Docker

Rafael Obelheiro

23/11/2023

Máquinas virtuais vs contêineres

- Máquinas virtuais e contêineres são formas diferentes de virtualização
- Uma MV implementa a abstração de uma máquina isolada
  - virtualização em nível de sistema
- Um contêiner implementa a abstração de um SO isolado
  - virtualização em nível de SO
- Contêineres são mais leves que MVs
  - mais instâncias no mesmo hardware
  - tamanho menor facilita transferências e migrações via rede

Máquinas virtuais vs contêineres

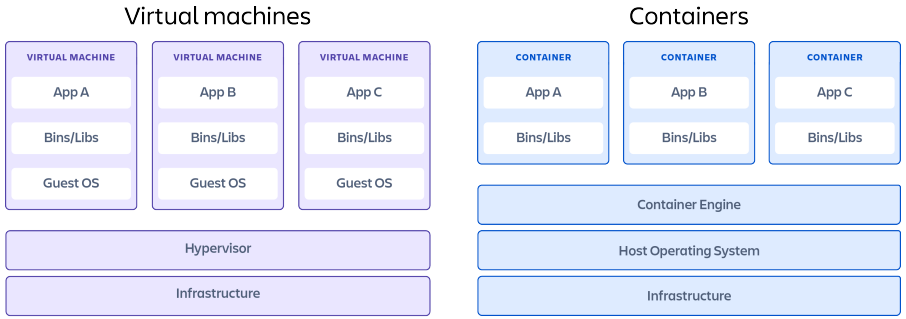

Figure 1:

Docker

- Docker é uma ferramenta para empacotar e distribuir aplicações como contêineres
- Uma aplicação conteinerizada tipicamente contém
  - o código da aplicação, devidamente configurado
  - componentes necessários para a execução → dependências
    - ★ interpretador, MV de aplicação
    - ★ bibliotecas e frameworks
    - ★ pacotes de software
  - configurações do SO
    - ★ arquivos e usuários
    - ★ variáveis de ambiente
    - ★ configuração de rede
- O contêiner pode ser executado em qualquer máquina que aceite seu formato

Conceitos de Docker

- Uma **imagem** é um template com o SA de um contêiner
  - um SA em camadas usando unionfs[1]
- **Camadas** representam mudanças no SA em diferentes momentos
  - uma camada pode ser compartilhada (reusada) por múltiplas imagens
- Um **contêiner** é uma imagem em execução
  - o SA do contêiner combina a imagem (R/O) com armazenamento temporário (R/W)
- Um **registro de imagens** é um repositório onde o Docker busca imagens solicitadas pelo usuário
  - default é o Docker Hub (https://hub.docker.com)

[1] https://lwn.net/Articles/396020/

Arquitetura do Docker

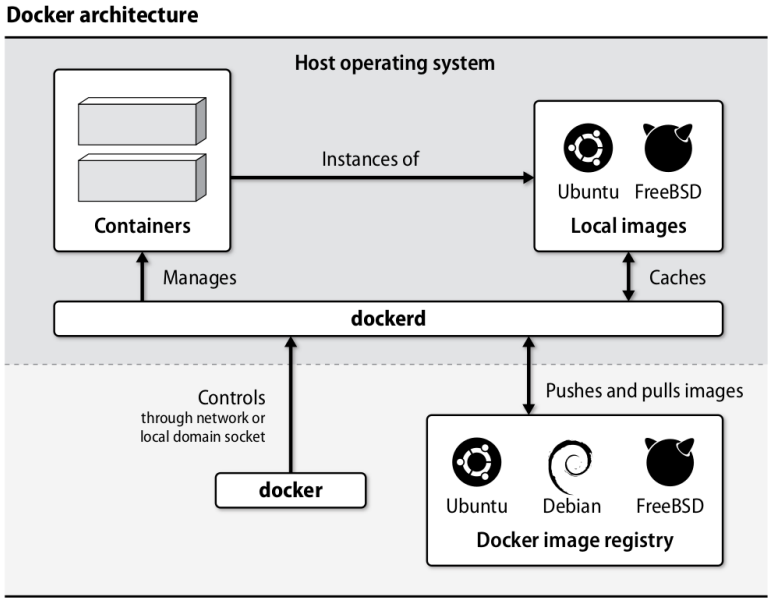

Figure 2:

Baixando e executando imagens

```
### Baixa imagem (debian)
$ sudo docker pull debian

### Instancia imagem (debian), abrindo shell
$ sudo docker run -it debian bash

### Instancia imagem (debian), sem shell
$ sudo docker run debian

### Abre shell em um contêiner em execução (12ab)
$ sudo docker exec -it 12ab sh
```


Listando e parando contêineres

```
### Lista contêineres em execução
$ sudo docker ps

### Lista contêineres em execução e encerrados
$ sudo docker ps -a

### Encerra um contêiner em execução (12ab)
$ sudo docker stop 12ab

### Reinicia um contêiner encerrado (12ab)
$ sudo docker start 12ab
```

Monitorando contêineres

```
### Lista processos em execução em um contêiner (12ab)
$ sudo docker top 12ab

### Monitora recursos usados por um contêiner (12ab)
$ sudo docker stats 12ab

### Inspeciona configuração de um contêiner (12ab)
$ sudo docker inspect 12ab
```


Removendo contêineres

- Contêineres continuam ocupando espaço em disco depois de encerrados
  - possibilita que sejam reiniciados
  - outros recursos pendentes (volumes, logs)
  - devem ser removidos para liberar o espaço

```
### Remove um contêiner encerrado (12ab)
$ sudo docker rm 12ab

### Encerra e remove um contêiner (12ab)
$ sudo docker rm -f 12ab

### Remove todos os contêineres encerrados
###   (rm -f para encerrar e remover tudo)
$ sudo docker rm $(sudo docker ps -qa)
```


Outras opções úteis de `docker run`

- `--rm`: remove o contêiner automaticamente após encerrar
- `-d` (*daemonize*): executa o contêiner em background
- `--name`: especifica um nome para o contêiner
  - usado em outros comandos para referir-se ao contêiner
- `-h`: especifica o hostname do contêiner
- `-e FOO=bar`: atribui o valor `bar` à variável de ambiente `FOO`
  - múltiplas atribuições separadas por vírgulas

```
### Instancia xpto com nome abc, removendo ao encerrar
$ sudo docker run --rm --name abc xpto
```


Mapeando portas de rede

- Portas de rede no host podem ser mapeadas em portas de rede no contêiner
  - pode restringir a interfaces específicas do host (default é não restringir)
- Detalhes: https://docs.docker.com/network/

Exemplos

```
### 1234/tcp no hospedeiro -> 80/tcp no contêiner
$ sudo docker run -p 1234:80 xpto

### 1234/tcp no loopback do hospedeiro -> 80/tcp no contêiner
###   (cuidado com forwarding entre interfaces)
$ sudo docker run -p 127.0.0.1:1234:80 xpto

### 1234/udp no hospedeiro -> 500/udp no contêiner
$ sudo docker run -p 1234:500/udp xpto

### 1234/udp no hospedeiro -> 500/udp no contêiner +
### 1234/tcp no hospedeiro ->  80/tcp no contêiner
$ sudo docker run -p 127.0.0.1:1234:80 -p 1234:500/udp xpto

### Lista mapeamentos de porta (12ab)
$ sudo docker port 12ab
```


Gerenciando imagens

```
### Lista imagens locais
$ sudo docker images

### Mostra camadas de uma imagem
$ sudo docker history debian

### Mostra mudanças no SA do contêiner 12ab
$ sudo docker diff 12ab

### Salva imagem a partir de um contêiner
$ sudo docker commit -m "Salvando mudancas" 12ab nova-img

### Cria imagem usando Dockerfile
$ sudo docker build -t "nova-img" .

### Remove imagem (nova-img)
$ sudo docker rmi nova-img
```


Configurando uma imagem

- Uma imagem pode ser configurada manualmente
  - imagem é executada
  - contêiner é acessado e configurado
    - ★ via shell, por exemplo
  - imagem modificada é salva como uma nova imagem
    - ★ `docker commit`
- O recomendado é automatizar a configuração via Dockerfile
  - um arquivo com comandos que configuram uma nova imagem a partir de uma imagem base
  - a nova imagem é construída com `docker build`
    - ★ pode ser publicada em um registro de imagens (público ou privado)
- A configuração manual pode ser útil para escrever o Dockerfile

