Docker - Nivelando conhecimento - Parte 1

Hugo Posca

PagarMe

16 setembro 2016

Um problema

Colocar um sistema em produção

- Sistema Operacional específico
- O interpretador/compilador da linguagem
- Bibliotecas (próprias e de terceiros)
- O seu sistema

Um problema

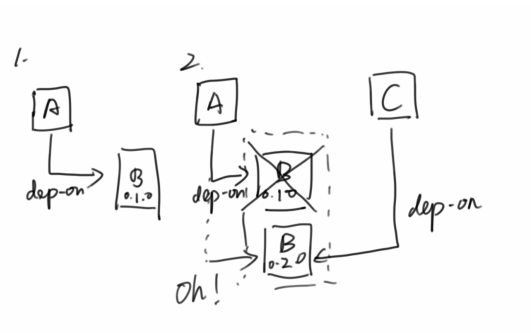
Colocar um sistema em produção

- Sistema Operacional específico
- O interpretador/compilador da linguagem
- Bibliotecas (próprias e de terceiros)
- O seu sistema

...

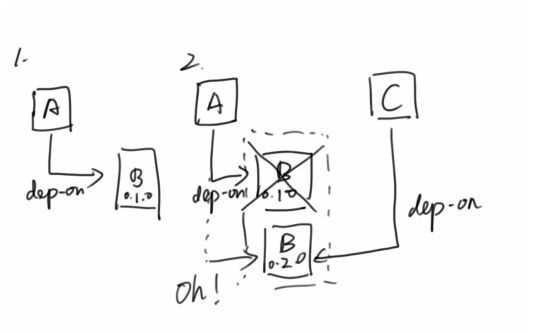
Qualquer pessoa que já tentou fazer isso sabe que ***"raramente"*** temos um problema chamado `Dependency Hell`, especialmente quando temos que colocar mais de um sistema em uma única máquina.

Dependency Hell



Quem nunca?

Dependency Hell



Quem nunca?

Quem nunca?

Não seria legal se existisse...

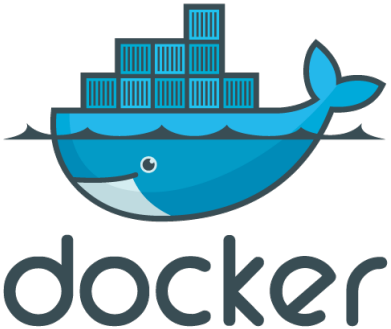
- Um jeito prático de se evitar esse tipo de problema?
- Um jeito de se deixar sistemas independentes realmente independentes?
- Ao mesmo tempo permitisse escalabilidade?
- Que evitasse o famoso "funciona na minha máquina"?
- E ainda por cima, fosse fácil de se "deployar"?

Não seria legal se existisse...

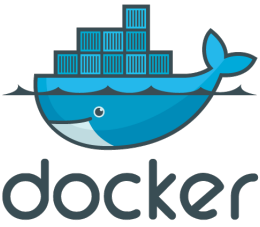
- Um jeito prático de se evitar esse tipo de problema?
- Um jeito de se deixar sistemas independentes realmente independentes?
- Ao mesmo tempo permitisse escalabilidade?
- Que evitasse o famoso "funciona na minha máquina"?
- E ainda por cima, fosse fácil de se "deployar"?

Ceus pobremas ci acabaranç!

Docker



Docker - História



- *dotcloud*, uma empresa de PaaS, fundada em 2010
- Versão 0.9 liberada como Open Source em março de 2013
- Já recebeu mais de $ 50 M em investimentos (em 2015)
- Parceria com várias empresas, incluindo RedHat e IBM

Docker - O que é?

"Ferramenta de empacotamento de uma aplicação e suas dependências em um container virtual que pode ser executado em um servidor Linux"

- Ambiente de execução auto-contido
- Kernel compartilhado com o Host
- Isolamento dos demais *containeres*
- Baixo overhead e tempo de boot

Docker - O que é?

"*chroot* com esteróides"

Chroot

- Troca o diretório '/' de um processo e seus filhos
- Programas que rodam com um ambiente "*chroot*ado"não conseguem acessar arquivos e comandos fora do ambiente
- Comumente chamado de *chroot jail*

Docker - O que é?

