Docker: O advento da Conteinerização

Jorge Viegas

Bacharelado em Ciência da Computação – Universidade do Vale do Rio dos Sinos (UNISINOS) – Campus São Leopoldo
93022-718 – São Leopoldo – RS – Brasil

jorgegv@edu.unisinos.br

Resumo. *Este artigo apresenta um breve estudo sobre como funciona o modelo de Conteinerização, modelo de virtualização muito utilizado depois da criação do Docker, a ferramenta em destaque no artigo. O foco do artigo é estudar as implementações da API de Containers contidas no Kernel do Linux, bem como suas implicações em outros sistemas operacionais. A comparação entre as máquinas virtuais e containers também serão aboradadas.*

1. Introdução

O conceito de conteinerização surgiu com a necessidade de transportar com segurança e confiabilidade aplicações de um ambiente computacional para outro. Com o advento da Computação em Nuvem, este tipo de problema se faz recorrente nos modelos computacionais modernos.

O Docker pode ser definido como um conjunto de ferramentas e como o próprio "ecossistema" dos Containers no ciclo de vida e de implementação de uma aplicação. Suas funcionalidades são baseadas nos conceitos de Linux Containers, que foram incorporados ao Kernel do Linux recentemente, cerca de dez anos.

2. Containers

Um Container é um conjunto isolado de processos, no contexto do Docker, criado a partir de uma Imagem. Uma Imagem é um "pacote" leve e independente, constituído do código-fonte de uma aplicação e de todas as dependências de que ele necessita para ser executado. Os containers podem ser considerados uma abstração no nível de aplicação que agrupa o código, suas dependências, binários, bibliotecas e arquivos de configuração em tempo de execução. Ao utilizar Containers, conseguimos abstrair a preocupação com Sistemas Operacionais e Infraestrutura do desenvolvimento de aplicações.

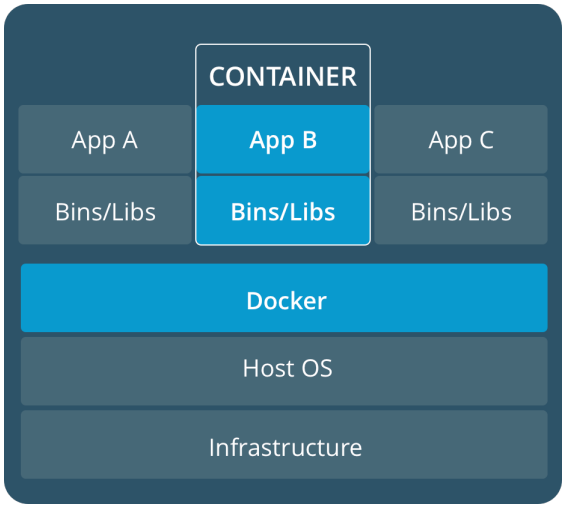

Figura 1. Onde um Container atua em um Sistema Operacional?

Fonte: https://www.docker.com

Containers sempre aparecem muito ligados ao Docker, por ser a principal e mais difundida aplicação de conteinerização. Porém, o conceito de Container foi criado e adicionado ao Kernel do Linux há mais de 10 anos, com o projeto LXC (Linux Containers). Alguns outros Sistemas Operacionais e grupos de pesquisa também já contavam com técnicas de virtualização no nível do SO antes mesmo do Linux, como por exemplo o Solaris Containers e o FreeBSD jails.

2.1. Open Container Initiative

Com todas as opções de Containerização, uma padronização das especificações de um Container se fez necessária. Em 2015 foi criada a Open Container Initiative, baseada nas linhas de pesquisa e apoio da Linux Foundation. O propósito era criar uma especificação e padrões de criação e formato de Containers para todas as plataformas computacionais. A base do estudo e criação dos padrões foi a tecnologia do Docker. Alguns patrocinadores deste projeto são as gigantes: Google, IBM, Microsoft e Oracle.

2.2. Imagens Docker

No Docker, um container é uma instância em tempo de execução de uma Imagem. Uma Imagem representa um pacote executável e transportável de uma aplicação, incluindo todo seu código-fonte, dependências e arquivos de configuração. Um Container é onde a aplicação em si será executada, uma representação da imagem em memória. Os containers são completamente isolados do sistema hospedeiro, conseguindo acessar apenas arquivos e portas se lhe for concedido o acesso. Os containers conseguem executar a aplicação acessando recursos nativos do Sistema Operacional, ao contrário das Máquinas Virtuais, que acessam o hospedeiro apenas através do Hypervisor.

3. Linux Containers

Linux Containers (ou LXC) se refere à API de containerização nativa do Linux. Os Linux Containers são fortemente baseados no conceito de "namespaces" e grupos de controle (cgroups) do Linux. Até mesmo por esse motivo, o funcionamento do Docker em outros sistemas operacionais que não tenham base UNIX é prejudicado. A tecnologia do Docker não substitui o LXC, muito pelo contrário, o Docker é baseado e adiciona algumas funcionalidades de alto-nível ao controle de Containers do Linux.

