.

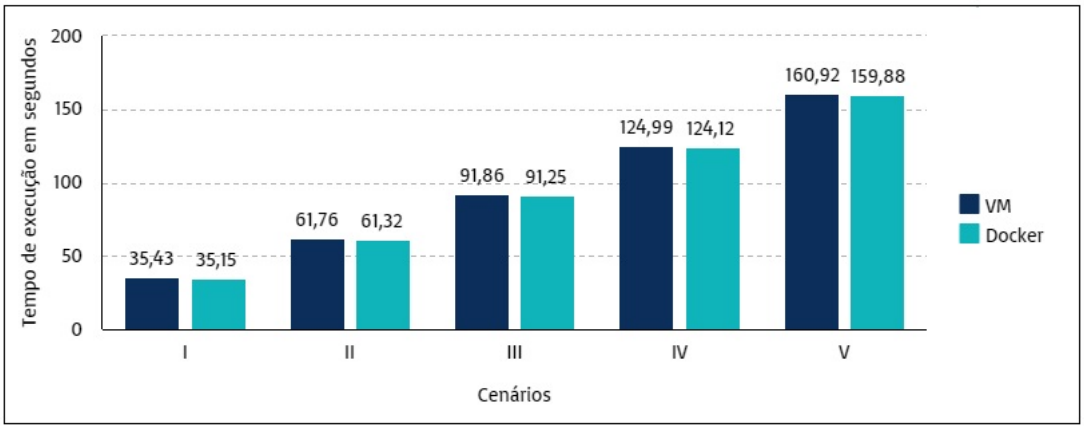

Figura 3.4: Resultado dos testes de performance de CPU para os cenários de teste com números primos.

Fonte: Elaborado pela autora, baseado em dados extraídos de: [1]

3.5 Artigo 5 - "*Performance comparison analysis of Linux Container and Virtual Machine for building cloud*", 2014.

No Artigo 5 (A5) [32], os autores realizam as comparações utilizando dois servidores de computação em nuvem, sendo um para Máquina Virtual, em uma plataforma Openstack, e o outro para Docker. Como sistema operacional foi utilizado o Linux Ubuntu 14.04.

Os autores realizam também testes de velocidade de inicialização, porém esse critério não será abordado no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o KVM como a ferramenta de VM e o *Docker Cloud* para o ambiente de containers. A escolha do primeiro teve relação direta com a facilidade de aplicação de políticas de sistema, de rede, de usuário, de segurança e por suportar a utilização de diversos SO's. Porém, cabe ressaltar que essa ferramenta pode causar maior desperdício dos recursos computacionais, uma vez que realiza a distribuição de imagens até mesmo dentro dos mesmos SO's utilizados.

Já para o segundo cenário, a escolha do *Docker Cloud* se deu por ser uma ferramenta de fácil implementação e distribuição, não havendo a necessidade de utilização de um SO convidado. Como desvantagem, ele só é capaz de virtualizar e distribuir sistemas operacionais Linux, restringindo assim sua utilização.

Os testes de performance de CPU são apresentados na Figura 3.5, e foram realizados testes para medir o tempo médio de processamento para calcular os fatoriais 100.000! e 200.000! utilizando um código em linguagem *python*. Os resultados foram medidos

após a repetição por 100 vezes dos cálculos, onde o Docker obteve um melhor resultado nos 2 cenários, com uma diferença de aproximadamente 5% entre os cenários para o cálculo fatorial de 100.000! e de aproximadamente 9% para o cálculo fatorial de 200.000!. Resultando em uma performance média de 7% a mais no tempo de consumo de CPU para VM's do que pelo Docker.

