COMPARATIVO DE DESEMPENHO ENTRE VIRTUALIZAÇÃO E CONTÊINERES UTILIZANDO DOCKER

COMPARATIVE PERFORMANCE BETWEEN VIRTUALIZATION AND CONTAINERS USING DOCKER

Marcelo Scarpin da Fonseca

Graduando em Redes de Computadores pela Fatec Bauru

E-mail: marcelo.scarpin.fonseca@gmail.com

Gustavo Cesar Bruschi

Docente na Fatec Bauru

E-mail: gustavo@bruschi.net

RESUMO:

Embora o assunto virtualização seja amplamente abordado e conhecido mundo afora, a utilização de contêineres, por alguns, ainda não é completamente habitual. Docker é uma das ferramentas mais populares, que visa a praticidade, selando em um container todas as partes necessárias de uma aplicação. Este artigo traz como proposta fazer uma comparação de desempenho entre um ambiente “conteinerizado” e um ambiente de virtualização, para isso, foram realizadas três etapas de testes analisando processamento e mais três etapas de testes analisando a memória RAM. Conclui-se que os resultados com Docker sobre a virtualização foram aproximadamente 50% mais eficientes para o processador e 86% mais eficientes para a memória, concluindo que ele pode ser atrativo em várias situações.

Palavras-chave: Virtualização. Performance. Docker.

ABSTRACT:

Although the subject of virtualization is widely discussed and known worldwide, the use of containers, by some, is not yet completely habitual. Docker is one of the most popular tools, which aims at practicality, sealing in a container all the necessary parts of an application. This article proposes to make a performance comparison between a “containerized” environment and a virtualization environment, for this, three test rows were analyzed analyzing processing and three more test rows analyzing RAM memory. The results obtained show that Docker's results on virtualization were approximately 50% more efficient for the processor and 86% more efficient for memory, concluding that it can be attractive in several situations.

Keywords: *Virtualization. Performance. Docker.*

1 INTRODUÇÃO

A virtualização em si não é novidade. Ela começou nos anos 70 com os caros e gigantescos *mainframes,* e nessa época era comum que os computadores possuíssem sistemas operacionais diferentes, mesmo aqueles do mesmo fabricante. Com a dificuldade da distribuição de *softwares* entre os *mainframes,* surgiu o que pode ser chamado de início da virtualização. Na realidade, as aplicações eram acompanhadas de todo o ambiente operacional necessário para a sua execução. Em outras palavras, a IBM criou um ambiente padronizado e uniforme para a execução de aplicações em suas máquinas: uma máquina virtual. E esse foi o caminho usado pela IBM na linha 370 de seus *mainframes* e sucessores, assim, simplificando a migração de aplicações entre *mainframes* e criando um ambiente mais uniforme para a criação de aplicações (SILVA, 2007).

Existem várias arquiteturas de virtualização, entre elas virtualização total, paravirtualiazação, virtualização assistida por *hardware*, e finalmente virtualização em nível do sistema operacional. Este último é a arquitetura de virtualização que é indiretamente abordada neste trabalho (PRADA D. L. et al. 2017).

Diferentemente da paravirtualização e da virtualização total essa forma de arquitetura de virtualização não depende de uma *Virtual Machine Monitor* (VMM). Deste modo o sistema operacional isola múltiplas áreas (*User-Spaces*) chamadas de contêineres. Apesar de conter algumas vantagens em relação à virtualização total e paravirtualização a virtualização baseada em contêineres vem com questões de segurança (PRADA D. L. et al. 2017).

Mesmo que a virtualização baseada em contêineres não seja nenhuma novidade, Docker surgiu relativamente recentemente no mercado, ele tem sido usado em algumas aplicações populares como Ebay e Spotify. Tendo como diferencial já ter nascido com recursos atuais, que ferramentas antigas focadas em virtualização baseada em container não tinham, como por exemplo:

a) Operar facilmente com ferramentas *third-party;*

b) Criar mais ambientes virtuais, no mesmo hardware, que outras tecnologias de virtualização em nível de sistema operacional;

c) As aplicações podem ser feitas em contêineres leves que podem operar sem modificações em quase qualquer lugar;

d) Provem interfaces para controlar e criar contêineres, de maneira simples e relativamente segura.

Este trabalho buscou comparar a performance entre uma ferramenta a base de contêineres e uma ferramenta de virtualização (respectivamente Docker e Virtual Box). Para realizar essa comparação foi utilizada a ferramenta de *benchmark* SysBench. As três etapas de testes (três para o processador e mais três para a memória RAM) foram feitas em cada ferramenta testada, a aplicação também fez três testes nativamente no sistema operacional como base de controle de comparação. O produto desta ferramenta de *benchmark* é medido em tempo: Quanto menor o tempo

de execução maior o resultado da performance do teste. O objetivo é detectar qualquer diferença de performance entre as ferramentas, e com isso comprovar que a ferramenta Docker é mais eficiente que a ferramenta de virtualização, os resultados colhidos mostram que os resultados do Docker sobre a virtualização foram aproximadamente 50% mais eficientes para o processador e 86% mais eficientes para a memória.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Virtualização

As máquinas virtuais, também chamadas de partições virtuais, permitem que a execução de várias tarefas aconteça de maneira simultânea, isto é, vários processos e aplicações necessitam compartilhar os mesmos recursos. Sendo assim uma partição virtual é na prática um ambiente operacional autossuficiente, mesmo se baseando nas características do hardware e software da máquina hospedeira. Isto é apenas possível devido ao *Virtual Machine Monitor* (VMM) (OLIVEIRA, 2015).

