Minicurso Docker

13 de junho de 2019

Gian Lucas Cavalcante de Lima

Lógica Sistemas de Informação

O que é Docker?

Docker é uma plataforma de código aberto, escrito em Go e criado pela Google, para facilitar os processos de desenvolvimento e implantação de uma aplicação.

O que é Docker?

Docker é uma plataforma de código aberto, escrito em Go e criado pela Google, para facilitar os processos de desenvolvimento e implantação de uma aplicação.

- O que o Docker faz é criar um ambiente sandbox, isolado do resto do sistema aonde uma aplicação vai rodar.

O que é Docker?

Docker é uma plataforma de código aberto, escrito em Go e criado pela Google, para facilitar os processos de desenvolvimento e implantação de uma aplicação.

- O que o Docker faz é criar um ambiente sandbox, isolado do resto do sistema aonde uma aplicação vai rodar.
- Chamado *Container*.

Como funciona?

Uma aplicação é encapsulada com suas dependências dentro de um container, aonde ela irá rodar isolada do resto do sistema.

Como funciona?

Uma aplicação é encapsulada com suas dependências dentro de um container, aonde ela irá rodar isolada do resto do sistema.

- não corre risco de sofrer interferências do meio externo.

Como funciona?

Uma aplicação é encapsulada com suas dependências dentro de um container, aonde ela irá rodar isolada do resto do sistema.

- não corre risco de sofrer interferências do meio externo.
- Similar a uma máquina virtual.

Então Container é uma Máquina Virtual?

Então Container é uma Máquina Virtual?

Containers não são máquinas virtuais, porém ambos possuam o mesmo objetivo.

Então Container é uma Máquina Virtual?

Containers não são máquinas virtuais, porém ambos possuam o mesmo objetivo.

Qual a diferença?

A principal diferença entre eles esta na sua arquitetura.

Então Container é uma Máquina Virtual?

Containers não são máquinas virtuais, porém ambos possuam o mesmo objetivo.

Qual a diferença?

A principal diferença entre eles esta na sua arquitetura.

- Numa Máquina Virtual é feita a emulação de toda uma máquina guest, desde o software ao hardware, em cima de uma máquina host.

Então Container é uma Máquina Virtual?

Containers não são máquinas virtuais, porém ambos possuam o mesmo objetivo.

Qual a diferença?

A principal diferença entre eles esta na sua arquitetura.

- Numa Máquina Virtual é feita a emulação de toda uma máquina guest, desde o software ao hardware, em cima de uma máquina host.
- Para fazer isso as VMs usam de um artifício chamado de Hypervisor.

Então Container é uma Máquina Virtual?

Containers não são máquinas virtuais, porém ambos possuam o mesmo objetivo.

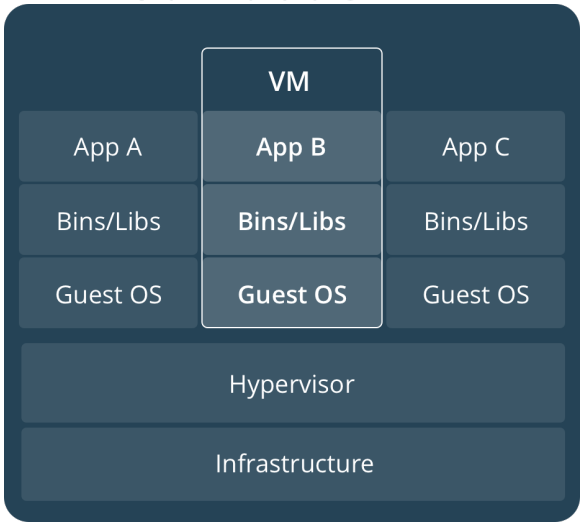
Qual a diferença?

A principal diferença entre eles esta na sua arquitetura.

