

# docker

**Introdução ao Docker**

Cristian Souza

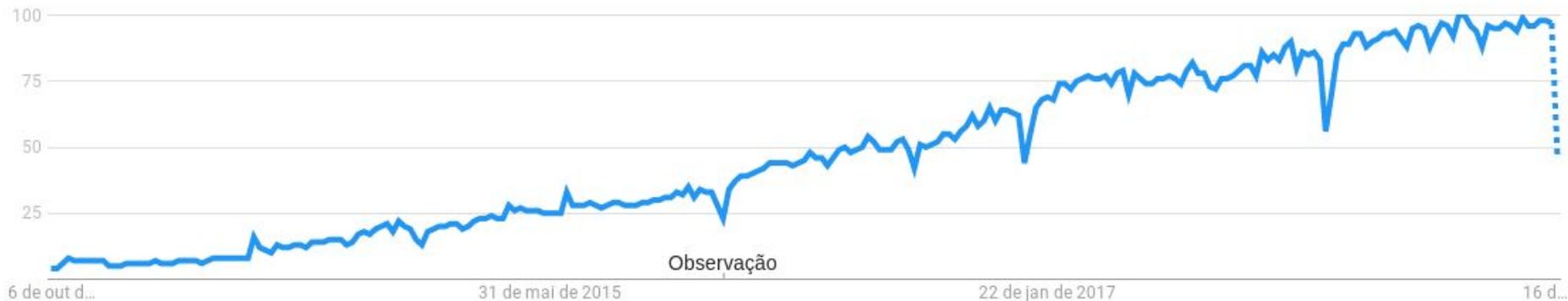
[cristian.sh](http://cristian.sh)

# História

- Desenvolvimento iniciado por Solomon Hykes dentro da empresa dotCloud (hodiernamente, Docker, Inc.), na França;
- Lançamento *open source* em 13 de março de 2013;
- Escrita na linguagem Go;
- Versões: Docker CE (*community edition*) e EE (*enterprise edition*);
- Desde a versão 0.9, a biblioteca ***libcontainer*** é usada para utilizar diretamente os recursos de virtualização fornecidos pelo Kernel;
- Também utiliza interfaces de virtualização abstrata através do ***libvirt***, ***Linux Containers (LXC)*** e ***systemd-nspawn***;
- Crescimento de 3100% em dois anos.

# Interesse ao longo do tempo

Interesse ao longo do tempo ?



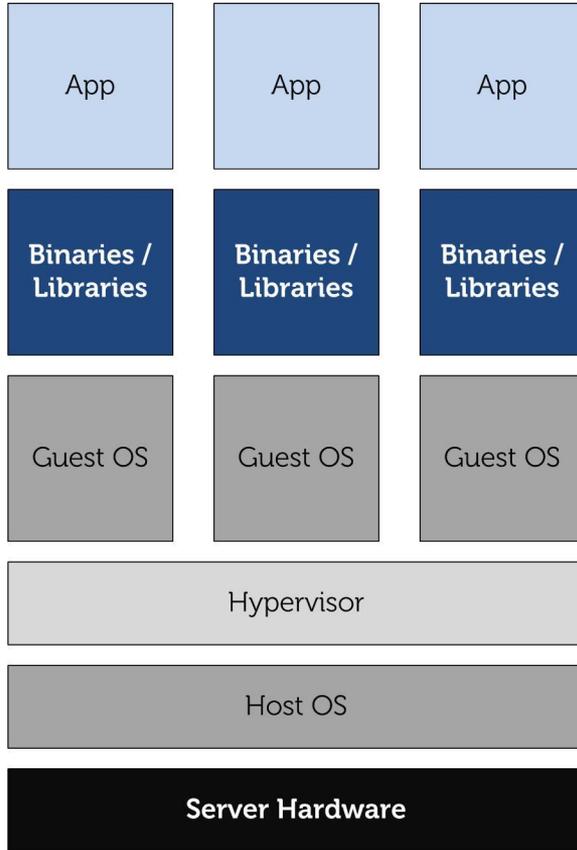
# Quem está usando Docker?



