Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

JOÃO ROBERTO RODRIGUES FILHO

DOCKER - A REVOLUÇÃO DA VIRTUALIZAÇÃO COM O USO DE CONTAINERS

Assis - SP
2016

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

JOÃO ROBERTO RODRIGUES FILHO

DOCKER - A REVOLUÇÃO DA VIRTUALIZAÇÃO COM O USO DE CONTAINERS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal do Ensino Superior de Assis – IMESA e Fundação Educacional do Município de Assis – FEMA, como requisito parcial para a obtenção do Certificado de Conclusão.

Orientador: Dr. Almir Rogério Camolesi

Área de Concentração: Informática, Virtualização, Linux

Assis - SP
2016

FICHA CATALOGRÁFICA

RODRIGUES F., João Roberto
Docker: A Revolução da Virtualização com o Uso de Containers / João Roberto Rodrigues Filho. Fundação Educacional do Município de Assis - FEMA - Assis, 2016.
Numero de paginas: 47
Orientador: Dr. Almir Rogério Camolesi.
1. Docker. 2. Virtualização. 3.Linux Container.

CCD: 001.6
Biblioteca da FEMA

DOCKER: A REVOLUÇÃO DA VIRTUALIZAÇÃO COM O USO DE CONTAINERS

JOÃO ROBERTO RODRIGUES FILHO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: Dr. ALMIR ROGÉRIO CAMOLESI

Examinador: ESP. DOMINGOS DE CARVALHO VILLELA JUNIOR

Assis/SP
2016

DEDICATÓRIA

Dedico este trabalho à minha família, que depositou confiança em minha luta. À namorada, que teve paciência em todos os momentos. Aos amigos, que sempre me apoiaram. Às canecas de café, que me mantiveram acordado nos momentos mais difíceis. E a todos que souberam compreender a minha ausência temporária.

AGRADECIMENTOS

Agradeço a todos os professores, que por todo esse tempo dividiram seus conhecimentos, sem os quais eu jamais chegaria a esta conquista.

À Etec Pedro D'Arcádia Neto, de Assis-SP, a qual proporcionou os primeiros conhecimentos que encorajaram esta jornada.

Ao professor Almir, o qual acompanha desde o primeiro ano e recomendou o objeto de estudo do presente trabalho, um tema atual e ainda pouco discutido.

A todos que, direta ou indiretamente, contribuíram para esta realização.

A experiência é uma lanterna dependurada nas costas, que apenas ilumina o caminho já percorrido.

Confúcio

RESUMO

Tendo sido desenvolvido pela DotCloud, o Docker é uma tecnologia de código aberto que revolucionou o ambiente de desenvolvimento de software, ao permitir criar, executar, testar e implementar aplicativos utilizando virtualização dentro de Containers. Com o Docker, o desenvolvedor tem a possibilidade de empacotar o software em uma imagem padronizada, de forma que esta contenha tudo que se fizer necessário para a execução do aplicativo desenvolvido: código, bibliotecas, ferramentas do sistema, configurações, dentre outros aspectos. Somando-se ao conceito de Virtual Machine, o Docker permite que a implementação de software se faça de forma rápida, confiável e estável, independente do ambiente em que ela é realizada.

Palavras chave: Docker, Virtualização, Linux Container

ABSTRACT

Developed by dotCloud, Docker is an open source technology that revolutionized the software development environment, allowing to create, run, test, and deploy applications by using virtualization in Containers. With Docker, the developer has the ability to package the software in a standard image so that it contains everything that is necessary for execution of the developed application: code, libraries, system tools, settings, all the things. Adding to the concept of Virtual Machine, the Docker enables the implementation of software and make it fast, reliable and stable, regardless of the environment in which it is held.

Keywords: Docker, Virtualization, Linux Container

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

FEMA Fundação Educacional do Município de Assis

IMESA - Instituto Municipal de Ensino Superior de Assis

CONTAINER: Contêiner

VMWARE: software que provê máquinas virtuais

VIRTUALBOX: software que provê máquinas virtuais

NGINX: Servidor Web e de Proxy

AUFS: *AnotherUnionFS*, sistema de arquivos

PHP: *Personal Home Page*, linguagem de programação

POSTGREE: Servidor de banco de dados.

MYSQL: Servidor de banco de dados

SUMÁRIO

Conteúdo

1. INTRODUÇÃO

No contexto de desenvolvimento de software, um dos requisitos para garantir a total funcionalidade dos programas ali produzidos é a certeza de que os mesmos terão sua total funcionalidade independentemente do sistema operacional, aplicações, hardware e demais variáveis a serem utilizadas no ambiente do cliente. Para que seja alcançado o objetivo da compatibilidade, faz-se essencial testar ao máximo o software produzido, simulando o maior número possível de condições que se façam presentes no local onde este será implantado.

Entretanto, não raramente, há condições que limitam o ambiente de testes em relação ao ambiente real do cliente. Complexidade do projeto, exigência de hardware específico, altos custos para uma simulação adequada, dentre outros fatores, exigem estratégias para que o desenvolvedor produza exatamente o que o cliente necessita, de forma a atender as expectativas deste.

Uma forma pertinente de solução para esta limitação do ambiente de testes surgiu com o conceito da virtualização de computadores, que consiste basicamente na criação de máquinas virtuais. Resumidamente, criar uma máquina virtual é a ação de utilizar um software que irá simular um novo computador, o qual será executado sobre o sistema operacional instalado na máquina física real. A virtualização, ao permitir a instalação das mais diferentes condições dentro de apenas uma maquina física, abre a opção de reproduzir os mais variados cenários, gerando assim a possibilidade da criação de um ambiente muito similar do cliente, ao mesmo tempo que gera redução de custos para a empresa desenvolvedora do software, a qual não precisará dispor de várias máquinas físicas com as mais diferentes configurações para representar o ambiente de aplicação dos softwares em desenvolvimento, tornando a implantação deste software mais eficiente e previsível para o desenvolvedor e cliente.

Sabemos que o ambiente de desenvolvimento de software não é o único beneficiado pelos conceitos de virtualização. Esta pode ser utilizada em muitos outros contextos, sendo desde um servidor físico hospedando as maquinas virtuais clientes, abrindo a oportunidade de uso de terminais em uma empresa de qualquer tamanho, até um simples ambiente educacional, para ensino de vários contextos de sistemas operacionais e suas aplicações.

Dentro do conceito de maquinas virtuais, temos o exemplo do Oracle VM VirtualBox [1], provavelmente o software de virtualização atual mais popular para simulação de ambientes rodando sobre outros ambientes. No VM Virtual Box podemos criar uma maquina virtual sobre o sistema operacional hospedeiro e nela instalar as mais variadas distribuições de Linux e Windows existentes no mercado, apenas para citar as duas possibilidades mais utilizadas no meio educacional.

Embora seja uma solução adequada para o uso da virtualização, ainda restam algumas arestas a serem aparadas nestes softwares. Uma delas é a performance. Máquinas virtuais possuem seu próprio sistema operacional, gerenciamento de memória, drivers de dispositivos, dentre outros devoradores de processamento e memória que nem sempre são necessários para a aplicação que se quer simular. São muitos procedimentos executados para alcançar o serviço desejado. E é aqui que o projeto Docker se sobressai.

O Docker segue o paradigma da virtualização sem propriamente o ser. Com a característica da utilização de containers, não é preciso executar todo um sistema operacional virtualizado, com todas suas dependências e seu kernel próprio, sobre o sistema operacional da máquina real para execução da aplicação desejada. O Docker irá ser executado sobre o sistema operacional instalado na máquina real, isolado desta, porém compartilhando o mesmo kernel, sendo por esta considerado apenas como um novo processo. E por sua característica de execução apenas como um processo isolado no sistema hospedeiro, o Docker elimina o carregamento de um novo sistema operacional por completo, com um novo kernel e dependências, como seria feito em outras formas de virtualização. Utilizando o conceito de Linux Containers para que apenas a aplicação que se faz necessária do sistema convidado seja carregada em cima do sistema hospedeiro, o Docker pouco impacta no desempenho da máquina física, tornando a operação mais rápida por compartilhar os mesmos recursos de hardware de forma segura e estável.

[1] Disponível em http://www.**virtualbox**.org

2. OBJETIVOS

1.2 OBJETIVOS GERAIS

O objetivo deste projeto é apresentar esta nova forma de virtualização, denominada Docker, com o intuito de descobrir as possibilidades que podem ser atingidas por este recente conceito.

A partir das pesquisas realizadas, somadas aos resultados gerados por este trabalho, será possível avaliar as reais vantagens da aplicação do Docker em complemento, ou por vezes substituição, das aplicações de virtualização utilizadas em ambientes de desenvolvimento de software.

1.2.2 OBJETIVOS ESPECÍFICOS

Temos também como objetivo no presente trabalho apresentar comparativos de desempenho em relação à outras maquinas virtuais existentes, tal qual os *hypervisors* VirtualBox e VMWare, visando de forma especial gerar dados para os seguintes pontos a serem verificados:

- ✓ Desempenho do computador: o quanto este é afetado por sistemas como o Virtual Box, VMWare e Docker;
- ✓ Praticidade na utilização: o tempo necessário, assim como acessibilidade, para o desenvolvedor acessar a maquina virtual e aplicação desejada, de forma a realizar os testes necessários.
- ✓ Analisar os resultados alcançados com a utilização da virtualização com Docker, VirtualBox e VMWare.
- ✓ Especificar um estudo de caso:
- ✓ Modelagem de uma situação apresentada;
- ✓Realização dos testes práticos;
- ✓ Documentação dos testes;

3. JUSTIFICATIVA

O uso de máquinas virtuais revolucionou o cenário dos ambientes de desenvolvimento de software. O Docker surgiu para ampliar essa revolução e, em se tratando de um conceito recente, existem dúvidas a respeito do mesmo.

O presente trabalho vem demonstrar que, independentemente da configuração existente na máquina disponibilizada para o ambiente de testes, o conceito de Linux Containers utilizado pelo Docker facilitará em muito a vida da equipe de desenvolvimento ao compartilhar dos mesmos recursos de hardware entre máquinas física e virtual, tornando todo o processo de testes menos dependente de um hardware poderoso, mais prático e acessível.

4. MOTIVAÇÃO

Sendo utilizado por grandes corporações de renome como HP, Microsoft, Twitter, Spotify, dentre outras, esta pesquisa busca verificar as reais vantagens da utilização do Docker no ambiente de desenvolvimento, assim como se esta é efetivamente uma solução viável, averiguando em especial as diferenças de desempenho deste em relação aos softwares de virtualização já existentes no contexto atual.

