VIRTUALIZAÇÃO E DOCKER

Luan Rubens Rodrigues Figueredo[1]
Gustavo dos Santos De Lucca[2]

Resumo: Este artigo apresenta os conceitos de virtualização, abrangendo definições dos mais variados tipos de virtualização no mercado, com suas vantagens e desvantagens. Também será exibido o conceito de LXC – Linux Containers, bem como o conceito de Docker, uma tecnologia que vem crescendo no mercado atual. Serão apresentadas definições, uma breve história e como utilizar os principais comandos. Visando o entendimento dos conceitos e aplicações do Docker, será realizada uma prática, onde efetuar-se-á a criação e gerenciamento de um container, que será utilizado a partir de uma máquina física.

Palavras-chave: Docker. Virtualização. Gerenciamento. Servidores.

1 INTRODUÇÃO

A virtualização é um dos assuntos mais importantes atualmente no mundo de Tecnologia da Informação (TI) (BURGUER, 2012). Surgiu da necessidade de reduzir o custo de infraestrutura e o consumo de energia. Antigamente muitas vezes em que se era necessário distribuir um novo software, por exemplo, era necessário adquirir uma nova máquina para executá-lo, pois as bibliotecas e outros recursos exigiam praticamente um sistema operacional dedicado. Com o surgimento da virtualização então foi possível usar mais de um sistema operacional sobre um mesmo hardware, gerando um reaproveitamento na infraestrutura criada (ALECRIM, 2012).

A virtualização pode ser considerada a tecnologia central de um datacenter, tornando-se essencial para qualquer empresa que lida com tecnologia. A tecnologia de virtualização iniciou-se com a publicação de um artigo em 1959, que tratava do uso de multiprogramação em tempo compartilhado. Inicialmente a virtualização utilizava o conceito de máquina virtual de processo, onde uma aplicação é executada sobre um sistema operacional e simula o comportamento de outro sistema. Outro marco para a

[1]Graduando em Engenharia da Computação na Faculdade SATC. E-mail: luanrubensf@gmail.com
[2]Ms. em Engenharia, Professor da Faculdade SATC. E-mail: gustavo.lucca@satc.edu.br

virtualização foi em 1998, com o surgimento da Vmware®. A virtualização pode ser considerada uma revolução para a área de tecnologia da informação (VERAS, 2016).

A virtualização vem sendo usada por várias organizações para melhorar a entrega de Serviços de TI. Entre as características da Virtualização, destaca-se a redução do consumo de energia, principalmente com refrigeração, reduzindo ainda o gasto com equipamentos e o espaço necessário para manter os servidores. A virtualização também permite fornecer alta disponibilidade de aplicações críticas, velocidade no *deploy* de serviços e migrações (BURGUER, 2012). Outro conceito que caminha junto com virtualização é o de Computação em nuvem, que é um paradigma que visa fornecer serviços distribuídos via Internet. Com o surgimento da virtualização e computação em nuvem, foram criados diversos outros conceitos e mecanismos visando melhorar o aproveitamento das técnicas.

Esta pesquisa visa contribuir com o estudo de tecnologias de virtualização e Docker. Busca-se com este artigo apresentar os conceitos necessários para o entendimento destas tecnologias, bem como apresentar uma comparação teórica sobre as mesmas. Visando atingir este objetivo também será apresentada uma prática com Docker. Conforme será apresentado, esta tecnologia é uma potencial escolha para diversos objetivos, principalmente no que tange ao desenvolvimento de aplicações. Também será possível perceber a praticidade e velocidade na utilização desta tecnologia.

2 VIRTUALIZAÇÃO

Virtualização é uma combinação de software e hardware visando a criação de máquinas ou ambientes virtuais em uma única máquina física (BURGUER, 2012). Na prática, as máquinas virtuais oferecem os mesmos serviços que uma máquina física, onde um sistema operacional pode ser utilizado com todos os seus recursos. Porém, máquinas ou ambientes virtuais só existem logicamente (ALECRIM, 2012). Com isso, é possível executar diversas máquinas sobre um mesmo hardware, o que aumenta o aproveitamento computacional e, consequentemente, diminui o consumo energético e infraestrutura.

2.1 CATEGORIAS DE VIRTUALIZAÇÃO

A Virtualização possui três grandes categorias. A primeira é a Nível de Hardware, onde a camada de virtualização é colocada diretamente sobre a máquina física e fornece às camadas superiores um hardware abstrato, similar ao original (VERAS; CARISSIMI, 2015). Tem-se como exemplo: Vmware ESX e Xen.

A segunda categoria apresentada é a Nível de SO, como característica, a camada de virtualização é executada entre o sistema operacional e as aplicações. As partições lógicas são vistas como máquinas isoladas, compartilhando o mesmo sistema operacional.

E ainda, a terceira categoria que é em nível de aplicação, onde a camada de virtualização é uma aplicação rodando dentro de um Sistema operacional. Define uma máquina abstrata virtual que executa aplicações, geralmente, de alto nível (VERAS; CARISSIMI, 2015). O maior exemplo é a JVM (Java Virtual Machine), que é uma máquina virtual que executa aplicações escritas na linguagem Java.

2.2 TIPOS DE VIRTUALIZAÇÃO

A virtualização de servidores é um processo no qual os sistemas operacionais e todas as suas aplicações passam a utilizar uma máquina virtual, e não mais um servidor físico, fazendo uso mais eficiente do hardware, permitindo maior agilidade e redução de custos (ROCHA, 2013). Sendo basicamente uma arquitetura que permite vários servidores virtuais executados em um único servidor físico. Cada servidor virtual é independente e isolado. A Figura 1 representa uma arquitetura de virtualização de servidores.

Figura 1: Virtualização de servidores
Fonte: Rocha (2013)

A virtualização de desktops é um conceito parecido com o de virtualização de servidores, ou seja, vários desktops virtuais são executados em um único servidor físico. Neste tipo de virtualização, os usuários não executam seus aplicativos e sistemas operacionais localmente. É adotado o modelo cliente-servidor, no qual todas as aplicações, processos e dados são mantidos de forma centralizada (ROCHA, 2013). A Figura 2 exibe uma estrutura básica de virtualização de desktops.

Figura 2: Virtualização de desktops
Fonte: CRMG – Network & Security (2014)

A virtualização de aplicações é uma forma de virtualização que permite que aplicações sejam executadas, em máquina local ou virtual, sem serem instaladas no dispositivo, ou seja, a aplicação roda centralizada e sem a necessidade de atualização e manutenção em diversos pontos (ROCHA, 2013). Isto permite uma configuração e

implementação centralizada da aplicação, o que melhora seu gerenciamento (VERAS; CARISSIMI, 2015).

A Figura 3 representa a arquitetura de virtualização de aplicações, onde a aplicação roda somente em uma máquina, sobre um sistema operacional, e os demais usuários apenas devem se conectar ao servidor para usar este serviço.