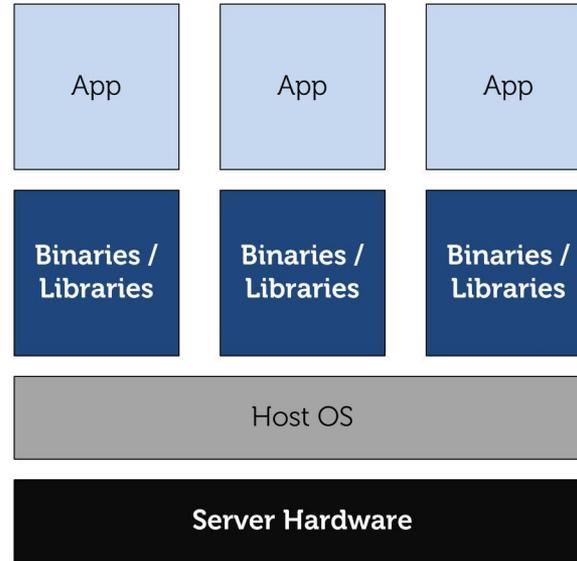
# Containers (LXC)

- A virtualização ocorre de forma menos isolada;
- Compartilha algumas partes do Kernel do sistema *host*, deixando uma menor sobrecarga;
- Exige menos que os tipos convencionais de virtualização (*bare metal* e *hosted*);
- Fazem uso do recurso **Cgroups** (*control groups*) para limitar e isolar o uso de CPU, memória, rede, armazenamento, etc.;
- Utilizam **Namespaces** para isolar grupos de processos para que eles não enxerguem processos de outros grupos ou *containers* no *host*;
- No caso do Docker, os *Namespaces* criados são:
  - ◆ **pid** → responsável pelo isolamento de processos;
  - ◆ **net** → responsável pelo controle das interfaces de rede;
  - ◆ **ipc** → responsável pelo controle de recursos de *InterProcess Communication*;
  - ◆ **mnt** → responsável pela gestão de pontos de montagem;
  - ◆ **uts** → responsável por isolar recursos do Kernel (**Unix Timesharing System**).

# Container x Virtual Machine



Virtualization



Containers

# Instalação do Docker

## 1. Atualizar os repositórios:

```
$ sudo apt-get update
```

## 2. Instalar pacotes para permitir que o apt use um repositório via HTTPS:

```
$ sudo apt-get install apt-transport-https ca-certificates curl  
software-properties-common
```

## 3. Adicionar a chave GPG do Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

## 4. Configurar o repositório estável:

```
$ sudo add-apt-repository \  
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) stable"
```

## 5. Instalar o Docker:

```
$ sudo apt-get update  
$ sudo apt-get install docker-ce
```

## 6. Configurar a execução para outros usuários:

```
$ sudo usermod -aG docker nome_do_usuario
```

## 7. Rodar um *container* de teste:

```
$ sudo docker run hello-world
```

# Comandos do Docker

→ Obtendo informações:

◆ `$ docker info`

→ Obtendo ajuda:

◆ `$ docker <comando> --help`

→ Listar *containers*:

◆ `$ docker ps` # apenas *containers* ativos

◆ `$ docker ps -a` # todos os *containers*

◆ `$ docker ps -qa` # exibe apenas os IDs

→ Listar imagens:

◆ `$ docker images -a`

→ Baixar uma imagem:

◆ `$ docker pull ubuntu:16.04`

→ Rodar um *container* e acessar seu *shell*:

◆ `$ docker run -i -t <imagem> /bin/bash`

# Comandos do Docker

→ Pesquisar uma imagem no Docker Hub:

◆ `$ docker search <nome> # --limit limita o número de resultados`

→ Remover uma imagem baixada:

◆ `$ docker rmi <id_da_imagem> # -f força a remoção`

→ Mostrar dados de um *container*:

◆ `$ docker inspect <nome_ou_id>`

→ Mostrar estatísticas de uso:

◆ `$ docker stats [nome_ou_id]`

→ Mostrar portas mapeadas:

◆ `$ docker port <id>`

# Comandos do Docker: Controlando *containers*

→ Criar um novo *container*:

◆ `$ docker run -it --name exemplo ubuntu`

● Instalar o *Apache*:

○ `# apt-get update && apt-get install apache2 -y`

→ Parar um *container*:

◆ `$ docker stop <id_ou_nome>`

→ Iniciar novamente um *container*:

◆ `$ docker start <id_ou_nome>`

→ Entrar em um *container*:

◆ `$ docker attach <id_ou_nome>`

→ Salvar alterações (uma vez que as alterações no *container* são voláteis):

◆ `$ docker commit <id> ubuntu/apache2`

→ Iniciar um serviço:

◆ `$ docker run -it -p 8080:80 --rm ubuntu/apache2 /bin/bash`

● `# service apache2 start`

# Comandos do Docker: Controlando *containers*

→ Reiniciar um *container*:

◆ `$ docker restart <id_ou_nome>`

→ Renomear um *container*:

◆ `$ docker rename <antigo> <novo>`

→ Matar um *container*:

◆ `$ docker kill <nome_ou_id>`

→ Atualizar as configurações/informações de um *container*:

◆ `$ docker update --cpus 2 <nome_ou_id>`

→ Pausar todos os processos em um *container*:

◆ `$ docker pause <nome_ou_id>`

→ Iniciar processos parados:

◆ `$ docker unpause <nome_ou_id>`

→ Criar uma *tag* para uma imagem:

◆ `$ docker tag <imagem> <imagem>:<tag>`

# Comandos do Docker: Controlando *containers*

→ Iniciar um *container* em *background*:

- ◆ `$ docker run -d -p 8080:80 ubuntu/apache2 /usr/sbin/apache2ctl -D FOREGROUND`

→ Remover um ou mais *containers*:

- ◆ `$ docker rm <id>`

- ◆ `$ docker rm -f <id>`

- ◆ `$ docker rm -f $(docker ps -qa)`

→ Enviar um comando a um *container*:

- ◆ `$ docker exec <id> <comandos>`

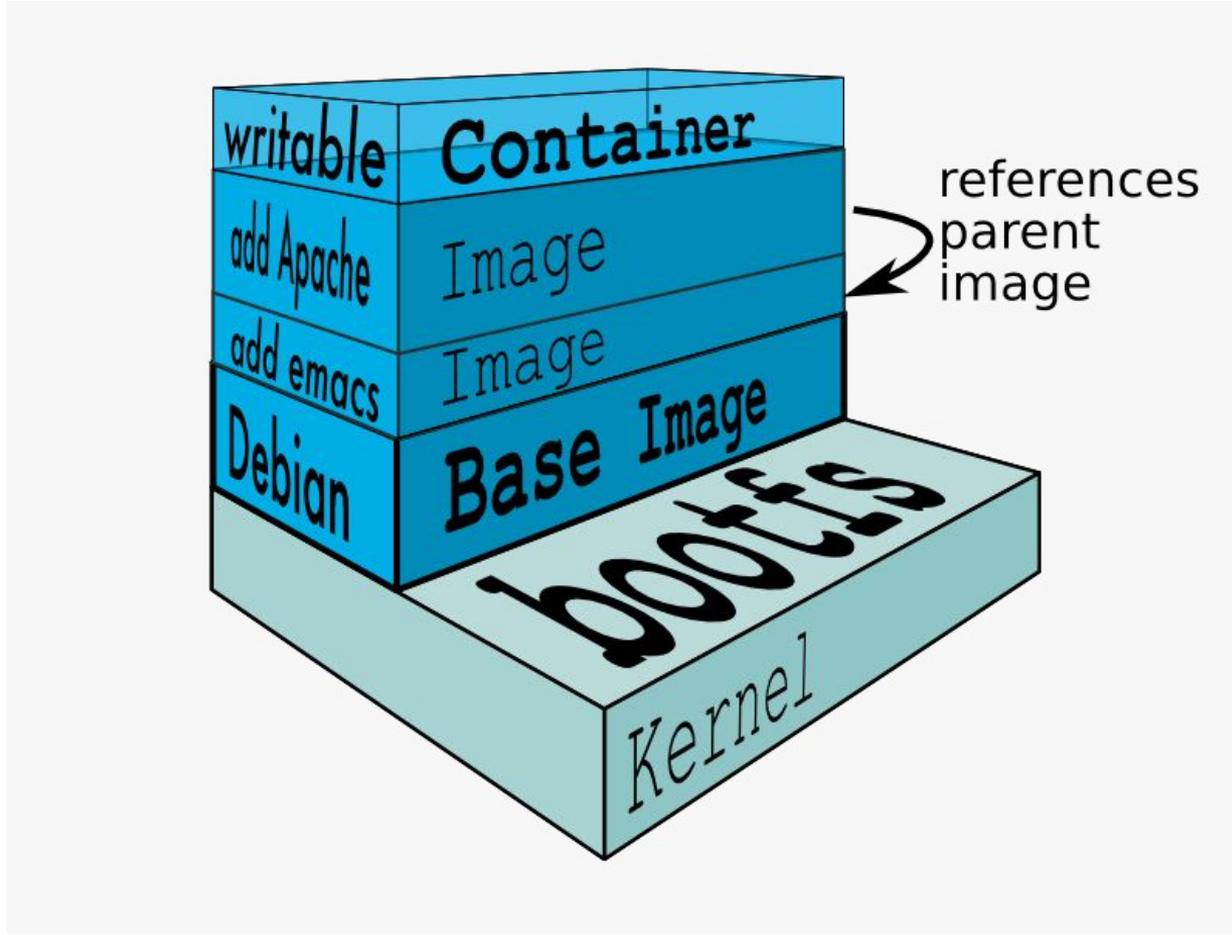
## Observação:

- O comando `run` possui muitos parâmetros, por exemplo: `-m` (limite de memória), `-e` (define uma variável de ambiente), `--link` (adiciona um *link* a outro *container*). Esses e outros parâmetros podem ser consultados com o comando:
  - `$ docker run --help`

# Criação de imagens

- É preciso fazer uso de um **Dockerfile**, que é um recurso utilizado para automatizar o processo de execução de tarefas no Docker.
- Funcionamento do processo de criação de imagens:
  - ◆ Ao criar um *container*, o Docker monta o **rootfs** em modo **read-only**;
  - ◆ O *rootfs* inclui a estrutura de diretórios típica do Linux (*/dev*, */proc*, *etc*, */bin*, */lib*, */tmp* e */usr*) e os programas necessários para execução de uma determinada aplicação;
  - ◆ O Docker utiliza o **unionfs** para criar uma camada de permissões **read-write** sobre a camada **read-only**.
- Programas instalados não existirão na imagem base. É criada uma nova camada acima da base contendo os programas instalados.

# Criação de imagens



# Dockerfile

→ Criação de um *Dockerfile* para o exemplo anterior de instalação do *Apache*:

```
1 FROM ubuntu
2 MAINTAINER Cristian Souza <crislianmsbr@gmail.com>
3 RUN apt-get update && apt-get install apache2 -y
```

Onde **FROM** indica a imagem que será utilizada; **MAINTAINER** indica quem irá manter a imagem; e **RUN** indica um comando a ser executado no *container*.

