Curso: Sistemas de Informação	
Disciplina: Entrega de Software	
Professor: Luiz Gustavo Dias	**Tipo:** Estudo Complementar
Objetivo: Entender sobre contêiner e docker	

DOCKER

Introdução

Considere um sistema constituído por diferentes serviços e componentes, como servidor web, banco de dados, mensageria, entre outros. A incompatibilidade de componentes, serviços e tecnologias à medida que um determinado sistema é atualizado gera conflitos entre os serviços que necessitam de bibliotecas específicas com versões distintas. Além disso, muitas vezes se requer um tempo significativo para a (re)configuração do sistema. Sendo assim, cada serviço precisa resolver suas dependências, principalmente com relação às bibliotecas com o sistema operacional subjacente. Nesse sentido, há necessidade de uma solução que viabilize a modificação ou a substituição dos componentes conflitantes sem afetar os demais e, portanto, sem comprometer o sistema como um todo. Essa solução é conhecida como Docker.

1. Docker

O Docker é uma solução que viabiliza a modificação ou a substituição de componentes conflitantes em um determinado sistema sem afetar os demais componentes e, principalmente, sem comprometer o sistema como um todo (SILVA, 2016). O Docker é um software e contêiner que fornece uma camada de abstração e automação, baseado, originalmente, em sistemas Unix. A Figura 1, a seguir, apresenta a estrutura de um contêiner.

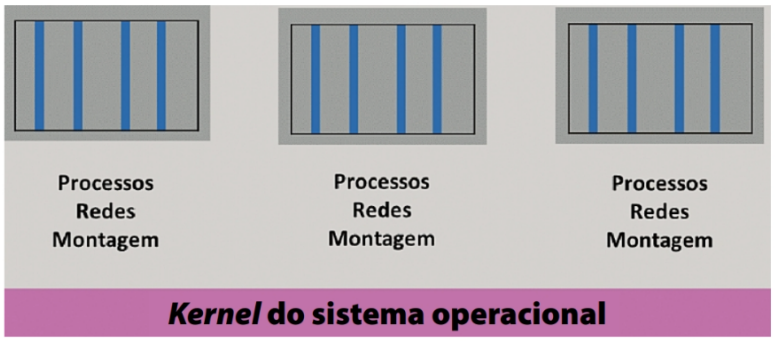

Figura 1. Estrutura básica de um contêiner.

Com base na Figura 1, pode-se perceber que os contêineres são ambientes completamente isolados de toda infraestrutura subjacente. Eles viabilizam processos próprios, isto é, possuem seus próprios serviços, sua própria interface de rede e sua própria montagem. Os contêineres são similares a máquinas virtuais, porém compartilham o mesmo kernel do sistema operacional (TANENBAUM; BOS, 2016) e possuem uma camada de software acima, chamada Docker.

Como visto, o Docker é uma solução baseada em contêineres originalmente baseada em sistemas Unix, cujo principal objetivo é empacotar aplicações em contêineres e enviá-los e executá-los em qualquer lugar, a qualquer momento, quantas vezes for necessário. Essa é uma das principais diferenças entre Docker, máquinas virtuais e contêineres, que serão abordados de forma mais detalhada no decorrer deste capítulo. Visando a apresentar as relações e diferenças entre esses conceitos, a Figura 2, a seguir, apresenta a estrutura básica de um Docker e de um contêiner.