5. PERSPECTIVA DE CONTRIBUIÇÃO

O Docker apresenta um grande potencial de crescimento em utilização pela comunidade de desenvolvimento como um todo - grandes nomes do meio, como a Microsoft, dedicam atenção especial ao projeto. Entretanto, por ser um conceito recente, ainda restam dúvidas sobre a utilização do mesmo.

Ao término deste trabalho, espera-se que os números apresentados pelos relatórios gerados sejam de utilidade para toda a comunidade de desenvolvimento, apresentando esta ferramenta como uma possibilidade para desenvolvedores e apreciadores da arte da programação.

Uma vez concluído, pretende-se difundir a pesquisa por meio de artigos, fóruns de desenvolvimento, comunidades de desenvolvedores, dentre outros meios de troca de informação, visando promover e compartilhar os conhecimentos e resultados alcançados, de forma a incentivar o projeto e promover sua divulgação.

6. ESTRUTURA DO TRABALHO

Pretende-se ter como base a seguinte organização de tópicos:

- **Capítulo 1 - Introdução:** aqui apresentando o tema, objetivos, justificativa e motivação para o desenvolvimento deste projeto de pesquisa.
- **Capítulo 2 - Aspectos gerais sobre virtualização:** Aqui pretende-se expor os conceitos de virtualização, seu histórico, evolução e situação atual.
- **Capítulo 3 - Conceitos do Docker:** Neste ponto, serão demonstrados
- conceitos do Docker, Linux Container, Docker Hub e demais conceitos a serem apresentados de forma a explicar o funcionamento deste.
- **Capítulo 4 - Apresentação dos testes teóricos:** Serão aqui demonstrados testes anteriores realizados e apresentados em fóruns e publicações especializadas.
- **Capítulo 5 - Análise Prática:** Neste capítulo, pretende-se demonstrar a metodologia que será aplicada ao realizar os testes de desempenho, e então realização destes testes seguindo os parâmetros pré-determinados.
- **Capítulo 6 - Conclusão:** O ponto onde serão comparados e demonstrados os resultados, buscando encontrar os pontos onde se faz vantagem e onde se faz viável a utilização do Docker.
- **Referências:** Toda a bibliografia utilizada na pesquisa.

7. ASPECTOS GERAIS DA VIRTUALIZAÇÃO

Para entendermos a ideologia aplicada pelo Docker, precisamos primeiramente conhecer e entender os conceitos de virtualização. Considera-se a década de 1960 como o início das pesquisas sobre máquinas virtuais, tendo como princípio o IBM M44/44X, que ainda não possuía uma virtualização completa, e tendo como ponto de sucesso o CP40, da mesma International Business Machines Corporation (IBM), que conseguiu atingir o objetivo da virtualização completa, a qual se dá quando temos hardware e software compatíveis com a virtualização. Neste primeiro momento, a virtualização tinha o objetivo específico de compartilhar sistemas de mainframes, na época fisicamente grandes, de valor elevado e com alto custo de manutenção para garantir seu funcionamento.

Anteriormente ao advento da virtualização, a aquisição de novos servidores físicos exigia um novo espaço físico na sala do servidor, da mesma forma que uma adequação do local em relação ao circuito elétrico de alimentação deste servidor, equipamentos para refrigeração do mesmo, ajustes no cabeamento de rede, horas de trabalho de uma equipe na montagem e configuração físicas do ambiente, assim como sua manutenção por toda a sua vida útil.

Via de regra, estes equipamentos não terão total utilização do seu potencial. Excetuando-se momentos de pico, grande parte da capacidade de processamento e memória de um servidor permanecem sem uso por períodos prolongados. E, estando ociosos, ainda que se considere que a tecnologia atual reduz o consumo de energia nestas condições, este servidor com baixa utilização de recursos ainda está ocupando espaço físico, utilizando energia elétrica, fazendo uso de sistemas de refrigeração, apenas para se manter ocioso. Mantém os custos operacionais para não realizar tarefa alguma.

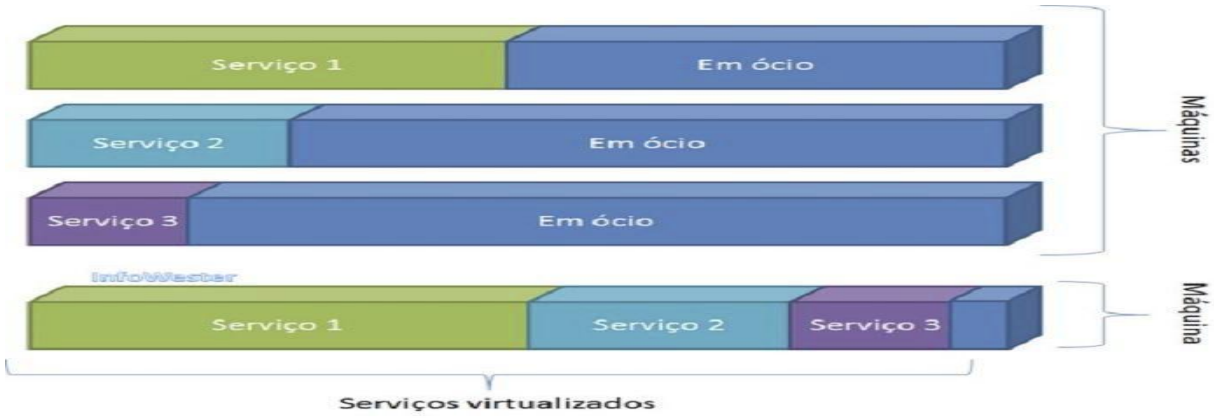

Figura 1: Demonstração de utilização de hardware em três serviços distintos, com e sem a virtualização.
Fonte: http://www.docker.com

Com a aplicação dos conceitos de virtualização, abre-se a possibilidade de que toda a capacidade de um servidor seja utilizada por vários sistemas. Mudanças e ampliações em um ou outro destes sistemas não necessariamente exigem a aquisição e configuração de novos equipamentos físicos. Utilizando-se da capacidade total de um servidor, iremos eliminar o custo que seria causado por gastos adicionais em novas aquisições de hardware.

Embora cercado de vantagens, podemos observar que apenas no final de década de 1990 foi introduzido no mercado um software de fácil utilização para virtualização, o VMware Virtual Platform para a plataforma x86, arquitetura dominante desde a década de 1980. Percebemos que o conceito criado na década de 1960 foi deixado de lado pelo mercado por um tempo razoável. Por outro lado, observando o cenário atual, notamos que a virtualização vingou de tal forma que vemos o princípio bastante difundido de Virtual Machine sendo ampliado e aplicado a várias áreas, sejam estas próximas ou mesmo distantes dos iniciais mainframes, abrangendo inclusive outras formas que podem, ainda que em outro contexto, beneficiar-se dos conceitos da virtualização. Exemplo disso pode ser percebido na Java Virtual Machine, um software que permite aos programas que foram escritos na linguagem de programação Java serem executados sobre qualquer plataforma compatível com esta máquina virtual - incluindo aqui computadores, smartphones, tablets e mesmo alguns itens distantes dos poderosos servidores da IBM, objetivo inicial destas pesquisas, tal qual simples eletrodomésticos que possuímos em nossas casas. A virtualização permitiu que sistemas computacionais fossem aplicados em benefício de projetos maiores, como na intercomunicabilidade da internet das coisas.

8. A VIRTUALIZAÇÃO NA PRÁTICA

Consideremos o funcionamento de um computador padrão x86, o mais comum no mercado atual, antes de apresentarmos o ambiente a ser aplicada a virtualização. Regra geral, o computador é um conjunto de dispositivos de hardware, preparado para receber um sistema operacional que irá interagir diretamente com este hardware, e sobre este sistema operacional serão instalados os mais variados aplicativos, de forma a tornar válido o uso do computador.

Diferente deste primeiro cenário, a virtualização ocorrerá quando houver o Monitor de Máquina Virtual (Virtual Machine Monitor, VMM) onde uma ou mais Máquinas Virtuais são criadas, sendo que cada unidade virtualizada será capaz de ter seu sistema operacional e aplicações instaladas, que irão funcionar sobre um computador virtual, existente apenas no contexto de software, sobre um hardware real, físico, com sistema operacional próprio. As características físicas deste hardware são simuladas pelo software, que o faz através de uma camada denominada abstração.

Neste contexto, os recursos do hardware real serão divididos ou compartilhados entre mais de um ambiente. Estes podem ou não realizar interações entre si, da mesma forma que podem simplesmente desconhecer a existência uns dos outros.

9. HYPERVISOR

Quem gerencia essa virtualização é a Hypervisor. Também conhecido como Monitor de Máquina Virtual (Virtual Machine Monitor, ou VMM) trata-se de um software que cria uma "camada" de software entre o hardware e o sistema operacional instalado na máquina física. O Hypervisor é o software que irá controlar e gerir o acesso dos sistemas operacionais aos recursos de hardware, recebendo os sistemas operacionais virtuais e gerenciando a divisão de processamento, memória e outros recursos de hardware entre estes, mantendo um sistema operacional virtualizado totalmente separado e independente de outro instalado na mesma máquina física. O software terá permissões para interagir diretamente no hardware com a finalidade de realizar ações como, entre outras, a comunicação direta com o processador,

memória, gerenciamento de instruções e acesso. Este modo de supervisor é necessário ao Hypervisor para executar (ou simular) as chamadas do sistema operacional, afim de que sejam alocados recursos organizados que serão divididos entre as máquinas virtuais, e mesmo determinar a ordem em que as requisições de hardware irão ocorrer.

A imagem a seguir exemplifica essa organização.