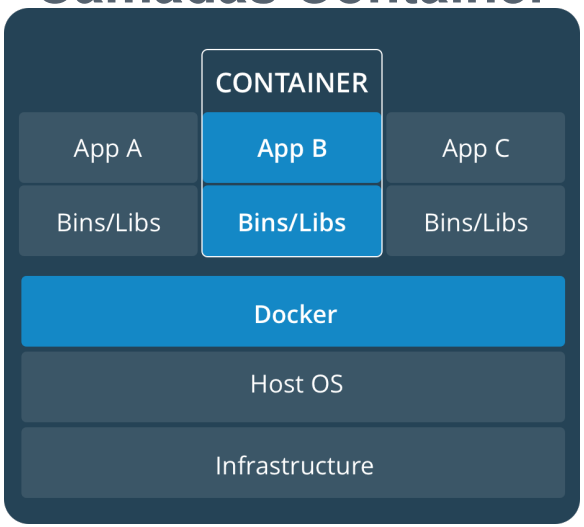
- Numa Máquina Virtual é feita a emulação de toda uma máquina guest, desde o software ao hardware, em cima de uma máquina host.
- Para fazer isso as VMs usam de um artifício chamado de Hypervisor.
- O Hypervisor é uma camada que fica entre o SO do Host e a máquina guest, e é ela que é responsável por fazer o intermédio entre os dois sistemas.

Container vs VM
Introdução

Camadas VM



Camadas Container



Por que Docker?
Introdução

Facilidade

Docker foi criado de forma a ser fácil de usar, tendo em vista que desenvolvedores, administradores de sistemas, e outros usuários possam tomar vantagem de containers para rapidamente construir e testar aplicações. Ele permite que qualquer um possa empacotar uma aplicação no seu computador e possa move-lo para outra máquina ou plataforma.

Velocidade

Containers são muito leves e rápidos. Como eles compartilham do kernel da máquina host, eles precisam de muito pouco recursos para funcionar.

Por que Docker?
Introdução

Velocidade
Containers são muito leves e rápidos. Como eles compartilham do kernel da máquina host, eles precisam de muito pouco recursos para funcionar.

Modular e Escalável
Docker torna fácil a divisão das funcionalidades da sua aplicação em containers individuais e o escalonamento dessas partes.

Por que Docker?
Introdução

Velocidade
Containers são muito leves e rápidos. Como eles compartilham do kernel da máquina host, eles precisam de muito pouco recursos para funcionar.

Modular e Escalável
Docker torna fácil a divisão das funcionalidades da sua aplicação em containers individuais e o escalonamento dessas partes.

Docker Hub
O Docker Hub é um vasto repositório aonde usuários e empresas compartilham suas próprias imagens, à disposição de todos usar e estudar.

Docker Management Commands
Introdução

Os comandos do Docker são divididos em gerenciadores e cada gerenciador tem seus subcomandos. O mesmo subcomando pode servir para mais de um gerenciador, por exemplo:

Docker Management Commands
Introdução

Os comandos do Docker são divididos em gerenciadores e cada gerenciador tem seus subcomandos. O mesmo subcomando pode servir para mais de um gerenciador, por exemplo:

```
$ docker container stop "id_do_container"
$ docker service stop "id_do_serviço"
```

Docker Management Commands
Introdução

Os comandos do Docker são divididos em gerenciadores e cada gerenciador tem seus subcomandos. O mesmo subcomando pode servir para mais de um gerenciador, por exemplo:

```
$ docker container stop "id_do_container"
$ docker service stop "id_do_serviço"
```

ou

```
$ docker container rm "id_do_container"
$ docker image rm "id_da_imagem"
```

Docker Management Commands
Introdução

`$ docker container`

O `container` é responsável por gerir os containers existentes, com ele você fará todas as ações que tenham relação a containers.

COMANDOS:

attach, commit, cp, create, diff, exec, export, inspect, kill, logs, ls, pause, port, prune, rename, restart, rm, run, start, stats, stop, top, unpause, update, wait

Docker Management Commands
Introdução

`$ docker container`

O `container` é responsável por gerir os containers existentes, com ele você fará todas as ações que tenham relação a containers.

COMANDOS:

attach, commit, cp, create, diff, exec, export, inspect, kill, logs, ls, pause, port, prune, rename, restart, rm, run, start, stats, stop, top, unpause, update, wait

`$ docker image`

O `image` é responsável pelas imagens, desde a criação, o download e pela gestão das imagens já existentes na máquina.

COMANDOS:

build, history, import, inspect, load, ls, prune, pull, push, rm, save, tag

Docker Management Commands
Introdução

`$ docker service`

O `service` é responsável pelos serviços. Caso seja necessário escalar seus serviços é por meio desse gerenciador que se faz.

