Comparativo de Desempenho e Funcionalidades entre Docker e Vagrant

Gustavo Ferraz Silveira

[1]Redes de Computadores
Faculdade de Tecnologia SENAC Pelotas
Rua Gonçalves Chaves 602 – 96015560 – Pelotas – RS – Brasil

gustavofsilveira@outlook.com.br

Abstract. *This project aims to perform a comparison between how Vagrant tools and Docker, taking into account, container size, consumption of server resources and also as its functionalities. For this, it is used, within its systems, diverse operating systems with varied services and applications, allowing a greater variety of tests to be done. We will also use various tools to monitor and collect the results obtained in the tests made, these tools are native and third-party.*
Keywords: Docker, Vagrant, container, features.

Resumo. *Este projeto tem como objetivo realizar um comparativo entre as ferramentas Vagrant e Docker, levando em conta, tamanho de container, consumo de recursos do servidor e também as suas funcionalidades. Para isso serão utilizados, dentro de seus containers, sistemas operacionais diversos com variados serviços e aplicações, permitindo uma variedade maior de testes a serem feitos. Também serão utilizadas diversas ferramentas para monitorar e coletar os resultados obtidos no testes feitos, essas ferramentas são nativas e de terceiros.*
Palavras-Chave: Docker, Vagrant, container, funcionalidades.

1. Introdução

O projeto LXC [LXC] (Linux *Container*) começou a ser desenvolvido em agosto de 2008. O objetivo do projeto era ser uma alternativa à tecnologia de Chroot, se tornando um meio termo entre virtualização e Chroot [Chroot], assim possibilitando a criação de ambientes Linux sem a necessidade de um kernel separado, assim diminuindo muito a ocupação em disco de cada sistema operacional. Esses *containers* isolados podem ser limitados tanto em uso de disco, quanto de memória RAM e CPU.

Uma das funcionalidades dos *containers* é o ”Kernel Namespaces”, que possiblita a abstração de processos dentro do kernel, com isso os processos ficam isolados dentro do kernel, permitindo vizualisar os pontos de montagem, id de processos, id de usuário, hostname e fila de processos isolados de outros processos.

Pode-se dizer que *containers* são uma forma de virtualização a nível de sistema operacional. Na Figura 1, é possível ver a diferença de virtualização e *container* na questão de alocação de recursos.

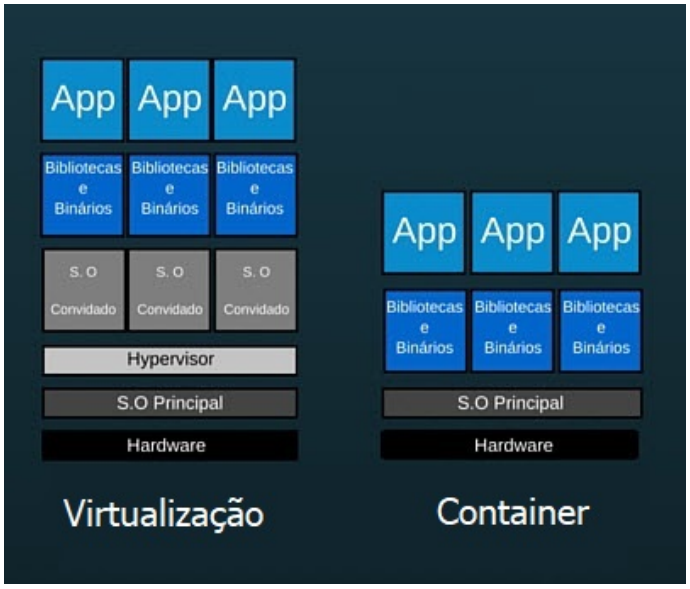

Figura 1. Virtualização e Container

2. Containerização

Os *containers* são uma forma de virtualização a nível de sistema operacional que permite rodar múltiplos "sistemas" isolados em um único sistema operacional real. Esses sistemas isolados conseguem ser, a partir da proteção dos containers, efetivamente isolados e limitados tanto em uso de disco, quanto memória RAM e CPU e só utilizam o necessário do hospedeiro, diminuindo muito o uso de espaço em disco.

Os *containers* possuem diversas funcionalidades.