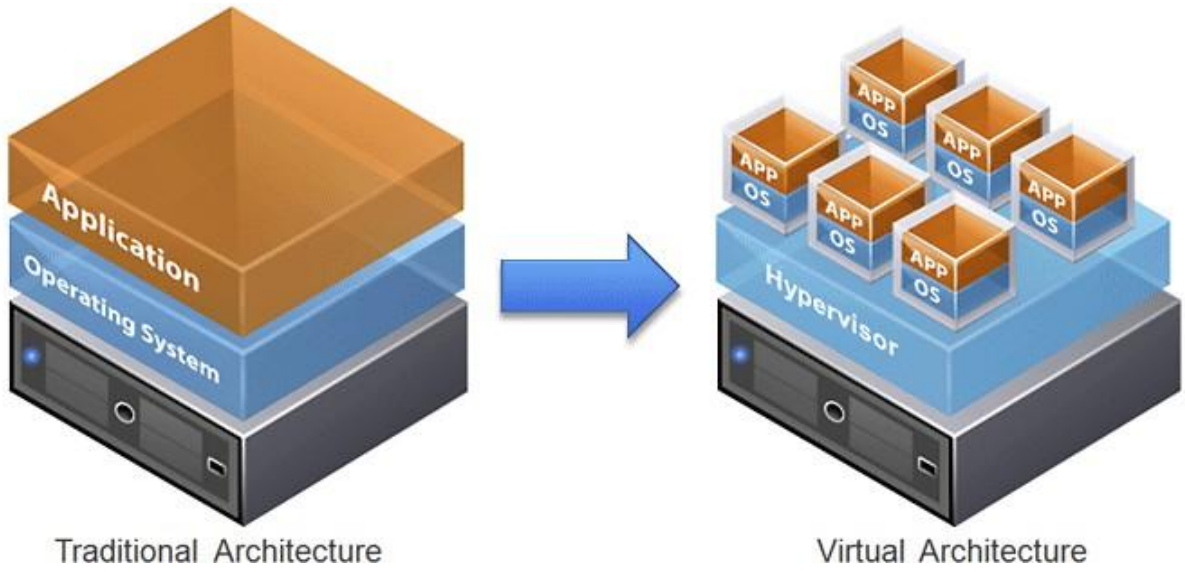

Figura 2: Ilustração de hardware com um sistema operacional e um segundo com hypervisor
Fonte: http://www.docker.com

Existem dois tipos de hypervisor. A mais conhecida no meio educacional é o tipo 2, estudado em aulas, que é aquela onde o hardware estará executando um sistema operacional diretamente no computador real, como um computador comum, com apenas um sistema operacional instalado. Podemos usar como exemplo um computador com o Microsoft Windows. Este irá hospedar uma VMM, o software Monitor de Maquina Virtual, podendo ser utilizado como exemplo o Oracle VirtualBox, sobre o qual temos uma máquina virtual (ou várias) simulando um computador real, onde iremos instalar outros sistemas operacionais, tal qual o Linux Ubuntu. Para todos os efeitos, este Linux Ubuntu instalado neste Oracle VirtualBox "enxergaria" este monitor de máquina virtual como sendo um computador real, com hardware próprio e exclusivo.

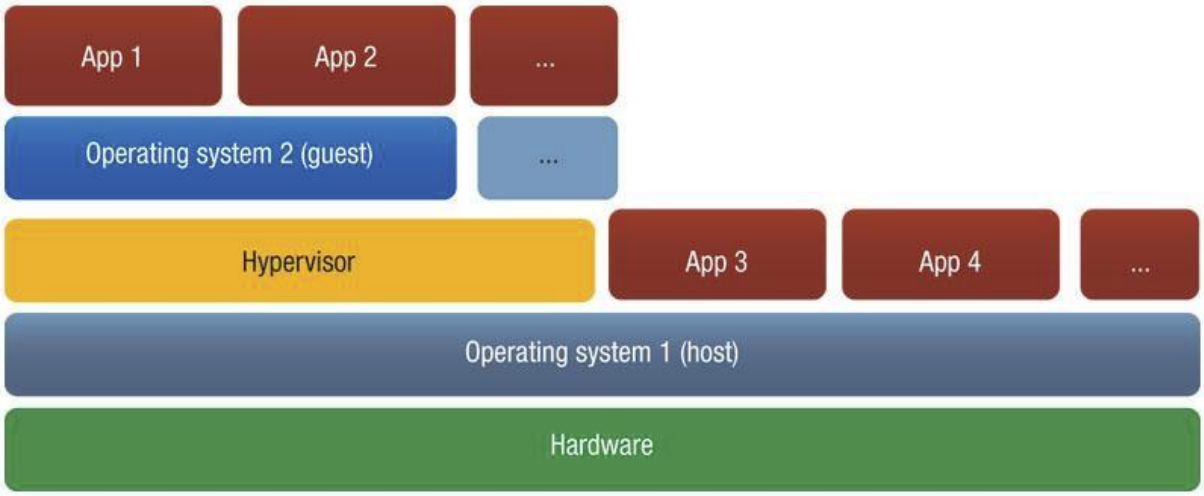

Figura 3: Exemplo de virtualização do tipo 2, com um sistema operacional executando o Hypervisor.
Fonte: http://www.docker.com

Outra possibilidade existente é a virtualização de tipo 1, um contexto onde o hardware não inicia sobre um outro sistema operacional que irá hospedar o software da Máquina Virtual, e sim a própria plataforma de virtualização será inicializada diretamente no hardware, sem a dependência de um sistema operacional para executá-la. Eliminando uma camada, há um ganho em desempenho das máquinas virtualizadas. Este é um contexto muito utilizado em servidores empresariais, até porque aqui temos primordialmente soluções comerciais, tais quais Microsoft HyperV, VMware ESX/ESXi, dentre outras.

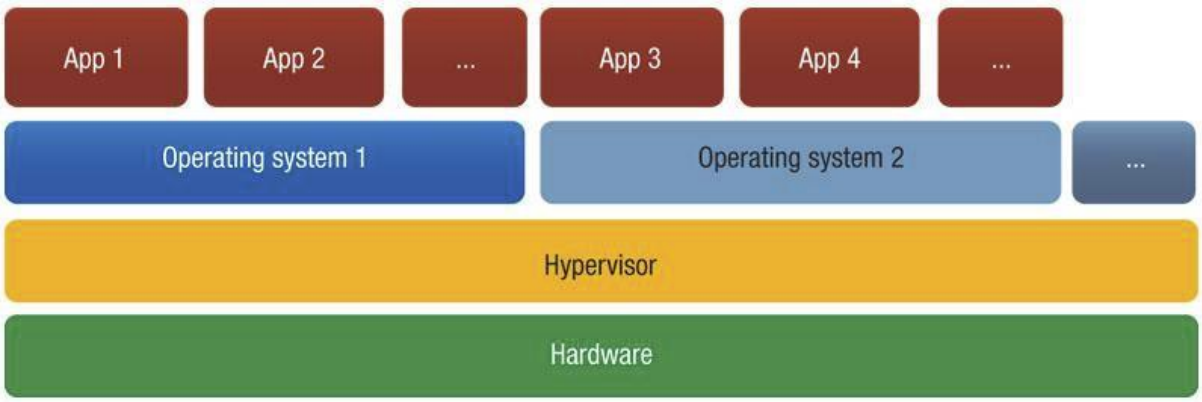

Figura 4: Exemplo de virtualização do tipo 1, com o Hypervisor executando diretamente no hardware
Fonte: http://www.docker.com

As vantagens do tipo 1 são óbvias, tanto no que se refere ao desempenho das Maquinas Virtuais, que não serão sistemas operacionais sendo executados sobre outro

sistema operacional, consumindo duas vezes mais processamento e memória, como na estabilidade, já que no tipo 2, se uma Virtual Machine é paralisada, todas estas Virtuais Machines instaladas sobre ele também terão problemas. No tipo 1, apenas a Virtual Machine com problema seria finalizada, mantendo-se todas as outras isoladas, em normal funcionamento.

Embora o tipo 2 tenha algumas desvantagens em desempenho e possa apresentar problemas de estabilidade, sendo assim o tipo menos utilizado, temos a máxima de que o melhor para a maioria não significa exatamente o melhor para todos. De fato, o ambiente de servidores é beneficiado pela virtualização de tipo 1, entretanto, outros ambientes são grandemente beneficiados pelo conceito de virtualização do tipo 2, como os utilizadores da Virtual Java Machine ou os entusiastas do objeto de estudo do presente trabalho: o Projeto Docker.

10. CHROOT

O processo *chroot* se trata de um recurso do sistema operacional Linux que permite, com isolamento, manter uma distribuição Linux secundária instalada dentro de outra distribuição primária. Esta segunda instalação é alocada em uma pasta do sistema, e não em partição, como seria o comum. Ademais, pode esse sistema secundário executar programas com um desempenho superior ao de uma Máquina Virtual em seu conceito primário.

Em razão da qualidade elevada na característica de criação e gerenciamento de usuários, grupos e permissões, os sistemas Unix e Linux tem a capacidade de permitir ou negar acesso aos componentes que não precisem ser compartilhados. Isso mantém o sistema secundário isolado do sistema principal, compartilhando apenas o que se faça essencial para seu funcionamento.

Tendo em vista que os processos que estivessem sendo executados dentro do ambiente do *chroot* seriam incapazes de ler processos externos, esta parece ser uma boa solução para a virtualização em sistemas Unix e Linux. Entretanto, caso seja utilizado o usuário *ROOT* dentro do *chroot*, mesmo que inicialmente isolado do sistema principal, ele tem a possibilidade de conseguir acesso ao mesmo. Sendo assim,

embora seja uma solução para testes de software, ainda não é a mais adequada para casos que possam comprometer a segurança do sistema. De uma forma primária, o *chroot* é a base para os Linux Containers.

11. LINUX CONTAINERS

Um Container, no ambiente Linux, assemelha-se muito ao conceito do chroot. Sendo executado no mesmo sistema operacional principal, sua função é basicamente isolar componentes do sistema, incluindo aqui o isolamento de rede. Aqui, no entanto, temos uma maior segurança oferecida pelos Linux Containers, já que o nível do isolamento oferecido pelo conceito dos Containers garante que o acesso esteja realmente desligado de todo e qualquer outro recurso que possa ser utilizado para controlar ou acessar o sistema principal, ou mesmo outros instalados na mesma máquina. Ademais, o container terá seu próprio PID (Process Identifier). Um PID é o um número gerado pelo sistema Linux para cada um dos processos que estejam ativos. Qualquer novo processo receberá um novo número, o que garantirá que cada processo tenha seu próprio PID, o qual não poderá ser atribuído a outro processo ou container simultaneamente.

Isso capacita os Linux Containers a gerenciarem de forma competente máquinas virtuais dentro do sistema hospedeiro, permitindo a divisão e gerência dos recursos de forma mais eficiente em questões de desempenho que o realizado pela virtualização tradicional. Por compartilhar o mesmo kernel do sistema operacional hospedeiro e , o Linux Container tem impacto mínimo em processamento e memória.

Conquanto ofereça um ambiente virtualizado, o Linux Container não é a máquina virtual em si. Este irá fornecer o ambiente com uma CPU própria, memória, rede, com todos esses parâmetros sendo passíveis de configuração pelo desenvolvedor, porém ainda diretamente ligado ao kernel hospedeiro. Sobre este ambiente ligado ao kernel do sistema hospedeiro, mas isolado deste, teremos a instalação das bibliotecas e todas as dependências que se façam necessárias, permitindo rodas vários sistemas operacionais em um único hospedeiro.