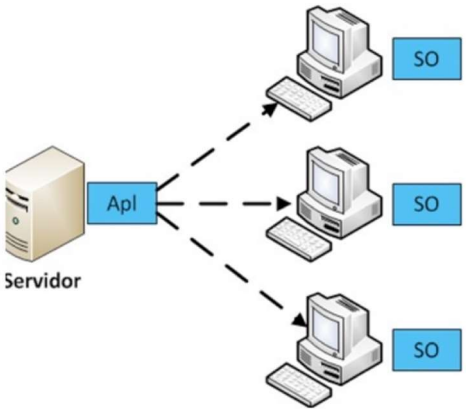

Figura 3: Virtualização de aplicação
Fonte: adaptado de Rocha (2013)

A virtualização de redes visa proporcionar ambientes lógicos de redes independentes, que são criados sobre uma única infraestrutura compartilhada de rede. Cada rede lógica fornece, a um grupo de usuários, serviços de redes semelhantes ao de uma rede não virtualizada. A rede lógica pode fornecer recursos dedicados e políticas de segurança independentes, o que envolve a segmentação da rede de transportes, dos dispositivos e serviços (VERAS; CARISSIMI, 2015).

2.3 HYPERVISORS

Um *hypervisor,* também chamado de Monitor de Máquina Virtual (Virtual Machine Monitor, VMM), pode ser definido como uma camada de software que é executada entre o hardware e o sistema operacional. É responsável por fornecer recursos da máquina física à máquina virtual, e permite que vários sistemas operacionais sejam executados em uma única máquina (MATTOS, 2008). A segurança dos recursos virtualizados e a agilidade na administração dos recursos computacionais, podem ser consideradas características necessárias de um

hypervisor. Os hypervisors podem ser classificados em Tipo I (*baremetal*) e Tipo II (*hosted*):

A. Tipo I (bare metal)

Opera diretamente sobre o hardware físico. Controla o hardware e o acesso do sistema operacional convidado (*guest* OS) (VERAS; CARISSIMI, 2015). Neste tipo, seu papel é compartilhar e gerenciar recursos entre as máquinas virtuais, de forma que cada máquina virtual possua seu próprio recurso, sem sofrer influência de outras máquinas virtuais. O kernel de uma máquina virtualizada com *hypervisor* tipo I, pode ser dividido: modo núcleo, onde fica instalado o sistema operacional, e modo usuário, onde os processos das máquinas virtuais são executados (ALVES, 2008).

B. Tipo II (hosted)

Não opera diretamente no hardware físico. É uma aplicação, executada como um processo do sistema operacional nativo, que fornece um ambiente de execução para outras aplicações (VERAS; CARISSIMI, 2015). A camada de virtualização possui um hardware virtual, que é criado a partir de recursos fornecidos pelo hardware do sistema operacional nativo. Possui seus recursos divididos com outros processos, pois funciona como qualquer outro processo rodando no sistema operacional nativo. A Figura exibe a arquitetura dos dois tipos de *hypervisors*.

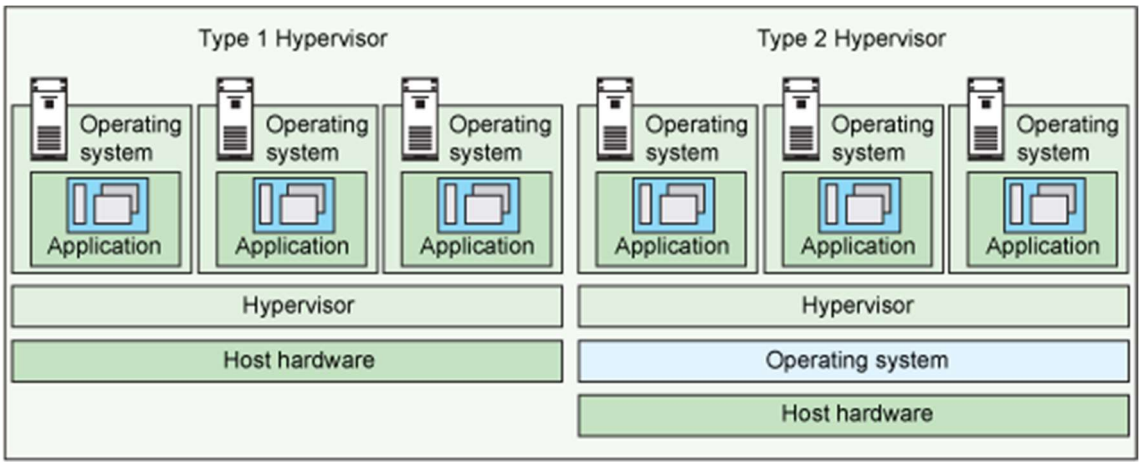

Figura 4:Tipos de *hypervisors*
Fonte: Tholeti (2011)

A virtualização pode ser realizada de maneiras diferentes. Cada forma de implementação possui vantagens e desvantagens. A necessidade do projeto é o que

deve determinar qual forma de virtualização será mais adequada para a solução. No hypervisor tipo II, a virtualização pode ser classificada em: virtualização total, paravirtualização, e virtualização assistida por hardware (VERAS, 2016).

2.3.1 Virtualização total

A virtualização total é uma máquina virtual que possui um sistema operacional sem qualquer alteração. Na virtualização total o sistema operacional executa instruções sensíveis, que são instruções que só podem ser executadas em modo núcleo, ou kernel, pois podem alterar o estado do sistema (ALVES, 2008).

A virtualização total possui algumas desvantagens, como seu desempenho, que é afetado pelo fato de o *hypervisor* ter que verificar todas as instruções sensíveis do sistema operacional convidado e substituí-las por operações equivalentes a serem executadas no hardware (VERAS; CARISSIMI, 2015). Outra desvantagem está no número elevado de dispositivos a serem suportados pelo VMM. A virtualização total possui um mecanismo de dispositivos genéricos, que funciona na maioria dos casos, mas não garante total eficiência. E ainda, possui problemas gerados pela implementação de várias máquinas virtuais, pois um sistema operacional convencional não foi criado para disputar recursos de hardware com outras aplicações, como outros sistemas, e isto pode, consequentemente, diminuir o desempenho na execução dos ambientes (MATTOS, 2008).

Este tipo de virtualização facilita na migração de ambientes virtuais entre máquinas físicas. Isto devido a se tratar de virtualização completa, não existindo dependência dos recursos das máquinas físicas. O isolamento das máquinas virtuais também facilita a gestão de segurança (VERAS; CARISSIMI, 2015).

2.3.2 Paravirtualização

Nesta implementação o sistema operacional convidado é modificado, intencionalmente, para que suas instruções sensíveis sejam executadas por meio de chamadas de *hypervisor* (*hypercalls*) (ALVES, 2008). Com isso o ganho em desempenho é significativo, visto que o VMM não necessita testar cada instrução.

Um ponto importante deste formato é que os dispositivos de hardware são acessados por drivers da máquina virtual, não necessitando de *drivers* genéricos. A principal desvantagem da Paravirtualização é a necessidade de alteração do sistema operacional convidado, devido à complexidade e ao acesso ao código fonte do mesmo, visto que nem todos os sistemas operacionais disponibilizam o código para modificações.

2.3.3 Virtualização assistida por hardware

A virtualização assistida por hardware foi um conceito criado pela Intel® e AMD® que visa melhorar o desempenho da solução como um todo. Esta tecnologia elimina os problemas da virtualização total e da Paravirtualização, e ainda fornece um maior desempenho na execução de instruções sensíveis por parte dos sistemas operacionais convidados (VERAS; CARISSIMI, 2015).