O VMM, também conhecido como *hypervisor,* é um software que possibilita o gerenciamento e criação das máquinas virtuais, e controla a operação de ambientes virtualizados em máquinas hospedeiras. Ele é incumbido do gerenciamento da utilização dos recursos do sistema hospedeiro, assim sendo ele é a peça de software que governa a utilização da memória, processadores e dispositivos de entrada e saída relacionados à máquina virtual. Um ponto notável que deve ser destacado é que o VMM é responsável pelo escalonamento das máquinas virtuais de modo similar a um escalonador de processos em um sistema operacional. Obviamente esse escalonamento só acontece quando mais de uma máquina virtual está sendo executada ao mesmo tempo (LAUREANO; MAZIEIRO, 2008).

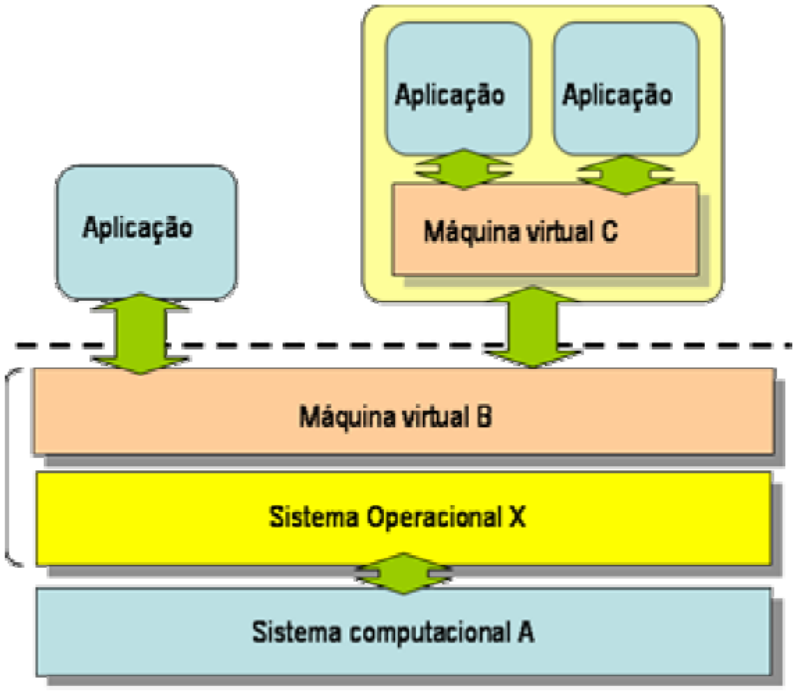
A virtualização em um modo geral tem vários benefícios, os quais podem refletir diretamente em redução de custos em ambientes corporativos, por exemplo:

a) Economia de espaço físico em *data centers*;
b) Economia de energia elétrica, com a centralização de vários servidores em apenas um só virtualizando todos os outros, existe um menor consumo de energia elétrica por parte do hardware, e também em menos uso de sistemas de refrigeração;
c) A possibilidade de usar aplicações escritas para uma plataforma, em uma outra plataforma;
d) A possibilidade de gerenciar recursos de maneira centralizada;
e) A facilidade de se fazer o *backup* e sua restauração;
f) A facilidade para a instalação, migração e remoção (SILVA, 2007).

Embora as arquiteturas de virtualização tenham certas semelhanças, elas se distinguem nos métodos utilizados e no nível de abstração. Serão abordadas nesse

trabalho apenas algumas das principais arquiteturas de virtualização (BUI, 2015). A figura 1 ilustra um exemplo básico de virtualização.

Figura 1 - Princípio básico de máquinas virtuais



Fonte: CAISSIMI, A. cap. 4. 26º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2012.

2.1.1 Virtualização em Nível de Sistema Operacional

A principal característica dessa arquitetura de virtualização é que não necessita de um VMM, o sistema operacional tende a isolar várias “áreas de usuários” as *User-Spaces*, as quais são denominadas de contêineres (SILVA, 2007).

Esta arquitetura é viabilizada principalmente pelo uso de uma entidade chamada de *Namespace*, ele é um conjunto de caracteres que permite diferenciar diferentes objetos, ou seja, é um identificador único, um ambiente abstrato. Isso possibilita que usuários diferentes utilizem espaços diferentes em um mesmo sistema operacional, sendo assim o isolamento do sistema como um todo é garantido pois tornam-se incapazes de se comunicar (SILVA, 2007).

Sendo assim, um contêiner pode ser definido como uma peça que usa todos os *Namespaces* necessários em conjunto, sendo assim, há apenas um *kernel*, o kernel do sistema hospedeiro, que de uma maneira isolada é compartilhada pelos contêineres (OLIVEIRA, 2015).

Mais um ponto notável associado a distinção dessa arquitetura das demais, além de não necessitar de um VMM, é que como não há um VMM não existe uma

pré-alocação dos recursos, os recursos são simplesmente controlados e limitados pelo kernel, sendo assim, todos os recursos podem ser dinamicamente alocados. Uma consequência pratica disso é que se um recurso não está sendo usado ele pode ser alocado para outro contêiner (OLIVEIRA, 2015).

Das críticas à virtualização em nível de sistema operacional, a que mais se destaca para BUI (2015) é a dificuldade de se manter um isolamento eficaz em diferentes aplicações compartilhando o mesmo *kernel*.

2.1.2 Virtualização Assistida por Hardware

Também conhecida como Virtualização Nativa, ela usa o suporte de hardware para a virtualização, com o objetivo de ajudar a reduzir o impacto da camada de virtualização utilizando novos recursos pontualmente incorporados no processador. Os maiores representantes dessa arquitetura são o Intel VT e AMD-V (OLIVEIRA, 2015).

