Imagem vs Container

- Imagem: são os binários, bibliotecas e o código-fonte que formar a sua aplicação
- Container: é uma instância de uma imagem rodando, pode-se ter vários conteiners rodando a mesma imagem

Comandos básicos para gerenciamento de containeres

`docker container run --publish 80:80 nginx`

--publish expõe a porta 80 da maquina local e envia todo o tráfico vindo desta para um executável rodando dentro do container na porta 80

`docker container run --publish 80:80 --detach nginx`

--detach diz para o docker rodar em background, é gerado um ID único toda vez que é rodado um novo container

`docker container ls`

listagem de todos os containers que estão rodando no momento

`docker container start 690f`

execução de um container existente com código 690f, onde 690f se refere ao digitos iniciais do ID do container

`docker container stop 690f`

para de rodar o container

`docker container ls -a`

listagem de todos os containers existentes, vale destacar que o comando "docker container run" sempre gera um novo container, enquanto que, "docker container start" começa a rodar um container existente

`docker container run --publish 80:80 --detach --name webhost nginx`

criando um novo container especificando um nome para ele, por padrão é gerado um nome aleatório

`docker container logs webhost`

exige os logs de um container

`docker container top webhost`

pode-se ver o processo do container "webhost" rodando

`docker container rm 63f 690 0de`

excluindo múltiplos containeres indicando-os pelas iniciais de seus ID's "63f 690 0de", detalhe que os containeres deve estar parados para serem excluidos

`docker container rm -f 63f`

exclui-se um conteiner, a flag "-f" permite que o docker force a exclusão de um container que está sendo executado

`docker container inspect 63f`

detalhes das configurações de um container, retorna um JSON com todos os metadados

`docker container stats`

monitoramento de perfomance, uso dos recursos, para todos os containers

`docker container --help`

listagem de comandos disponíveis para gerenciar containeres

O que acontence quando é utilizado o comando "docker container run --publish 80:80 nginx"?

1. Procura-se a imagem localmente no cache de imagens
2. caso não encontre, a imagem será procurada em um repositório (por padrão, Docker Hub).
3. é realizado download da ultima versão (isto é "nginx:latest").
4. Então, é criado um container baseado nesta imagem e prepara-se para inicia-la.
5. é dado um IP virtual em uma rede privada dentro do docker engine
6. abre a rede na porta 80 no host e na porta 80 no container
7. container é finalmente iniciado

É necessário usar SSH para acessar um container? A resposta é não, mas como ter uma linha de comando dentro de um comando?

`docker container run -it`

inicia um novo container em modo interativo (a entrada do container é mantida aberta) com um `tty` alocado (pseudo terminal). Vale destacar que ao sair do terminal o processo do container será finalizado.

`docker container run -it --name my-server nginx bash`

container de nginx com nome my-server será aberto com um bash aberto

`docker container exec -it mysql bash`

executa um comando adicional em um container que está sendo executado. Ao sair do terminal, o container continuará com sua execução normalmente, diferentemente dos comandos run e start.

`docker container run --rm -it alpine sh`

inicia container com alpine (distro de linux) com o sh aberto. **flag –rm roda o container uma única vez e excluí ele após o uso, economizando espaço no HD, útil para testes.**

Conceitos básicos de rede do docker

- cada container se conecta em uma rede virtual privada "bridge" (rede padrão)
- cada rede virtual roteia pelo firewall NAT no IP do host
- todos os containers em uma rede virtual podem se "falar" entre eles sem a flag -p (alias para --publish)
- observação: os containeres usam um IP próprio e não o IP da máquina hospedeira, ou seja, a rede é encapsulada

`docker container run -p 80:80 --name webhost -d nginx`

cria um container webhost com a image nginx, rodando em background com port 80 no host e port 80 dentro do container

`docker container port webhost`

exibe os tráficos de rede pelas portas dentro do container webhost

`docker network ls`

exibe as networks existentes no host.

`docker network inspect bridge`

realiza a inspeção de uma rede, no caso a bridge

`docker network create my_app_net`

cria uma nova rede privada com host chamado de "my_app_net"

`docker container run -d --name new_nginx --network my_app_net nginx`

cria um container que roda em backgroud, com nome new_nginx, utilizando a nova rede criada my_app_net usando a imagem nginx.

`docker network inspect my_app_net`

lista informações sobre a rede my_app_net, podendo visualizar os containeres que estão presentes nela.

`docker network connect my_app_net 078b`

conecta container com ID de iniciais 078b à rede criada my_app_net

`docker container inspect 078b`

informações como as de rede sobre o container 078b

Gerenciamento de imagens

As imagens podem ter diferentes tags que representam uma mesma versão, por default sempre será baixado a última versão taggeada com "lastest"

`docker image ls`

listagem das imagens que já foram baixadas

`docker pull alpine`

Com este comando baixa-se a imagem da última versão do Alpine. Observação: Alpine é uma distro de Linux que pesa em torno de 5 MB, então você terá uma experiência bem limitada com ele, inclusive ele não possuí o `bash`, apenas o `sh`

`docker pull node`

o docker irá baixar a versão mais recente do node (15.2.0)

`docker pull node:12.19.0`

com a tag 12.19.0 o docker irá procurar pelo node referente à versão desta tag, para visualizar mais tags basta procurar pela imagem no Docker Hub

`docker pull node:alpine` com a tag alpine, iremos baixar a imagem mais recente do node em uma imagem muito mais leve por ser feita em cima da distro alpine.