Neste formato o sistema operacional convidado tem acesso direto aos recursos do hardware, melhorando ainda mais o desempenho. Como todos os recursos para execução das máquinas virtuais são fornecidos, as alterações dos SOs também não são mais necessárias. Basicamente a virtualização assistida fornece um modo de privilégio adicional, não presente nos outros tipos de virtualização.

Uma limitação desta tecnologia é a necessidade de um apoio explícito no CPU host, sendo que nem todos os processadores hoje no mercado possuem esta funcionalidade.

3 COMPUTAÇÃO EM NUVEM

Junto com as práticas de virtualização surge o conceito de computação em nuvem, que é um conceito que pode ser aplicado nos cenários de TI para fornecer produtos de forma escalável e distribuída (LEWIS, 2010). A virtualização é um dos recursos fundamentais para que o conceito de computação em nuvem possa ser aplicado, pois através dela é possível obter maior escalabilidade, flexibilidade e redução de custos.

Computação em nuvem é um paradigma de computação distribuída que tem foco em prover serviços distribuídos e escaláveis sobre a Internet, para um grande

número de usuários (LEWIS, 2010). Estes serviços podem fornecer desde estruturas, ou máquinas completas, até aplicações simples, que são apenas acessadas pelos clientes.

3.1 MODELOS DE SERVIÇO

Os serviços fornecidos pela nuvem podem ter diferentes formas de entrega, e podem ser divididos em IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*). Cada uma das formas de serviço possui suas próprias características e fornecem diferentes níveis de responsabilidade para o cliente e fornecedor (VERAS, 2015). A Figura 5 exibe as responsabilidades do cliente e do fornecedor em cada uma das categorias mencionadas, bem como uma breve comparação com o modelo tradicional (*On-Premisses*), onde o cliente tem responsabilidade por todas as camadas de serviço.

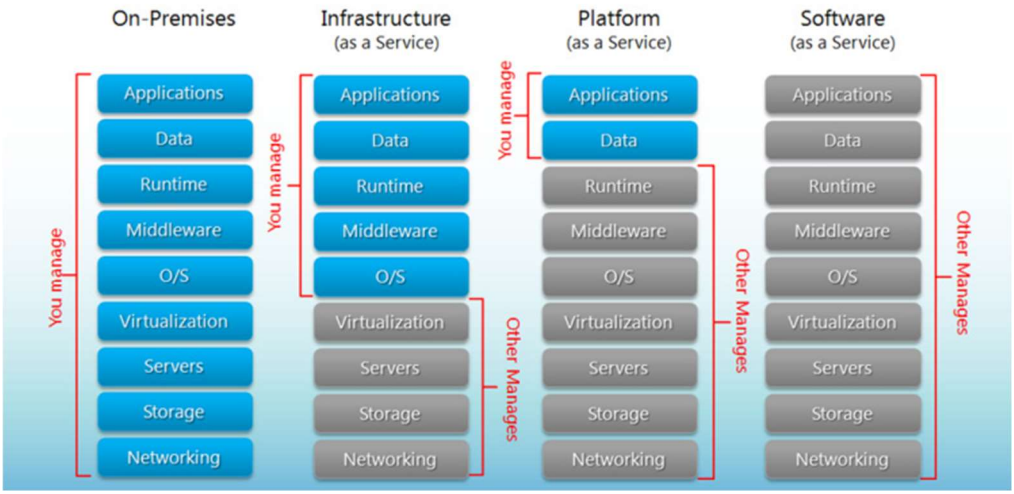

Figura 5: Responsabilidades na computação na nuvem
Fonte: Stamey (2017)

IaaS fornece uma estrutura computacional como serviço, sendo disponibilizada através da Internet. Funciona, de maneira genérica, como um hardware fornecido ao cliente. Uma grande vantagem de usar IaaS, é que a empresa não precisa comprar toda uma estrutura robusta para montar um servidor ou um Datacenter, mas sim, alugar esta estrutura de um provedor e pagar somente o que é utilizado (LEWIS, 2010). O Microsoft Azure é um exemplo de IaaS.

O PaaS fornece a estrutura computacional fornecida para o cliente, visando o processamento e armazenamento de aplicações que serão executadas e disponibilizadas na nuvem. Este modelo também fornece meios para a comunicação dos aplicativos disponibilizados. O provedor fica responsável pela manutenção da estrutura e da camada de software fornecida, assim o cliente fica livre para desenvolver e disponibilizar seus próprios serviços. Tem-se como exemplos de PaaS o Microsoft Azure e o AppEngine da Google (VERAS, 2015).

No modelo SaaS, são disponibilizadas aplicações completas através da nuvem, sendo uma alternativa para o processamento e armazenamento local. Estas aplicações podem ser de interesse coletivo ou sob demanda para determinados clientes. Toda a estrutura de armazenamento, redes, sistemas operacionais e servidores são de responsabilidade do provedor de serviços. O cliente possui baixo nível de responsabilidade nesse modelo. São exemplos de SaaS o Gmail, da Google e o SalesForce.com (VERAS, 2015).

3.2 MODELOS DE IMPLANTAÇÃO DE NUVEM

Existem diferentes modelos de implantação de computação na nuvem, os quais as empresas podem optar para alocar suas aplicações ou serviços. Os tipos disponíveis são nuvem privada, nuvem pública, nuvem comunitária e nuvem híbrida (VERAS, 2015).

A nuvem pública é disponibilizada e mantida por terceiros, geralmente organizações públicas ou grupos industriais, que geralmente, possuem grande capacidade de processamento e armazenamento. Funciona com um modelo de pagamento por demanda, ou seja, o cliente paga apenas os recursos que utilizar. Podem-se destacar como vantagens da nuvem pública o baixo custo inicial, facilidade no gerenciamento e economia de escala, devido ao modelo de pagamento por uso (VERAS, 2015).

A nuvem privada é um modelo no qual a infraestrutura é de uso exclusivo de uma organização. Geralmente os serviços oferecidos pela organização são de uso exclusivo da mesma, não sendo acessados publicamente. O modelo de nuvem privada ainda oferece dois modelos básicos, sendo a nuvem privada hospedada na empresa e a nuvem privada hospedada em provedores de serviço. No primeiro, a

estrutura é mantida e fica localizada na própria empresa. Este modelo é interessante quando deve-se ter um controle rígido da nuvem, ou quando exigido por regras da empresa. Já no segundo, o cliente utiliza uma estrutura de terceiros para hospedagem de sua nuvem privada, sendo interessante para aplicações gerais e aplicações críticas. Tem-se como vantagens da nuvem privada o maior controle, maior qualidade no serviço, facilidade em integração e um custo total mais baixo (VERAS, 2015).

Na nuvem comunitária, a infraestrutura de nuvem é compartilhada entre diversas organizações que possuem interesses comuns. Também pode ser gerenciada por organizações que fazem parte do grupo, ou por terceiros. A estrutura também pode estar localizada em terceiros ou em membros do grupo (VERAS, 2015).