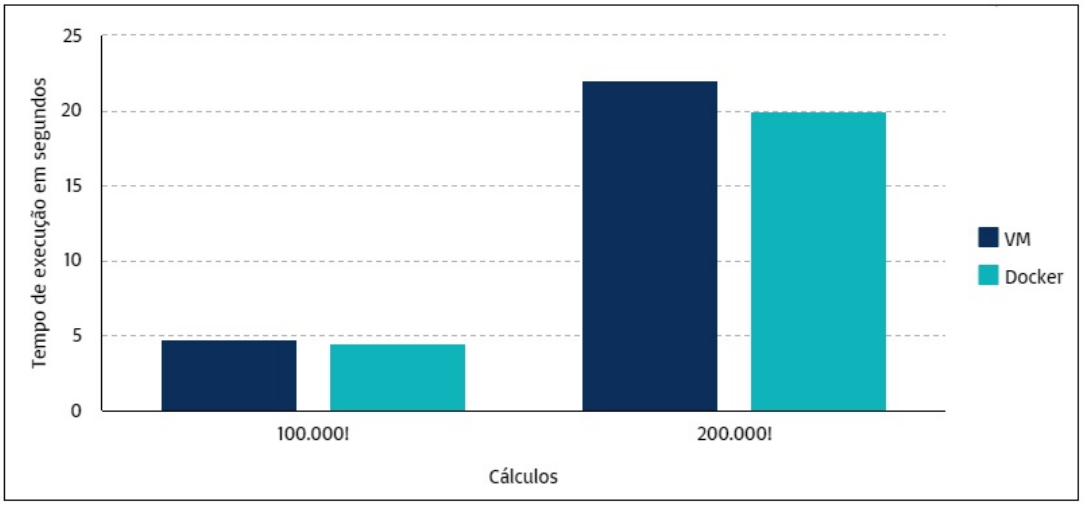

Figura 3.5: Resultado dos testes de performance de CPU para os cálculos fatoriais de 100.000! e 200.000!.
Fonte: Elaborado pela autora, baseado em dados extraídos de: [32]

Como resultado final, os autores concluem que, para a pesquisa em questão, o Docker possui mais vantagens de uso que o VMM quando considerada a análise de processamento, uma vez que, por não possuir um SO convidado, há uma economia na utilização dos recursos de CPU, fazendo com que o tempo de inicialização, geração e distribuição de imagens sejam menores quando comparados a máquinas virtuais.

3.6 Artigo 6 - "*Comparison between Openstack virtual machines and Docker containers in regards to performance*", 2020.

No Artigo 6 (A6) [33], o autor realiza as comparações de performance com foco em consumo de CPU utilizando um processador *Intel Core* i7-6700K CPU @ 4.00GHz e 16 GB RAM.

Os autores realizam também testes de performances de disco, tempo de inicialização e taxa de transferência, porém esses critérios não serão abordados no escopo deste trabalho,

uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o *VirtualBox* em um *Ubuntu Server* 18.04 como a ferramenta de VM, enquanto para o Docker foi utilizada a versão 19.03.11 implantado através do *OpenStack* para o ambiente de containers.

Os testes de performance de CPU são apresentados na Figura 3.6, e foram realizados com base em três cenários: busca dos usuários constantes em uma tabela *MySQL*, adição de uma linha nesta tabela e o carregamento e armazenamento de um arquivo na base. Para todos os testes foram considerados cenários com 2, 3 e 4 instâncias sendo executados em *looping* por 45 segundos.

Nas análises realizadas neste artigo, o Docker obteve um melhor resultado para todos os cenários, sendo que a utilização dos recursos de CPU se tornam ainda mais eficientes quanto mais expandidas as instâncias foram, uma vez que, para duas instâncias, houve um consumo de cerca de 21% a mais de CPU pela VM quando comparada ao Docker; enquanto para quatro instâncias houve um consumo de cerca de 39% a mais de CPU pela VM:

Para a obtenção dos percentuais anteriormente apresentados, foi realizada primeiramente a média de consumo entre as três ações acima previstas, para cada um dos cenários de quantidade de instâncias. Ou seja, foi calculado o percentual de consumo médio para a ação de busca dos usuários na tabela *MySQL* por 4, 6 e 8 instâncias; o percentual de consumo para adição de linhas na tabela em cada cenário de instâncias, e assim sucessivamente. Após a obtenção desses números e utilizando-os como base, foi realizado novo cálculo considerando a média total para VM e para Docker, assim encontrando um resultado de consumo médio de CPU 30% maior pelas VM's em relação ao Docker.

Como resultado final, os autores concluem que, para a pesquisa em questão, o Docker possui um desempenho melhor em relação ao uso da VMM, uma vez que, à medida que os cenários foram sendo escalonados, os percentuais de utilização de CPU entre as duas ferramentas ficaram cada vez mais afastados entre si.