2.1. Kernel Namespaces:

Possibilita a abstração de processos dentro do kernel, isso quer dizer que um processo ou grupo de processo é isolado dentro do kernel, podendo visualizar os pontos de montagem, id de processos, id de usuário, hostname e fila de processo, isolados de outros processos ou grupo de processo.

2.2. Apparmor *and* SELinux:

Responsável por carregar politicas de acesso no host, isso quer dizer que são definidas regras de acesso a determinados arquivos/diretórios dentro do host, isso garante que em uma possível brecha de segurança, um container não consiga visualizar ou executar algo malicioso no host.

2.3. Seccomp policies:

Realiza uma filtragem das chamadas de sistema (*syscall*) e aceita ou não essa chamada. Como o host e *container* compartilham do mesmo kernel, é necessário algumas políticas especificas a nível de interface do kernel para que o hóspede não consiga escalar privilégios dentro do sistema host.

2.4. Chroots (pivot_root):

Responsável pelo mapeamento de diretórios e ponto de montagem do sistema container, é a chamada de sistema que cria a árvore de diretórios que o container terá acesso.

2.5. Kernel Capabilities:

Todos os sistema UNIX dividem os processos em duas grandes categorias, processo privilegiados, sendo executados como root, e processos não privilegiados, executados com usuário comum. Por padrão todos os *containers* executam comandos não privilegiados, ou seja, um processo dentro de um container é executado como root dentro do host, mesmo que este esteja como root dentro do container. Sendo assim, a partir do kernel 2.2, foram inseridos dispositivos que possibilitam ao container executar de forma privilegiada alguns comandos, para esses dispositivos foi dado o nome de Capabilities ou CAP.

2.6. Cgroups:

É responsável pelo controle de uso dos recursos por processo/grupo de processo, podendo assim executar diferentes containers com diferentes limites de uso (memória, cpu, I/O).

Devido a essas características é que chamamos de container todo sistema criado utilizando essa tecnologia, pois o sistema fica isolado dentro de uma caixa de recursos alocados exclusivamente para ele, e claro limitado a esses recursos.

3. Ferramentas:

Foram selecionadas duas ferramentas para a realização desse comparativo, são elas Docker e Vagrant, ambas ferramentas que utilizam o conceito de *containers*.

3.1. Docker

Docker é uma plataforma Open Source escrito em GO [GO], que é uma linguagem de programação de alto desempenho desenvolvida dentro do Google, que facilita a criação e administração de ambientes isolados.

O Docker [Docker] possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um container, e a partir disto o ambiente inteiro torna-se portável para qualquer outro Host que contenha o Docker instalado, desta forma facilita a realização de testes e de modificações por mais de uma pessoa. Isso reduz muito o tempo de *deploy* de alguma ferramenta ou até mesmo aplicação, pois não tem a necessidade de ajustar o ambiente para o funcionamento do serviço, o ambiente será sempre o mesmo, basta configurar na primeira vez e depois replicar quantas vezes quiser.

Outra facilidade do Docker é pode criar suas imagens a partir de arquivos de definição chamados Docker Files [Dockerfiles]. Uma imagem refere-se a uma lista de camadas, empilhadas uma acima da outra, formando a base do *container*. A imagem é imutável, porém facilmente estendida. Na Figura 2 pode-se ver, de maneira mais facil, como funciona o *container*. Ele utiliza o kernel do host como base do *container*, onde será utilizado um sistema operacional como base da imagem e logo após os serviços que estão disponíveis na imagem.