Existe também uma junção dos modelos disponíveis, esta podendo ser de dois ou mais modelos. Esta nuvem é chamada de nuvem hibrida. Cada modelo ainda funciona de forma única, porém são conectados por meio de tecnologias padronizadas ou privadas. Isto permite a conectividade entre as nuvens disponíveis. A nuvem híbrida ainda exige uma camada a mais de gerenciamento, visando conciliar nuvens públicas e privadas (VERAS, 2015).

Um dos mecanismos de virtualização disponíveis é o LXC (Linux Containers), que possui um funcionamento parecido com o de uma máquina virtual. Visando entendimento de Docker e sua arquitetura, para o desenvolvimento da pesquisa, torna-se fundamental o entendimento desse mecanismo de virtualização.

4 LXC – LINUX CONTAINERS

É um mecanismo leve de virtualização em nível de Sistema Operacional, não tendo a necessidade de executar várias máquinas virtuais com a emulação de seus hardwares (ORACLE, 2012). São ambientes de execução que possuem CPU isolada, memória, blocos I/O e recursos de rede, mas compartilham o mesmo kernel do sistema operacional. O Linux Container tem um funcionamento parecido com o de uma máquina virtual, mas é diferente em sua arquitetura, se fazendo mais leve, pois não é necessário executar todo um sistema operacional na máquina.

Uma máquina virtual teria sistemas operacionais duplicados, sendo um para cada instância e vários volumes de disco repetidos, pois cada instância da máquina virtual seria uma máquina isolada. Devido ao fato de os containers serem mais leves,

é possível executar de seis a oito vezes mais containers do que máquinas virtuais em um mesmo hardware (WANG, 2016).

A Figura 6 exibe a arquitetura de virtualização tradicional, onde cada máquina virtual possui seu próprio sistema operacional.

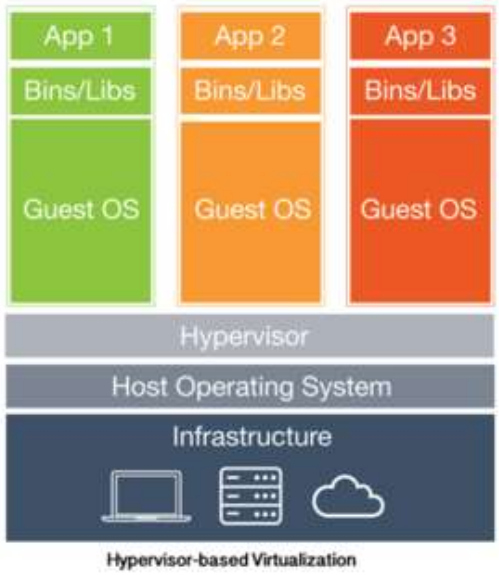

Figura 6: Virtualização tradicional com hypervisor
Fonte: Santhosh (2016)

A Figura 7 exibe a arquitetura da virtualização através de containers, onde todos compartilham o mesmo kernel do sistema operacional, não necessitando que cada container possua seu próprio sistema operacional sendo executado.

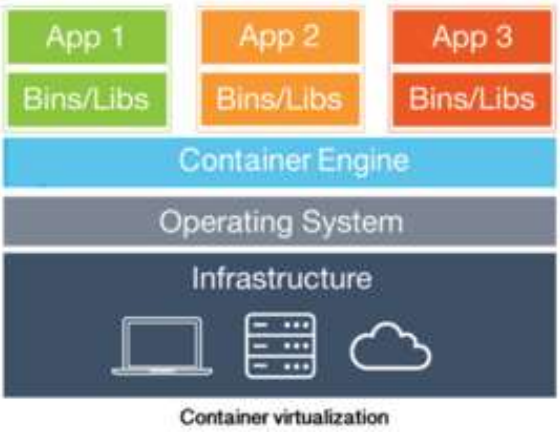

Figura 7: Arquitetura de virtualização com container
Fonte: Santhosh (2016)

Para um entendimento efetivo do funcionamento do LXC, é necessário abordar os conceitos de *cgroups* e *namespaces* do sistema operacional Linux. *Namespaces*, foi um conceito desenvolvido pela IBM que basicamente engloba um conjunto de recursos e os fornece a um processo, fazendo com que estes recursos pareçam dedicados ao processo. Processos de um mesmo *namespace* conseguem ver as alterações nos recursos (WANG, 2016). O conceito de *cgroups* foi desenvolvido pela Google e fornece o isolamento e gerenciamento de recursos do sistema, como CPU e memória, para um determinado grupo de processos (WANG, 2016).

5 DOCKER

Docker é uma plataforma *open source* que automatiza o *deploy* de aplicações em containers. Foi criado pela dotCloud Inc., empresa especialista em PaaS (TURNBULL, 2014). Em 2013 a dotCloud resolveu tornar *open source* o core de sua plataforma, dando origem ao Docker. Inicialmente o Docker era simplesmente um encapsulamento do LXC integrado ao *Union Filesystem*, porém seu crescimento foi muito rápido e o projeto foi muito bem aceito pela comunidade. Com isso a dotCloud passou a se chamar Docker e a primeira versão oficial da plataforma foi lançada (VITALINO; CASTRO, 2016).

Quando a versão 1.0 do Docker foi lançada, empresas grandes, como Spotify, já estavam o utilizando em larga escala. Posteriormente Amazon Web Services (AWS) e Google passaram a oferecer suporte ao Docker em suas soluções de nuvens (VITALINO; CASTRO, 2016). Por ser um projeto *open source* qualquer pessoa poderia contribuir com melhorias e correções de bugs, o que trouxe transparência e agilidade para o projeto.

Um dos diferencias do Docker é que ele possui uma *engine* para o *deploy* de aplicações, sendo executada em um ambiente virtualizado de container. Ele foi desenvolvido para ser um ambiente leve e rápido no qual pode-se executar aplicações e também workflows para o *deploy* das mesmas (TURNBULL, 2014).

Geralmente, um container Docker, leva menos de um segundo para ser inicializado, pois não é necessário um *hypervisor*, nem iniciar um sistema operacional *guest*, o que aumenta a performance e a escalabilidade do ambiente. Para iniciar um

container, é necessário somente uma máquina executando um sistema operacional compatível com Linux e o Docker (TURNBULL, 2014).

O principal objetivo do Docker é proporcionar múltiplos ambientes isolados, dentro de uma mesma máquina, mas acessível do mundo externo (GOMES; SOUZA, 2015). A 8 exibe um ambiente Docker.

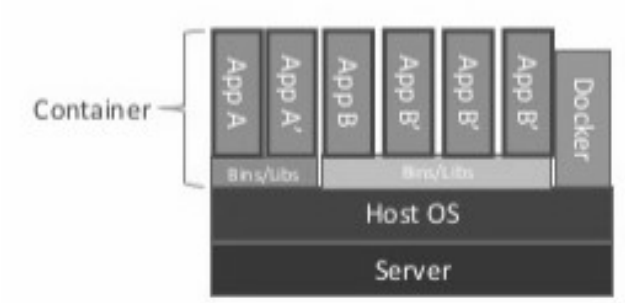

Figura 8: Ambiente Docker.
Fonte: Gomes e Souza (2015).

No desenvolvimento de aplicações, é possível criar ambientes iguais para desenvolvimento, testes e produção, de maneira simplificada usando containers Docker, o que facilita o desenvolvimento e manutenção de aplicações e de seus ambientes.