"*chroot* com esteróides"

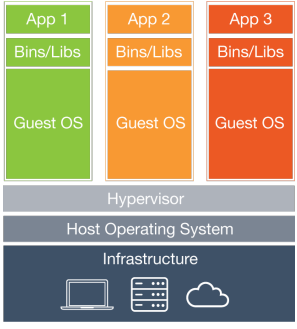
Chroot

- Troca o diretório '/' de um processo e seus filhos
- Programas que rodam com um ambiente "*chroot*ado"não conseguem acessar arquivos e comandos fora do ambiente
- Comumente chamado de *chroot jail*

Tudo isso através do uso de *containeres*

Seria então o docker uma máquina virtual?

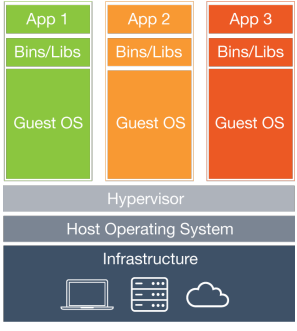
Não! - Containeres possuem uma arquitetura diferente, que permite maior portabilidade e eficiência.



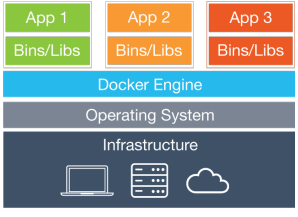
VM

Seria então o docker uma máquina virtual?

Não! - Containeres possuem uma arquitetura diferente, que permite maior portabilidade e eficiência.



VM



Containers

Tecnologias e ideias utilizadas

AKA: Por que só no Linux?

Tecnologia	Ano da primeira versão
docker	2013
LXC	2008
cgroups	2007
libvirt	2005
apparmor	1998
...	...
chroot	1979
...	...
container	1933

O que é um container?

Um container nada mais é do que uma caixa de metal em que você pode colocar o que couber lá dentro...

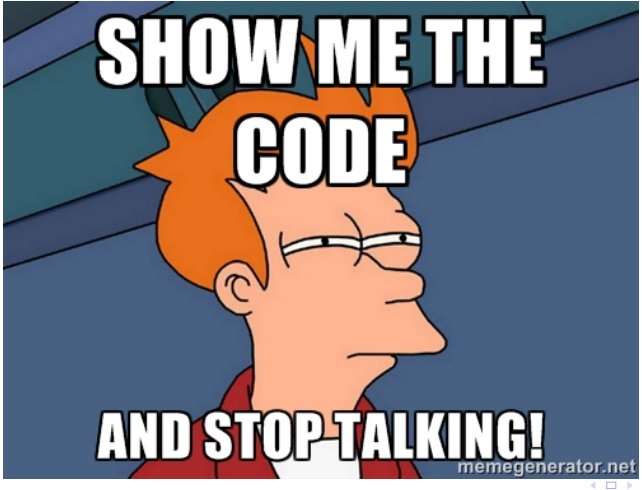
O que é um container?

Com tamanhos padrões e interfaces comuns, onde guindastes e guinchos podem ser acoplados para colocá-los em navios ou caminhões...

O que é um container?

E que tem que ser alocados e gerenciados com cuidado!

Stop the blah blah blah...



Instalando o docker

```
$ wget -qO- https://get.docker.com/ | sh
$ sudo usermod -aG docker <usuario>
```

Ou os instaladores do docker.com

Subindo um container

Para se subir um container basta executar um *docker run* dizendo qual imagem deve ser utilizada:

```
$ docker run --name container-teste ubuntu:14.04
```

Subindo um container

```
$ docker run --name container-teste-2 \
  ubuntu:14.04 \
  /bin/echo 'Hello World'
```

Pergunta: Por que o meu container não fica em pé?

Por que ele não fica em pé?

Mantra dos containeres

"Containeres só são executados enquanto o comando especificado está ativo."

Deixando um container em pé - Alternativa 1

Um container iterativo:

```
$ docker run -ti --name container-teste-3 \
  ubuntu:14.04 /bin/bash
```

Deixando um container em pé - Alternativa 2

Um container daemonizado:

```
$ docker run -d --name container-teste-4 \
  ubuntu:14.04 /bin/bash -c \
    "while true; do \
      echo hello world; \
      sleep 1; \
    done"
```

Construindo sua própria imagem

Imagem

Uma *imagem* é um *modelo/template/"ISO"/"VDI"* somente leitura, que é utilizado para se subir um container.