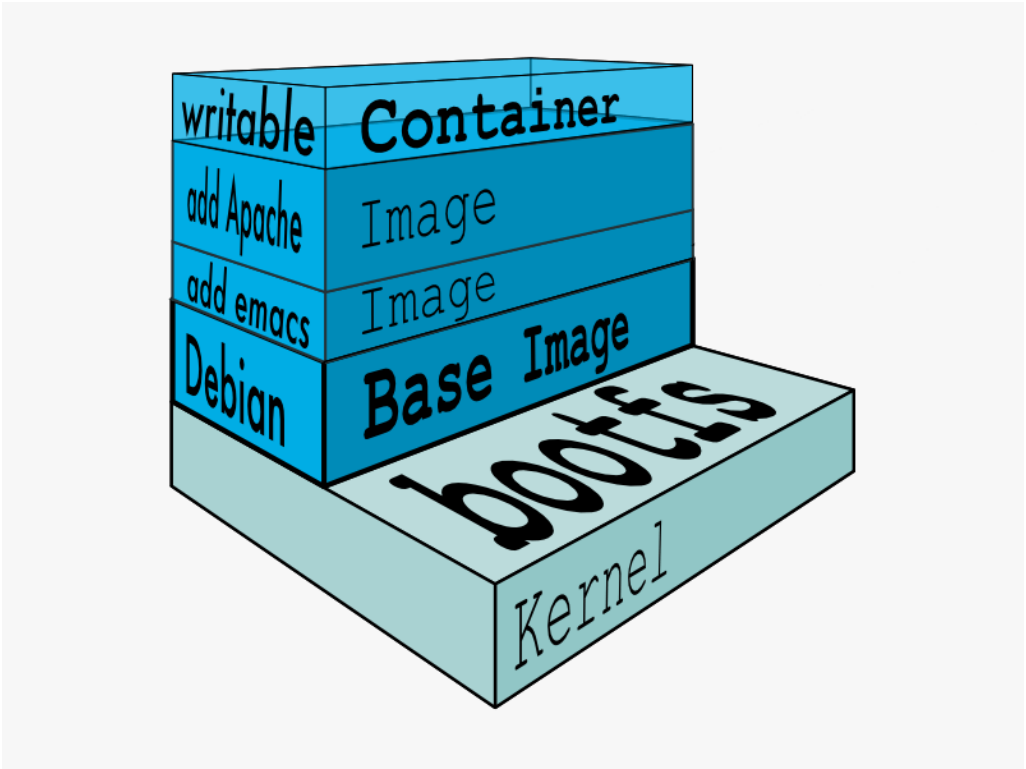

Figura 2. Imagens e Container

Por utilizar como *backend default* o LXC, é possível definir limitações de recursos por *container*, como por exemplo limitar em 1 Gb a memória RAM de um *container*.

É possível encontrar muitas imagens atualizadas e com suporte oficial utilizando o Docker Hub [DockerHub]. Esta plataforma tem repositórios oficiais de diversas ferramentas e aplicações, como MongoDB, Nginx, Apache, MySQL, PostgreSQL, entre outros. Também é possóvel criar suas próprias imagens e disponibilizar nesta plataforma.

3.2. Vagrant

O Vagrant [Vagrant] é uma ferramenta, escrita em Ruby [Ruby], que permite que sejam criados, rapidamente, diversos tipos de ambientes virtuais para realização de testes, desenvolvimento de aplicações, provisionamento de ambiente e disponibilização de serviços utilizando soluções de virtualizações como hospedeiro, por exemplo o VMWare e Virtualbox e também sistemas operacionais Linux.

Assim como o Docker, esta ferramenta possui a facilidade de levar tranferir esse ambiente para qualquer lugar sem dificuldades, basta ter o Vagrant em funcionamento do local de destino e realizar algumas pequenas adaptações.

Os *containers* são chamados de *box*, no Vagrant. Existem diversos repositórios, na internet, com imagens para criação de *Box* no Vagrant.

4. Ambiente de Testes

Para realizar os testes foram utilizados dois servidores, ambos com o sistema operacional Ubuntu Server 16.04 (Recomendado tanto pelo Docker quanto pelo Vagrant) com 2 Gb de Memória RAM, 50 Gb de Disco e 2 cores de CPU. Em um dos servidores foi instalado o Docker e no outro foi instalado o Vagrant. Foram utilizados, para realizar os testes, os serviços Apache, MySQL, ferramentas de *stress* e as próprias funcionalidades de cada ferramenta.

4.1. Apache

O Apache [Apache] HTTP *Server Project* é um projeto para desenvolver e manter um servidor HTTP de código aberto para sistemas operacionais modernos. O objetivo deste projeto é fornecer um servidor seguro, eficiente e extensível que forneça serviços HTTP em sincronia com os padrões HTTP atuais.