5.1 COMPONENTES DO DOCKER

O Docker é composto por diversos componentes que são interligados para a criação da plataforma. Dentre estes componentes, destacam-se o Cliente-servidor Docker, as Docker Images, os *Registries* e os Containers. Juntos, esses componentes formam o core do Docker, e são indispensáveis no uso da plataforma. Para completar a arquitetura, também é possível ter uma API REST para efetuar a comunicação do Cliente Docker com o servidor ou *daemom*.

A Cliente-Servidor Docker

Docker utiliza uma arquitetura cliente-servidor. O *Docker client* se comunica com o *Docker server* ou *daemon*. Basicamente o cliente envia comandos para o servidor, que faz todo o trabalho pesado. É possível executar o *Docker client* e o

Docker server na mesma máquina, ou em máquinas separadas, onde o cliente se comunicaria com um servidor remoto.

A comunicação entre o *client* e o *daemon* é feita através de sockets ou através de uma API REST. A Figura 9 exibe a arquitetura básica do Docker.

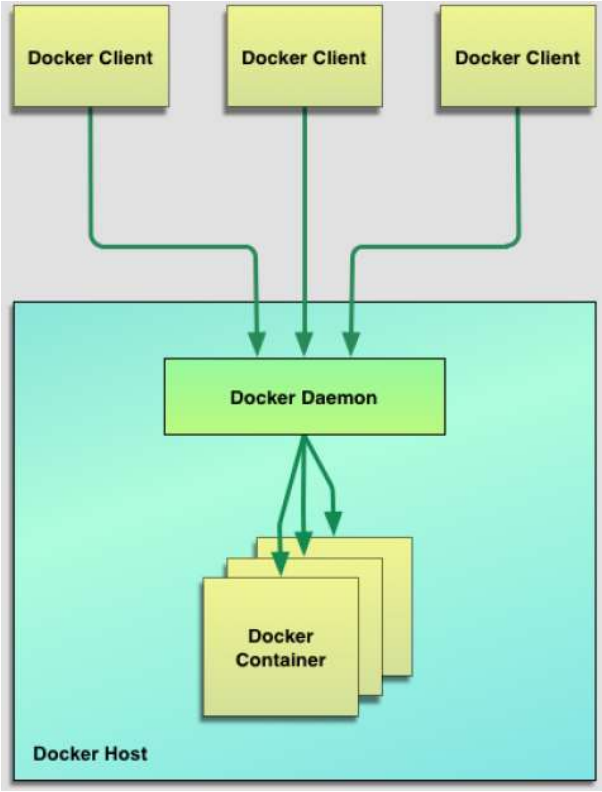

Figura 9: Arquitetura básica do Docker.
Fonte: Turnbull (2014).

B Docker Images

Docker Images é a base de qualquer container. Todo container é executado a partir de uma Docker Image. Elas que fazem a parte de construção do container ser possível. Neste arquivo de imagem (Docker Image) são escritas uma série de instruções que indicam como o container será construído. Podem ser consideradas o código fonte do container, pois indicam como ele deverá ser criado e configurado. Docker Images podem ser compartilhadas, armazenadas, atualizadas e são altamente portáveis (TURNBULL, 2014).

C Registries

É basicamente uma aplicação servidora que permite o armazenamento e distribuição das Docker Images. Existem dois tipos de *registries*: público e privado. O

registry público é chamado de Docker Hub e é mantido pela empresa Docker Inc. Qualquer pessoa pode criar uma conta no Docker Hub para compartilhar e armazenar suas próprias imagens. O Docker Hub contém mais de 10 mil imagens criadas por pessoas da comunidade, no mundo inteiro, disponíveis para o uso. No Docker Hub também está disponível o recurso para armazenar as imagens em modo privado. Nesse modo, é possível armazenar os arquivos da imagem e outros arquivos de informação proprietária, que são mantidos de maneira segura e podem ser compartilhadas somente com membros específicos. Também é possível criar um novo *registry* privado, pois o código utilizado para executar os *registries* é abertos e pode ser utilizados por qualquer pessoa (TURNBULL, 2014).

D Containers

Docker facilita a montagem e execução de containers, nos quais podem ser executadas aplicações e serviços. Os containers são iniciados a partir de imagens e podem conter um ou mais processos. Um container Docker é basicamente um ambiente de execução que possui um conjunto de operações padrão. Cada container possui um software que permite que um conjunto de operações seja executado, como por exemplo, criar, iniciar, parar, reiniciar e destruir um container. Para o Docker, não importa o conteúdo ou o que está sendo executado em cada container, ou seja, não importa se dentro do container está rodando um servidor web ou um banco de dados, cada container é carregado da mesma maneira. Onde o container está sendo executado também não interessa para a aplicação. Por exemplo, um container pode ser iniciado em um laptop, armazenado em um *registry* e o mesmo container ser iniciado em um servidor ou uma máquina virtual. Containers foram feitos para serem leves, portáveis e o mais genérico possível (TURNBULL, 2014).

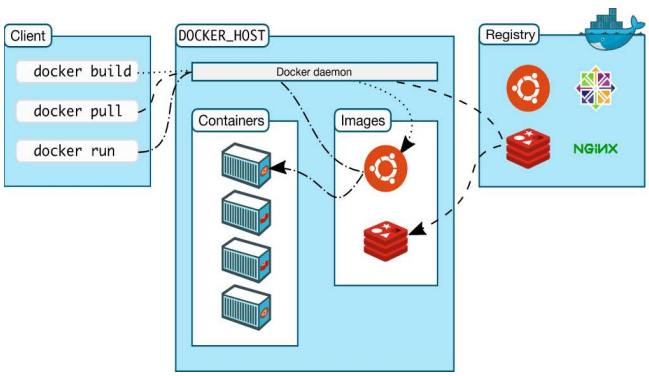

Figura 10: Arquitetura do Docker
Fonte: Docker Inc. (2016)

A Figura exibe uma arquitetura completa do Docker, exibindo o cliente, servidor, Docker Images, Containers e *registries*.

5.2 AMBIENTES DE EXECUÇÃO PARA DOCKER

Docker pode ser executado em qualquer máquina x64 executando um kernel de Linux, de preferência com uma versão 3.8 ou superior. Também é suportado por uma grande variedade de distribuições Linux, como Debian, Ubuntu, CentOS, Fedora, entre outros. Com um ambiente virtual, é possível executar o Docker em ambientes Windows ou Mac OS (TURNBULL, 2014).

Pelo fato de utilizar LXC internamente, é necessário que a versão do kernel suporte as funcionalidades de *cgroups* e *namespaces*. Para executar em sistemas operacionais que não possuem um *kernel* do Linux, é possível usar o Boot2Docker, é uma pequena máquina virtual com scripts para seu gerenciamento. Esta pequena máquina virtual fornece um servidor Docker local, no qual os clientes irão se conectar (TURNBULL, 2014).

A instalação do Docker se dá de maneira simplificada, através de linhas de comando em ambientes Linux e através do Boot2Docker, um software que possui um conjunto de ferramentas para execução nos demais ambientes suportados.