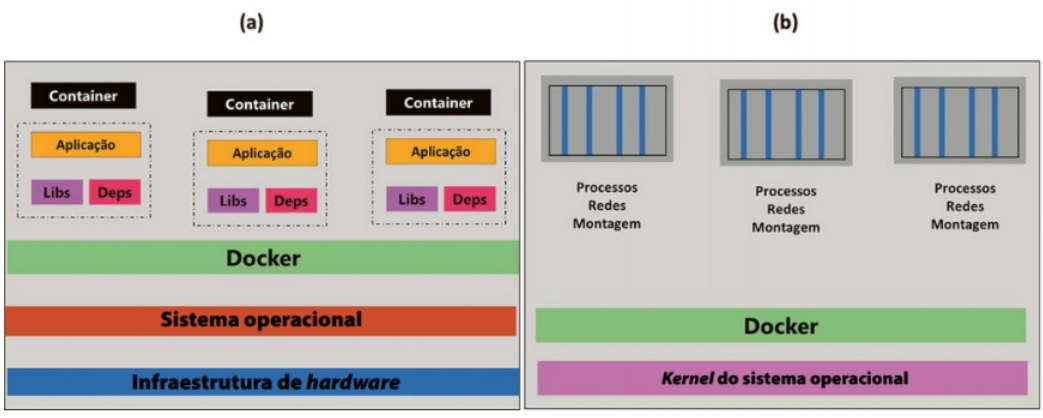

Figura 2. Relações e diferenças entre um (a) Docker e um (b) contêiner.

Conforme apresentado na Figura 2a, o Docker apresenta uma infraestrutura de hardware subjacente e, em seguida, o sistema operacional. A camada do Docker localizada acima do sistema operacional possibilita a execução de cada componente (ou aplicações) que constitui o sistema em um contêiner separado, com suas próprias bibliotecas e respectivas dependências, todas no mesmo sistema operacional, porém em contêineres separados. Já na Figura 2b, pode-se observar que os contêineres são ambientes completamente isolados, têm seus processos, suas próprias interfaces de rede e suas próprias montagens. No entanto, todos os contêineres compartilham o mesmo kernel do sistema operacional em que estão sendo executados, o qual é responsável por interagir com a infraestrutura de hardware subjacente.

Na camada acima, tem-se o Docker, por meio do qual é possível executar cada componente em um contêiner separado, com suas próprias bibliotecas e dependências, todas na mesma máquina virtual e no mesmo sistema operacional, porém em contêineres distintos. Por essa razão, é importante compreender as relações e diferenças entre contêineres e máquinas virtuais, apresentadas na Figura 3.

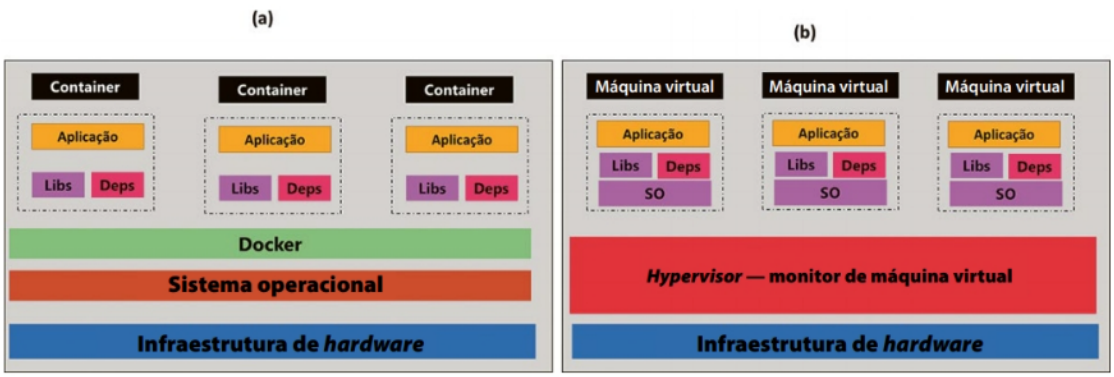

Figura 3. Relações e diferenças entre (a) contêineres e (b) máquinas virtuais.

