Docker

Container é o que vai conter a aplicação. Ela será executada dentro desse container que funcionará junto com o sistema operacional.

Docker é a tecnologia de **containers**:

1. Docker **engine**: Facilita deploy e execução das aplicações. Faz o intermédio com o SO.
2. Docker **compose**: definição e orquestração de múltiplos containers.
3. Docker **Swarm:** coloca múltiplos dockers hosts para trabalharem em um cluster.
4. Docker **Hub**: repositório com várias imagens diferentes para os containers.
5. Docker **machine**: permite instalar e configurar em hosts virtuais.

Imagem: uma série de instruções sobre o que deve ser feito para criar um container.

Comandos mais comuns:

1. docker version: exibe a versão do docker.
2. docker run NOME_DA_IMAGEM: cria um container com a respectiva imagem passada como parâmetro.
3. docker ps: lista todos os containers ativos no momento.
4. docker ps -a: lista todos os containers já criados, inclusive os que estão no estado "parado".
5. docker run -it NOME_DA_IMAGEM: atrela o terminal ao container, ou seja, coloca dentro do container.
6. docker start ID_CONTAINER: para iniciar um container.
7. docker stop ID_CONTAINER -t SEGUNDOS_DE_ESPERA: parar um container. Tem um parâmetro de tempo de espera antes de parar, o default é 10s. Não remove o container do computador.
8. docker start -a -i ID_CONTAINER: para iniciar um container com terminal atrelado.
9. docker rm ID_CONTAINER: remove um container.
10. docker container prune: para limpar todos os container inativos (stopped).
11. docker images: mostra as imagens.
12. docker rmi NOME_DA_IMAGEM: para remover uma imagem.
13. docker run -d NOME_DA_IMAGEM: roda a imagem em background (em segundo plano) e não trava o terminal.
14. docker run -d -P NOME_DA_IMAGEM: linka uma porta externa (localhost) para acessar um site.
15. docker port ID_CONTAINER: para pegar a porta.
16. docker run -d -P --name NOME_QUE_EU_QUISER NOME_DA_IMAGEM: para nomear um container. Posso usar esse nome no lugar do ID.
17. docker run -d -p PORTA:PORTA_DO_CONTAINER NOME_DA_IMAGEM: posso escolher a porta.
18. docker run -d -P -e VAR="VALOR" NOME_DA_IMAGEM: define uma variável de ambiente.
19. docker ps -q: retorna apenas os IDS.
20. docker stop -t 0 $(docker ps -q): para todos os ids que retornarem.
21. docker run -v "[CAMINHO_VOLUME_LOCAL:]CAMINHO_VOLUME_CONTAINER" NOME_DA_IMAGEM: cria um volume no respectivo caminho do container, caso seja especificado um caminho local monta o volume local no volume do container.
22. docker inspect ID_CONTAINER: retorna diversas informações sobre o container.
23. docker pull NOME_DA_IMAGEM: só baixa a imagem do repositório, não executa.
24. docker build -f Dockerfile: cria uma imagem a partir de um Dockerfile.
25. docker build -f Dockerfile -t NOME_USUARIO/NOME_IMAGEM .: constrói e nomeia uma imagem não oficial.
26. docker login: inicia o processo de login no Docker Hub.
27. docker push NOME_USUARIO/NOME_IMAGEM; envia a imagem criada para o Docker Hub.
28. docker pull NOME_USUARIO/NOME_IMAGEM: baixa a imagem desejada do Docker Hub.
29. hostname -i: mostra o ip atribuído ao container pelo docker (funciona apenas dentro do container).
30. docker network create --driver bridge NOME_DA_REDE: cria uma rede especificando o driver desejado.

31. docker run -it --name NOME_CONTAINER --network NOME_DA_REDE NOME_IMAGEM: cria um container especificando seu nome e qual rede deverá ser usada.
32. docker version: exibe a versão do docker que está instalada.
33. docker start ID_CONTAINER: inicia o container com id em questão.
34. docker-compose build: realiza o build dos serviços relacionados ao arquivo docker-compose.yml, assim como verifica a sua sintaxe.
35. docker-compose up: sobe todos os containers relacionados ao docker-compose, desde que o build já tenha sido executado.
36. docker-compose down: para todos os serviços em execução que estejam relacionados ao arquivo docker-compose.yml.
37. docker-compose ps: lista os serviços que estão rodando.