5.3 USANDO DOCKER

A interação com o Docker se dá por meio de linha de comando. O principal comando é o “docker”, que é o responsável por se comunicar com o *daemon* da solução (servidor), toda instrução enviada ao servidor é iniciada com este comando, sendo que o mesmo deve estar mapeado nas variáveis de ambiente do sistema operacional utilizado.

Figura 11: Executando comandos do Docker
Fonte: Elaborada pelo autor

A **11**, exibe um terminal com alguns comandos executados. Nota-se que o comando inicia com a palavra “docker”, que indica que a instrução é enviada ao servidor Docker.

5.4 COMANDOS DOCKER

No próprio site do Docker, é possível encontrar uma vasta documentação sobre todos os comandos que podem ser executados, bem como seus parâmetros, funcionalidades e descrições (DOCKER Inc., 2016). A seguir são exibidos alguns comandos básicos, em sua forma de uso simplificada, pois os comandos Docker aceitam vários parâmetros de configuração.

A. *docker images*

Lista as imagens disponíveis no repositório Docker local.

B. *docker pull [imagem]*

Adiciona uma imagem ao repositório Docker local, onde [imagem] é o nome da imagem que deverá ser adicionada. As imagens ficam disponíveis no Docker Hub.

C. *docker rmi [imagem]*

Comando utilizado para remover uma imagem do repositório local, onde [imagem] é o nome ou ID da imagem que deve ser removida.

D. *docker run [imagem]*

Um dos principais comandos, é utilizado para criar os containers, onde [imagem] é o nome da imagem, a partir da qual será gerado o container. Caso a

imagem não esteja no repositório local, é efetuado o download da mesma e o container é criado. Pode ser usado o parâmetro “--name” para especificar um nome para o container. Também existem diversos outros parâmetros.

E. docker ps

Este comando lista todos os containers que estão em execução no momento. Pode ser utilizado o parâmetro “-a” para exibir todos os containers, estando em atividade ou não.

F. docker rm [container]

Comando responsável por remover um container do servidor Docker. Onde [container] é o nome ou ID do container que deve ser removido. Também é possível utilizar uma variante deste comando para remover todos os containers, passando os seguintes parâmetros “docker rm -f $(docker ps -a -q)”.

G. docker stats [container]

Exibe as informações do uso atual de hardware pelo container, onde [container] é o nome ou ID do container que deve ser monitorado. Comando muito útil para o monitoramento dos containers.

H. docker build

Comando utilizado para a criação de containers a partir de um Dockerfile em um determinado contexto.

I. Mapeamento de portas para o container

É possível mapear portas do servidor para uma porta específica do container. Para isso é utilizado o parâmetro “–p” na execução do container, como no comando “docker run –p 3306:3306 mysql”. Isto indica que a porta 3306 do servidor é aberta e está mapeada para a porta 3306 do container iniciado.

5.5 DOCKERFILE

O Dockerfile é um arquivo de texto que contém instruções para a criação de imagens. Basicamente é especificado um conjunto de comandos para a criação de uma imagem que, posteriormente, pode ser utilizada para criação de containers. É possível criar novas imagens a partir de outras já existentes (TURNBULL, 2014).

Para efetivamente criar a imagem é necessário usar o comando “docker build”, referenciando o Dockerfile e com os parâmetros necessários. Para execução do comando também é necessário um contexto, que é basicamente o conjunto de arquivos existentes no diretório ou na URL que indica o Dockerfile. Todo o processamento de construção da imagem a partir do Dockerfile é feito no *daemon*, ou seja, quando o comando é iniciado, todo o contexto é enviado ao servidor Docker, para a criação da imagem. Por este motivo é aconselhado a criação dos Dokcerfiles a partir de diretórios vazios, para evitar que muita carga seja enviada ao servidor (DOCKER Inc., 2016).

6 IMPLEMENTAÇÃO

Para melhor entendimento dos conceitos abordados foi realizada uma atividade prática. O objetivo foi utilizar Docker para ter um container rodando um serviço de banco de dados MySQL que possa ser acessado. Para implementação foi utilizado um computador com o sistema operacional Windows 7, 4 GB de memória RAM e processador Intel Core i5.

6.1 INSTALAÇÃO DO DOCKER

Como o computador utilizado não possui uma versão de Linux instalado, é necessário executar o Docker a partir de uma máquina virtual. Com este intuito foi instalado o Docker Toolbox, que é um conjunto de ferramentas que instala o necessário para rodar o serviço em ambientes que não possuem sistema operacional Linux. O Docker Toolbox utiliza o VirtualBox para criar o ambiente simulado.

A instalação é simples e intuitiva. Instalando todos os componentes mostrados disponíveis no Docker Toolbox já é o suficiente para ter o ambiente rodando na

máquina. Ao abrir o terminal do Docker, é possível verificar em qual IP que ele está sendo executado. Como a execução é feita dentro de uma máquina virtual, pode-se acessar o Docker através de um IP. A Figura exibe o IP configurado no ambiente criado para a prática, que, neste caso, é 192.168.99.100.

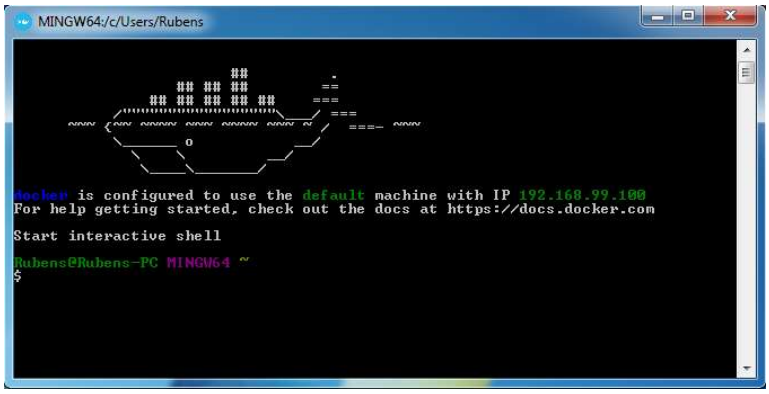

Figura 12: Inicialização do terminal Docker
Fonte: Elaborada pelo autor

Com a instalação do Docker ToolBox também foi instalado o Kitematic for Windows, que é um software para criação e gerenciamento de containers bem como para a configuração do Docker. A interação com o software se dá através de uma interface gráfica, o que facilita a usabilidade para os usuários não tão avançados. Conforme exibido na Figura 13, o software possui uma interface que possibilita o gerenciamento, não necessitando o uso do terminal para a maioria dos comandos. A versão do software utilizado ainda não estava em sua versão final, mas já se mostrou extremamente útil para o uso rápido dos containers.