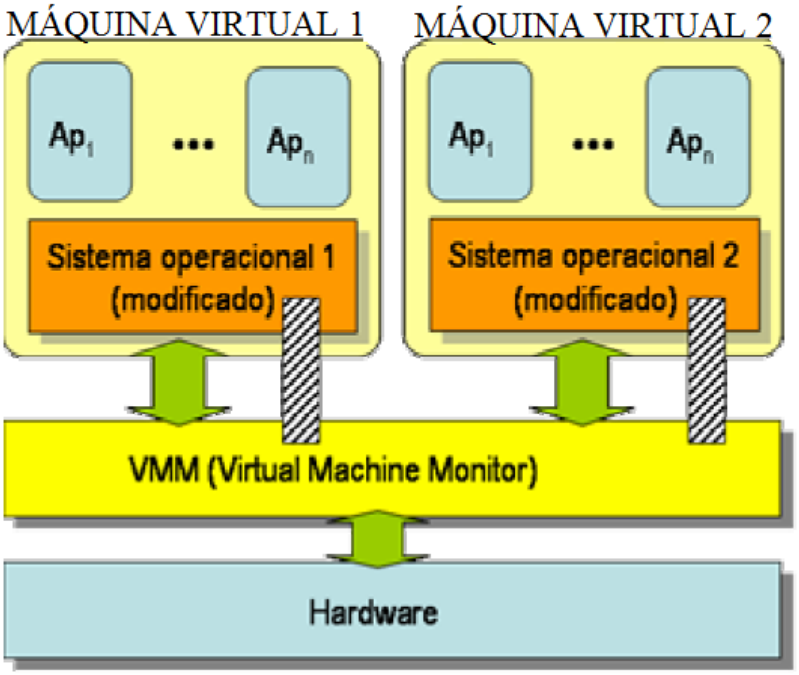
Com a arquitetura Intel VT, ela basicamente eliminou a necessidade da tradução da camada de software, essa arquitetura funciona fazendo com que o VMM tenha um controle parcial sobre o comportamento do CPU, na prática há um bit de controle que quando em estado “0” significa que o processador pode executar as instruções irrestritamente, quando em estado “1” o processador entende que está executando uma máquina virtual, desse modo algumas instruções privilegiadas são desabilitadas, e o processador da preferência ao armazenamento em cache invés de memória RAM (para tornar mais veloz o compartilhamento do tempo de processamento das demais máquinas virtuais por exemplo) (SILVA, 2007).

Essa arquitetura permite que todos os sistemas operacionais sejam capazes de funcionar sobre a mesma CPU, pois o acesso é realizado diretamente ao *hardware*, sendo assim ela remove a necessidade da emulação de um processador, assim sendo ela traz a possibilidade de vários sistemas operacionais não modificados possam ser executados paralelamente. Isso a torna diferente da técnica de virtualização total pois nesta última é possível executar um sistema operacional em um processador emulado (LAUREANO; MAZIEIRO, 2008).

2.1.3 Paravirtualização

Nessa arquitetura de virtualização o sistema operacional tem “consciência” de que está sendo virtualizado, tanto que ele necessita ser modificado para funcionar, são rearranjadas as instruções que teriam acesso direto aos recursos físicos de modo que elas sejam substituídas por chamadas diretas ao VMM. O grande diferencial desta arquitetura é que o VMM fornece uma interface de comunicação para operações consideradas críticas ao *kernel*, como por exemplo o gerenciamento de memória (PRADA D. L. et al. 2017). A figura 2 esboça a paravirtualização.

Figura 2 - Paravirtualização



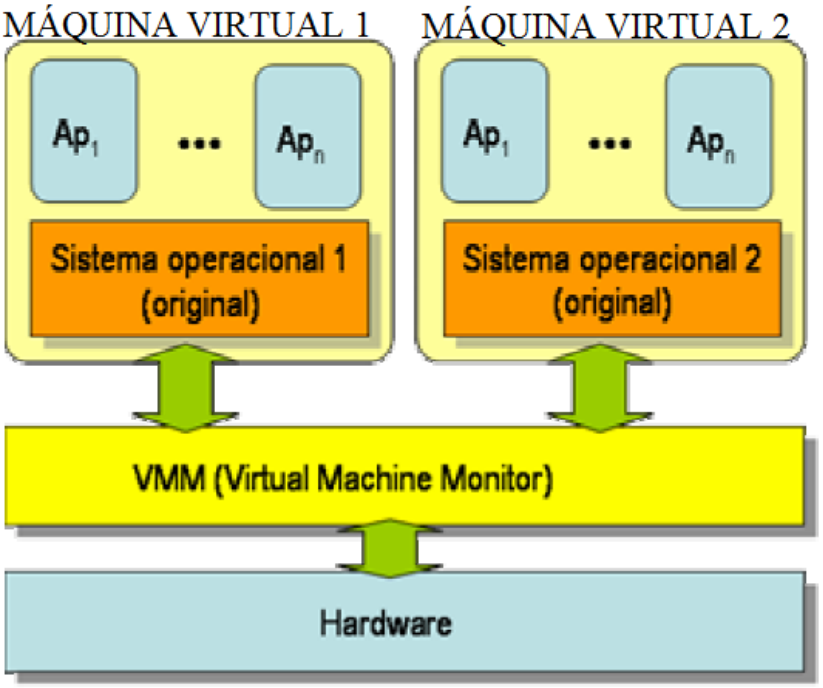
Fonte: CAISSIMI, A. cap. 4. 26º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2012.