O docker não teria muita utilidade se só pudéssemos utilizar imagens a partir de sistemas operacionais.

Ele permite que construamos nossas próprias imagens e a utilizemos como base para os containeres, utilizando um arquivo chamado Dockerfile.

Dockerfile

```
FROM ruby:2.2.2
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 561F
RUN apt-get update && apt-get install -y apt-transport-https \
    ca-certificates nodejs
RUN apt-get update && apt-get install -y passenger
RUN echo America/Sao_Paulo > /etc/timezone && \
    dpkg-reconfigure --frontend noninteractive tzdata
WORKDIR /app
COPY . /app
VOLUME /app
EXPOSE 3000
CMD ["passenger", "start", "--port", "3000"]
```

Dockerfile - Instruções

```
FROM imagem[:tag] # A partir de qual imagem estamos nos baseando
RUN comando # Basicamente o que escrevemos em um script bash
WORKDIR /app # Diretorio "raiz" para os comandos seguintes
COPY . /app # Copia arquivos para dentro do container
VOLUME /app # Volumes expostos para fora do container
EXPOSE 3000 # Portas liberadas para fora do container
CMD ["comando", "parametros", ...]
# Que comando deve ser executado assim que um container sobe
```

Dockerfile - Construindo uma imagem própria

Depois de se ter um Dockerfile construído é preciso se construir a imagem propriamente dita:

```
$ docker build -t minha-imagem:1.0 .
```

OBSERVAÇÃO!

Sim, é um ponto no final do comando!

Dockerfile - Subindo um container a partir de uma imagem própria

E agora sim podemos subir um container a partir dela:

```
$ docker run -tid --name meu-container minha-imagem:1.0
```

Coisas interessantes que já podemos fazer

Com o nosso conhecimento não muito avançado já podemos fazer coisas úteis como por exemplo:

- Subir um servidor de Team Fortress 2
- Subir um servidor de Minecraft
- Brincar de "máquinas virtuais"
- Compilar seus programas em apenas uma linha de comando
- Compilar essa apresentação:

```
docker run --rm -i -v $(pwd):/data blang/latex \
  /bin/bash -c "pdflatex -shell-escape apresentacao.tex && \
                pdflatex -shell-escape apresentacao.tex"
```

Docker - Mais uma vantagem - Deploy

Depois que você estiver familiarizado com o docker o seu processo de deploy pode ser totalmente automatizado. E passará por passos como:

```
docker build ...
docker push ...
```

E no seu servidor:

```
docker pull ...
docker run ...
```

Containeres e seus arquivos

Ensinamento filosófico

"Arquivos vivem (e morrem) no contexto dos containers"

- Podemos conectar `mount points` a outros containeres, a volumes ou a um diretório em nossa própria máquina.
- Assim, os arquivos estarão a salvo em um lugar dentro de nossas máquinas e não ficarão tão ligados ao ciclo de vida dos containeres.

Containeres e seus arquivos

Ensinamento filosófico

"Arquivos vivem (e morrem) no contexto dos containers"

- Podemos conectar `mount points` a outros containeres, a volumes ou a um diretório em nossa própria máquina.
- Assim, os arquivos estarão a salvo em um lugar dentro de nossas máquinas e não ficarão tão ligados ao ciclo de vida dos containeres.

Outro ensinamento filosófico

Se o seu container apagar arquivos do `mount point`, eles também serão apagados na sua máquina... cuidado!

Containeres e seus arquivos

Para isso utilizamos a flag ‘-v’:

```
docker run --name postgres \
  -v /mnt/pgdata:/var/lib/postgresql/data/pgdata \
  -e LANG=en_US.utf8 \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -e POSTGRES_PASSWORD=postgres \
  -p 5432:5432 -d postgres:9.3.6
```

Importante:

```
-v diretorio-na-sua-maquina:diretorio-dentro-do-container
```

Containeres e suas portas

Para se acessar portas dentro de um container temos que utilizar a flag ‘-p’

```
docker run --name postgres \
  -v /mnt/pgdata:/var/lib/postgresql/data/pgdata \
  -e LANG=en_US.utf8 \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -e POSTGRES_PASSWORD=postgres \
  -p 5432:5432 -d postgres:9.3.6
```

Importante:

```
-p porta-da-sua-maquina:porta-dentro-do-container
```

Acessando um container

Ok... mas e se por acaso precisarmos entrar dentro de um container para entender o que está acontecendo?
Em uma máquina virtual poderíamos dar um *ssh* e acessar a máquina.