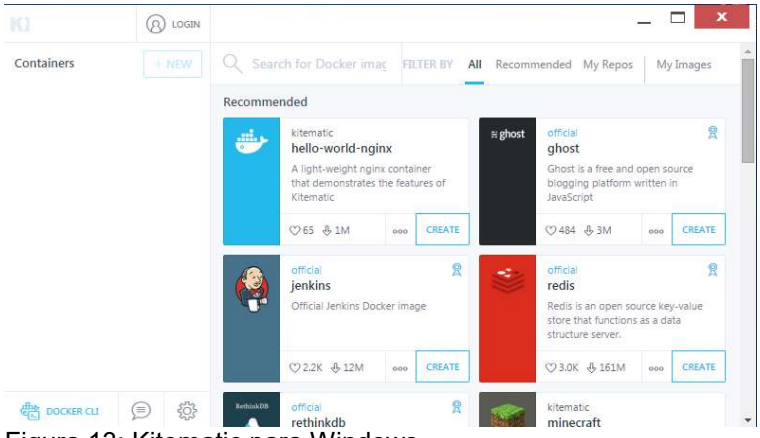

Figura 13: Kitematic para Windows
Fonte: Elaborada pelo autor

As visualizações de alguns dados de gerenciamento são mostradas de forma mais organizada, devido ao poder que a interface de usuário trás. O Kitematic também foi criado pela empresa Docker Inc. e é distribuído gratuitamente juntamente com o Docker Toolbox, para as plataformas Windows 7 ou superior e Mac OS X 10.8 ou superior (DOCKER Inc., 2016).

6.2 CRIAÇÃO DE CONTAINERS

Para criação e gerenciamento dos containers foi utilizada como principal ferramenta o terminal Docker, onde os comandos são dados por linhas de comando. A Figura exibe o terminal Docker.

Pode-se utilizar o comando “docker os -a” para verificar os containers existentes no ambiente. A 14 mostra que ainda não há nenhum container criado no ambiente.

Figura 14: Execução do comando para listar containers.
Fonte: Elaborada pelo autor.

Para criação de um container, é necessário ter uma imagem. No site do Docker Hub é possível encontrar as imagens previamente criadas e disponibilizadas para uso. No software Kitematic também é possível visualizar imagens disponíveis para uso.

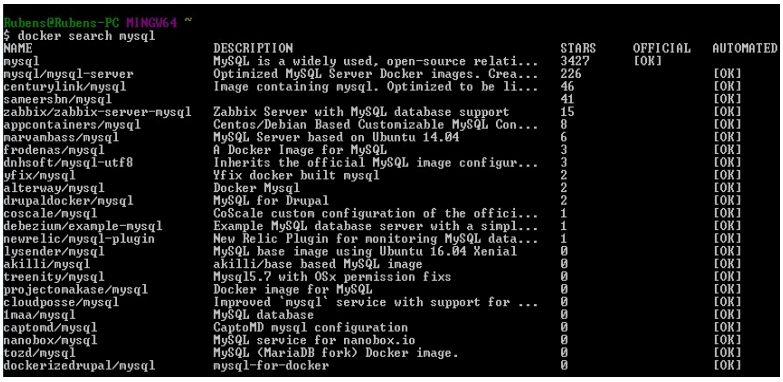

Figura 15: Listagem de imagens do MYSQL
Fonte: Elaborada pelo autor

Como o objetivo é executar um servidor de banco de dados, é necessária uma imagem que forneça este tipo de serviço. Já existe uma imagem criada para um banco de dados MYSQL, assim como para outros diversos tipos de serviços. Com o comando “docker search mysql” é possível verificar todas as imagens criadas com o nome passado na pesquisa. Conforme a 15, é possível verificar que há diversas imagens disponíveis, e, na primeira linha, é possível ver que a imagem “mysql” é a oficial criada. Esta é imagem que será utilizada.

É possível utilizar o comando “docker pull [image]” para efetuar o download da imagem, onde [image] é o nome da imagem que deve ser baixada, neste caso, “mysql”. Após executar o comando, um download da imagem e dos recursos necessários é iniciado.

Após finalizar o download da imagem foi necessário criar o container. É utilizado o comando “docker run [image]”, onde [image] é o nome da imagem, neste caso, “mysql”. O comando executado para a criação do container foi “docker run -p 3306:3306 --name mysql-container -e MYSQL_ROOT_PASSWORD=root -d mysql”. O parâmetro “-p” indica que a porta 3306 do host está aberta e deve ser mapeada na porta 3306 do container. O parâmetro “--name” dá nome ao container, neste caso, o nome dado foi “mysql-container”. Já o parâmetro “-e” é responsável por atribuir o valor a alguma variável. Neste caso, na variável “MYSQL_ROOT_PASSWORD” foi atribuído o valor “root”, que será a senha do usuário “root” da instância criada do MYSQL. E o parâmetro “-d” indica que o container irá ser executado em background, sem que seja necessário ficar conectado ao terminal. Por último, é possível ver o nome da imagem no comando, que é “mysql”. Na Figura , é possível ver o resultado do comando de criação do container.

Figura 16: Comando para criação do container MYSQL
Fonte: Elaborada pelo autor

O container foi criado e já está em execução. Para verificar a execução do container, foi utilizado o comando “docker os”, que exibe os containers que estão em execução. Pode também ser utilizado o parâmetro “-a”, opcionalmente, para mostrar não somente os containers em execução, mas todos do servidor Docker. A Figura 16,

exibe o resultado do comando, é possível perceber que só há um container em execução, com o nome de “mysql-container” e que foi criado a partir da imagem “mysql”. Alguns outros dados do container também são exibidos, como ID do container, quando foi criado, há quanto tempo está em execução, entre outros.

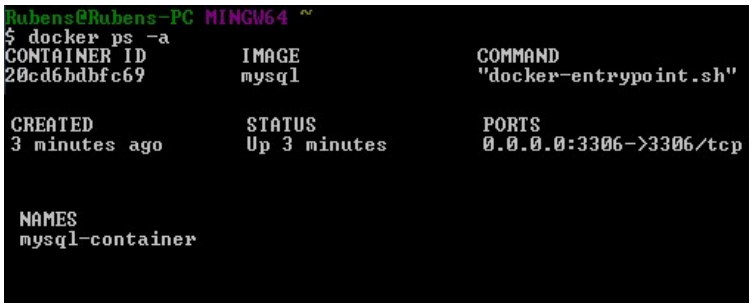

Figura 17: Execução do comando de listagem de containers
Fonte: Elaborada pelo autor

Através do software Kitematic é possível perceber que o container está criado e pode ser gerenciado. Algumas informações sobre o container também são exibidas no software.

Para fazer o monitoramento do container, é possível utilizado o comando “docker stats [container]”, onde [container] é o nome ou ID do container que deve ser monitorado. A Figura , exibe alguns dados do container em execução, como CPU utilizada, quantidade de memória RAM, entre outros. Este comando se fez útil no monitoramento dos containers.

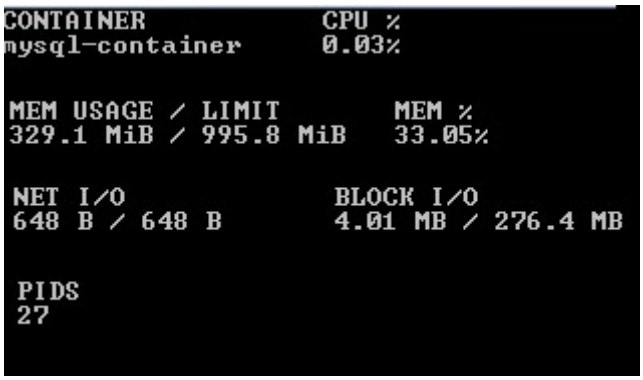

Figura 18: Resultado do comando “docker stats” para o container criado.
Fonte: Elaborada pelo autor.