Na Figura 3b, pode-se perceber uma camada mais baixa de infraestrutura de hardware, uma camada superior denominada monitor de máquina virtual, também conhecida como hypervisor, e, em seguida, cada máquina virtual supervisionada (TANENBAUM; BOS, 2016). Cada máquina virtual (VM, virtual machine) possui seu próprio sistema operacional (SO), bibliotecas (Libs) e dependências (Deps) dentro dela. Esse aspecto pode ocasionar uma sobrecarga no sistema/aplicação quando considerada uma maior utilização dos recursos subjacentes. Já na Figura 3a, pode-

se observar a infraestrutura de hardware subjacente e o Docker instalado no sistema operacional, em que o Docker gerencia os contêineres que são executados com suas próprias aplicações, bibliotecas e dependências independentes.

Além da vantagem sobre a utilização do sistema, outra vantagem importante da utilização de um Docker é que o processo de configuração ocorre apenas uma vez, isentando que o sistema seja reconfigurado ou modificado a cada atualização ocorrida no sistema em questão. Portanto, cada uma das estratégias de virtualização possui vantagens e desvantagens em diferentes aspectos, principalmente em termos de utilização, tamanho e inicialização (Figura 4).

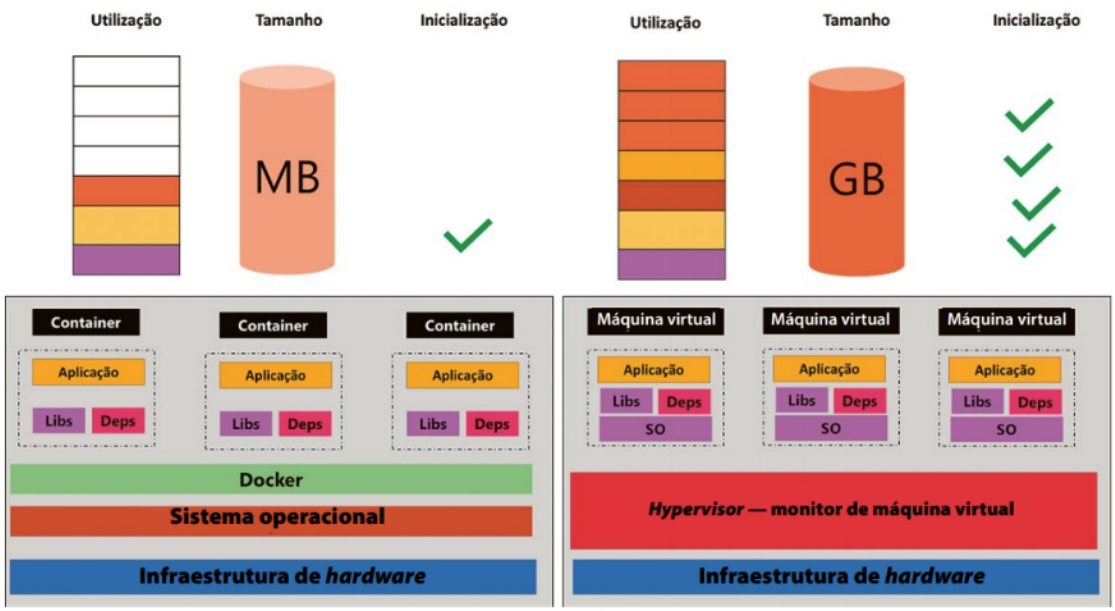

Figura 4. Contêineres vs. máquinas Virtuais: inicialização, tamanho e utilização.

A comparação apresentada na Figura 4 considera que existem vários sistemas operacionais virtuais e kernels executados em VMs, as quais, por sua vez, acabam consumindo mais espaço em disco (tamanho). Desse modo, as VMs tornam-se mais pesadas nesse contexto, de modo que, por exemplo, o tamanho de uma VM pode requerer alguns gigabytes, ao passo que os Dockers são considerados mais leves por requererem apenas alguns megabytes de armazenamento. Esse aspecto também permite que os Dockers apresentem uma inicialização mais leve, ou seja, eles podem ser inicializados em questão de segundos, ao passo que as VMs podem demorar

alguns minutos. Em contrapartida, as VMs provêm um desacoplamento completo do sistema operacional e do hardware subjacente.

