Introdução

Digamos que você acabou de criar um novo app e quer deixar o mundo todo usar. Como é possível fazer isso?

Primeiramente, o app que você criou está rodando no seu próprio computador. Isso é legal, mas não é bom que as pessoas acessem o seu app *através* do seu computador. Imagina só? Você quer desligá-lo, mas não pode pois outras pessoas dependem dele.

Essa discussão já tivemos: precisamos de um servidor. Entretanto, isso gera outro problema, especialmente para *softwares* mais complexos: **O servidor é um computador diferente do seu.**

Ou seja, ele possivelmente está rodando outro *sistema operacional*, ou talvez em outra versão, talvez não tenha todos os programas instalados que o seu tem, ou em versões diferentes...

Isso pode dar uma grande dor de cabeça para instalar todos os requisitos que o seu app novo precisa. Isso também cria uma dependência no **servidor no qual você está rodando o app**, no sentido de 'se não for esse servidor, não funciona'.

Para isso, foi criado o conceito de **Conteinerização**.

O que é conteinerização?

Essencialmente, é o conceito de **isolar** uma aplicação das outras, de forma que tudo o que ela precise esteja dentro do *container*, **e nada mais**.

Além de um grande ganho na segurança do sistema (i.e. se alguém comprometer a segurança da aplicação, ela ganha acesso total ao container, mas não à maquina inteira), essa prática facilita muito a distribuição da aplicação em um ou mais servidores. Isso porque ao invés de distribuir a aplicação aos servidores, é possível distribuir **um container com a aplicação**, contendo todos os requisitos do software junto ao próprio software.

A diferença entre um container e uma VM

A primeira pergunta que eu me fiz quando li essa ideia foi "Peraí, qual a diferença entre um container e uma máquina virtual?" já que os conceitos são muito parecidos.

Para alguém que está usando o produto final cegamente, é completamente possível imaginar containeres de linux como micro-VMs, e trabalhar com isso.

Mas somos hackers. Queremos saber como as coisas funcionam. E antes de tudo, precisamos falar

um pouco sobre como um sistema operacional funciona.

Todo sistema operacional tem um núcleo. O Windows tem o kernel NT, o OS X e o iOS usam o kernel Darwin, o GNU/Linux e o Android usam o Linux. (Kernel é a palavra inglesa de núcleo, usarei kernel e núcleo intermitentemente)

O kernel é responsável por fazer as partes mais essenciais de um sistema operacional, como por exemplo:

- O sistema de arquivos (ler e escrever em arquivos, e como transcrever isso para protocolos que um HD ou SSD entende);
- Lidar com todas as redes do computador (conversar com a placa de Wi-Fi, descobrir para qual IP é preciso mandar o pacote);
- Administração de processos (multi-threading, escalonamento de processos)

A parte mais superficial do sistema operacional lida com a interface do usuário para esse núcleo. Em outras palavras, transcrever cliques em um arquivo ou toques em uma tela para operações nucleares.

Dito isso, **O que é uma VM?**

Uma VM tem como sua parte principal uma camada que traduz instruções entre um núcleo de sistema operacional e outro. Essa camada é chamada de **Hypervisor**, várias outras responsabilidades como salvar o estado da máquina virtual em um arquivo, mas a responsabilidade principal é **transcrever comandos dados por um programa dentro da máquina virtual** para **comandos no sistema operacional externo**, de maneira segura.

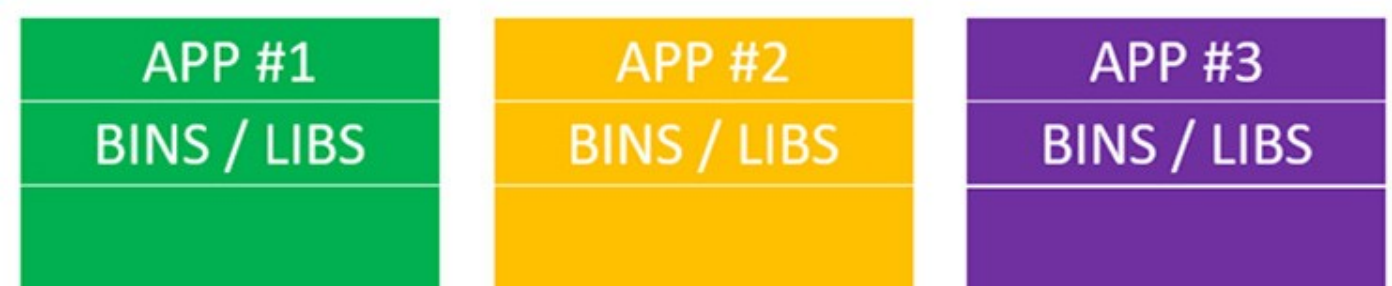
Isso implica que para *n* máquinas virtuais rodando num computador, há *n + 1* núcleos de sistema operacional rodando, junto a no mínimo 1 Hypervisor para transcrever instruções das máquinas virtuais para a máquina real.

Containeres fazem isso um pouco diferente. Ao invés de executar vários núcleos de sistema operacional e transcrever instruções entre eles, um container executa instruções utilizando **o mesmo kernel** que o da máquina física.