3.7 Artigo 7 - "*An updated performance comparison of virtual machines and Linux containers*", 2015.

No Artigo 7 (A7) [34], os autores realizam as comparações de performance utilizando como máquina um servidor *IBM System x3650 M4 Server* com dois processadores *Intel*

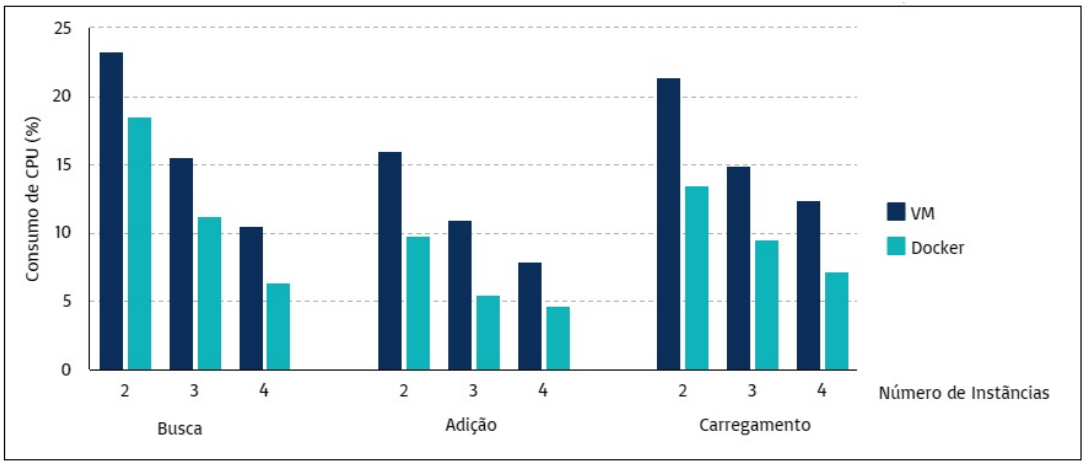

Figura 3.6: Resultado dos testes de consumo de CPU para 2, 3 e 4 instâncias para as operações de Busca, Adição e Carregamento.
Fonte: Elaborado pela autora, baseado em dados extraídos de: [33]

Sandy Bridge-EP Xeon @ 2.4-3.0 GHz com 16 núcleos e 256GB de RAM. Como sistema operacional foi utilizado o Linux Ubuntu 13.10 na versão 64 bits.

Os autores escolheram o KVM como a ferramenta de VM. A versão do Docker utilizada foi a 1.0 em um ambiente de containers.

Os autores realizam também testes de sobrecarga de memória, disco e rede, porém esses critérios não serão abordados no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os testes de performance de CPU são apresentados na Figura 3.7, e foram realizados em um ambiente configurado para medir a execução do *MySQL* nos dois cenários com o aumento gradual de usuários simulados, avaliando o consumo de CPU por segundo conforme as chamadas ao *MySQL* iam sendo executadas. À medida que o número de chamadas aumentou, foi possível observar um aumento de consumo da CPU até determinado ponto, quando há uma estabilização do consumo. Durante os testes foi possível identificar um consumo, em termos totais, 38% maior pelo KVM em relação ao Docker.

Como resultado final, os autores concluem que, para a pesquisa em questão e em relação a performance de CPU, o Docker possui mais desempenho que o KVM para todos os cenários apresentados no estudo.

