Fundamentos de Docker

CONTAINERS:
Processos - É um programa que está em execução no sistema operacional onde tem as informações para controle de sua execução.

Namespaces - Ferramenta dentro do container para agrupar processos de forma que não seja possível interferir em outros processos fora do name spaces.
O PROBLEMA PODE INTERFERIR DE FORMA QUE ACABE OS RECURSOS DO PC.

Cgroups - Permite o controle dos recursos do sistema alocado para evitar o problema acima de interferência por meio dos recursos limitados do sistema.

Resumo: Containers é um processo isolado em execução em uma máquina host que está isolado de todos os outros processos da máquina host.

Container != VM - container não tem virtualizado e nem separação de hardwares

Imagens - Contém tudo que for necessário para executar um container, dependências, configurações, scripts, binários, etc. além disso ela é imutável, para altera-la é necessário criar outra. Elas podem ser reutilizadas em diferentes ambientes e em diferentes equipes, facilitando a colaboração e implantação de aplicativos.

Dockerfile - Arquivo de configuração usado para definir como a imagem Docker deve ser construída. Lista as instruções para instalar dependências , copiar arquivos, configurar variáveis, executar comandos dentro dos containers etc.

Registros Docker - As imagens podem ser armazenadas em registros/repositórios online onde as imagens podem ser distribuídas

Volumes Docker - é uma maneira de associar um volume de memoria da maquina host para um contêiner, quando o contêiner é excluído o volume permanece pois está associado ao volume da maquina host. (Pode ter vários contêineres no mesmo volume)

Commands docker:
Docker ps - Mostra a lista de containers ativos
Docker ps -a - Mostra todos os containers ativos e desativados.
Docker image pull <nome da imagem> - Atualiza a imagem/baixa.
Docker image pull <nome da imagem>:<Caso queira especificar a versão da imagem> - Atualiza a imagem/baixa.
Docker images - Vê todas as imagens
Docker container create <nome da imagem> - cria o container com base na imagem.
Docker start <id ou nome do container> - executa o container.
Docker start -a <id ou nome do container> - executa o container e mostra a entrada e a saída no terminal do container.
Docker run <nome da imagem (hello-world)> - Baixa a imagem > executa a imagem e executa o container.

Docker stop <id ou nome do container> - Para a execução de um container.
Docker start -ai <id ou nome do container> - Starta um container em modo interativo.
Docker rm <id ou nome do container> - Deleta um ou mais containers dependendo do tanto de id passado. NÃO SE PODE FAZER A REMOÇÃO DE UM CONTAINER QUE ESTÁ ATUALMENTE EM EXECUÇÃO (**Apenas se usar o --force/-f**)
Docker logs <id ou nome do container> - Lista os logs de um container caso não tenha um terminal atrelado/travado ao container já mostrando.
docker container inspect <id ou nome do container> - Mostra todas as informações de um container através de um json.
Docker exec <id ou nome do container> <command linux> - Permite rodar comandos do Linux dentro do container.
Docker exec -it <id ou nome do container> bash - Permite que eu acesse o terminal Linux do container.
Docker images/ Docker image ls - lista todas as imagens montadas disponíveis no root.
Docker image rm <id ou nome do container> - remove a imagem do sistema root
Docker volume Create <nome que deseja criar> - Cria o volume com um nome
Docker volume ls - Lsita todos os volumes criados
docker volume inspect <nome do volume> - Mostra informações detalhadas sobre o volume especificado através do json GERA UM VOLUME DENTRO DE UMA PASTA DOCKER VOLUMES (/var/lib/docker/volumes/teste/_data)
Docker build -t <nome da imagem> . - Faz a build de uma imagem. o -t é para passar um nome para a imagem e o '.' é para falar que o dockerfile está na raiz de onde é executado o comando.
docker build -t dotnet-pj2 -f.docker/Dockerfile . - -f especifica o diretório onde o Docker file está e o . é o diretório raiz onde vai ser executado o comando, no caso se pode definir o diretório manualmente também.
Docker login - Para logar no docker cli com a conta de seus repositórios do docker.
Docker push <nome da imagem> - Posta a imagem nos repositórios do docker.

Flags do comando docker run:
-it - -i (--interactive) → Mantém a entrada padrão (stdin) aberta, permitindo que você envie comandos para o container. -t (--tty) → Aloca um terminal virtual (pseudo-TTY), permitindo que você tenha uma experiência interativa semelhante a um terminal real.bash: Especifica o comando que será executado dentro do container (abre um shell Bash).
Isso mesmo! O -t (--tty) cria um terminal virtual (TTY), garantindo que a experiência no container seja semelhante a um terminal normal do Linux.
Explicação:
O Docker normalmente não cria um terminal automaticamente ao executar um container.
O -t aloca um pseudo-terminal, permitindo que a saída dos comandos seja formatada corretamente (exibir prompt, cores, manipulação de cursor etc).
--rm - Caso passe essa flag na criação de um container, o container em questão será eliminado/removido assim que ele sair de execução automaticamente.
--name - Caso passe essa flag na criação de um container um nome pode ser passado para o container ao invés de ficar de gerar de forma aleatória facilitando assim sua identificação. (Docker run it --name <nome do container> ubuntu bash).
-d/--detach - Destrava o terminal caso exista preferência, o container continua em exe mas o terminal fica destravado mesmo com o /docker-entrypoint.