Ou seja, para *n* containeres rodando na mesma infraestrutura, há apenas *1* núcleo de sistema operacional compartilhado entre os containeres. E mesmo assim, o administrador de containeres (os quais vamos falar nesse capítulo, que incluem **dockerd**, **LXC**, **podman**, **rkt**, entre outros) dá um jeito de criar uma experiência isolada entre os containeres.

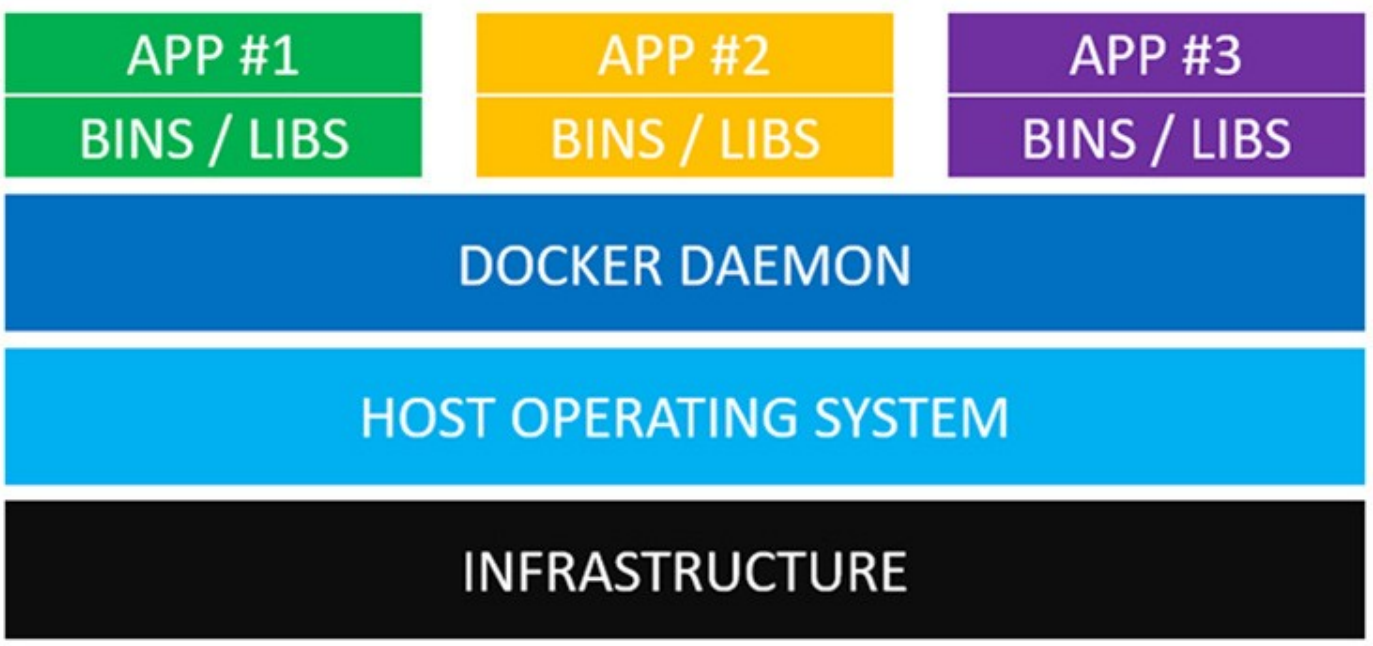
Para visualizar isso melhor, peguei os gráficos desse artigo.

Arquitetura de VMs



Note que há vários "Guest OS" sendo transcritos pelo Hypervisor para o sistema operacional da máquina física.

Arquitetura de Containeres (Docker, no caso)



Em contraste com a arquitetura de VMs, não há mais do que um sistema operacional rodando na infraestrutura. O daemon do Docker (`dockerd`) tem a responsabilidade de isolar as chamadas de sistema dos containeres, para que um não interfira na vida do outro.

Em termos práticos, Máquinas virtuais possibilitam rodar **núcleos** diferentes, o que possibilita coisas como Windows dentro de Linux (e vice-versa). Isso vem a um custo grande: máquinas virtuais são mais pesadas do que containeres, tanto em consumo de memória (mantendo dois sistemas operacionais em memória) quanto armazenamento (Uma VM básica pode ter uns 700MB de tamanho).

Um container possibilita criar ambientes isolados que usam o mesmo núcleo de SO do que a máquina anfitriã. Para rodar containeres de Docker em um Windows, por exemplo, o Docker executa secretamente uma VM de linux na qual o `dockerd` executa dentro. Com isso, os containeres são muito mais leves em termos de RAM, armazenamento e tempo de *startup* do que uma VM.

No nosso exemplo acima de construir um app, o ideal é construir um container linux que rodará numa máquina física GNU/linux. Nesse capítulo, vamos ver como.

Docker I: um prólogo prático

Revision #5
Created Sat, Aug 24, 2019 3:08 PM by razgrizone
Updated Tue, Sep 10, 2019 12:29 PM by Cainotis