4.2. MySQL

MySQL é o banco de dados de código aberto mais popular do mundo. Pelo seu desempenho e facibilidade de uso o MySQL [MySQL] se tornou uma das principais escolhas para de banco de dados para se utilizar em aplicativos baseados na web.

4.3. Stress

Essa ferramenta Stress [Stress] testa o *stress* do sistema operacional de várias formas selecionáveis. Essa ferramenta possui mais de 175 testes de *stress*.

4.4. Versionamento

Ambas as ferramentas possuem a função de versionamento nativamente. Essa função serve para salvar o que foi modificado no sistema operacional do *container* desde o último versionamento feito, podendo ser o estado inicial do *container*. Nestes testes serão utilizadas suas funções de versionamento para analisar qual ferramenta apresentará o melhor desempenho para realizar a tarefa.

5. Ferramentas de coleta de recursos

Para realizar a coleta de dados e monitorar recursos dos servidores, foram utilizadas algumas ferramentas de terceiros que possuem integração com as ferramentas, permitindo assim obter os resultados de maneira mais acertiva e visual. Foram também utilizadas ferramentas nativas, tanto do Docker quanto do Vagrant.

5.1. cAdvisor

O cAdvisor [cAdvisor] é uma ferramenta *Open Source* criada para utilização em *containers* e funciona para gerenciar e monitorar o consumo de recursos de cada *container*. com esta ferramenta é possível verificar a utilização de memória RAM, disco, CPU, rede, entre outros. Também é possível utilizar alguns recursos de gerenciamento, como pro exemplo realizar um *commit* no *container*. Essa ferramenta tem integração exclusiva com o Docker, não tendo suporte para o Vagrant. Na Figura 3 vemos a interface apresentada durante o monitoramento feito em um *container*

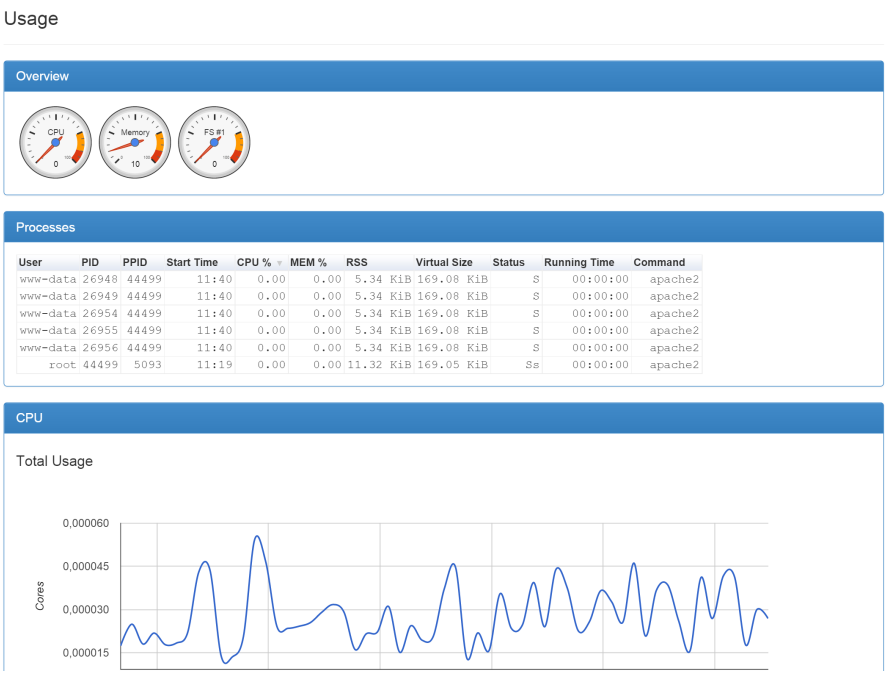

Figura 3. Interface Web cAdvisor

5.2. Rancher

O Rancher [Rancher], é ferramenta que provê uma interface de gerenciamento web para o docker, com ela fica muito mais fácil gerenciar os *containers* e as aplicações e serviços instalados neles, assim como permiti gerenciar a rede isolada que os *containers* utilizam. Esta ferramenta possui a versão free e a paga. Na Figura 4, pode-se ver como é a interface da ferramenta.