As consequências práticas têm efeitos positivos e negativos. O principal efeito positivo é que geralmente existe um ganho de desempenho quando comparada com a arquitetura de virtualização total. O principal efeito negativo é que o sistema operacional necessita ser modificado para esta arquitetura, e nem todos os sistemas operacionais disponibilizam o código fonte como por exemplo os sistemas operacionais da família Microsoft (SILVA, 2007).

2.1.4 Virtualização Total

A virtualização total também é chamada de emulação de hardware por alguns, isso porque para um sistema funcionar nessa arquitetura o *hardware* tem que ser completamente emulado, sendo assim o VMM necessita receber e traduzir em tempo real as instruções para a máquina virtual, essa arquitetura necessita que todas as características do hardware alvo sejam fielmente emuladas para a máquina virtual, isso inclui por exemplo todo o conjunto de instruções de *I/O* e acesso à memória (SILVA, 2007). A figura 3 é uma ilustração aproximada da virtualização total.

Figura 3 - Virtualização total



Fonte: CAISSIMI, A. cap. 4. 26º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2012.

Isso implica que nessa arquitetura o sistema operacional não necessariamente tem “consciência” de que está sendo executado em um ambiente virtualizado pois o *hardware* está fidedignamente sendo reproduzido pelo VMM, assim sendo não há necessidade de modificação de sistemas operacionais, o que é uma vantagem para sistemas operacionais de código fechado como por exemplo os da família Microsoft (PRADA D. L. et al. 2017).

Um ponto que vale a pena ser abordado é a desvantagem dessa arquitetura em relação as demais, ela tem uma notável inferioridade em relação ao seu desempenho devido a essa necessidade de uma emulação completa do *hardware*, o que leva a uma perda de desempenho (OLIVEIRA, 2015).

2.2 Contêineres

SILVA (2016) afirma que o contêiner em si é uma reunião de uma aplicação com suas dependências que estão compartilhando o mesmo kernel de um sistema *host*. Contêineres podem ser considerados similares às máquinas virtuais, porém como utilizam o mesmo *kernel* da máquina *host* conseguem ter um desempenho mais satisfatório.

Os contêineres costumam ser mais compactos quando comparados às outras aplicações, pois usam grande parte dos recursos do sistema host possuem um *overhead* menor (BUI, 2015).

Para LAUREANO e Mazieiro (2008) existe uma diferença chave quando comparamos contêineres com máquinas virtuais. Nas máquinas virtuais ocorre a virtualização de um novo sistema operacional e a emulação de todo um hardware para o seu funcionamento pleno, utilizando recursos da máquina host, o contrário acontece quando utilizamos contêineres, pois os recursos são apenas compartilhados sendo assim é possível rodar vários contêineres no mesmo *host*. O que seria impraticável com uma máquina virtual usando o mesmo *host*.

Com os contêineres também é possível de um ambiente ou uma aplicação para um contêiner facilitando drasticamente a sua portabilidade para outra plataforma necessitando apenas ter o Docker instalado, por exemplo. Com essas características citadas acima é notável que é possível diminuir drasticamente o tempo de implantação de uma aplicação devido à não necessidade de ajustar a aplicação para todo um novo ambiente, podendo ela ser replicada em vários ambientes diferentes com pouquíssimas alterações (BUI, 2015).

É mútuo entendimento de que o desenvolvimento de uma nova aplicação apresenta vários obstáculos durante seu ciclo de desenvolvimento, além de todo o esforço para portar uma aplicação para diferentes ambientes, todavia também podem ocorrer problemas com a escalabilidade de aplicações.

Para SILVA (2016), o ato de “contêinerizar as aplicações” mostra-se uma maneira de facilitar o desenvolvimento de uma aplicação. Certos sistemas operacionais baseados em Unix estão há mais de uma década utilizando conceitos de conteinerização.

Segundo SILVA (2016), tecnologias de virtualização que utilizam contêineres só são possíveis com uma padronização de ambiente. A própria proposta do Docker traz embutida em si a ideia de padronização, pois assim a aplicação pode funcionar em um ambiente padronizado e pode, na prática, desprezar as informações dos recursos de arquitetura da sua máquina host. Deste modo as premissas de desenvolvimento podem ser simplificadas, do mesmo modo que para o sistema host todo contêiner é “igual” sendo tratado praticamente como uma caixa preta, pois praticamente não existem discriminações.

Ao citar Docker é importante citar também o conceito de imagem nesse contexto, ao montar uma virtualização de qualquer aplicação é indispensável uma imagem, que representa fielmente as configurações dos ambientes virtualizados a partir desta mesma imagem. Para traçar um paralelo com as máquinas virtuais, uma metáfora justa seria comparar o arquivo ISO com a imagem e o ambiente virtualizado com o contêiner. Um grande acervo de imagens pode ser encontrado no Docker Hub, lá é possível baixar várias imagens gratuitamente (BUI, 2015).

3 MATERIAIS E MÉTODO

Com o objetivo de realizar um comparativo de desempenho da ferramenta Docker em relação às soluções de virtualização, foram feitos experimentos utilizando a ferramenta Docker baseada em contêineres, a ferramenta de virtualização Virtual Box, e o teste também foi feito no sistema operacional hospedeiro para efeitos de comparação, como uma amostra de controle.

Quanto aos testes realizados no Docker e nativamente no sistema operacional foram realizados em um computador com as seguintes especificações:

a) Ubuntu Desktop 18.04.4 LTS;
b) Processador de seis núcleos fx(tm) 6300 de 3,5 GHz;
c) Memória RAM de 4GB DDR 3 1600 MHz. (Apenas um pente de memória);
d) Partição do HD com 25GB.