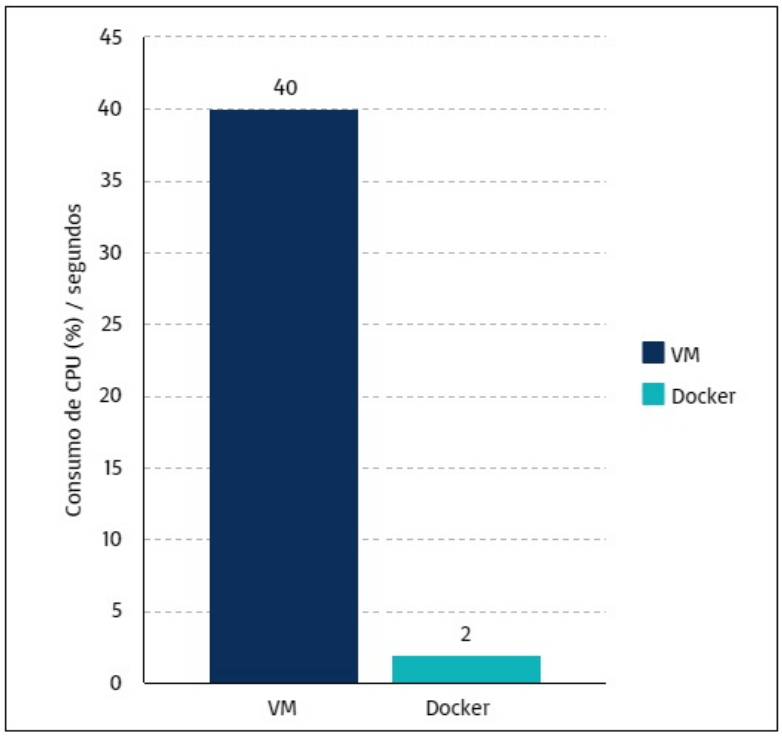

Figura 3.7: Resultado dos testes de performance de CPU para transferências simultâneas. Fonte: Elaborado pela autora, baseado em dados extraídos de: [34]

3.8 Artigo 8 - "*Docker versus KVM*: Uma análise de desempenho de disco para pequenos arquivos", 2016.

No Artigo 8 (A8) [35], o autor realizou as comparações de performance com foco em consumo de CPU utilizando como máquina um computador *Lenovo ThinkCentre* M91p 4518 com processador Core i5 2400 @ 3.1 GHz e 8 GB RAM. Como sistema operacional foi utilizado o Linux Ubuntu 14.04.

Os autores realizam também testes de performance de disco, porém esse critério não será abordado no escopo deste trabalho, uma vez que há apenas o critério de performance de CPU em comum entre todos os estudos utilizados neste trabalho.

Os autores escolheram o KVM como a ferramenta de VM e o Docker para o ambiente de containers.

Para as análises foram executados 100 testes para três diferentes cenários, com 4, 6 e 8 instâncias para cada uma das duas tecnologias. Dentre os cenários testados, foi avaliado o consumo percentual de CPU, por minuto, conforme as ações de escrita, reescrita, leitura, releitura, escrita e leitura randômicas de arquivos eram executados. Os valores apresen-

tados pelo autor foram encontrados considerando-se a quantidade de arquivos utilizados e o número de containers e VM's em cada um dos cenários testados.

Os testes de performance de CPU são apresentados na Figura 3.8 e, para o cenário de 4 instâncias, houve um consumo médio 31% maior pelo Docker do que pelo VM. Já para 6 instâncias houve uma queda considerável no consumo de CPU pelo Docker, deixando uma lacuna, em termos totais, de apenas 2% entre ele o KVM. Em relação ao teste com 8 instâncias, houve um consumo médio maior de CPU pelo KVM do que o Docker, se invertendo o consumo de recursos, sendo, em termos totais, 9% maior pelo KVM que pelo Docker.

Para a obtenção dos percentuais anteriormente apresentados, foi realizada primeiramente a média de consumo entre as seis ações acima previstas, para cada um dos cenários de quantidade de instâncias. Ou seja, foi calculado o percentual de consumo médio para a ação de escrita em 4, 6 e 8 instâncias; o percentual de consumo para ação de reescrita nas instâncias, para a ação de leitura, e assim sucessivamente. Após a obtenção desses números e utilizando-os como base, foi realizado novo cálculo considerando a média total para VM e para Docker, assim encontrando um resultado de consumo de CPU 21% maior pelas VM's em relação ao Docker.

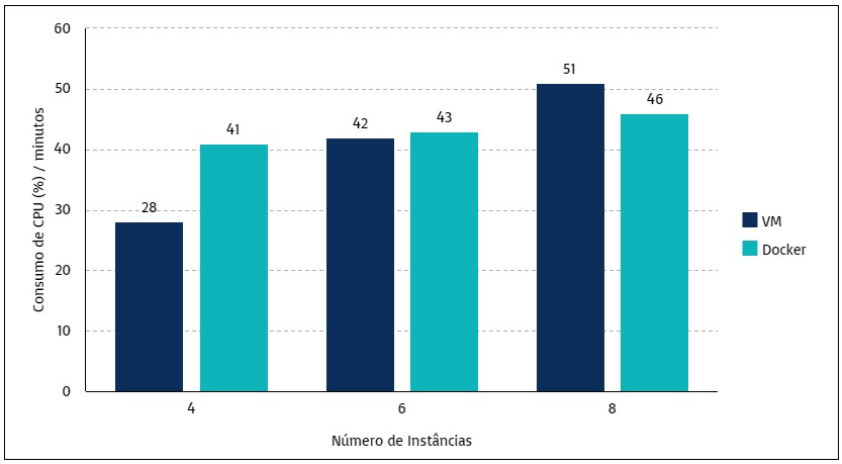

Figura 3.8: Resultado dos testes de consumo de CPU para 4, 6 e 8 instâncias.
Fonte: Elaborado pela autora, baseado em dados extraídos de: [35]