As diferenças entre um Sistema Operacional instalado em uma máquina física ou virtual e o Container são as seguintes:

O Sistema Operacional possui:

- Bootfs: Boot FileSystem;
- Bottloader e Kernel;
- Rootfs: Root Filesystem;
- Restante dos arquivos do sistema.

A passo que o Linux Container apenas

- Bootfs: (LXC, aufs/btrfs), utilizando o mesmo Kernel do hospedeiro.
- Rootfs: Sistema de arquivos.

Podemos assim notar que os Linux Containers vieram para agregar novas possibilidades à virtualização, que não busca substituir, mas complementar a utilização de Máquinas Virtuais.

O Linux Container é a base para o Docker, sendo as vantagens percebidas na utilização muito similares:

- Velocidade de execução;
- Boot da máquina sendo efetuado em segundos;
- Economia de recursos da máquina hospedeira;
- Processos executados internamente ao container são enxergados pelo hospedeiro como um processo no próprio hospedeiro;
- Pode-se utilizar muitos containers sem prejuízo de desempenho.

12. VAGRANT

Embora não diretamente relacionado com o projeto Docker, o Vagrant possui algumas características que o tornam precursor deste. Este surgiu com o objetivo de resolver problemas provenientes das diferenças de versões entre sistemas entre os

desenvolvedores e simplificar o trabalho com Máquinas Virtuais, facilitando o processo de instalar e configurar maquinas virtuais.

O Vagrant não é a Máquina Virtual propriamente, sendo apenas uma forma de automatizar a configuração das mesmas. Com o Vagrant, você cria uma Máquina Virtual completa e personaliza esta da forma que definir ser necessária, com os pacotes e versões de aplicativos necessários para o desenvolvimento, como PHP, MySql, PostGreSQL, dentre outros necessários para o projeto onde o desenvolvedor estiver trabalhando. Terminada essa configuração, a Maquina Virtual é enviada para um servidor ou para a internet, e distribuída entre toda a equipe de desenvolvimento.

Isso irá assegurar que a equipe por completo tenha exatamente a mesma versão de todos os aplicativos, o que garante o fim do problema da compatibilidade e diferenças entre sistemas e computadores entre desenvolvedores de uma equipe. Todos terão exatamente a mesma Máquina Virtual, espelhada entre a equipe, reduzindo ou mesmo anulando problemas com incompatibilidades que atormentam equipes de desenvolvimento de software, acabando com o conhecido problema do "em minha máquina funcionou".

O Vagrant trabalha com o conceito de boxes, que são imagens pré-configuradas de sistemas operacionais que irão servir como base para a construção de novas Máquinas Virtuais. Uma vez geradas, essas boxes podem ser enviadas para um repositório, o Vagrant Cloud. Este repositório foi adicionado na versão 1.5.0 do Vagrant, e se trata de um serviço de compartilhamento de imagens da própria Hashicorp, desenvolvedora do sistema. Desta forma, boxes podem ser adicionados em outros computadores com a simples instalação do software do Vagrant e a utilização de um comando específico, que irá baixar do repositório a imagem da máquina Virtual já com as configurações desejadas.

13. DOCKER

Somando os paradigmas utilizados pela virtualização, *chroot*, Linux Containers e Vagrant, o Docker surgiu em 2013 para concentrar o melhor dos recursos característicos de todos estes conceitos citados anteriormente, produzindo assim uma solução leve e eficaz para atender a necessidade dos desenvolvedores e clientes.

O Docker é apresentado como uma projeto de licença open source que possibilita ao desenvolvedor a criação, execução, testes e implementação de aplicativos desenvolvidos com o uso de Linux Containers. Este container armazenará o software com as configurações padronizadas pelo desenvolvedor, que disponibilizará um mesmo pacote contendo o código, execução, ferramentas, bibliotecas e tudo que se faça necessário no contexto a ser atingido.

Embora o conceito de virtualização esteja presente, o Docker não pode ser exatamente considerado uma máquina virtual, por não seguir a filosofia de virtualização tradicionalmente utilizada. Se considerarmos um ambiente virtualizado comum, este terá um hypervisor contando um Sistema Operacional completo (ou vários), cada qual com seu kernel e bibliotecas próprias, interagindo ou não entre eles.

No Docker isolaremos os recursos do Sistema Operacional hospedeiro, porém este irá fornecer o seu kernel para o container Docker. Esta possibilidade se torna viável em razão de o Docker utilizar o conceito de Linux Containers, explicado em capítulo anterior. Por utilizar este conceito, será executado de forma mais leve e rápida, consumindo menos recursos de hardware. Iniciar um container é algo como iniciar um novo processo isolado no sistema operacional hospedeiro, onde o próprio Docker torna-se um processo dentro do Sistema, sendo este diretamente instalado sobre a máquina física. Essa é a principal razão de a inicialização de uma máquina Docker se mostrar tão rápida em comparação com máquinas virtuais com hypervisor.

É uma solução voltada para administradores e desenvolvedores de sistemas, obviamente não sendo exclusivamente a estes. A dotCloud, atualmente nomeada Docker, é a desenvolvedora do projeto. É distribuído sob a definição de Open Source, o que garante que seu código fonte será disponibilizado e licenciado com uma licença de código aberto no qual o usuário terá o direito de estudar, modificar e distribuir o software sem custos para qualquer um, e abrangendo qualquer finalidade.

Em especial por esta característica, a sua imagem Docker pode ser customizada e distribuída livremente, sem restrições de licença, da mesma forma que o desenvolvedor pode alterar as imagens para que atenda suas necessidades, com a mesma liberdade oferecida pelos sistemas Linux.

O Docker foi escrito na linguagem de programação Go, que é uma linguagem de programação de alto desempenho desenvolvida pelo Google e definida com licença de código livre em novembro de 2009. Segue uma sintaxe similar ao C, conquanto contenha algumas particularidades, sendo uma destas facilitar o propósito de criação e administração de ambientes isolados.

13.1 DOCKER HUB

Uma vez customizadas, estas imagens podem ser mantidas na máquina, colocadas em um servidor local ou mesmo há a possibilidade de distribuí-las no serviço de repositório Docker Hub[1], um servidor de registro em nuvem mantido pela DotCloud. Este serviço requer um cadastro, que é gratuito, que uma vez efetuado permite acesso ao repositório onde o usuário pode construir, testar e armazenar suas imagens. Ademais, o Docker Hub permite que se faça buscas por imagens já criadas por outros usuários de forma que, uma vez encontrada a imagem com o serviço que se deseja executar, apenas se faz necessário realizar o download da mesma e executá-la localmente.

Essa busca por imagens já existentes pode ser feita de duas maneiras. A mais simples consiste em, no terminal de comando do computador local, executar o comando abaixo.

- *docker search*

Podemos exemplificar a busca por imagens Ubuntu. Neste caso, utiliza-se "*docker search ubuntu*", onde será retornada uma lista de imagens que possuam esta palavra chave procurada.

A outra forma é acessar o site do Docker Hub[1] e, no campo Search, digitar os termos relacionados à imagem que se deseja obter.

[1] Docker hub: http://hub.docker.com

Há também os repositórios oficiais. Trata-se de repositórios certificados, mantidos por empresas fornecedores Linux (Canonical, Oracle, Red Hat), as quais mantém imagens de seus sistemas base, o que garante confiança de que aquela é uma imagem otimizada e atualizada pelos próprios desenvolvedores destas.

13.2 VIRTUAL MACHINE E DOCKER

O desenvolvedor que trabalha em múltiplos projetos que possuam dependência um com outro, em algum momento precisará isolar os ambientes, de forma a evitar conflitos com versões de bibliotecas, bancos de dados, dentre outros. O Vagrant era até então utilizado para a criação de ambientes de desenvolvimento portáveis, que possam ser distribuídos e que tivessem a garantia de espelhamento. Entretanto, Máquinas Virtuais possuem cada qual um distinto Sistema Operacional, seu próprio Kernel, gerenciamento de memória, processos, uso de recursos, de forma que ao se imaginar uma situação onde são mantidos vários ambientes ativos, estes consumiriam, se não todo, ao menos uma grande parte do processamento.

Compartilhando os recursos do Sistema Operacional hospedeiro, o Docker garante que muitas instâncias possam ser virtualizadas sem impactar bruscamente na performance do hardware. A imagem abaixo demonstra este menor uso de recursos.

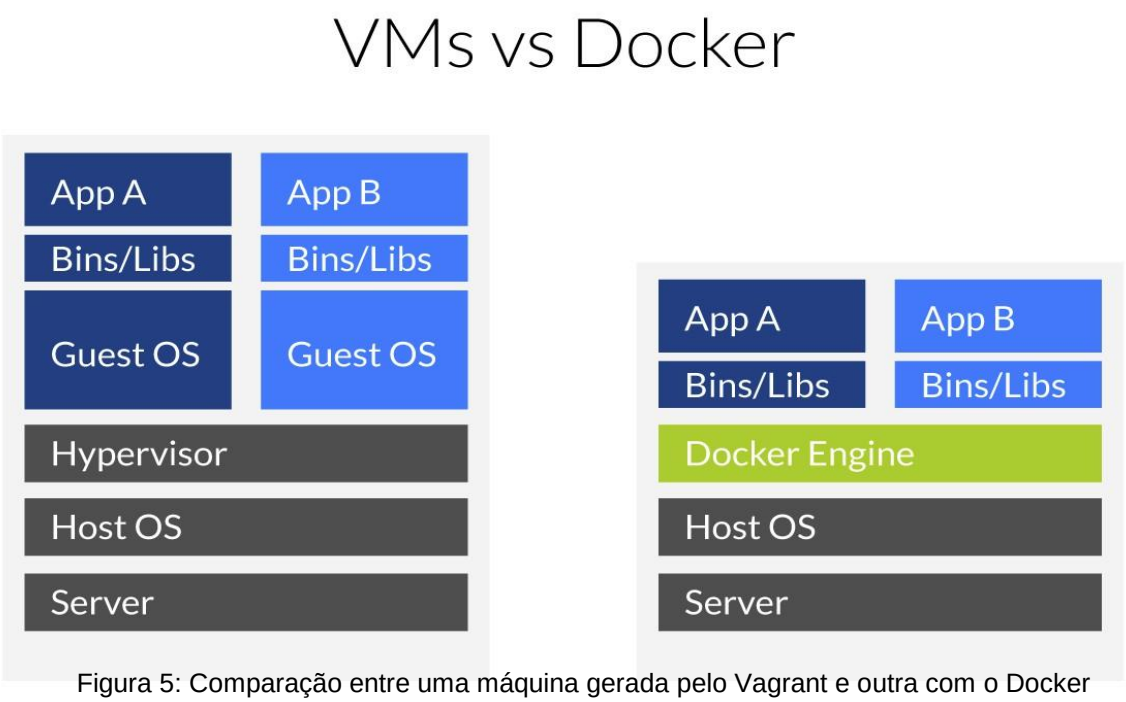

Figura 5: Comparação entre uma máquina gerada pelo Vagrant e outra com o Docker

Fonte: http://www.docker.com

O Docker oficialmente ainda não possui interface gráfica, de forma que sua utilização se dá por comandos no terminal.