COMANDOS:

create, inspect, logs, ls, ps, rm, rollback, scale, update

Docker Management Commands
Introdução

`$ docker service`

O `service` é responsável pelos serviços. Caso seja necessário escalar seus serviços é por meio desse gerenciador que se faz.

COMANDOS:

create, inspect, logs, ls, ps, rm, rollback, scale, update

`$ docker swarm`

O `swarm` é responsável pela gestão do swarm, como criar uma orquestração, adicionar e remover máquinas ao cluster.

COMANDOS:

ca, init, join, join-token, leave, unlock, unlock-key, update

Em prática

Criando containers

```
$ docker container run \
    "options"     \
    "image"       \
    "command"
```

Em prática

Criando containers

```
$ docker container run \
    "options"     \
    "image"       \
    "command"
```

Listando containers

```
$ docker container ls
```

`-a, --all` Lista todos os containers existentes.

Em prática

Criando containers

```
$ docker container run \
    "options"     \
    "image"       \
    "command"
```

Listando containers

```
$ docker container ls
```

`-a, --all` Lista todos os containers existentes.

Apagando containers

```
$ docker container rm \
```

"container_id" ou *"container_name"*

Em prática

Vamos para o terminal..

Dockerfile

Um *Dockerfile* é como uma *blueprint* de uma imagem, ele é um arquivo de texto com as instruções de passo a passo para a construção daquela imagem.
O *Dockerfile* é constituito por uma série de linhas com palavras-chaves com as quais você define os passo a passo da construção.

Dockerfile

Um *Dockerfile* é como uma *blueprint* de uma imagem, ele é um arquivo de texto com as instruções de passo a passo para a construção daquela imagem.
O *Dockerfile* é constituito por uma série de linhas com palavras-chaves com as quais você define os passo a passo da construção.

PALAVRAS-CHAVES:

FROM, RUN, CMD, LABEL, EXPOSE, ENV, ADD, COPY, ENTRYPOINT, VOLUME, USER, WORKDIR, ARG, ONBUILD, STOPSIGNAL, HEALTHCHECK, SHELL

Dockerfile

Um *Dockerfile* é como uma *blueprint* de uma imagem, ele é um arquivo de texto com as instruções de passo a passo para a construção daquela imagem.
O *Dockerfile* é constituito por uma série de linhas com palavras-chaves com as quais você define os passo a passo da construção.

PALAVRAS-CHAVES:

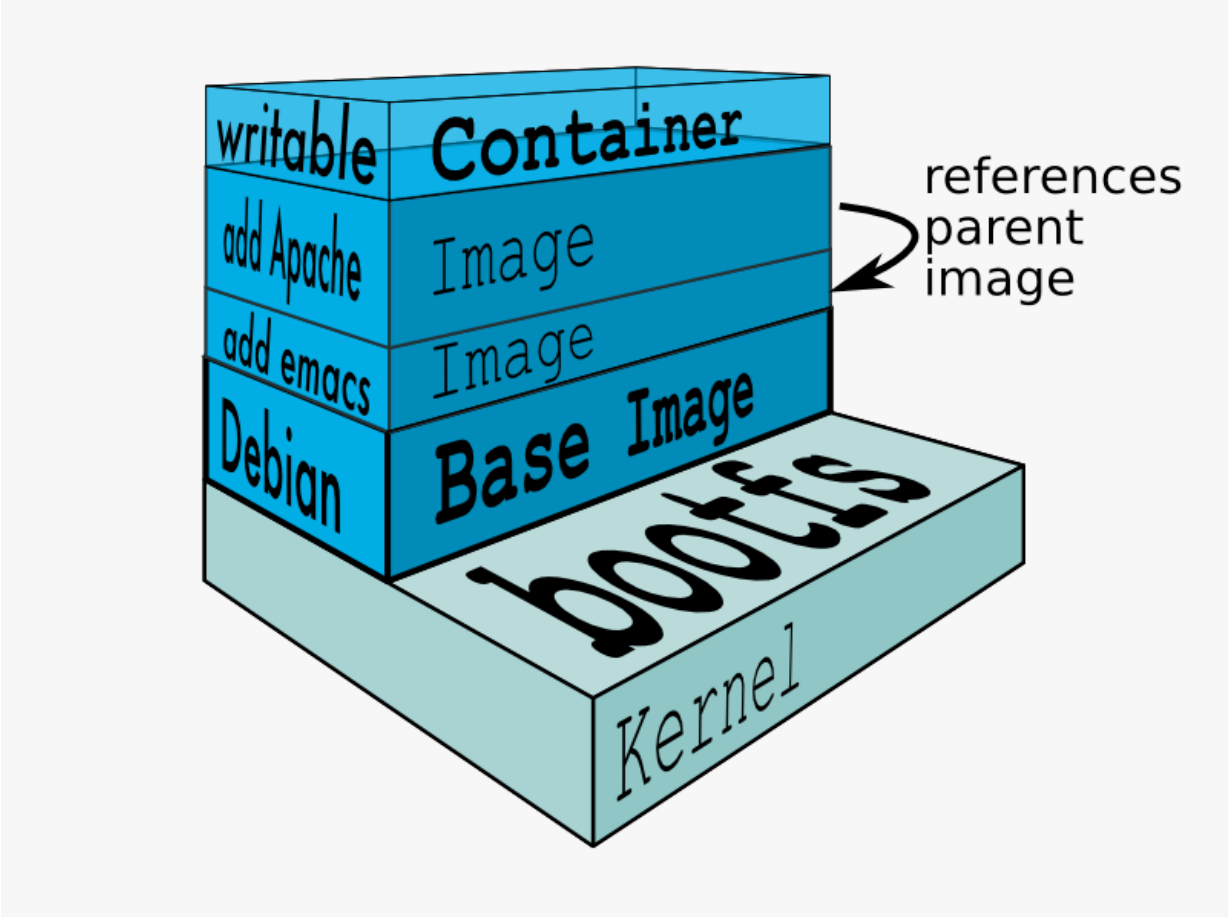
FROM, RUN, CMD, LABEL, EXPOSE, ENV, ADD, COPY, ENTRYPOINT, VOLUME, USER, WORKDIR, ARG, ONBUILD, STOPSIGNAL, HEALTHCHECK, SHELL

Após escrever o Dockefile basta rodar `$ docker image build "caminho_para_Dockerfile"` para construir a imagem.