Como existem vantagens e desvantagens em ambas as estratégias, atualmente é utilizada uma solução que acopla as duas estratégias em um mesmo ambiente, permitindo, assim, explorar as vantagens de cada uma delas em um mesmo objetivo (Figura 5).

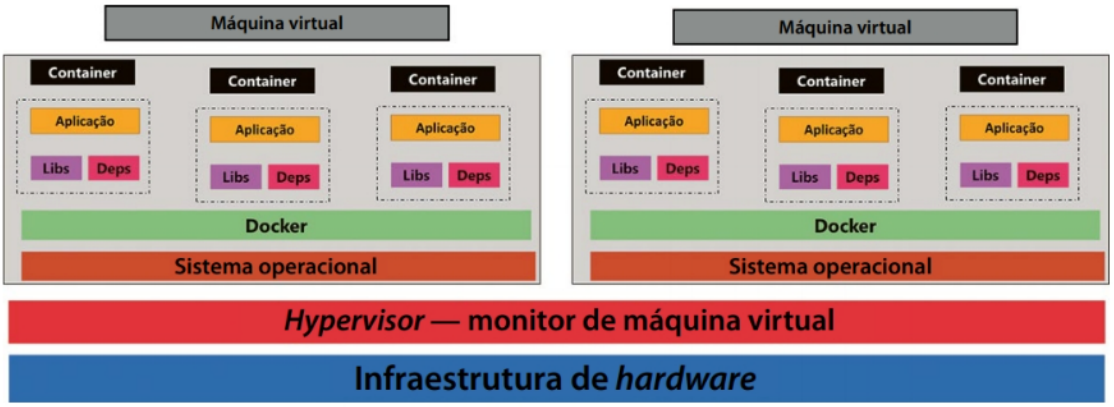

Figura 5. Estratégia de utilização de contêineres e máquinas virtuais.

A Figura 5 representa a estratégia mais condizente com o cenário atual, com milhares de contêineres em execução dentro de VMs gerenciados por Dockers hosts. Dessa forma, é possível explorar os benefícios das duas estratégias em uma única solução, isto é, as vantagens da virtualização em oferecer a flexibilização de montar e desmontar contêineres sob demanda (escalonamento rápido) aliadas aos benefícios do Docker, que propicia o gerenciamento de diferentes aspectos, como, por exemplo, aplicativos, bibliotecas, dependências e conflitos.

Por fim, salienta-se que Dockers, contêineres e virtualização são estratégias que se complementam, visando a suprir diferentes desafios dos sistemas distribuídos atuais, como, por exemplo, compatibilidade entre tecnologias, componentes e dependências de bibliotecas. A estratégia de utilização de contêineres e VMs tem sido amplamente utilizada nos sistemas distribuídos atuais, de modo a usufruir dos benefícios das três estratégias em um único ambiente.

2. Arquitetura do Docker

Primeiramente, foram apresentados os principais conceitos, bem como as características e vantagens do uso do Docker no desenvolvimento de DevOps (KIM et al., 2018). A fim de finalizar essa visão geral do Docker, a Figura 6, a seguir, apresenta a arquitetura dessa estratégia.