Os comandos básicos para início do uso do Docker estão listados abaixo:

- *docker pull [nome da imagem]* - realiza o download da imagem Docker desejada;
- *docker images* - exibe uma lista com as imagens disponíveis na máquina, as quais já foram feitas o download e estão localmente instaladas;
- *docker run [nome da imagem]* - inicializa a imagem escolhida;
- *docker ps* - exibe uma lista com os containers em execução;
- *docker ps -a:* visualizar todos os containers, mesmo os que não estão sendo executados;
- *docker exec [id do container] [comando]* - executa comandos no container, não estando logado dentro deste container;
- *docker commit* - utiliza o container que está sendo executado e gera uma imagem do mesmo, a qual pode ser distribuída;
- *docker inspect* - apresenta as propriedades do container que está sendo executado;
- *docker rm -f <nome_container>* - Remover um container;
- *docker rm -f $(docker ps -a -q)* - Remover TODOS os containers;
- *docker rmi -f <nome_imagem>* - Remover uma imagem.

Outros comandos serão apresentados, em momento oportuno.

14. INSTALAÇÃO DO DOCKER EM UM UBUNTU 14.04 LTS

Inicialmente, seguindo as recomendações dos desenvolvedores, precisamos de uma máquina com uma distribuição Linux instalada, sendo essencial que esta tenha a arquitetura x86, 64 bits.

Não é citada uma única distribuição recomendada, sendo possível instalar em um computador com o sistema operacional Linux Ubuntu, CentOS, Arch linux, Debian, Red Hat, openSUSE, dentre outros. Faz-se essencial o kernel 3.10 ou mais recente do Linux, conforme recomendação dos desenvolvedores do Docker, disponível em [*https://docs.docker.com/engine/installation/linux/ubuntulinux*].

> *"O Docker requer uma instalação de 64 bits, independente da sua versão do Linux. Além disso, seu kernel deve ser no mínimo o 3.10. Versões mais recentes que o kernel 3.10 também são suportadas."*

Foi escolhido o Linux Ubuntu 14.04 LTS como sistema operacional em razão da extensa documentação e suporte a este, porém em qualquer outra distribuição os resultados seriam idênticos, com alguma variação podendo surgir apenas nos comandos utilizados para instalação do Docker.

O Ubuntu é um Linux baseado no Debian, patrocinado pela empresa Canonical Ltd., que é a responsável por manter o suporte e atualizações ao mesmo. A equipe desenvolvedora do Ubuntu tem como objetivo oferecer um sistema que qualquer pessoa possa utilizar sem dificuldades, abrangendo inclusive o usuário iniciante em informática ou mesmo em sistemas Linux.

Considerando que já se tenhamos a máquina física com o sistema operacional Ubuntu instalado, segue-se a recomendação de executar o comando abaixo para atualização de pacotes do sistema Ubuntu, afim de obtermos as versões mais novas, estáveis e compatíveis com os mais recentes aplicativos.

- *sudo apt-get update*

Onde o comando "sudo" dará a permissão de administrador do sistema, necessária para a instalação de pacotes, o "apt-get" é o comando responsável pelo download e instalação dos pacotes, e o "update" irá verificar as atualizações. Este é o único passo de configuração no Linux Ubuntu 14.04 LTS na máquina hospedeira para seguirmos com a instalação do Docker. Nada mais é necessário.

Iniciamos então a verificação da versão do kernel do Linux que temos instalado na máquina, utilizando um comando no terminal. O Ubuntu 14.04 utilizado aqui possui a versão 3.19 do kernel. Para averiguar qual é a versão do kernel, abra o terminal de comando e digite no console:

- *uname -r*

Deve-se verificar a arquitetura do sistema operacional, uma vez que existe a exigência de que o sistema operacional Hospedeiro utilize a arquitetura de 64 bits para correta instalação do Docker. Existem maneiras de que a instalação se faça em sistemas operacionais de 32 bits, porém estes não são suportados oficialmente. Para verificar a arquitetura do seu Linux, digite no terminal de comando:

- *uname -m*

14.1 INSTALANDO O DOCKER

Seguindo os parâmetros indicados pela bibliografia utilizada, instalaremos o Docker em um computador com o sistema operacional Linux Ubuntu 14.04 LTS, 64bits, utilizando o comando abaixo:

- *$ curl https://get.docker.com > /tmp/install.sh*
- *$ cat /tmp/install.sh*

Utilizando-se do comando “cat”, podemos verificar os detalhes da imagem. Estando tudo conforme o esperado, continuaremos a instalação com:

- *$ chmod +x /tmp/install.sh*
- *$ /tmp/install.sh*

Isso inicializa o script de instalação do Docker. Realizado o procedimento a instalação será executada e finalizada. Uma forma simples de verificar se o Docker foi instalado é verificando a versão do mesmo através do comando:

- *docker version*

Este comando resultará na seguinte mensagem:

beto@ubuntu:~$ docker version
Client:
Version: 1.11.2
API version: 1.23
Go version: go1.5.4
Git commit: b9f10c9
Built: Wed Jun 1 21:47:50 2016
OS/Arch: linux/amd64

Server:
Version: 1.11.2
API version: 1.23
Go version: go1.5.4
Git commit: b9f10c9
Built: Wed Jun 1 21:47:50 2016
OS/Arch: linux/amd64

Uma outra forma de verificar se o docker está instalado e sendo executado é através do comando que resultará na seguinte saída:

- *docker ps*

beto@ubuntu:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

Este comando traz informações a respeito dos containers que estão sendo executados, na seguinte ordem.

Container ID: a identificação dos containers que estão sendo executados;
Image: a imagem que este container estará utilizando;
Command: o comando que o container estará executando;
Created: o tempo que este comando foi criado;
Status: se ele está sendo executado ou parado.
Ports: informações referentes às portas utilizadas;
Names: o nome do container.

Para realizarmos o primeiro teste de executar uma imagem Docker, utilizaremos uma imagem do Debian disponibilizada no Docker Hub para criar um novo container, que irá exibir no terminal a mensagem “Hello, World”.

beto@ubuntu:~$ docker run debian echo "Hello World"
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
5c90d4a2d1a8: Pull complete

Digest:
sha256:8b1fc3a7a55c42e3445155b2f8f40c55de5f8bc8012992b26b570530c4bded9e
Status: Downloaded newer image for debian:latest
Hello World
beto@ubuntu:~$

A mensagem “*Unable to find image 'debian:latest' locally*” indica que não havia esta imagem do Debian no sistema, e foi preciso fazer o download da mesma. O Docker verifica o repositório online e, em “*latest: Pulling from library/debian*”, realiza o download da imagem, provisiona e inicia um container, executa a imagem e o comando requisitado *echo “Hello World*” é apresentando no terminal, através da mensagem “Hello World”. Após, ele encerra o container e voltamos para nossa máquina.

Afim de desconsiderar o tempo de download, que só é realizado uma vez, executaremos o comando novamente e percebemos que toda essa ação leva apenas cerca de um segundo para provisionar o container, iniciar o Debian, executar o comando echo, apresentar em tela e encerrar o container.

```
beto@ubuntu:~$ docker run debian echo "Hello World"
Hello World
beto@ubuntu:~$
```

Quanto tempo levaríamos para iniciar uma máquina virtual com o Debian já previamente instalado, executar o comando que apresenta uma mensagem em tela e finalizar a mesma? Não muito tempo, porém bem mais que um segundo.

Para verificarmos a imagem instalada, podemos utilizar o comando “*docker info*”, que irá apresentar o número de containers criados, executando, pausados ou parados. Um container “*created*” é o que foi inicializado com o comando “*docker create*”, mas ainda não foi inicializado. O status “*stopped*” indica que há um container, porém o mesmo não possui processos sendo executados. Este container pode ser reinicializado com o comando “*docker start*”, sem que seja necessário realizar novo download da imagem.

Faz-se essencial citar, neste ponto, que podemos utilizar uma imagem para criação de nosso container, e através do "*commit*" nesse container, podemos criar uma imagem. Embora entrelaçados, os conceitos de imagem e containers são distintos.

O Docker faz o uso de imagens para realizar a criação de containers. Por esta razão, há uma certa confusão entre "imagem" e "container". Um container não é uma imagem. Regra geral, a imagem irá conter um ambiente linux mínimo e será utilizada para um microserviço específico, podendo ter como exemplo um servidor web, um servidor de banco de dados, ou como no exemplo citado, apenas apresentar uma mensagem em tela. Quando o Docker executa a imagem, ele irá criar um novo container no sistema operacional hospedeiro, para que esta imagem esteja disponível dentro deste container, totalmente isolada de todo o restante do sistema operacional da máquina hospedeira.

14.2 INSTALAÇÃO DO WEBSERVER NGINX DENTRO DO CONTAINER

Para demonstrar um exemplo além da simples mensagem "Hello World" no terminal de comando, podemos efetuar a instalação de um servidor de web NGINX, que é utilizado como servidor HTTP e servidor de E-mail, sendo um dos mais populares servidores Web na atualidade. Este servidor irá ser executado dentro de um container com a imagem do Linux Ubuntu 14.10, sobre o qual será instalado o servidor NGINX, e este ao final irá apresentar uma página default do servidor.

O sistema operacional hospedeiro não possui a imagem, de forma que se faz necessário realizar o download da mesma. Estando logados como *root* no sistema, faremos através do comando:

- *docker run -i -t --name nginxserver ubuntu /bin/bash*

O comando "*docker run*" inicia a imagem ou, se for o caso, indica ao Docker que não possuímos a imagem e precisamos do download da mesma. O parâmetro " *-i* " indica que o comando está sendo executado em modo interativo, a passo que o parâmetro seguinte " *-t* " indica para o sistema operacional alocar um TTY simulado. O parâmetro "*--name nginxserver*" dará um apelido ao container iniciado, e o comando seguinte irá indicar a imagem que desejamos executar, "*ubuntu /bin/bash*", sendo que o */bin/bash* é o processo que irá ser executado.
Na data de instalação, o Ubuntu utilizado foi o 14.10, portanto, somos colocados dentro do container com Ubuntu 14.10, com o shell "Bash" sendo executado. O nome de usuário no terminal é alterado, demonstrando que estamos em uma outra máquina, esta virtualizada.