Todavia, para os testes realizados na máquina virtual foi adicionado mais memória RAM ao sistema hospedeiro. Isso foi feito para possibilitar que a máquina virtual tenha a mesma quantidade de memória disponível para o teste que o contêiner Docker, porém na máquina virtual foi alocado apenas 4GB de memória RAM para evitar qualquer tipo de diferença entre os testes. Sendo assim, as configurações do sistema hospedeiro da máquina virtual são:

a) Ubuntu Desktop 18.04.4 LTS;
b) Processador de seis núcleos fx(tm) 6300 de 3,5 GHz;
c) Memória RAM de 8GB DDR 3 1333 MHz. (Agora com um pente de memória de 4 GB DDR 3 1333 MHz adicionado);
d) Partição do HD com 25GB.

Tanto no container Docker quanto na máquina virtual foram realizados os mesmos testes, com a mesma versão do SysBench. O container Docker está rodando em uma máquina com Ubuntu 18.04.4 Desktop LTS, a máquina virtual tem 4GB de memória RAM alocada e um disco virtual dinamicamente alocado de 10 GB, está rodando no mesmo sistema operacional citado acima, porém o sistema operacional virtualizado é a versão *server,* a fim de não desperdiçar recursos de processamento e comprometer os testes.

Foram realizados os testes com o software de *benchmark* SysBench, o qual é uma ferramenta desenhada para realizar um teste preciso e simples de *benchmark*, é uma ferramenta modular, multiplataformas que pode operar em *multi-thread*. No teste de CPU o programa processa uma quantidade especificada por parâmetros de números primos (o que demanda um grande esforço do processador), (os cálculos são realizados na base de 64 *bits*). Já para o teste de memória, o programa faz testes sequenciais de escrita ou leitura, pelos parâmetros é definido o tamanho do bloco de memória a ser lido ou escrito, e a quantidade total de memória a ser lida ou escrita. O produto de todos os testes de *benchmark* realizados deste trabalho é a quantidade de tempo (em segundos), deste modo, quanto menor a quantidade de tempo de um resultado, mais eficiente o resultado é. (O tempo é inversamente proporcional à eficiência do resultado.)(Kopytov, 2009).

Foram realizadas três etapas de testes em cada plataforma (Docker, Virtual Box e nativamente no Ubuntu) para medir a performance do processador e mais três etapas de testes para medir a performance da memória RAM.

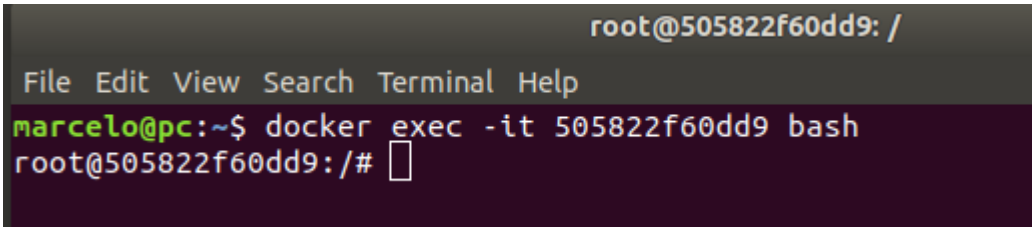
Para realizar os testes no Docker foi necessário primeiramente fazer o download da imagem Docker oficial do Ubuntu para a partir dela criar um contêiner e iniciá-lo em modo interativo. A Figura 4 exemplifica o início da execução do contêiner em modo interativo, esses comandos são explicados na Tabela 1. Após esse procedimento o SysBench pôde ser instalado no contêiner Docker com o comando **apt install sysbench**. Este exato mesmo comando foi utilizado para instalar o SysBench no Ubuntu da máquina virtual e nativamente no próprio sistema hospedeiro.

Tabela 1 – Análise dos comandos utilizados referentes ao Docker

Comando	Explicação
docker ps -a	Lista os processos referentes ao Docker, incluindo estado (parado ou executando), ID e nome do contêiner
docker pull ubuntu	Faz o download da imagem oficial do Ubuntu, disponível no DockerHub
docker run -it ubuntu bash	Inicia um novo contêiner da imagem Ubuntu previamente baixada (o parâmetro -it realiza uma execução interativa com o usuário)
docker start 505822f60dd9	Inicia o contêiner de id 505822f60dd9 (o contêiner Ubuntu recém criado)
docker exec -it 505822f60dd9 bash	Inicia a aplicação bash em modo interativo a partir do contêiner com id 505822f60dd9

Autoria própria.

Figura 4 – Exemplo do início da execução do contêiner em modo interativo



Autoria própria.

Para os testes de processador os parâmetros utilizados no programa sysbench foram: **sysbench --test=cpu --cpu-max-prime=20000 run**, explicados mais detalhadamente na Tabela 2.

Tabela 2 – Análise dos comandos utilizados para iniciar os testes do processador

Parâmetros	Explicação
--test=cpu	Indica que o teste a ser realizado será o de processamento
--cpu-max-prime=20000	Especifica a quantidade de números primos a serem processados

Autoria própria.

Para os testes de memória RAM os parâmetros utilizados no programa SysBench foram: **sysbench --test=memory --memory-block-size=1M --memory-total-size=100G --num-threads=1 run**. Estes parâmetros são explicados na Tabela 3.

Tabela 3 – Análise dos comandos utilizados para iniciar os testes da memória RAM

Parâmetros	Explicação
--test=memory	Indica que o teste a ser realizado será o de memória
--memory-block-size=1M	Indica o tamanho do bloco a ser movimentado
--memory-total-size=100G	Indica a quantidade total de memória a ser testada
--num-threads=1	Indica a quantidade de *threads* a serem processadas simultaneamente

Autoria própria.