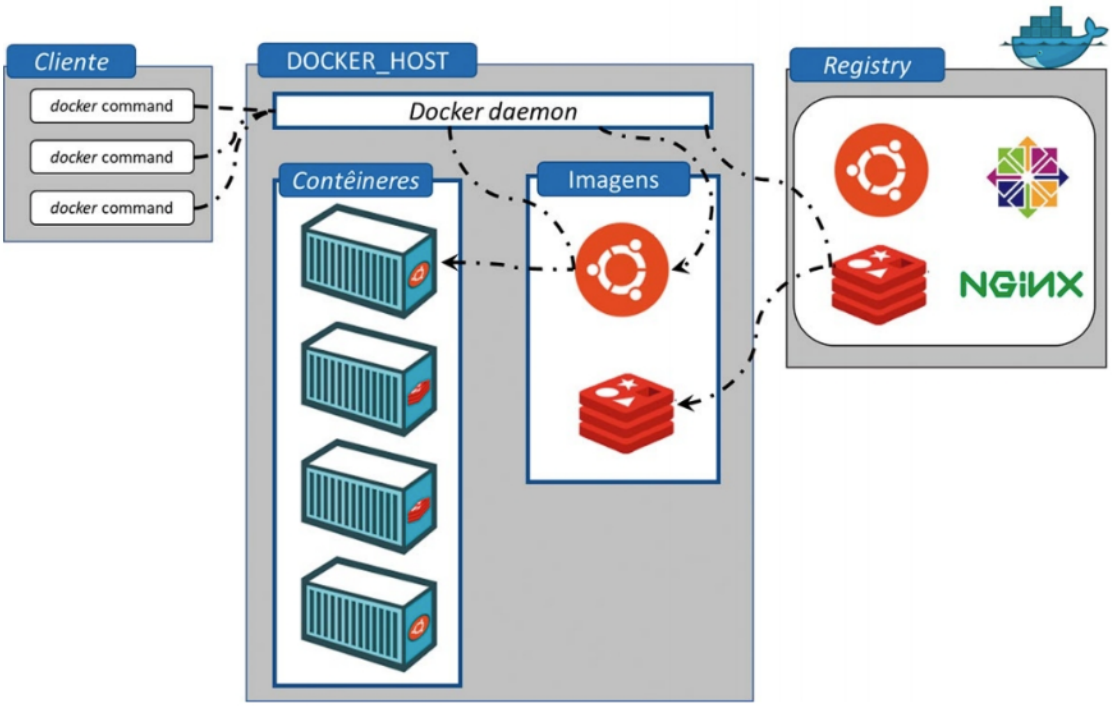

Figura 6. Arquitetura do Docker.
Fonte: Adaptada de Docker (2020).

Com base na Figura 6, é importante destacar que o Docker utiliza uma arquitetura cliente–servidor. O cliente Docker se comunica com o daemon Docker, que faz o trabalho pesado de construção (comando docker builid), execução e distribuição de seus contêineres Docker. O Docker daemon (também conhecido como dockered) escuta as solicitações da API Docker e gerencia objetos Docker, como imagens, contêineres, redes e volumes. O daemon também pode se comunicar com outros daemons para gerenciar os serviços Docker. Além disso, o cliente Docker e o daemon podem ser execu-tados no mesmo sistema, assim como é possível conectar um cliente Docker a um daemon remoto do Docker. O cliente Docker e o daemon se comunicam usando uma API REST, por soquetes (neste exemplo, UNIX) ou uma interface de rede. Um registro Docker armazena imagens Docker. O Docker Hub é um

registro público que qualquer pessoa pode usar, e o Docker está configurado para procurar imagens no Docker Hub por padrão, sendo possível executar seu próprio registro privado. A seguir, serão apresentados os principais pré-requisitos e etapas para a instalação do Docker com base no Docker Hub.

3. Incorporando o Docker

Antes de apresentar os principais passos para a instalação do Docker, é importante apresentar uma visão geral do Docker de alto nível e ressaltar a necessidade desse software neste contexto (DOCKER, 2020). A Figura 7, a seguir, apresenta um exemplo de um sistema hipotético atual com aplicações, componentes e serviços com e sem contêineres e a camada de software Docker.