Também pode-se utilizar o comando abaixo para confirmação de que se está em um container com um sistema diferente da máquina host, conforme imagem.

- *cat /etc/issue*

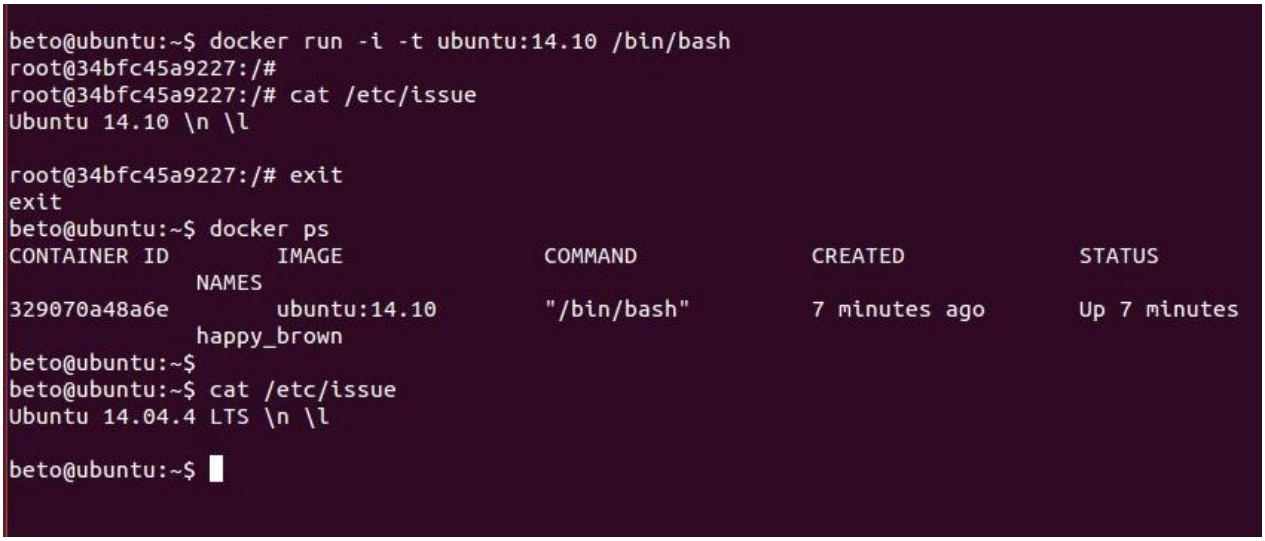

Figura 6: instalação, entrada no container e saída do container com Ubuntu 14.10, instalado sobre a máquina física com o sistema operacional Ubuntu 14.04 LTS

Adicionalmente, para podermos sair do container sem o finalizar, mantendo o container ativo, assim como suas aplicações, utilizamos a combinação de teclas "*CTRL + P + Q*".

Para voltar ao container que está em execução, utilizamos um comando e o container ID (na imagem acima, "*34bfc45a9227*"):

- *docker attach 34bfc45a9227*

Demonstrada a instalação e existência do container, prossegue-se com a instalação do NGINX, estando dentro do container, com o seguinte comando:

- *apt-get update*
- *apt-get install -y nginx*

Após realizado o download do NGINX, para verificarmos se a instalação dentro do container foi bem sucedida, utilizamos:

- *nginx -v*

Containers tem como característica a volatilidade. Afim de tornar esta instalação permanente, pode-se realizar um "*commit*" do container. Esta ação irá gravar o estado atual do container na imagem utilizada.

Para isso, deveremos utilizar o comando, seguindo do nome do container:

- *sudo docker commit 34bfc45a9227 ubuntu/nginx*

Ao executar o commit, uma nova imagem é criada baseada na inicial. A nomenclatura "*ubuntu/nginx*" será definida para a nova imagem. Pode-se verificar a criação da mesma com o comando abaixo:

- *docker images*

Em seqüência, utilizando-se da imagem com o NGINX instalado, iremos configurá-lo para apresentar uma página local padrão do NGINX. Utilizando o argumento "*-p*" e definindo a porta de comunicação 80 do container com a porta 8080 da máquina física, podemos informar que estas portas estarão mapeadas e abertas, estabelecendo conexão entre as duas máquinas, real e virtual. Esta ação será configurada estando logado como root, utilizando a seqüência de comandos:

- *docker run -it -p 8080:80 ubuntu/nginx /bin/bash*

Estando dentro do Container, inicializamos o servidor Nginx:

- *service nginx start*

Com isso, o servidor NGINX estará instalado dentro do container em que é virtualizada uma máquina Ubuntu, comunicando-se com a máquina física, onde temos um Ubuntu 14.04 LTS, e provendo a página de inicialização do NGINX, na porta indicada.

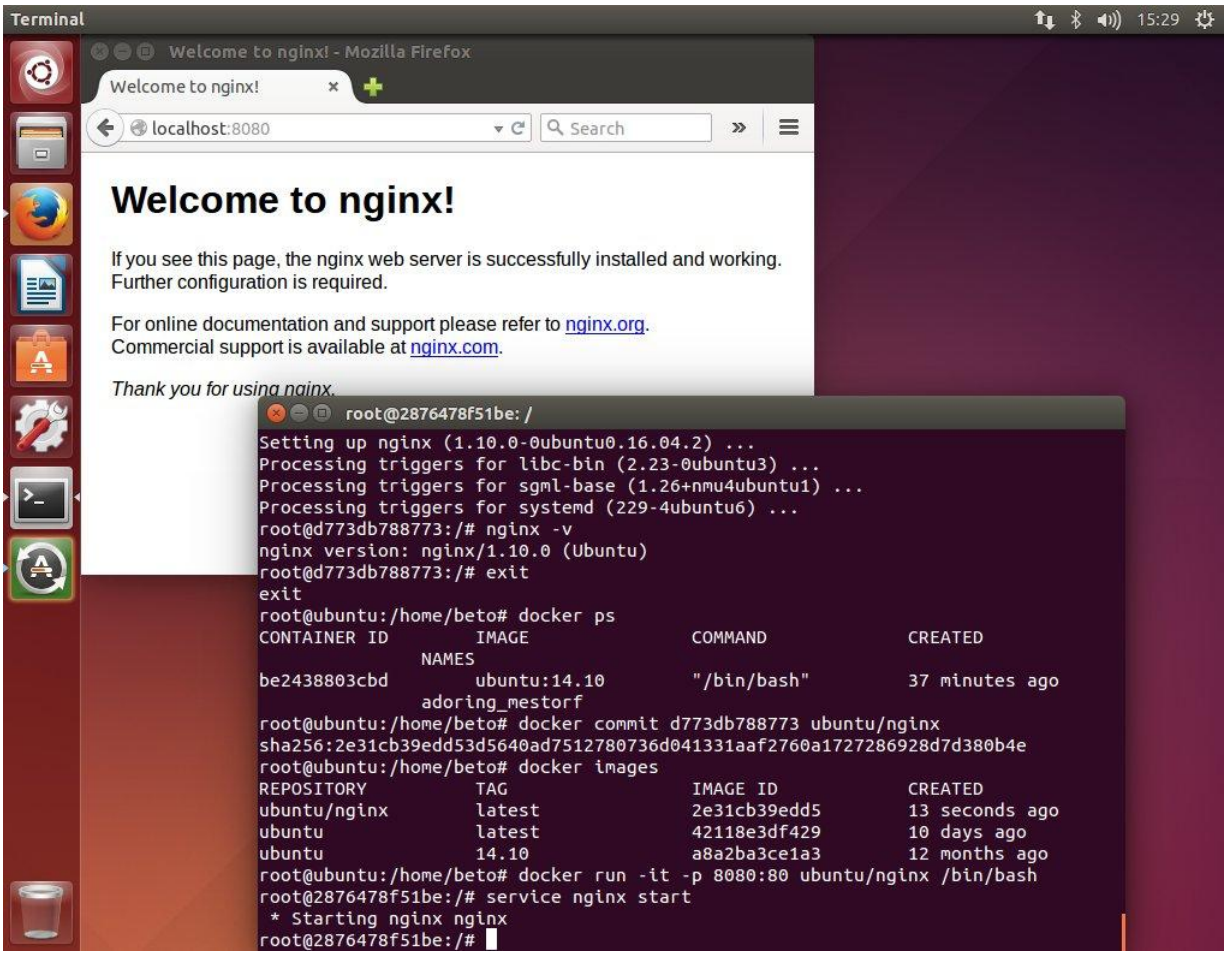

Figura 7: Docker em Linux Ubuntu 14.04 LTS, com um container Docker com Ubuntu 14.10, executando o servidor NGINX

O que ocorre se fecharmos este container, utilizando o simples comando "*exit"*? O servidor é finalizado, e a página do NGINX não mais responderá na porta 8080, tornando o servidor indisponível para o sistema operacional hospedeiro - que não mais estará hospedando o container. O containter é finalizado, o serviço interrompido e os recursos alocados para execução deste container são liberados.