Como resultado final da pesquisa, o autor concluiu que conforme o número de instâncias aumentava, o consumo de CPU foi se acentuando para ambos os cenários, porém, em maior índice para o KVM, demonstrando que, para a pesquisa em questão, em termos de escalabilidade, o Docker supera o KVM para a maior parte dos cenários que foram propostos no estudo.

3.9 Resumo resultados

Conforme analisado anteriormente, segue abaixo Tabela 3.1 com o resumo dos principais resultados encontrados nos artigos analisados quanto a performance das tecnologias máquina virtual e container Docker para consumo de CPU:

Tabela 3.1: Resumo dos resultados.

Artigo	**Diferença percentual no cenário de teste**	**Tecnologia mais vantajosa para o cenário avaliado**	**Critério de avaliação**
A1	4% relativo	Docker	Gflops por segundo
A2	0,13% relativo	Docker	Tempo de execução em segundos
A3	47% relativo	Docker	Tempo de execução em segundos
A4	0,7% relativo	Docker	Tempo de execução em segundos
A5	7% relativo	Docker	Tempo de execução em segundos
A6	30% absoluto	Docker	Percentual de tempo de CPU por segundo
A7	38% absoluto	Docker	Percentual de tempo de CPU por segundo
A8	21% absoluto	Docker	Percentual de tempo de CPU por segundo

Conforme apontado nas análises anteriores e na Tabela 3.1, podemos perceber que os containers Docker obtiveram melhores resultados em praticamente todos os testes realizados para performance de CPU, com exceção dos cenários iniciais de instâncias, sinalizando que, em termos de escalabilidade, o Docker tende a possuir vantagens em relação a VM.

Frisamos que os artigos estudados utilizam métricas diferentes, o que pode dificultar a comparação entre eles. Porém, para os que possuem a mesma métrica, foi possível observar resultados consistentes para a indicação do Docker como a tecnologia que apresentou os melhores resultados em geral.

Salientamos ainda que, apesar das métricas e formas percentuais serem diferentes entre os artigos apontados, a comparação entre eles se justifica uma vez que, individualmente, todos analisam aspectos de performance de CPU e todos apontaram o Docker com um desempenho superior a VM para a maioria dos cenários, mais uma vez apontando resultados consistentes e justificando assim a utilização deles para a comparação escolhida como escopo deste trabalho.

A julgar por todos os estudos apresentados e demais literaturas, pode-se julgar que containers podem ser uma melhor alternativa em relação à virtualização quando observado apenas o aspecto de desempenho. Porém, é importante analisar outros aspectos na hora de escolher qual tecnologia utilizar, uma vez que outros critérios também devem ser levados em consideração, como o aspecto de consumo de memória RAM, Disco, Rede e Segurança, por exemplo.

No próximo capítulo, iremos apresentar as conclusões a que chegamos com este trabalho.

Capítulo 4

Conclusão

A virtualização está muito presente em instituições escolares e outras organizações. Esse fato pode ser observado pelo aumento de empresas que oferecem ao mercado soluções para gerenciamento de ambientes virtuais e, também, devido ao poder computacional que aumentou consideravelmente nos últimos anos. Essa evolução trouxe diversos benefícios, porém causou um aumento de custo para algumas empresas que necessitavam adquirir grandes máquinas que por muitas vezes ficavam ociosas por longos períodos, aumentando assim o custo de manutenção para os usuários, uma vez que muitas arquiteturas são definidas com base em medições de picos de utilização de um sistema.

As máquinas virtuais permitem que um único equipamento seja composto por diversas máquinas hospedeiras independentes, que podem ser do tipo total ou parcial. Embora traga muitos benefícios, tais como redução de custos com equipamentos, distribuição dos recursos, entre outros, também está sujeita a falhas e problemas de performance.

Afim de atingirmos o objetivo geral proposto para este trabalho, que foi de realizar um estudo comparativo de performance entre as tecnologias Máquina Virtual e Container Docker, apresentamos neste trabalho um breve histórico da evolução das técnicas de virtualização e, de forma mais detalhada, das duas tecnologias propostas.