-p/--publish - publica a porta de um container para um host / Se tem um container que se chama nginx que disponibiliza uma porta para acessar, mas não tem como acessá-la no host/pc local para fazer isso se usa o -p, está é a sintaxe:
docker run (-p <porta do host que se quer associar (8080)>:<Porta que o container libera (80)>)
OBS: O nginx é um container que disponibiliza uma porta existem outros e dá para criar container com portas disponíveis.
— network <nome da rede> - Cria um container com acesso à rede especificada.

- **BINDMOUNTS:** São alocações de volumes de forma específica e controlada por você:
  -v <alguma rota de volume do root>:<alguma rota de volume para o container> - Esse parâmetro possibilita compartilhar um volume do sistema root para o container, vira de fato uma pasta compartilhada, onde tudo que altera nessa pasta pelo root altera no container e vice versa.
  --mount type=<tipo de alocação (bind)>,source=<alguma rota de volume do root>,target=<alguma rota de destino para o container> - Essa flag também faz o processo de compartilhamento de volume porém ela é mais verbosa/explícito e mais precisa deixando escolher o tipo de compartilhamento.
  OBS -v --mount - PODE CRIAR PASTAS NO PROCESSO DE SELECIONAR O LOCAL DO CONTAINER, ENTÃO AO INVÉS DE ESCOLHER UMA PASTA QUE JÁ EXISTE PODE CRIAR UMA BASTA APENAS ESCOLHER O CAMINHO E DAR O NOME DA PASTA NO FINAL QUE VAI COMPARTILHAR O VOLUME. SÃO CHAMADOS DE BINDMOUNTS ESSE MÉTODO DE COMPARTILHAMENTO
- **VOLUMES:** Diferente do bindmounts o manuseamento e alocação fica toda por conta do docker
  --mount type=volume,source=<Nome do volume que se quer usar>,target=<alguma rota de destino para o container>
  -v data:<alguma rota de destino para o container>
  OBS: QUANDO SE SOUBER QUAL PASTA ESPECÍFICA APONTAR, USAR O BINDMOUNT, SE NÃO SOUBER OU NÃO NECESSITE DE APONTAR ALGO ESPECÍFICO, USE O VOLUME PARA FAZER A ASSOCIAÇÃO. VOLUMES SÃO MAIS FÁCEIS DE SER GERENCIADOS POIS O DOCKER ATUA EM CIMA DELES NO GERENCIAMENTO DO CLI. É POSSÍVEL O QUE VÁRIOS CONTAINERS ACESSEM O MESMO VOLUME.

DOCKERFILE - É um arquivo que é utilizado pelo docker para gerar imagem:
Docker build -t <nome da imagem > - Faz a build de uma imagem. o -t é para passar um nome para a imagem e o '.' é para falar que o dockerfile está na raiz de onde é executado o comando. Exemplo Dockerfile:

```
#TODA IMAGEM DOCKER PARTE DE UMA OUTRA PRÉ EXISTENTE
#O FROM É UTILIZADO PARA PEGAR A IMAGEM BASE
FROM golang:1.21-alpine

#QUANDO SE QUER EXECUTAR UM COMANDO NO TERMINAL DO CONTAINER NA BUILD
DA IMAGEM, UTILIZA-SE O RUN
```

```
RUN apt-get update
RUN apt-get install -y vim

# WORKDIR SERVE PARA DEFINIR O DIRETÓRIO DE TRABALHO PADRÃO DO
CONTAINER, OU SEJA, O DIRETÓRIO QUE SERÁ ACESSADO QUANDO O CONTAINER
FOR INICIADO, ATÉ MESMO OS COMANDOS RUN/DE TERMINAL SERÃO EXECUTADOS
NESTE DIRETÓRIO
WORKDIR /app

#O COPY COPIA OS ARQUIVOS DO DIRETÓRIO LOCAL PARA O DIRETÓRIO DO
CONTAINER, ONDE O . SIGNIFICA QUE SERÁ COPIADO OU PEGO PARA O DIRETÓRIO
RAIZ
COPY index.html .
# NESTE CASO O INDEX.HTML SERÁ COPIADO PARA O DIRETÓRIO RAIZ DO
CONTAINER, MAS SE FOSSE . . O DIRETÓRIO RAIZ DO ROOT SERA COPIADO PARA
O DIRETÓRIO RAIZ DO CONTAINER

#O ENTRYPOINT É OS COMANDOS QUE SERÃO EXECUTADOS QUANDO O CONTAINER FOR
INICIADO, ELE NÃO PODE SER ALTERADO DE FORMA ALGUMA DEPOIS DA IMAGEM
SER GERADA
ENTRYPOINT [ "go", "run", "main.go" ]

#CMD [ "go", "run", "main.go" ]
#O CMD ELE TAMBÉM É UM COMANDO QUE É EXECUTADO QUANDO O CONTAINER FOR
INICIADO, MAS ELE PODE SER ALTERADO QUANDO SE PASSA UM COMANDO DE
TERMINAL NA EXECUÇÃO DO CONTAINER.
DESSA FORMA OS DOIS SE COMPLEMENTAM, POIS O CMD PODE SER PASSADO COMO
PARÂMETRO PARA O ENTRYPOINT.
```