4 RESULTADOS E DISCUSSÃO

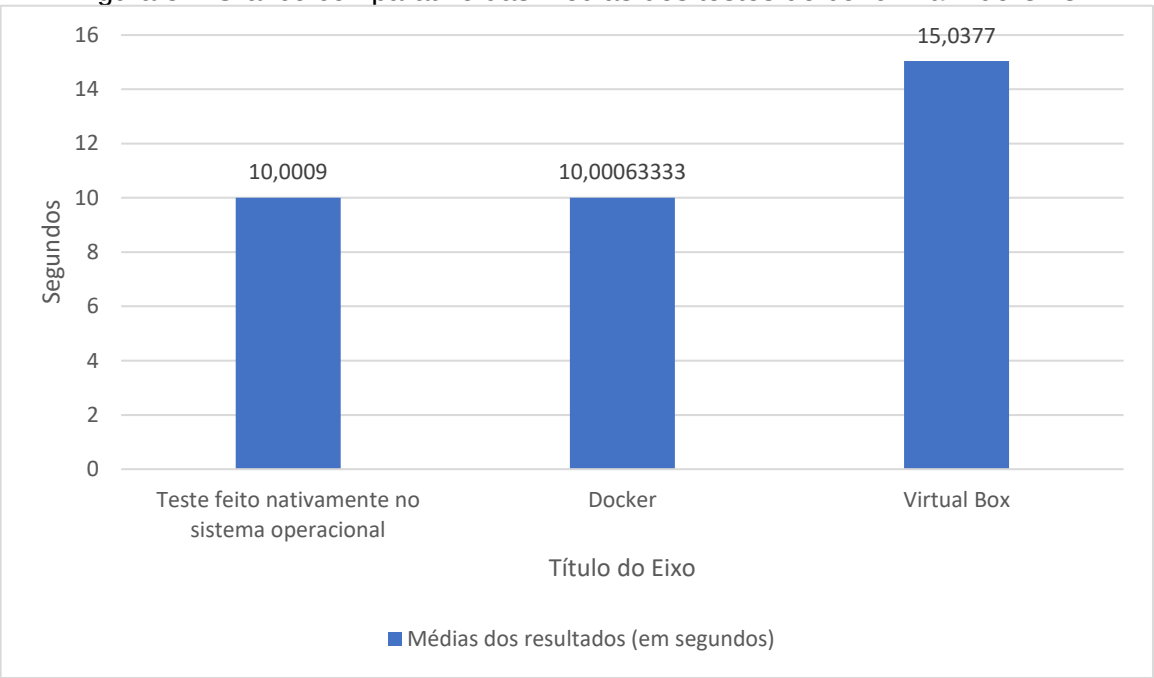
A partir dos métodos apresentados no capítulo 3, os resultados são apresentados nas Tabela 4 e 5. A Tabela 4 apresenta as medições brutas de cada etapa de testes de CPU e suas médias, bem como a Figura 5, que expõe os resultados graficamente para melhor entendimento. Neste ponto já é possível notar que: Enquanto as diferenças das médias de tempo entre o teste executado nativamente e o teste executado na ferramenta Docker é inferior a 2 milissegundos, a diferença entre as médias dos testes do Virtual Box com o Docker e a diferença entre as médias dos testes do Virtual Box com os testes nativos é de mais de 5 segundos. Como apresentado na Tabela 6 e no gráfico da Figura 7, a média dos resultados obtidos pelo Virtual Box é, aproximadamente, 50,36% maior que a média dos resultados do Docker e dos testes realizados diretamente no sistema operacional.

Tabela 4 – Resultados coletados dos testes de *benchmark* do CPU (em Segundos)

Rodada de testes	1	2	3	Média
Nativo	10,00	10,00	10,00	10,00
Docker	10,00	10,00	10,00	10,00
Virtual Box	15,07	15,09	14,95	15,04

Dados coletados a partir do resultado do Programa SysBench.

Figura 5 – Gráfico comparativo das médias dos testes de *benchmark* do CPU



Autoria própria.