→ Para gerar uma imagem:

```
◆ $ docker build -t apache2 <caminho_dockerfile>
```

→ Testando a imagem gerada:

```
◆ $ docker run -d -p 8080:80 apache2 /usr/sbin/apache2ctl  
-D FOREGROUND
```

```
◆ curl -IL http://localhost:8080
```

# Dockerfile: Mais opções

→ **EXPOSE**: opção utilizada para mapear portas (parâmetro -P);

◆ Exemplo: **EXPOSE** 80

◆ `$ docker run -d -P apache2 /usr/sbin/apache2ctl -D FOREGROUND`

◆ É aberta uma porta aleatória no *host*

→ **ADD**: utilizado para copiar arquivos para o *container*;

◆ Exemplo: **ADD** <arquivo> <destino\_no\_container>

→ **WORKDIR**: altera a área de trabalho padrão do Docker (/);

◆ Exemplo:

```
ADD ./ /projeto
```

```
WORKDIR /projeto
```

→ **CMD**: inicia um serviço no *container*;

◆ Exemplo: **CMD** /usr/sbin/apache2ctl -D FOREGROUND

◆ Com o **CMD**, o *bash* está sendo chamado dessa forma: `/bin/sh -c`

# Dockerfile: Mais opções

→ **ENTRYPOINT**: similar ao **CMD**. Entretanto, chama o comando ou *script* diretamente (e não o *bash*):

◆ Exemplo:

```
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

**Observação:** ao alterar o Dockerfile, as novas atualizações serão adicionadas à imagem anterior, sem a necessidade de realizar o processo de *build* completo novamente.

# Logs

- É possível verificar os *logs* dos gerados pelas aplicações sem precisar acessar os *containers*. O parâmetro **logs** retorna os *logs* dos *containers*:
  - ◆ Configuração no Dockerfile:
    - **RUN** ln -sf /dev/stdout /var/log/apache2/access.log
    - **RUN** ln -sf /dev/stderr /var/log/apache2/error.log
  - ◆ Visualizar os logs gerados fora do *container*:
    - `$ docker logs <id>`

# Exportação e importação de *containers*

1. Fazer um *commit* da imagem personalizada:

a. `$ docker commit <id> <nome_da_nova_imagem>`

2. Exportar para um arquivo com o parâmetro **save**:

a. `$ docker save <nome_da_nova_imagem> > /tmp/<nome>.tar`

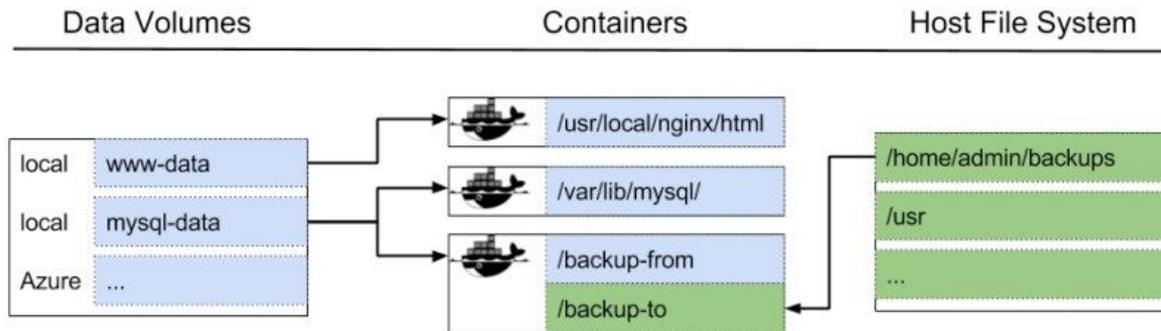
3. Importar em outro *host*:

a. `$ docker load < /tmp/<nome>.tar`

# Volumes

- O Docker permite que diretórios no *container* sejam mapeados no sistemas de arquivos do *host*;
  - É possível compartilhar um volume entre vários *containers*;
  - Qualquer alteração no volume é feita de forma direta;
  - A opção `-v` é utilizada para informar o diretório no *host*;
  - É possível configurar as permissões com as opções `ro` e `rw`.
- ◆ Exemplo:

- `$ docker run -it -v <dir_origem>:<dir_container>:<permissão> ubuntu /bin/bash`



# Volumes - Dockerfile