Para se conectar à instância criada do MYSQL, foi utilizado um software chamado MYSQL Workbench. Este software foi criado especificamente para se

conectar a instâncias do MYSQL e está disponível para download gratuito no site do mesmo.

O servidor Docker, conforme já mencionado, está sendo executado no IP 192.168.99.100, e o container do MYSQL foi mapeado para a porta 3306. Logo, com estes dados, é possível se conectar à instância que está sendo executada no container. Após a conexão ter sido realizada, foi possível utilizar o MYSQL normalmente.

7 CONCLUSÕES

Computação em nuvem é atualmente um paradigma fundamental para a disponibilização de aplicações e serviços. Um dos pilares para a implementação deste paradigma é a virtualização. Cada modelo de virtualização possui propostas e objetivos diferentes, não sendo possível definir o melhor modelo disponível. Para a alternativa de virtualização também se tem o Docker, que é o foco desta pesquisa.

A instalação do Docker foi feita de maneira simples e prática, nenhum problema foi encontrado nesta etapa. A forma de uso do Docker também é simples e em poucos minutos foi possível ter uma instância do MYSQL configurada e executando na máquina local.

O tipo de ambiente mais indicado para execução do Docker, é um ambiente Linux, pois nos demais é necessária uma máquina virtual para obter um kernel Linux e então executar o Docker, o que causar uma perda de desempenho, devido à camada adicional da máquina virtual.

Apesar da prática apresentada ser simples, foi possível perceber o poder do Docker, pois possui alta escalabilidade e fácil gerenciamento dos containers. Caso fosse necessário, por exemplo, um container de um servidor web, com poucos comandos seria possível ter este servidor executando e disponível para uso.

Na parte de desenvolvimento de aplicações, o Docker também é uma potencial escolha, pois é muito simples e rápido iniciar todo o ambiente. Neste caso o desenvolvedor não precisaria ficar horas configurando e aprendendo a configurar diversos software e ambientes, simplesmente poderia ser criada uma imagem ou um Dockerfile e usá-los em todos os terminais de desenvolvimento.

É possível também usar a tecnologia para o espelhamento de ambientes, tendo em vista que, se todos os servidores usarem a mesma imagem ou Dockerfile para criação dos containers, todos terão o mesmo ambiente de execução, sendo diferenciados pela sua capacidade física.

Embora o uso inicial do Docker seja feito de forma facilitada, é necessário um conhecimento avançado para a aplicação de determinados conceitos, como: criação de Dockerfiles, criação de um *Registry* privado e o uso de ferramentas avançadas de monitoramento.

A documentação fornecida pelo próprio site do Docker é ampla e facilmente entendida. A comunidade Docker também é muito ativa na Internet, com diversos fóruns e tutoriais que auxiliam no aprendizado.

REFERÊNCIAS

ALECRIM, Emerson. **O que é virtualização e para que serve?** Disponível em: <http://www.infowester.com/virtualizacao.php>. Acesso em: 21 nov. 2016.

ALVES, Luiz Augusto da Silva. **Tipos de virtualização**. 2008. Disponível em: <http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2010_2/luizaugusto/tipos_arq.html>. Acesso em: 21 nov. 2016.

ALVES, Luiz Augusto da Silva. **Tipos de Virtualização**. 2008. Disponível em: <http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2010_2/luizaugusto/tipos_tec.html>. Acesso em: 21 nov. 2016.

BUCHANAN, Jim. **Regras de compliance que é preciso saber antes de entrar em cloud**. 2013. Disponível em: <http://computerworld.com.br/gestao/2013/07/05/regras-de-compliance-que-e-preciso-saber-antes-de-entrar-em-cloud>. Acesso em: 13 dez. 2016.

BURGER, Thomas. **The Advantages of Using Virtualization Technology in the Enterprise**. Disponível em: <https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise>. Acesso em: 21 nov. 2016.

CRMG - Network & Security. **Virtualização de Servidores**. 2014. Disponível em: <http://www.crmg.com.br/virtualizacao-de-servidores>. Acesso em: 14 dez. 2016.

DOCKER DOCS. **Docker documentation.** 2016. Disponível em: <https://docs.docker.com/>. Acesso em: 21 nov. 2016.

GOMES, Rafael; SOUZA, Rodrigo. **Docker - Infraestrutura como código, com autonomia e replicabilidade**. Salvador: Universidade Federal da Bahia.

HARRYS, Torry. **Cloud computing**: An Overview. New Jersey: Torry Harris. Disponível em: <http://www.thbs.com/downloads/Cloud-Computing-Overview.pdf>. Acesso em: 21 nov. 2016.

LEWIS, Grece. **Basics About Cloud Computing**. Pittsburgh: Carnegie Mellon University, 2010. Disponível em: <http://resources.sei.cmu.edu/asset_files/WhitePapßßer/2010_019_001_28877.pdf>. Acesso em: 21 nov. 2016.

VERAS, Manoel; CARISSIMI, Alexandre. **Virtualização de Servidores**. Rio de Janeiro: Rede Nacional de Ensino e Pesquisa - Rnp, 2015.

VERAS, Manoel. **Computação em nuvem**. Rio de Janeiro: Brasport, 2015.

VERAS, Manoel. **Virtualização**: Tecnologia Central do Datacenter. Rio de Janeiro: Brasport, 2016.

MATTOS, Diogo Menezes Ferrazani. **Virtualização**. 2008. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/OqueohypervisorouVMM(VirtualMachineMonit.html>. Acesso em: 21 nov. 2016.

MATTOS, Diogo Menezes Ferrazani. **Virtualização**: VMWare e Xen. Rio de Janeiro: Urfj, 2008. Disponível em: <http://www.gta.ufrj.br/grad/08_1/virtual/artigo.pdf>. Acesso em: 21 nov. 2016.

ORACLE. **About Linux Containers**. Disponível em: <https://docs.oracle.com/cd/E37670_01/E37355/html/ol_about_containers.html>. Acesso em: 21 nov. 2016.

SANTHOSH, Ramya. **What is Docker, Difference between Docker and VM, Installation of Docker and its usage**. 2016. Disponível em: <https://techglimpse.com/docker-installation-tutorial-centos/>. Acesso em: 14 dez. 2016.

STAMEY, Laura. **IaaS vs. PaaS vs. SaaS Cloud Models**: Differences & Examples. 2017. Disponível em: <http://www.hostingadvice.com/how-to/iaas-vs-paas-vs-saas/>. Acesso em: 09 jul. 2017.

THOLETI, Bhanu P. **Learn about hypervisors, system virtualization, and how it works in a cloud environment**. 2011. Disponível em: <https://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/index.html>. Acesso em: 09 jul. 2017.

TURNBULL, James. **The Docker Book**. Creative Commons, 2014.

VITALINO, Jeferson Fernando Noronha; CASTRO, Marcus André Nunes. **Descomplicando o Docker**. Rio de Janeiro: Brasport, 2016.

WANG, Chenxi. **Containers 101:** Linux containers and Docker explained: A brief introduction to lightweight, portable, flexible Docker containers and why developers love them. 2016. Disponível em: <http://www.infoworld.com/article/3072929/linux/containers-101-linux-containers-and-docker-explained.html>. Acesso em: 21 nov. 2016.