Com docker o mais correto é se utilizar um *docker exec* ou um *docker attach*:

```
docker exec -ti nome-do-container comando
```

O que nos permite fazer um:

```
docker exec -ti nome-do-container bash
root@098815607b17:/# # Agora estamos dentro do container
```

Acessando um container

Como o *docker exec* foi originalmente feito para se executar comandos pontuais dentro de um container podemos ter uma noção da saúde de um container com um:

```
$ docker exec -ti meu_container bash -c 'top -b -n 1'

top - 02:19:47 up 7 days, 14:54,  0 users,  load average: 1.83, 1.76, 1.81
Tasks:   3 total,   1 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s): 41.6 us, 10.9 sy,  0.0 ni, 47.1 id,  0.3 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem:  16394080 total,  6907524 used,  9486556 free,   111040 buffers
KiB Swap:  3998716 total,   566312 used,  3432404 free.  2338964 cached Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  1 root      20   0   46436   6672   2948 S   0.0  0.0   0:00.09 irb
  6 root      20   0   20232   1992   1516 S   0.0  0.0   0:00.04 bash
  28 root      20   0   21796   1220    960 R   0.0  0.0   0:00.04 top
```

* Também podemos utilizar o *docker stats*

Linkando containeres

Links permitem que containeres se comuniquem de forma segura.

Para que isso funcione precisamos:

1. Subir containeres com *–name* (Não precisa de verdade, mas ajuda)
2. Subir o container que quer se conectar com os outros com um *–link*
3. Subir os containeres na **ordem certa**

```
docker run -d --name db training/postgres
docker run -d -p 5000:5000 --name web \
  --link db training/webapp python app.py
```

E no ambiente de desenvolvimento?

Para um ambiente de desenvolvimento geralmente fazer todos esses passos na mão seria algo que tiraria boa parte da vantagem de agilidade do Docker...

Por isso existe o `Docker Compose`, mas que é tema para uma próxima talk!

Espiadinha rápida do docker-compose.yml

```
web:
  build: .
  dockerfile: docker/development/Dockerfile
  command: passenger start --port 3000
  hostname: web.meuapp
  ports:
  - 3000:3000
  volumes:
  - .:/app
  working_dir: /app
  links:
  - db
  - redis
  - mail
  environment:
    BUNDLE_APP_CONFIG: /app/.bundle
    RDS_DB_NAME: app_development
    RDS_HOSTNAME: db
    RDS_PASSWORD: root
    RDS_PORT: 3306
```

```
db:
  environment:
    MYSQL_ROOT_PASSWORD: root
  image: centurylink/mysql
  ports:
  - 3306
  volumes:
  - .data:/var/lib/mysql
  - .:/db
redis:
  image: redis
  ports:
    - 6379
mail:
  image: mailhog/mailhog
  environment:
    VIRTUAL_HOST: mail.meuapp.dev
    VIRTUAL_PORT: 8025
```

Docker - Limpando a bagunça

Depois de um tempo sua máquina pode ficar “ligeiramente” poluída, cheia de containeres mortos e imagens incompletas. Faça uma limpeza:

Apagando containeres que já morreram

```
docker rm -v $(docker ps -a -q -f status=exited)
```

Apagando imagens soltas

```
docker rmi $(docker images -f dangling=true -q)
```

Limpando volumes esquecidos

```
docker run -v /var/run/docker.sock:/var/run/docker.sock -v
/var/lib/docker:/var/lib/docker –rm martin/docker-cleanup-volumes
```

Docker - Filosofia de vida

Pense na diferença entre gado e animais de estimação!

Sua infraestrutura deve ser composta de componentes que você possa tratar como gado: auto-suficientes, facilmente substituiveis e gerenciáveis às centenas ou milhares.

Ao contrário de servidores físicos ou máquinas virtuais, containers podem subir, serem replicados, destruídos e gerenciados com uma flexibilidade muito maior.

Referências

- Documentação do Docker
- Docker - Hello World
- Dockerfile
- Docker Hub
- Docker Compose
- Instalação - Mac OS X
- Instalação - Windows

Pensamento da noite