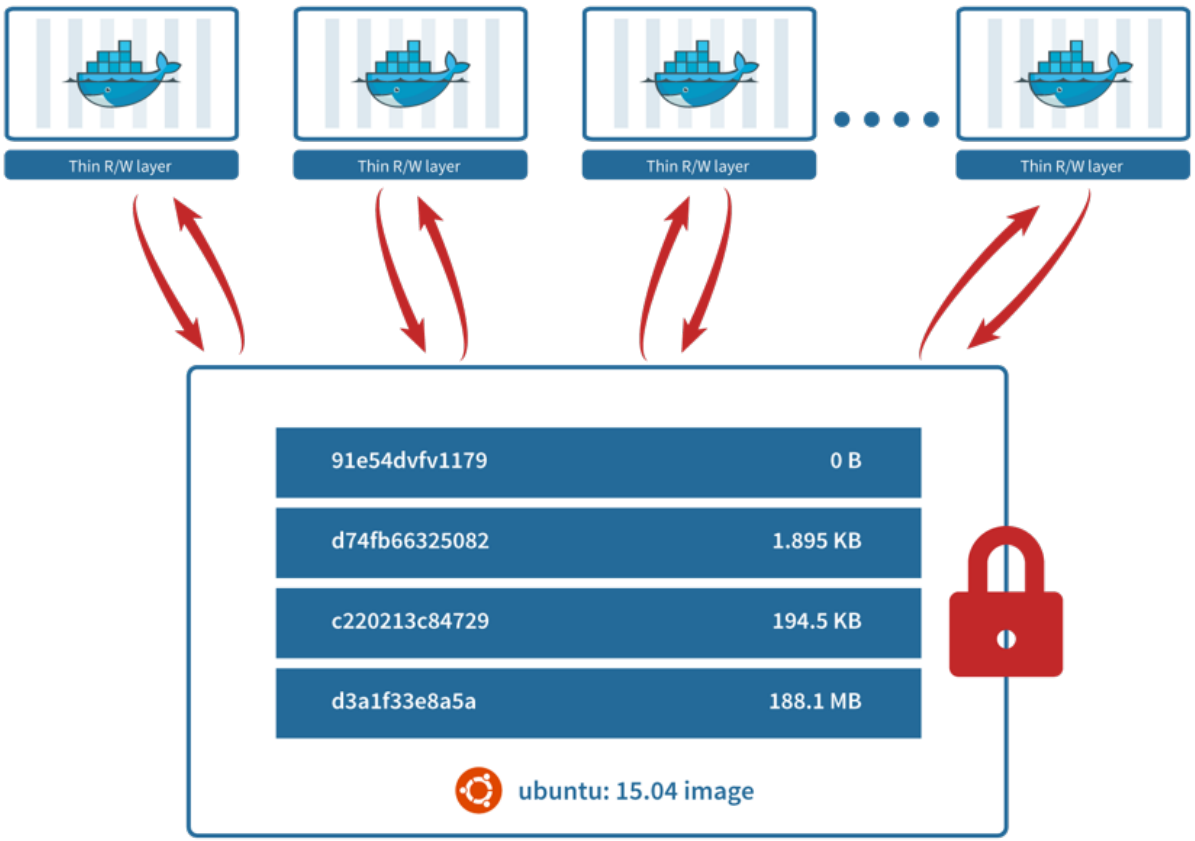
Camadas de uma Imagem

- Os containers do Docker funcionam baseado em camadas, essas camadas são pequenas imagens que juntas constituem uma docker image.
- Ao criar uma imagem de um container, o Docker compila cada parte do sistema numa camada e cria uma subimagem para cada camada.
- Dessa forma ele consegue reaproveitar essas subimagens para a criação de uma nova imagem que tenha camadas em comum com outra imagem já existente.

Criando Imagens
Indo Além



Criando Imagens
Indo Além



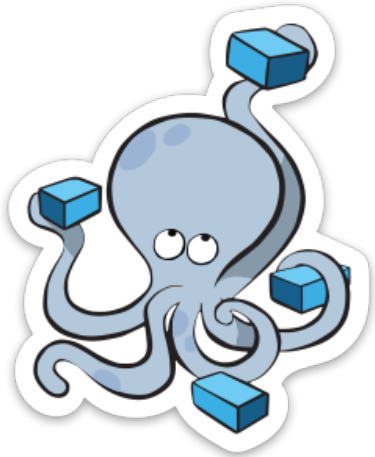
Criando stacks

Um stack é um cluster de containers que trabalham em conjunto para formar uma aplicação.
Existem duas formas de se criar uma stack.

Criando stacks

Um stack é um cluster de containers que trabalham em conjunto para formar uma aplicação.
Existem duas formas de se criar uma stack.

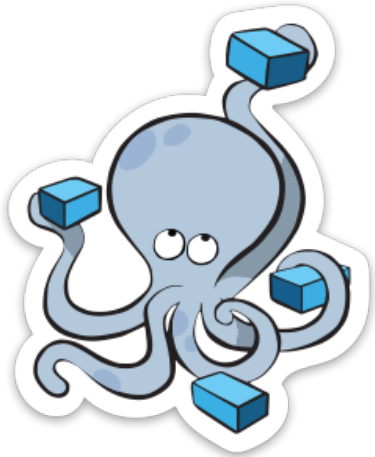
Docker Compose



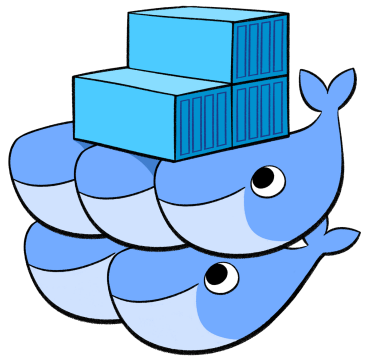
Criando stacks

Um stack é um cluster de containers que trabalham em conjunto para formar uma aplicação.
Existem duas formas de se criar uma stack.

Docker Compose



Docker Stack



Fim!

pratique sem medo

labs.play-with-docker.com