Layered File System: sistema de camadas do docker. Uma imagem é composta de camadas que podem ser reaproveitadas em outras imagens. **As imagens são bloqueadas para escrita**. Ao criar um container estamos criando em cima de uma imagem que já existe. Ele cria uma nova camada read/write acima da imagem principal.

Volumes: lugar onde salvamos os dados dos containers. Aponta para o docker host. Cria-se uma pasta dentro do container e o que estiver escrito nesta pasta passa a ser escrito no docker host. Assim não perdemos os dados.

Containers são voláteis, isto é, ao removê-los removemos os dados juntos. Para deixar os dados persistentes devemos usar Volumes. Os volumes salvos não ficam no container e sim no Docker Host.

Construindo nossas próprias imagens:

Dockerfile: ensina o docker a criar uma imagem a partir da nossa aplicação. É um arquivo de texto. Monta sua imagem a partir de outra imagem base já existente. Posso criar vários arquivos com o nome que eu quiser com extensão .dockerfile.

FROM NOME_DA_IMAGEM:VERSION: inicia o arquivo. Imagem da biblioteca necessária no projeto.
MAINTAINER: nome da pessoa que mantém a imagem.
ENV NOME_VAR=VALOR: para setar variáveis de ambiente.
COPY . NOME_PASTA: adiciona o código fonte da aplicação. O ponto refere-se à raíz do projeto seguido do local onde irá ser salvo.
WORKDIR DIR: diretório onde os comandos devem ser executados.
RUN seguido dos comandos: executa um comando quando a imagem estiver sendo construída.
ENTRYPOINT [COMANDO]: comando para iniciar a aplicação. Comando de entrada. Executado assim que carrega o container. Pode ser substituído pelo RUN COMANDO (ex. RUN npm start).
CMD [PARAMS]: recebe os parâmetros extras para o ENTRYPOINT.
EXPOSE PORT: diz a porta que o container irá usar.

Para criar a imagem tem que fazer o build: docker build -f NOME_DO_DOCKERFILE -t NOME_QUE_VOU_DAR_A_IMAGEM **.** (use um ponto no final).

Para testar, crie um container com a nova imagem. **Subindo** a imagem no **docker hub**:
1. cria uma conta no docker hub
2. docker login
3. docker push NOME_DA_IMAGEM

As imagens são **read-only** sempre. Um container é uma instância de uma imagem. Para guardar as alterações a docker engine cria uma nova layer em cima da última layer da imagem.

As imagens criadas pelo Docker acessam a mesma rede, porém apenas através de IP. Ao criar uma rede é possível realizar a comunicação entre os containers através do seu nome. Durante a criação de uma rede precisamos indicar qual driver utilizaremos, geralmente, o driver bridge.

Docker compose:

Para múltiplos contêineres. **Serve para subirmos vários containers de forma automática.** O arquivo **docker-compose.yml** é um arquivo de texto que descreve tudo que deve acontecer para subir essa aplicação. Informamos todo o processo de subir a aplicação. Usamos tab para identar o arquivo. É parecido com json.

version:versão: primeiro comando é sempre a versão.
services:nome_serviço: adicionamos os serviços, que são as diferentes partes da aplicação
build:
dockerfile:
caminho_arquivo,
context: caminho por onde começar a caçar o serviço (ex.: raíz),
image:nome_image,
container_name:nome_container,
ports: "porta:porta",
network: nome_network,
networks:nome_network:driver: para criar a network
image: nome_image: caso o serviço seja criado a partir de uma imagem ao invés de ser buildado através do dockerfile.
depends_on: nome_serviço: indica que determinado serviço depende que a algum outro serviço suba primeiro para funcionar. Determina a ordem em que os serviços serão buildados.