Dockerfile

Abbreviated list of Dockerfile instructions

Instruction	What it does
ADD	Copies files from the build host to the image[a]
ARG	Sets variables that can be referenced during the build but not from the final image; not intended for secrets
CMD	Sets the default commands to execute in a container
COPY	Like ADD, but only for files and directories
ENV	Sets environment variables available to all subsequent build instructions and containers spawned from this image
EXPOSE	Informs **dockerd** of the network ports exposed by the container
FROM	Sets the base image; must be the first instruction
LABEL	Sets image tags (visible with **docker inspect**)
RUN	Runs commands and saves the result in the image
STOPSIGNAL	Specifies a signal to send to the process when told to quit with **docker stop**; defaults to SIGKILL
USER	Sets the account name to use when running the container and any subsequent build instructions
VOLUME	Designates a volume for storing persistent data
WORKDIR	Sets the default working directory for subsequent instructions

a. The source can be a file, directory, tarball, or remote URL.

Figure 3:

Exemplo

- Dockerfile

```
# Version: 0.1
FROM ubuntu:latest
LABEL maintainer="rafael.obelheiro@udesc.br"
RUN apt-get update; apt-get install -y nginx
RUN echo 'Estou no conteiner' \
>/var/www/html/index.html
EXPOSE 80
```

- Construindo e instanciando a imagem

```
$ sudo docker build -t "nginx:dockerfile" .
$ sudo docker run -p 5001:80 -d \
nginx:dockerfile nginx -g "daemon off;"
```


Volumes

- O SA de um contêiner combina a imagem (R/O) com armazenamento temporário (R/W)
- Um **volume** é um diretório independente, que pode ser escrito, mantido fora do unionfs
  - armazenamento persistente
  - pode ser compartilhado por vários contêineres
  - conteúdo existente na imagem é copiado para volume quando o contêiner é instanciado pela 1a vez
- Volumes são criados e gerenciados pelo Docker
  - armazenados sob `/var/lib/docker/volumes`

Gerenciamento de volumes

```
### Cria um volume (teste-vol)
$ sudo docker volume create teste-vol

### Executa ubuntu com /var/log associado ao volume
###   teste-vol
$ sudo docker run -it -v teste-vol:/var/log ubuntu bash

### Lista volumes
$ sudo docker volume ls

### Mostra informações do volume (teste-vol)
$ sudo docker volume inspect teste-vol

### Remove volume (teste-vol)
$ sudo docker volume rm teste-vol
```


Bind mounts

- Em alguns casos, deseja-se compartilhar um diretório entre o contêiner e o hospedeiro
  - contêiner pode ver modificações feitas por processos externos
- Volumes são gerenciados pelo Docker
  - permissões normalmente restritas
- Bind mounts permitem associar um diretório no contêiner a um diretório no hospedeiro
  - conteúdo no contêiner é mascarado pelo conteúdo do hospedeiro
  - contêiner verá mesmas permissões do hospedeiro, mas usuários/grupos podem não ser consistentes
  - diretório do hospedeiro precisa iniciar por /

```
### Instancia sgbd com /var/lib/postgresql mapeado em
###   /home/pgsql (no host)
$ sudo docker run -v /home/pgsql:/var/lib/postgresql sgbd
```

Logs

- Saída e erro padrão (`stdout`, `stderr`) de um contêiner são conectados a logs do Docker
  - `docker logs`
- O armazenamento efêmero dos contêineres não casa bem com aplicações que geram logs diretamente em `/var/log`
  - uma solução é estabelecer um link simbólico dos arquivos para `/dev/{stdout,stderr}`
  - `/var/log` também pode ser conectado a um volume ou bind mount

Exemplos

```
### Consulta os logs do contêiner (12ab)
$ sudo docker logs 12ab

### Consulta os logs do contêiner (12ab), com timestamps
$ sudo docker logs -t 12ab

### Monitora os logs do contêiner, com timestamps
$ sudo docker logs -f -t 12ab
```