3.1. Namespaces

Um namespace encapsula recursos globais do sistema em uma abstração que faz com que os processos dentro deste namespace tenham seus próprios recursos globais de sistema isolados. Mudanças nos recursos globais (rede, sistema de arquivos, processos) são visíveis apenas para outros processos que são membros do mesmo namespace. Um dos usos de m namespace é implementar Containers. Linux Containers utilizam a implementação de namespaces para prover isolamento entre os processos. Isso permite que sejam criados processos com namespaces diferentes do padrão, fazendo com que os processos dentro do namespace acessem os recursos globais sem saber que está "encapsulado" dentro de um grupo específico de acesso.

O Linux oferece seis tipos diferentes de Namespaces: IPC, Network, Mount, PID, User e UTS. Ainda há um sétimo tipo de namespace chamado "CGROUP", que é um caso especial por também ser usado no controle de usuários padrão do Linux. Quando criamos um novo processo através da função clone(), podemos especificar através de flags (parâmetros) quais serão os namespace de que este processo fará parte. Se utilizarmos por exemplo, o comando clone() com a flag CLONE_NEWIPC, o Kernel criará um novo processo e também um novo namespace de tipo IPC (Comunicação entre processos). Os processos criados dentro de um namespace de IPC são visíveis somente para os processos que também fazem parte do mesmo namespace, não podendo receber mensagens "externas", encapsulando assim a comunicação entre processos.

4. Conteinerização x Virtualização

Conteinerização e Virtualização são dois tipos de virtualização que são aplicados em duas diferentes camadas se um sistema. A conteinerização atua no nível de aplicação, utilizando todos os recursos do kernel do Sistema Operacional hospedeiro, enquanto a virtualização adiciona complexidade com o Hypervisor, que é a camada de software responsável por controlar e realizar a comunicação entre a Máquina Virtual e o Hardware ou entre a Máquina Virtual e o próprio Sistema operacional hospedeiro. Ambos conceitos podem ser aplicados ao mesmo tempo, tendo uma Máquina Virtual rodando os Containers ao invés do sistema hospedeiro.

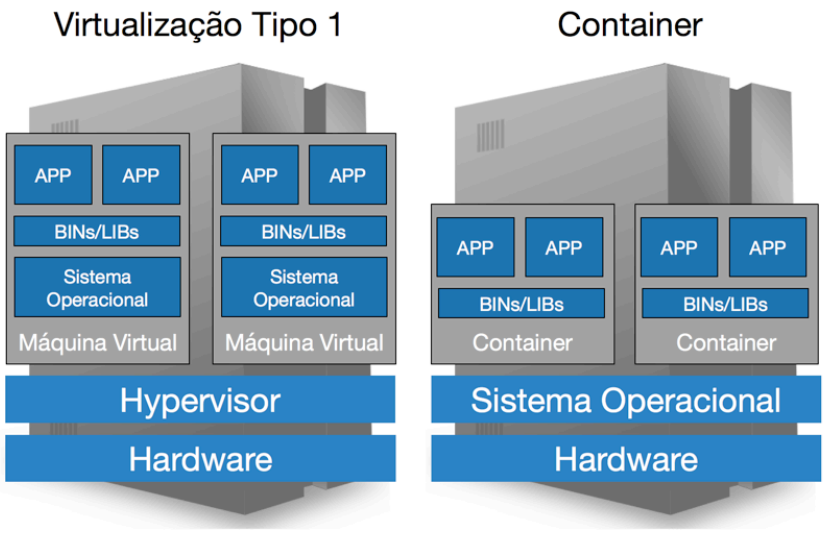

Figura 2. Diferenças das camadas de um Ambiente Computacional com Virtualização e Containers

Fonte: Universidade Federal de Lavras (UFLA)

2.1. Vantagens e Desvantagens

As duais principais diferenças entre os dois tipos de Virtualização são a performance e o isolamento, principalmente pela arquitetura distinta. Em performance, a Conteinerização tem vantagem por utilizar recursos do Kernel do Sistema Operacional hospedeiro, enquanto uma Máquina Virtual adiciona toda a complexidade de ter um Sistema Operacional inteiro, muitas vezes não usando nenhum recurso de software do hospedeiro. Por esse motivo, os arquivos de Containers são muito menores em relação às imagens de Virtualização, que ao invés de conterem apenas a aplicação, contém o SO inteiro. O acesso ao Hardware também fica prejudicado, uma vez que para acessá-lo a VM deve obrigatoriamente acessar o Hypervisor, enquanto os Containers utilizam o acesso padrão do SO hospedeiro.

Uma das desvantagens dos Containers em relação às VMs é o isolamento da aplicação, pois por rodar no mesmo Kernel, acessar e compartilhar os mesmos recursos demanda controle e isolamento complexos, enquanto nas VMs o próprio Hypervisor se encarrega do controle de acesso e isolamento em relação ao Hospedeiro.

Referências

Felter, Wes., Ferreira, A., Rajamony, R. and Rubio, J. “An Updated Performance Comparison of Virtual Machines and Linux Containers”. Julho 2014, IBM Research Division, Austin Research Laboratory, Disponível: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

Linux Manual Pages. Disponível: http://man7.org/linux/man-pages/man2/clone.2.html. Acesso: novembro/ 2017.

Docker Official Documentation. Disponível: https://docs.docker.com. Acesso: novembro/ 2017.

Linux Containers Official Website. Disponível: https://linuxcontainers.org/. Acesso: novembro/ 2017.