```
FROM ubuntu
MAINTAINER Cristian Souza <cristianmsbr@gmail.com>
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update -qq && apt-get install -y mysql-server
ADD my.cnf /etc/mysql/conf.d/my.cnf
RUN chmod 664 /etc/mysql/conf.d/my.cnf
ADD run /usr/local/bin/run
RUN chmod +x /usr/local/bin/run
VOLUME ["/var/lib/mysql"]
EXPOSE 3001
CMD ["/usr/local/bin/run"]
```

- `$ docker build -t mysql .`
- `$ docker run -d -p 3001:3001 -e MYSQL_ROOT_PASSWORD=<PASS> mysql`
- `$ mysql -h 127.0.0.1 -u root -p`

# Volumes - Mais opções

- É possível utilizar a opção `--volumes-from` para especificar os volumes de um *container* que serão compartilhados com outros:
  - ◆ `docker run -it --volumes-from <id> ubuntu`
- Remover um volume do disco:
  - ◆ `docker rm -v <nome>`
- Backup de um volume:
  - ◆ `docker run --volumes-from <id> -v \`  
`$(pwd) :/backup ubuntu tar cvf /backup/backup.tar \`  
`/var/lib/mysql`
- Restauração de um volume:
  - ◆ `docker run --volumes-from <id> -v \`  
`$(pwd) :/backup ubuntu tar xvf /backup/backup.tar`

# Rede

- O Docker, por padrão, cria a interface **docker0** no *host*;
- Essa interface serve como uma ponte virtual para encaminhamento de pacotes de forma automática para outras interfaces que estejam conectadas;
  - ◆ `$ brctl show`
- De forma aleatória, o Docker escolhe um endereço privado de IP e uma máscara de sub-rede;
- O Docker configura de forma automática as regras de *firewall* no **iptables**;
  - ◆ `$ sudo iptables -L -n`

# Alterando configurações de rede

→ Para alterar as configurações, é preciso parar o serviço do Docker e configurar uma nova interface de rede:

- ◆ `$ sudo service docker stop`
- ◆ `$ sudo ip link add name br0 type bridge`
- ◆ `$ sudo ip addr add 192.168.0.1/24 dev br0`
- ◆ `$ sudo ip link set dev br0 up`
- ◆ **Alterar o arquivo `/etc/docker/daemon.json`**
- ◆ **Reiniciar o Docker**

# Comunicação entre *containers*

→ Exemplo:

◆ Criar dois *containers* com o MySQL instalado:

- `$ docker run -d -e MYSQL_ROOT_PASSWORD=senha --name db1`
- `$ docker run -d -e MYSQL_ROOT_PASSWORD=senha --name db2`

◆ Acessar o *bash* de um e se conectar com o banco de dados do outro:

- `$ docker exec -it db1 /bin/bash`
- `$ mysql -h <ip> -u root -p`

→ Também é possível utilizar a opção **link** para estabelecer a comunicação entre *containers*:

◆ `$ docker run -d -e MYSQL_ROOT_PASSWORD=senha --name mysql`

◆ `$ docker run -d --name teste --link mysql:<apelido> ubuntu`

# Docker Hub

- Enviar uma imagem para um repositório:
  - ◆ \$ docker login
  - ◆ \$ docker push <usuário>/<imagem>

# Próximos passos

→ Sugestões para aprofundamento:

- ◆ Weave: ferramenta para configuração de rede em *hosts* diferentes;
- ◆ Docker Compose: ferramenta criada para reduzir a complexidade de configuração e isolamento dos ambientes com Docker;
- ◆ Docker Swarm e Kubernetes: ferramentas para orquestração de *containers*.



*"That's all Folks!"*