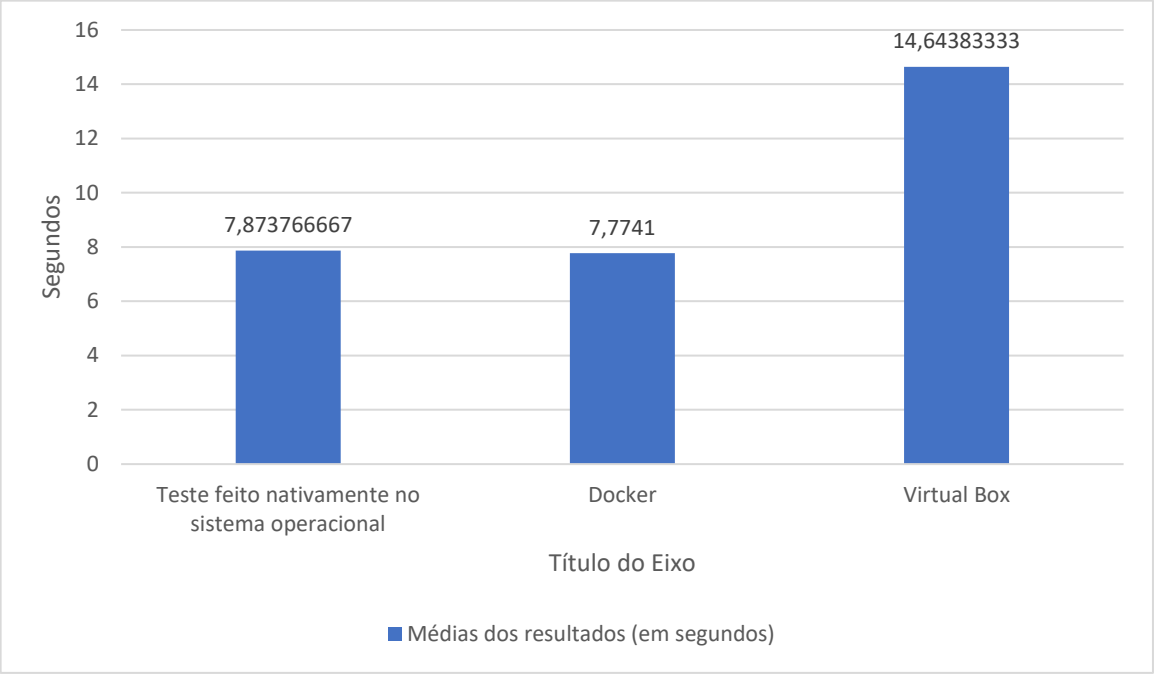
Já a Tabela 5 apresenta as medições brutas de cada etapa de testes de memória RAM e suas médias, assim como a Figura 6, que expõe os gráficos relacionados para melhor entendimento. Também é possível notar algo muito similar aos testes realizados no processador: Enquanto as diferenças das médias de tempo entre o teste executado nativamente e o teste executado na ferramenta Docker é inferior a 0,099 segundos, a diferença entre as médias dos testes do Virtual Box com o Docker e a diferença entre as médias dos testes do Virtual Box com os testes nativos do sistema hospedeiro é de mais de 6 segundos. As variações percentuais dos resultados da memória RAM também são similares às variações percentuais dos testes de CPU, expostos na Tabela 6. A Tabela 7 expõe as variações percentuais entre as médias dos testes de memória RAM, ou seja, expõe que enquanto a diferença percentual entre os testes do Docker e os feitos nativamente no sistema operacional é de aproximadamente 1,28%, a diferença entre os testes do Docker e Virtual Box, e nativamente no sistema operacional e Virtual Box é, aproximadamente e respectivamente, 88,36% e 85,98%.

Tabela 5 – Resultados coletados dos testes de *benchmark* de memória RAM (em Segundos)

Rodada de testes	**1**	**2**	**3**	**Média**
Nativo	8,05	7,80	7,77	7,87
Docker	7,79	7,77	7,76	7,77
Virtual Box	14,57	14,65	14,71	14,64

Dados coletados a partir do resultado do Programa Sysbench.

Figura 6 – Gráfico comparativo das médias dos testes de *benchmark* da memória RAM



Autoria própria.

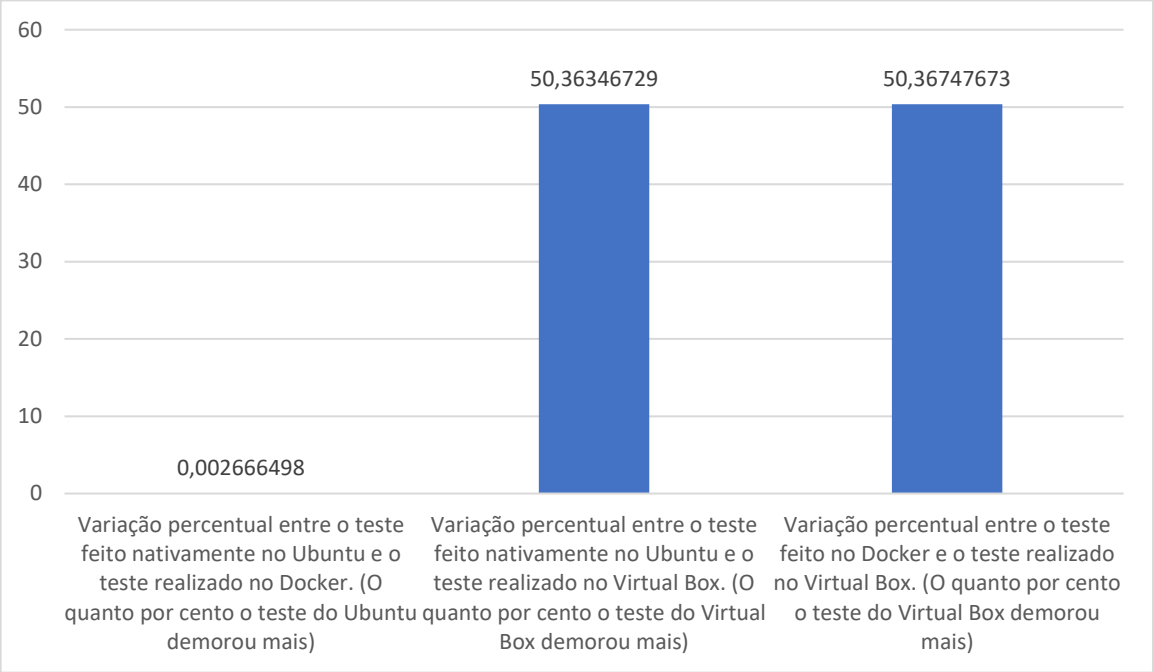
Fórmula da variação percentual, **V.P. = (Maior Valor – Menor Valor) / Menor Valor x 100**, foi utilizada para gerar as Tabelas 6 e 7, utilizando como dados de entrada as médias apresentadas nas Tabelas 4 e 5. O objetivo de utilizar a variação percentual neste trabalho é apresentar, de forma compreensível, as diferenças entre os resultados.