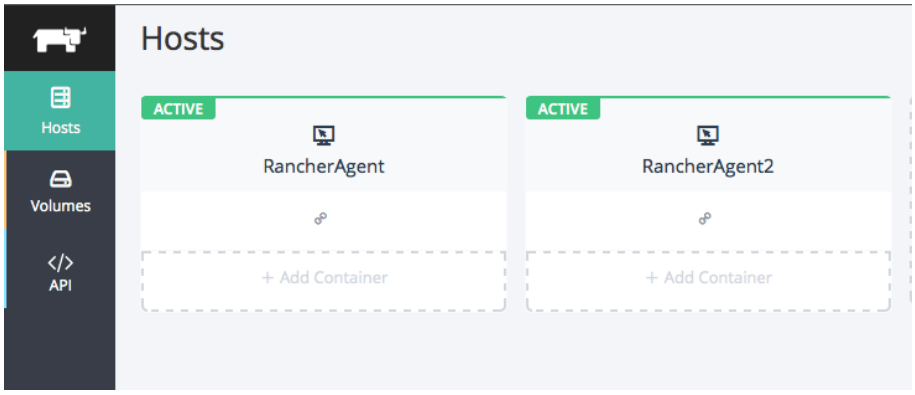

Figura 4. Interface Web Rancher

5.3. Docker Stats

Ferramenta nativa do Docker, o Docker Stats [DockerStats] mostra o que cada *container* está consumindo de recursos do servidor hospedeiro. Essa ferramenta foi utilizada durante alguns testes de *stress* feitos nos *containers*

5.4. Spotlight

O Spotlight [Spotlight] é uma ferramenta que monitora o consumo de hardware do servidor hospedeiro do Docker e do Vagrant, com essa ferramenta é possível acompanhar o consumo detalhado dos recursos do servido como uso de memória, uso de disco, uso de CPU, I/O, usuários conectados, entre outras informações. Neste caso a ferramenta foi conectada no servidor que estava servindo como host cada uma das ferramentas e durante os testes de *stress*, foram obtidos alguns resultados. Na Figura 5 vemos a interface da aplicação.

Figura 5. Interface Spotlight

6. Resultados

Depois de feitos os testes em todas com todas as ferramentas e funcionalidades citadas anteriormente, foram obtidos os resultados.

6.1. Tempo de Versionamento - Teste 1

Depois de instalados os serviços Apache e MySQL e adicionados alguns arquivos e pastas no sistema operacional, configuradas algumas aplicações que utilizam comunicação direta com o banco de dados, para simular um ambiente real, foi realizado o versionamento e os resultados de desempenho estão no gráfico da Figura 6, onde é possível ver que o Vagrant levou cerca de 11 segundos e o Docker 5 segundos.

A diferença de tempo entre as ferramentas foi bem grande, o Docker levou menos que a metade do tempo que o Vagrant para realizar o versionamento de um *container*, no caso do Vagrant *Box*. com o mesmo sistema operacional e mesmas modificações feitas.

Em testes com maior número arquivos adicionados e com o *container* mais tempo em funcionamento, o Docker mostrou melhor desempenho, com resultados sempre na metade do tempo que o Vagrant.