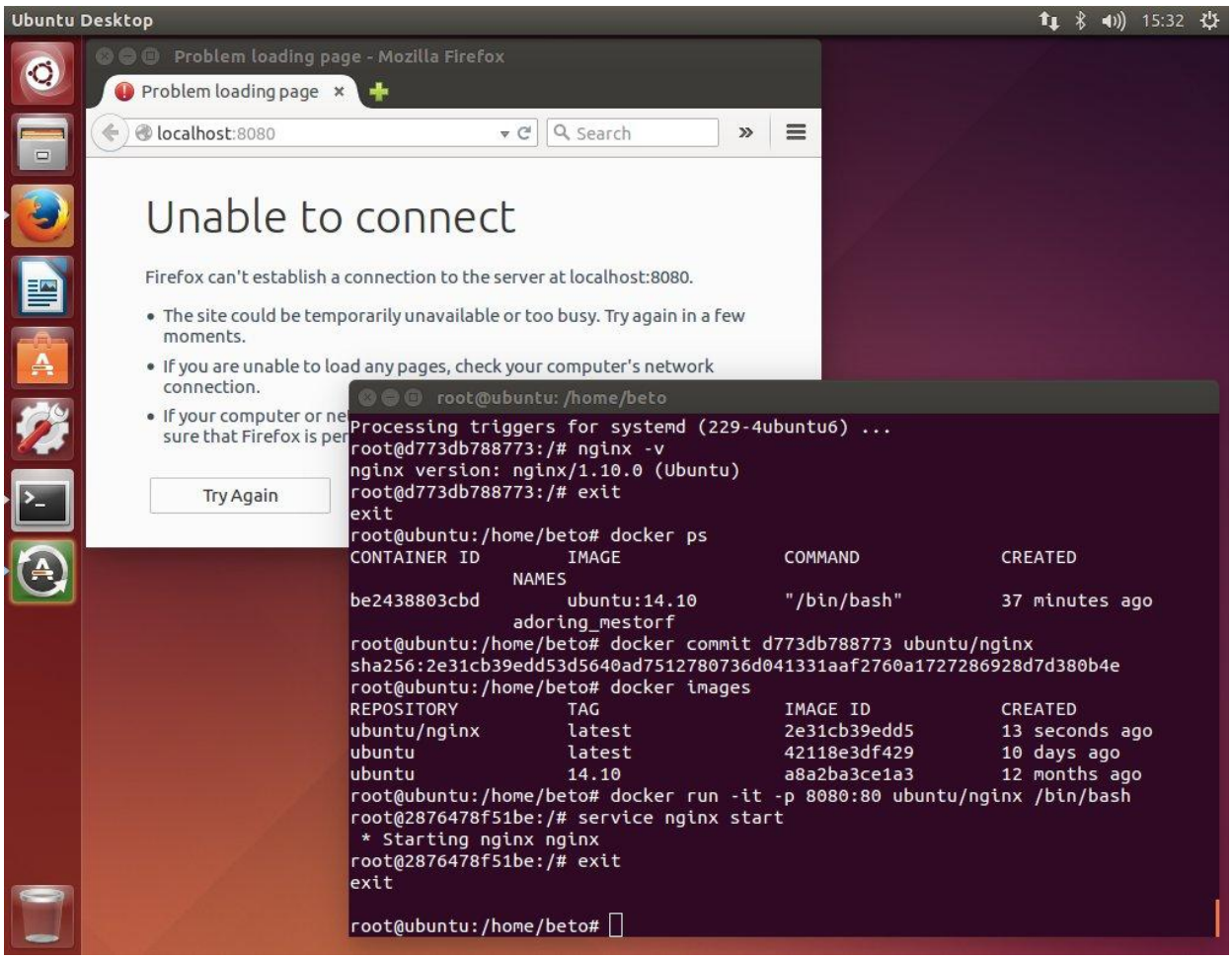

Figura 8: Docker sendo executado em Linux Ubuntu 14.04 LTS, onde o comando "*exit*" finalizou a execução do container com o servidor NGINX

14.3 UBUNTU 14.04 NO VWWARE PLAYER EXECUTANDO O NGINX

Uma linha de comando é suficiente para inicializar o servidor Nginx, e um comando irá finalizar o mesmo.

Para efeito de comparação, foi utilizada uma imagem do mesmo Ubuntu 14.04 LTS instalada no VMware Workstation Player, sobre a qual foi configurado o mesmo Webserver Nginx.

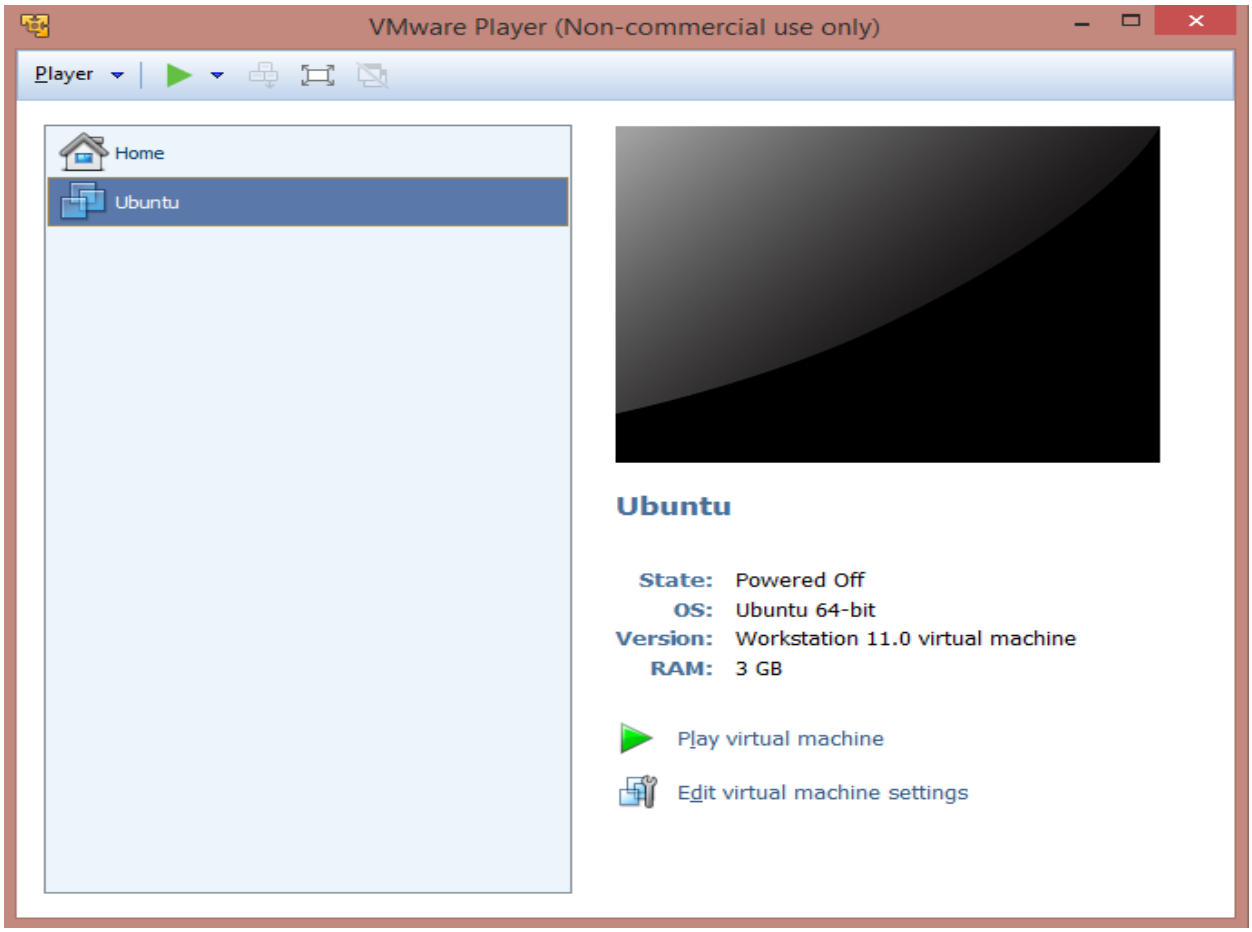

Figura 9: VMware com o Ubuntu instalado, com o Webserver Nginx já configurado

Considerando que o mesmo serviço já esteja totalmente configurado no container Docker e no VMware Workstation Player, temos que o serviço é inicializado no Docker ao tempo da digitação de uma linha de comando no terminal, que irá inicializar a imagem e o serviço, a passo que no VMware Workstation Player, também após realizarmos todas as configurações necessárias, tivemos o tempo de carregamento do Hypervisor, sobre o qual é inicializado um novo sistema operacional completo, um novo kernel, suas bibliotecas, aplicações, para só então o serviço estar disponível.

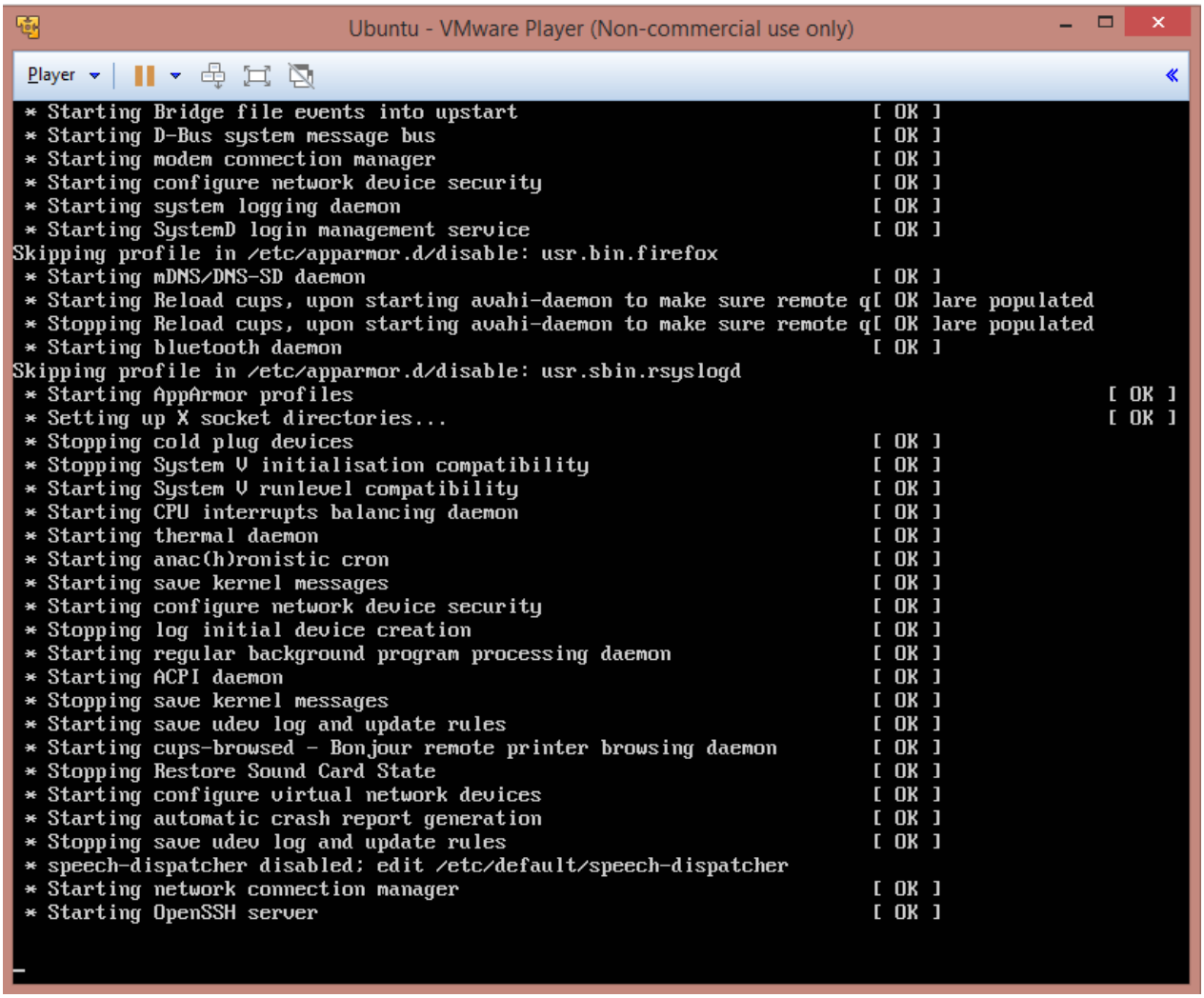

Figura 10: Carregamento do sistema Linux Ubuntu 14.04 LTS no hypervisor VMware Player

Finalmente, temos o sistema operacional carregado e podemos então inicializar o serviço do webserver Nginx, apresentado em imagem abaixo.