Máquinas Virtuais são computadores de *software* com a mesma funcionalidade que os computadores físicos. Graças a isso, elas ganharam um importante papel no mercado de Infraestrutura como Serviço (IaaS), ficando ao alcance de muitos usuários que necessitavam de maior mobilidade em seus sistemas.

Posteriormente ao surgimento das máquinas virtuais, houve o surgimento de containers, que é um ambiente isolado em uma máquina. A tecnologia de containers fez com que o uso dos recursos de *hardware*, como CPU, RAM e transmissão de dados via rede, pudessem ser compartilhados entre vários *softwares*. O uso dessa tecnologia estendeu o conceito de Infraestrutura como Serviço (IaaS) para Containers como Serviço (CaaS).

Realizamos então a análise de oito publicações envolvendo máquinas virtuais e containers Docker, sob uma perspectiva de performance de CPU. Nos testes analisados, containers Docker saíram-se melhor na maior parte das avaliações, com exceção dos cenários iniciais de instâncias, o que sinaliza que, em termos de escalabilidade, para os artigos analisados, o Docker possui vantagens em relação a VM. Apesar dos artigos utilizarem métricas diferentes entre si, foi possível observar resultados consistentes para o desempenho do Docker, demonstrando assim o poder da ferramenta em diversos cenários de teste e sob as óticas de desempenho e consumo de CPU.

Embora a virtualização traga diversos benefícios, faz-se necessário um estudo do cenário onde elas serão aplicadas, de forma a avaliar as vantagens e desvantagens da aplicação de cada uma das técnicas, levando-se em consideração também o aspecto de consumo de memória RAM, Disco, Rede e Segurança.

Como trabalhos futuros desejamos realizar a implementação das duas tecnologias apresentadas, de forma a realizar um estudo prático comparativo entre elas, além de acoplar nos testes a tecnologia Kubernetes para orquestração dos containers Docker.

Referências

[1] Yadav, R. R.; Sousa, E. T. G.; Callou G. R. A.: *Performance comparison between virtual machines and docker containers.* IEEE Latin America Transactions, 16, 2018. 1, 26, 27

[2] VMware: *Glossário: O que é uma máquina virtual?* Disponível em: <https://www.vmware.com/br/topics/glossary/content/virtual-machine.html>. Acesso em: 01 dez 2023. 1

[3] Zhang, Q.; Cheng, L.; Boutaba R.: *Cloud computing: state-of-the-art and research challenges.* J Internet Serv Appl, 7–18, 2010. 1

[4] Herrera-Izquierdo, L.; Grob, M.: *A performance evaluation between docker container and virtual machines in cloud computing architectures.* CEDIA, Maskana, 8, 127-133, 2017. 1, 7, 12, 22, 23

[5] Damasceno, J. C.: *Construindo uma nuvem privada com openstack.* Anais do IX ENUCOMP. Parnaíba: FUESPI, 1, 9-36, 2016. 2, 6, 8, 15

[6] Oliveira, M. M.: *Como fazer pesquisa qualitativa.* Petrópolis, RJ; Vozes, 2007. 2

[7] Thiollent, M.: *Metodologia da pesquisa.* São Paulo,SP; Cortez: Autores Associados, 1986. 3

[8] Martins, B. C.: *Cooperação e livre fluxo da informação: A influência da cultura hacker na definição dos padrões da comunicação mediada por computador.* UniRevista. Universidade Federal do Rio de Janeiro – UFRJ, RJ, 1: 3, 2006. 5

[9] McIlroy, M. D.; Pinson, E. N.; Tague B. A.: *Unix time-sharing system: Forward.* The Bell System Technical Journal, EUA, 57: 6, 1978. 5

[10] Mattos, G. O.: *Aspectos de desempenho da computação paralela em clusters e grids para processamento de imagens.* Tese (Doutorado). Universidade Federal de Pernambuco. Recife, PE, 2008. 5

[11] Randal, A.: *The ideal versus the real: Revisiting the history of virtual machines and containers.* ACM Computing Surveys, 53: 1, 1-31, 2020. 5

[12] Machado, F. B., Maia L. P.: *Arquitetura de sistemas operacionais.* Rio De Janeiro: Livros Técnicos e Científicos Editora S.A.-LTC, 5, 2017. 6