Tabela 6 – Resultado da variação percentual entre as médias (*benchmark* do CPU)

V.P.	Nativo	Docker
Docker	0,002666498%	#
Virtual Box	50,36346729%	50,36747673%

Autoria própria.

Figura 7 – Gráfico comparativo das variações percentuais entre as médias dos testes de *benchmark* de CPU



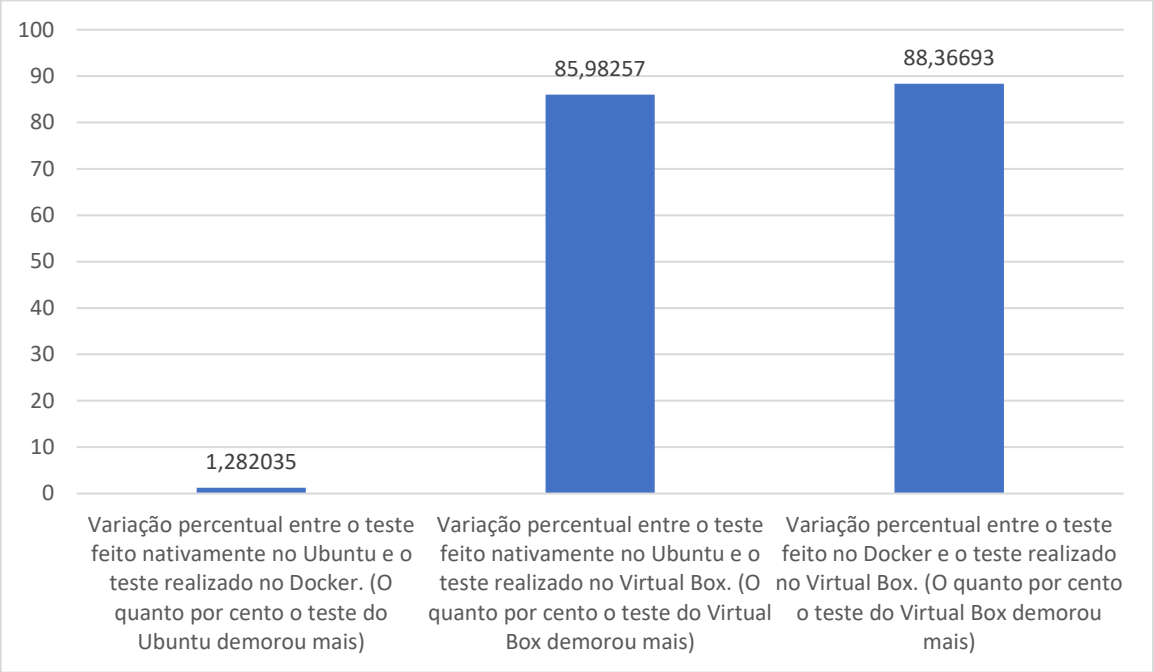
Autoria própria.

Tabela 7 – Resultado da variação percentual entre as médias (*benchmark* da memória RAM)

V.P.	**Nativo**	**Docker**
Docker	1,28%	#
Virtual Box	85,98%	88,37%

Autoria própria.

Figura 8 – Gráfico comparativo das variações percentuais entre as médias dos testes de *benchmark* de memória RAM



Autoria própria.

5 CONCLUSÃO

A partir dos resultados apresentados no capítulo 4, coletados de modo descrito no capítulo 3, é possível concluir que a ferramenta de contêineres Docker utiliza os recursos do sistema hospedeiro de forma mais eficiente que a ferramenta de virtualização Virtual Box, mostrando significativas diferenças de performance tanto quando a comparação é feita entre os testes de memória RAM quanto quando a comparação é feita entre os testes de CPU. De modo simultâneo, as diferenças entre os testes de controle feitos nativamente no sistema hospedeiro e os testes realizados na plataforma Docker têm uma diferença praticamente nula, como por exemplo no caso da variação percentual entre as médias dos testes de CPU entre Docker e o sistema hospedeiro, onde a diferença é de 0,002666498%, mostrando que, se há, existe pouca diferença de performance entre executar uma aplicação via Docker e nativamente no sistema operacional.

6 REFERÊNCIAS

BUI, T. **Analysis of Docker Security**. 13 jan. 2015. Disponível em: https://arxiv.org/pdf/1501.02967.pdf . Acesso em: 24 out. 2019.

OLIVEIRA, I. C**. Aprimorando a Elasticidade de Aplicações de Banco de Dados Utilizando Virtualização em Nível de Sistema Operacional**. 2015. Disponível em:

https://pdfs.semanticscholar.org/c47d/73033d335ebd7f50a3f01d0761ad0d93a95b.pdf . Acesso em: 24 out. 2019.

LAUREANO, M. A. P; MAZIERO, C. A. **Virtualização**: Conceitos e Aplicações em Segurança. 4ª ed. Curitiba, PR, 2008.

SILVA, R. F. **Virtualização de Sistemas** . Dez. 2007. Disponível em: https://www.lncc.br/~borges/doc/Virtualizacao%20de%20Sistemas%20Operacionais.TCC.pdf . Acesso em: 24 out. 2019.

Kopytov, A. **SysBench manual**. 2009. Disponível em: https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf . Acesso em: 12 mai. 2020.

SILVA, W. F. **Aprendendo Docker**. 1ª ed. São Paulo, SP, 2016.

PRADA D. L. et al. **DOCKER**: Apresentação da ferramenta. IV congresso catarinense de ciência da computação P47, 2017.