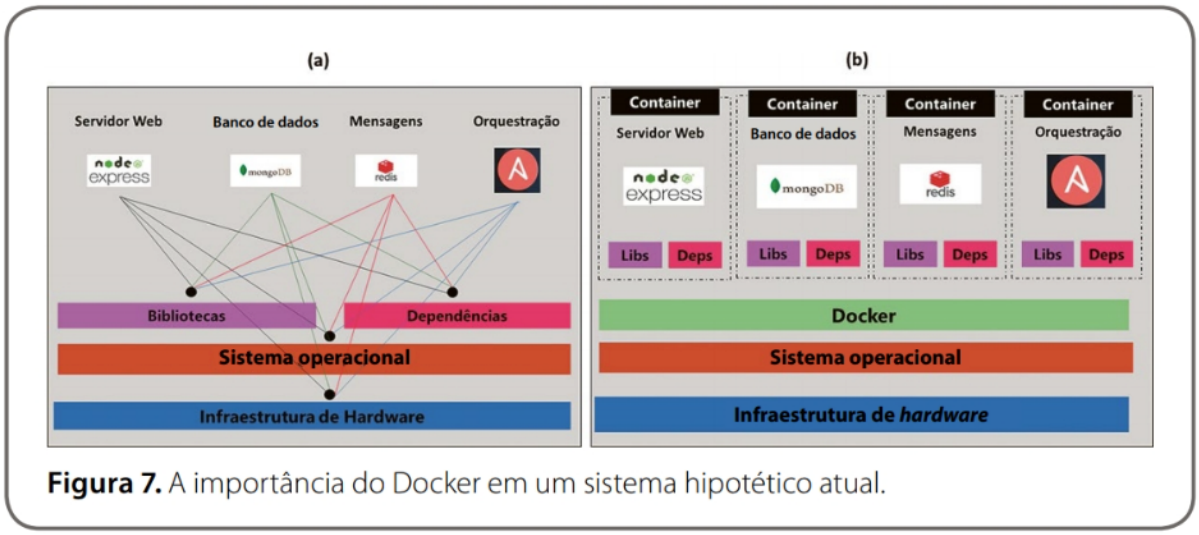

Figura 7. A importância do Docker em um sistema hipotético atual.

Na Figura 7, é apresentado um sistema hipotético, com e sem contêineres e Docker, constituído dos seguintes componentes: um servidor web com Node Express, banco de dados com MongoDB, sistemas de mensagens com Redis e serviço de orquestração com Ansible. Na Figura 7a, pode-se observar uma série de conflitos, principalmente em termos de compatibilidade de todos os serviços, aplicações e componentes com todas as camadas subjacentes, desde bibliotecas e dependências, passando pelo sistema operacional até a camada mais baixa de infraestrutura de hardware (bare metal). Dessa forma, é nítido que há necessidade de se garantir que todos os diferentes e distintos serviços, componentes e aplicações sejam compatíveis com a versão do sistema ope-racional. Na Figura 7b, por sua vez, é possível constatar a real importância do Docker e dos contêineres quando da execução de cada

componente em contêineres (containerize applications) com suas próprias dependências e bibliotecas, todos em contêineres separados, porém na mesma VM e no mesmo sistema operacional.

Em termos práticos, o Docker é necessário para resolver alguns aspectos significativos que fazem parte dos sistemas atuais, os quais envolvem distintas tecnologias e diversas aplicações (de ponta a ponta do sistema) e refletem problemas de compatibilidade e dependências de bibliotecas, tempos longos de configuração dos ambientes e ambientes heterogêneos de desenvolvimento. Por meio dos contêineres, torna-se possível modificar ou alterar esses componentes sem afetar os outros, ou até mesmo o sistema como um todo. A configuração desses ambientes (também conhecidos como contêineres dockers) não é uma tarefa fácil, pois eles encontram-se em nível de abstração muito baixo, porém a camada Docker oferece uma ferramenta de alto nível, com várias funcionalidades, tornando o acesso, o manuseio e o funcionamento básico dessa estratégia mais fácil. Dessa forma, é importante ressaltar que a configuração de um Docker é extremamente importante nesse contexto, a fim de oferecer aos desenvolvedores alternativas simples de manipulação, desenvolvimento e gerência de um sistema, com comandos simples após a sua instalação.