Imagens e as suas camadas

A construção de imagens se baseiam em camadas, que é um conceito fundamental para o funcionamento e otimização dela pois cada camada é cacheada para construções futuras, economizando tempo e processamento para o build das imagens. Tal funcionamento, é importatissimo na hora de construir os Dockerfiles.

Tempo de vida dos containeres e persistencia de dados

- **Data volumes**: opção de configuração para um container que cria um espaço especial fora do sistema de arquivos do container para armazenar dados únicos, sendo útil para criar containeres com aplicações de banco de dados por exemplo.
- **Bind mounts**: compartilhamento ou montagem de um diretório do hospedeiro, ou arquivo, em um dado container, sendo útil para que o docker "enxergue" uma pasta de arquivos de um projeto sendo desenvolvido por exemplo.

`docker volume prune`

limpa volumes **não** usados

`docker volume ls`

listagem dos volumes existentes, mesmo após a exclusão dos containers os volumes persistem e não são excluídos, protegendo todos os dados do container

`docker volume create --help`

Criar um volume é necessário antes de usar "docker run -v"

`docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-db:/var/lib/mysql mysql`

-v flag permite especificar um novo volume a ser criado para um container, caso fosse apenas "-v /var/lib/mysql" ele seguiria o padrão do Dockerfile do mysql, que já cria um volume neste local, mas ao adicionar "mysql-db" o volume é nomeado. Observação: -d roda container em modo detach, isto é, em "plano de fundo". flag -e exporta variavel de ambiente necessária para executar o mysql (geralmente as variaveis de ambiente que temos que exportar são destacadas nos repositorios das imagens no Docker Hub)

Observações Bind Mounting

- Como dito "montamos" um diretório/arquivo presente em nossa máquina dentro do file system do container, possibilitando o acesso de dentro do container aos arquivos do nosso computador.
- Útil para o desenvolvimento, utilizando um diretorio no host ao mesmo tempo que as dependencias rodam em um container do docker.
- Mesma forma para usar o volume colocando a flag -v mas ao invés de nomear um volume, deve-se colocar o caminho do diretório atual no host começando com /, exemplo "-v /User/my-super-app:/var/lib/mysql"

`docker container run -d --name nginx -p 80:80 -v \$(pwd):/usr/share/nginx/html nginx`

Cria-se um container utilizando um bind mount com o diretório atual (comando `pwd` retorna diretório atual)

`docker container exec -it nginx bash`

podemos testar a criação de arquivos no hospedeiro que terão sua alteração verificada dentro do container também.

Tornando nossa vida mais fácil com "receitas"

- O docker possui uma ferramenta chamada `docker-compose` que possibilita que escrevemos arquivos em um domínio de linguagem específico para realizar instruções que usariamos em um docker container run da vida de forma mais "automatizada", por exemplo:

docker-compose.yaml

```
version: '3.7'

services:
  site:
    build: my-super-node-app-image
    ports:
      - 8080:8080
    volumes:
      - deps:/usr/src/app/node_modules
      - ./:/usr/src/app

volumes:
  deps:
```

- criamos um serviço nomeado como `site` que é um container construído com a imagem `my-super-node-app-image`, publicando as portas 8080 tanto para o host quanto para a rede privada do docker (destacando que o docker-compose cria uma rede privada), utilizando um volume para as dependências do `node_modules` presentes em projetos node e montando a pasta atual `.` para "enxergar" pastas como `src` que pode conter código fonte do projeto.
- mais informações podem ser encontradas na documentação

`docker-compose up`

seta volumes/networks e starta todos os containers, enfim, realiza todas as intruções do docker-compose.yml

`docker-compose up -d`

flag -d, roda em background os containeres

`docker-compose logs`

exibe os logs de todos os containers que estão rodando

`docker-compose down`

derruba os containers

`docker-compose down -v`

para e exclui os containers e também exclui os volumes (esse comando em muitos momentos é importante, por exemplo, em projetos node muitas vezes atualizamos as dependencias dos projetos e acaba que temos alguns bugs em relação à consistencia dos node_modules, nisso precisamos limpar o volume que contém os node_modules e contrui-lo novamente)

`docker-compose up --build`

sobe os containers buildando sempre uma nova imagem

Observações importantes

- BUSQUEM INFORMAÇÕES NA DOCUMENTAÇÂO DO DOCKER, ela é bastante completa e pode tirar a maioria de suas dúvidas, existem milhares de outros comandos e coisas úteis para aprender