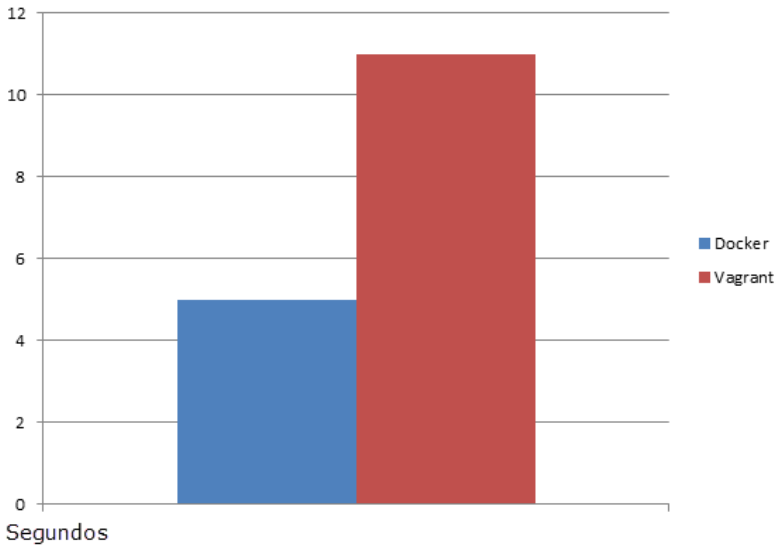

Figura 6. Gráfico de Tempo para Versionamento

6.2. Tempo de Versionamento - Teste 2

Em outro teste feito, para analisar o desempenho durante o versionamento, foi utilizada a ferramenta Stress para criar uma grande carga no processamento do *container* e, durante esse *stress* criado, realizar o versionamento local em ambas as ferramentas, Docker e Vagrant.

O Docker levou cerca de 75 segundos, enquanto o Vagrant, cerca de 85 segundos, como pode-se ver na Figura 7.

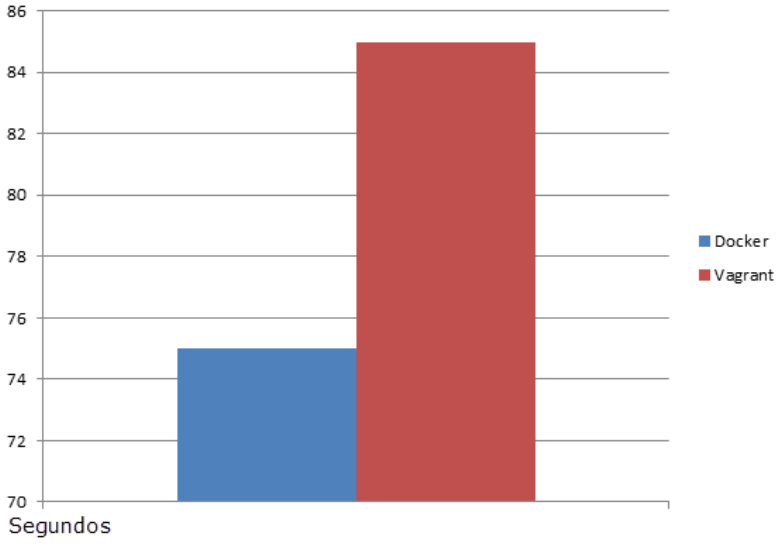

Figura 7. Gráfico de Tempo para Versionamento

6.3. Controle de Recursos

Ambas as ferramentas, Docker e Vagrant, possuem nativamente controle para restringir o uso de hardware de seus *containers*. Durante testes de *stress* feitos em seus *containers*,

utilizando o comando *"While"* e a ferramenta *"stress"*, tanto Docker quanto Vagrant cumpriram seu papel e não deixaram que seus *containers* utilizassem cota de hardware maior que a configurada inicialmente. Em um dos testes os *containers* foram configurados para restringir a 1GB de memória em cada ferramenta e, com as ferramentas acima citadas, foram feitos testes de *stress* forçando ao uso de 2GB, porém os *containers* em nenhum momento passaram do valor estipulado na configuração.

6.4. Funcionalidades e Recursos de cada ferramenta

O Docker possui uma vasta lista ferramentas de terceiros que são desenvolvidas específicamente para funcionamento nele e que auxiliam bastante em diversas tarefas do cotidiano, como por exemplo ferramentas para gerenciamento web dos *containers*, ferramentas para monitoramento de *containers*. O Docker possui, também, bastante documentação na internet e no seu site oficial.

O Vagrant possui bem menos ferramentas de terceiros para agregar no seu uso diário, assim como também possui bem pouca documentação se comparado ao Docker.

o Vagrant pode ser utilizado diretamente em um virtualizador, o que aumenta a variedade de sistemas operacionais que podem ser utilizados em *box* em um mesmo hospedeiro. Essa funcionalidade, permite que possam ser utilizados kernel ligeiramente diferentes.

A variedade de comandos nativos para gerenciar seus *containers* no Docker, também é mais vasta que a do Vagrant, o que faz muita diferença na hora de escolher qual ferramenta utilizar no ambiente.