[13] Motyczka, L. B.; Ferrari, E. R.; Destefani L.; Maran V.: *Um comparativo de consumo elétrico entre arquiteturas x86 e arm em servidores de bancos de dados.* Sirc Conference. Santa Maria, RS, 2013. 6

[14] Veras, M.: *Componente central do datacenter componente central do datacenter.* Brasport. Rio de Janeiro, RJ, 2010. 6, 7, 8, 10

[15] Takeuti, A. Y.: *Virtualização em ambientes corporativos.* Perspectivas em Ciências Tecnológicas. Fatece - SP, 3: 3, 30-54, 2014. 7

[16] Ghannoum, R. G.; Rodrigues, F. B.: *Virtualização de servidores: Vantagens e desvantagens.* Revista Mirante, Anápolis, GO, 11: 6, 2018. 8, 10

[17] Damasceno, J. C.: *Ucloud: uma abordagem para implantação de nivem privada para a administração pública federal.* Tese (Doutorado). Universidade Federal de Pernambuco, UFPE, PE, 2015. 8

[18] Carissimi, A: *Virtualização: da teoria a soluções.* 26º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos–SBRC. UFRGS. Rio Grande do Sul - RS, 57: 6, 173-207, 2008. 9, 11

[19] Red Hat, Inc: *O que é um container linux?* Disponível em: <https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container>. Acesso em: 07 mai 2023. 11, 12

[20] Docker, Inc: *Docker overview.* Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em: 07 mai 2023. 12, 13, 14, 15, 16

[21] Docker, Inc: *Use containers to build, share and run your applications.* Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em: 07 mai 2023. 12

[22] Red Hat, Inc: *Docker: desenvolvimento de aplicações em containers.* Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-docker>. Acesso em: 08 mai 2023. 12, 13, 17

[23] Garcia, W.; Pereira, F. G.: *Docker containers não são vm's.* 2º Seminário de Tecnologia Gestão e Educação. Faculdade Alcides Maya, Porto Alegre, RS, 5-10, 2019. 13

[24] W3Big: *Arquitetura docker.* Disponível em: <https://www.w3big.com/pt/docker/docker-architecture.html>. Acesso em: 09 jun 2023. 14

[25] Smith, R.: *Docker orchestration.* Packt Publishing Ltd, 2017. 15

[26] Docker, Inc: *Dockerfile reference.* Disponível em: <https://docs.docker.com/engine/reference/builder/>. Acesso em: 08 jun 2023. 15

[27] Braga, A. R.; da Silva, P. R. X.; Soares J. M.; Gomes D. G.: *Ócio produtivo: Explorando a ociosidade de clusters virtuais para execução de aplicações do tipo saco de tarefas.* X Workshop em Clouds e Aplicações. Fortaleza - CE, 10, 2012. 16

[28] Coleman, M.: *Containers and vms together.* Disponível em: <https://www.docker.com/blog/containers-and-vms-together/>. Acesso em: 08 jun 2023. 16, 17

[29] Taurion, C.: *Cloud computing. computação em nuvem: transformando o mundo da tecnologia da informação.* Rio de Janeiro: Brasport, 2009. 21

[30] Li, Z.: *Comparison between common virtualization solutions: Vmware workstation, hyper-v and docker.* IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC). Greenville, SC, USA, 2021. 23, 24

[31] Potdar, A. M.; Narayan, D. G.; Kengond S.; Mulla M. M.: *Performance evaluation of docker container and virtual machine.* Procedia Computer Science, 171(08), 2020. 25

[32] Seo, K.; Hwang, H.; Moon I.; Kwon O.; Kim B.: *Performance comparison analysis of linux container and virtual machine for building cloud.* Advanced Science and Technology Letters. Networking and Communication, pp.105-111, 66, 2014. 27, 28

[33] Bonnier, V.: *Comparison between openstack virtual machines and docker containers in regards to performance.* Blekinge Institute of Technology. Karlskrona, Sweden, 2020. 28, 30

[34] Felter, W., Ferreira A. Rajamony R. Rubio J.: *An updated performance comparison of virtual machines and linux containers.* IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, TX, 2015. 29, 31

[35] Silva, R. C.: *Docker versus kvm: Uma análise de desempenho de disco para pequenos arquivos.* Universidade Federal do Ceará. Quixadá, CE, 2016. 31, 32