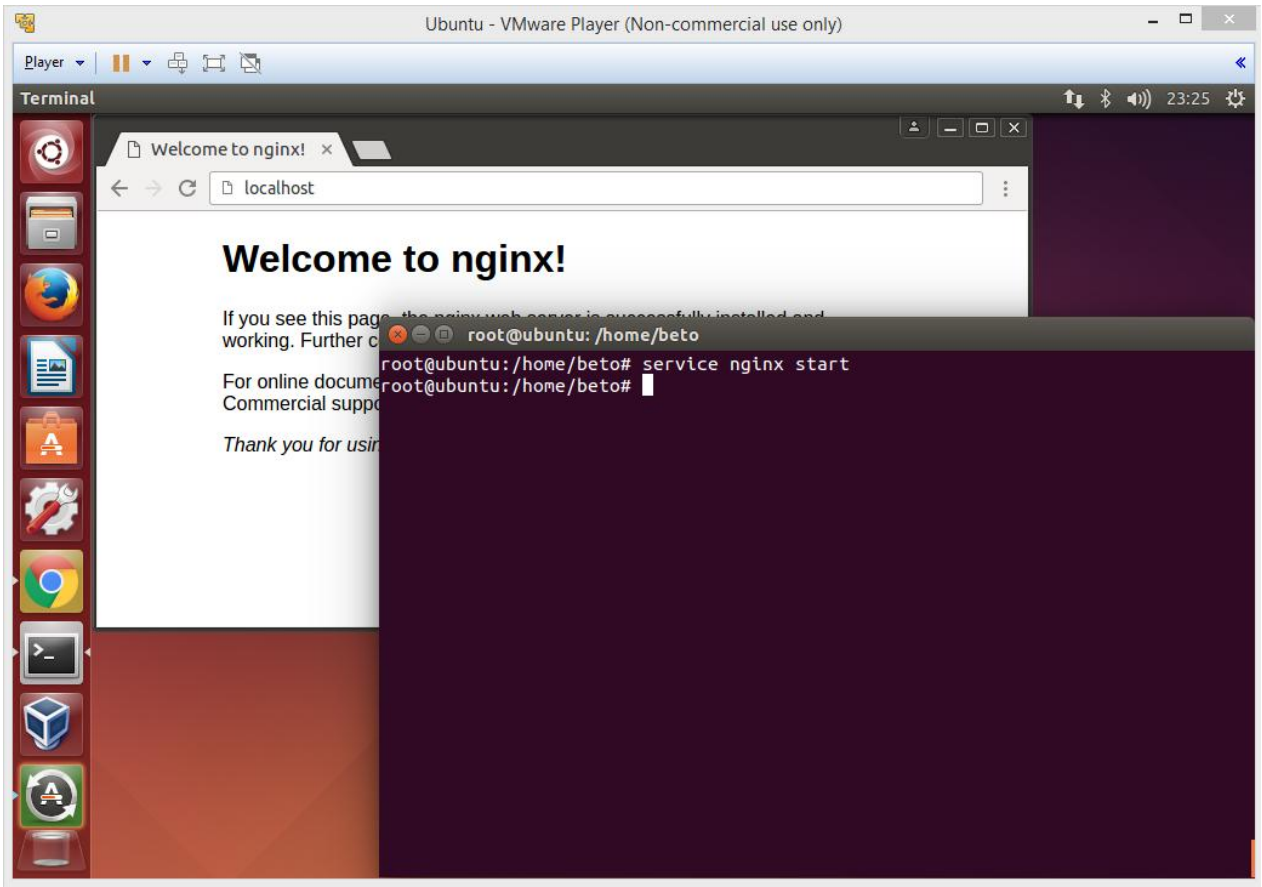

Figura 11: NGINX sendo executado no Linux Ubuntu 14.04 LTS, virtualizado no VMware Player.

15. ADMINISTRANDO SEU CONTAINER

Na instalação default do Docker, as configurações principais de seus contêineres ficam na máquina hospedeira, no diretório "/var/lib/docker". Este é o local onde se encontram os containers com suas respectivas configurações.

Uma série de comandos é utilizada para manipular e administrar containers. São citados abaixo os comandos utilizados na versão 1.11 do Docker:

- *docker attach* – Acessar dentro do container e trabalhar a partir dele.
- *docker build* – A partir de instruções de um arquivo Dockerfile eu possa criar uma imagem.
- *docker commit* – Cria uma imagem a partir de um container.
- *docker cp* – Copia arquivos ou diretórios do container para o host.
- *docker create* – Cria um novo container.
- *docker diff* – Exibe as alterações feitas no filesystem do container.
- *docker events* – Exibe os eventos do container em tempo real.
- *docker exec* – Executa uma instrução dentro do container que está rodando sem precisar atachar nele.
- *docker export* – Exporta um container para um arquivo .tar.
- *docker history* – Exibe o histórico de comandos que foram executados dentro do container.
- *docker images* – Lista as imagens disponíveis no host.
- *docker import* – Importa uma imagem .tar para o host.
- *docker info* – Exibe as informações sobre o host.
- *docker inspect* – Exibe r o json com todas as configurações do container.
- *docker kill* – Da Poweroff no container.
- *docker load* – Carrega a imagem de um arquivo .tar.
- *docker login* – Registra ou faz o login em um servidor de registry.
- *docker logout* – Faz o logout de um servidor de registry.
- *docker logs* – Exibe os logs de um container.
- *docker port* – Abre uma porta do host e do container.
- *docker pause* – Pausa o container.
- *docker ps* – Lista todos os containers.

- *docker pull* – Faz o pull de uma imagem a partir de um servidor de registry.
- *docker push* – Faz o push de uma imagem a partir de um servidor de registry.
- *docker rename* – Renomeia um container existente.
- *docker restart* – Restarta um container que está rodando ou parado.
- *docker rm* – Remove um ou mais containeres.
- *docker rmi* – Remove uma ou mais imagens.
- *docker run* – Executa um comando em um novo container.
- *docker save* – Salva a imagem em um arquivo .tar.
- *docker search* – Procura por uma imagem no Docker Hub.
- *docker start* – Inicia um container que esteja parado.
- *docker stats* – Exibe informações de uso de CPU, memória e rede.
- *docker stop* – Para um container que esteja rodando.
- *docker tag* – Coloca tag em uma imagem para o repositorio.
- *docker top* – Exibe os processos rodando em um container.
- *docker unpause* – Inicia um container que está em pause.
- *docker version* – Exibe as versões de API, Client e Server do host.
- *docker wait* – Aguarda o retorno da execução de um container para iniciar esse container.

16. CONCLUSÃO

O presente documento busca apresentar as vantagens, ainda que óbvias, da utilização dos mais variados recursos de virtualização, com o foco específico nesta recente tecnologia denominada Docker.

Após todos os estudos, espero ter demonstrado os pontos em que a utilização do Docker é vantajosa em relação a outras formas de virtualização, mas especialmente, em como este software desenvolvido por uma empresa, em conjunto da comunidade de software livre, pode unir e complementar todas as formas de virtualização já existentes, simplificando o trabalho do desenvolvedor em meio a tantas ferramentas e tecnologias necessárias para o desenvolvimento de software. Espero também ter demonstrado que é válida a máxima onde, no mundo da informática, novas tecnologias em regra vêm para somar, e não para substituir. Seguindo esta máxima, busco demonstrar que o Docker tem seu grande trunfo ao rodar aplicações, e não em emular uma máquina completa com sistema operacional próprio.

Embora já existentes anteriormente, foi com o Docker que pudemos ver uma revolução no uso de *Linux Containers*. A DotCloud trouxe ferramentas que facilitaram a administração destes *Containers,* até então de difícil instalação e configuração. Aplicações podem ser instaladas nestes *Containers*, possibilitando serviços de hospedagem mais simples e baratos, servidores com a mesma capacidade atendendo ainda mais clientes, máquinas tendo seus recursos otimizados e abrindo um leque de opções para os desenvolvedores. Tudo graças a esta revolução denominada Docker.

Cumprido este objetivo, aqui tratado como principal, tenho também a intenção de aprimorar este trabalho procurando conhecer com detalhes o dia a dia em ambientes de desenvolvimento, afim de poder especificar de forma mais precisa as necessidades de programadores e analistas, podendo concentrar ainda mais o foco neste cenário e nos ganhos específicos do uso de Docker e seus *Containers.*

REFERÊNCIAS

ADRIAN MOUAT, **Usando Docker - Desenvolvendo e Implantando Software com Contêineres**, 1ª Ed., São Paulo, SP, Brasil: Editora Novatec, 2016.

DANIEL ROMERO, **Containers Com Docker, Do Desenvolvimento À Produção**, 1ª Ed. São Paulo, SP, Brasil: Editora Casa do Código, 2015.

DOCKER WEBSITE, **What is Docker?**. Disponível em <https://www.docker.com/what-docker>. Acesso em 18/10/2015.

LEANDRO STOK, **Thinclient ou Desktop, quando e onde devo utilizar um ou outro?** Disponível em <http://www.tiespecialistas.com.br/2011/02/thinclient-ou-desktop-quando-e-onde-devo-utilizar-um-ou-outro>. Acesso em 19/10/2015.

IBM DEVELOPERSWORKS, **Virtualização de aplicativos, passado e futuro**. Disponível em <http://www.ibm.com/developerworks/br/library/l-virtual-machine-architectures>. Acesso em 19/10/2015.

JAMES TURNBULL, **The Docker Book: Containerization is the new virtualization**, disponível para Kindle em <http://www.amazon.com/The-Docker-Book-Containerization-virtualization-ebook/dp/B00LRROTI4/ref=cm_cr-mr-img>

Oracle VM VirtualBox: Test, Develop, and Demonstrate Across Multiple Platforms on one Machine. Disponível em <http://www.oracle.com/us/technologies/virtualization/virtualbox/overview/index.html. Acesso em 22/10/2015.

RAFAEL BERNARDES - (MVP, MICROSOFT COMMUNITY CONTRIBUTOR, MICROSOFT PARTNER), **Analisando a performance de máquinas virtuais no hyper-v com o perfmon**. Disponível em <http://social.technet.microsoft.com/wiki/contents/articles/5935.aspx>. Acesso em 22/10/2015.