7. Análise de Resultados

Cada ferramenta tem seu ponto forte e é adequada em determinado ambiente. No caso do Docker, é uma ferramenta completa de containerização, possui diversas funcionalidades nativas e tem um desempenho melhor na questão de deploy, versionamento e gerenciamento de *containers*, ele possui também muita documentação e várias ferramentas de terceiros que facilitam muito o uso diário de *containers*, como por exemplo o Rancher, que foi mostrado anteriomente.

O Docker possui também um repositório muito vasto de imagens, com uma vasta lista de serviços pré-instalados e configurados e isso agiliza muito o processo de criação e implementação de um *container* em um ambiente de produção.

Como ponto negativo do Docker, vale ressaltar a questão do Kernel, como ele utiliza o Kernel do hospedeiro para a criação dos *containers*, não é possível criar um *container* com um sistema operacional que utiliza um Kernel diferente do servidor hospedeiro.

O Vagrant demonstrou desempenho inferior ao do Docker na realização de tarefas como versionamento, mais específicamente metade do tempo em um dos testes e também possui pouca documentação. Ele tem como ponto forte a possibilidade de utilizar ele diretamente em um virtualizador e isso permite que ele utilize *containers* com sistemas operacionais que possuem Kernel diferente, isso consome um pouco mais de disco, porém é uma grande e importante característica.

Na questão de controle de recursos, ambas as ferramentas tiveram resultados satisfatórios, restringindo uso de rede, de memória RAM, uso de disco e de CPU. Nenhuma das ferramentas falhou nos testes em nenhum momento.

O único cenário que o Vagrant tem uma pequena vantagem em cima do Docker, é em ambientes onde existe a necessidade de utilização de sistemas operacionais com Kernel diferente.

8. Conclusões

Ambas as ferramentas são muito boas, porém durante os testes feitos o Docker apresentou melhor desempenho, mais funcionalidades, vasta lista de ferramentas de terceiros compatíveis e mais facilidade para uso e gerenciamento de seus *containers*.

Outro fato que pesa a favor do Docker, são as ferramentas de terceiros que são desenvolvidades especificamente para ele, assim não tendo como se utilizadas no Vagrant.

Em comparação ao Docker, existe muito pouca documentação do Vagrant na internet, o que faz com que o Docker leve a melhor neste ponto.

Cada uma das ferramentas tem seus pontos fortes e fracos, o que faz com que cada uma seja adequada para determinada situação ou peculiaridade em um ambiente.

Referências

Apache. The Apache Software Foundation. Disponível em: <https://www.apache.org/>. Acesso em: Junho de 2017.

cAdvisor. cAdvisor. Disponível em: <https://github.com/google/cadvisor>. Acesso em: Junho de 2017.

Chroot. A Brief History of Chroot. Disponível em: <http://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016>. Acesso em: Junho de 2017.

Docker. Docker. Disponível em: <https://www.docker.com/>. Acesso em: Maio de 2017.

Dockerfiles. Docker. Disponível em: <https://docs.docker.com/engine/reference/builder/>. Acesso em: Junho de 2017.

DockerHub. Docker. Disponível em: <https://hub.docker.com/>. Acesso em: Junho de 2017.

DockerStats. Docker. Disponível em: <https://docs.docker.com/engine/reference/commandline/stats/>. Acesso em: Junho de 2017.

GO. The GO Programing Language. Disponível em: <https://golang.org/>. Acesso em: Maio de 2017.

LXC. Linux Container, Infrastructure for container projects. Disponível em: <https://linuxcontainers.org/>. Acesso em: Junho 2017.

MySQL. The world's most popular open source database. Disponível em: <https://www.mysql.com/>. Acesso em: Junho de 2017.

Rancher. Simple, easy-to-use container management. Disponível em: <http://rancher.com/>. Acesso em: Junho de 2017.

Ruby. O melhor amigo do programador. Disponível em: <https://www.ruby-lang.org/pt/>. Acesso em: Junho de 2017.

Spotlight. Spotlight. Disponível em: <https://www.quest.com/>. Acesso em: Junho de 2017.

Stress. Ferramenta Stress. Disponível em: <http://kernel.ubuntu.com/ cking/stress-ng/>. Acesso em: Junho de 2017.

Vagrant. Development Environments Made Easy. Disponível em: <https://www.vagrantup.com/>. Acesso em: Junho de 2017.