Variável gerada dentro do dockerfile:
PORT - Permite definir qual porta o container vai está atuando de forma que se possa variar, para poder variar a porta, a imagem tem que ter uma variável port
docker run -d -e PORT=3000 -p 8080: 300 <Nome da imagem>

```
# ENV É UTILIZADO PARA DEFINIR VARIÁVEIS DE AMBIENTE NO DOCKERFILE
#ENV PORT=8080

#O EXPOSE É UTILIZADO PARA DEFINIR/DOCUMENTAR A PORTA QUE O CONTAINER
VAI ESCUTAR
EXPOSE 8080
#${PORT}
#UTILIZA A VARIÁVEL DE AMBIENTE PORT PARA DEFINIR A PORTA QUE O
CONTAINER VAI ESCUTAR
```

Multi-stage - Esse é o processo no qual é possível criar uma imagem pelo dockerfile e com essa imagem criar outra, muito usado para melhorar o desempenho de uma imagem

diminuindo seu tamanho. EXEMPLO:

```
FROM golang:1.21-alpine AS builder

WORKDIR /app

COPY main.go .

RUN go build main.go

FROM scratch AS runner

WORKDIR /app

COPY --from=bulder /app/main /app

EXPOSE 8080

CMD [ "./main" ]
```

Nesse caso criou executou a primeira imagem usando a golang para buildar o projeto main.go, mas depois a imagem a ser gerada e fato a outra imagem usando o scratch que é mais leve para apenas executar o arquivo copiado da imagem golang, proporcionando o mesmo resultado porém de forma mais rápida pelo fato do scratch ser mais leve.
OBS: PODE TER VÁRIAS IMAGENS SENDO EXECUTADAS NO PROCESSO DO DOCKERFILE, MAS SEMPRE A ÚLTIMA É A IMAGEM A SER GERADA DE FATO.

NETWORK DOCKER: Quando se precisa de dois containers se comunicando

- DRIVERS NETWORK:
    - **bridge (padrão)**: Cria uma rede isolada da máquina host para contêineres, permitindo que eles se comuniquem entre si.
    - **host**: Remove o isolamento de rede, fazendo com que o container consiga se comunicar com a máquina host. (FUNCIONA SO NO LINUX NA MÁQUINA HOST)
    - **overlay**: Permite comunicação entre containers em diferentes hosts, conectando múltiplos daemons Docker sem necessidade de roteamento no sistema operacional.
    - **ipvlan**: Dá controle total sobre endereços IPv4 e IPv6, permitindo VLANs e roteamento em camada 3 para integração com redes físicas.
    - **macvlan**: Atribui um endereço MAC ao container, fazendo-o parecer um dispositivo físico na rede, útil para aplicações legadas.
    - **none**: Isola completamente o container, sem acesso à rede do host ou de outros containers.

Docker network ls - Lista as networks criadas.
Docker network inspect <nome da rede> - Mostra todos os detalhes da network através de um json.
Docker network create <nome da rede> - Cria uma rede padrão que usa o bridge.
Docker network connect <nome da rede> <nome do container> - Conecta um container já existente em uma network.
Docker network disconnect <nome da rede> <nome do container> - Desconecta um container de uma network.

OBS: É possível que um container esteja conectado a 2 ou mais networks.
Para fazer resolução de domínio os containers tem que está em uma rede criada por você.
Commands Linux:
cat <nome do arquivo> Mostra todo o conteúdo do arquivo.
touch <nome do arquivo+extensão> - Cria um arquivo no diretório atual
vim <nome do arquivo+extensão> - Cria um arquivo no diretório atual
apt-get update - Atualiza o pacote de gerenciamento apt-get
RUN apt-get install -y <nome instalação> - Faz a instalação de algum pacote, o -y serve para não ter que dar permissão de instalação

Documenta qual porta o container está escutando:
EXPOSE 3984
info extra:
/docker-entrypoint. - Trava o processo de execução de um container para ele não ser finalizado / Processo fica rodando.
Em caso de colocar o mesmo nome em uma imagem na build que já existe no docker, a imagem que já tem no docker perde o nome (fica como none) e a nova imagem gerada vai ter esse nome herdado