Comandos Docker:

docker build [FLAGS] [DOCKERFILE\_PATH]: “Builda” uma imagem baseada nas instruções do Dockerfile. O parâmetro “.” para o [DOCKERFILE\_PATH] significa que o Dockerfile encontra-se na mesma pasta do diretório onde o CMD foi aberto. Em caso de mudanças no código ou em outros arquivos, a imagem deve ser **rebuildada!**

Flags:

–t nomeia a imagem a ser buildada

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker run [FLAGS] [CONTAINER\_NAME]: Inicia o contâiner informado

Flags:

–p [LOCAL\_PORT]:[DOCKER\_EXPOSED\_PORT] antes do nome da imagem informa a porta a ser utilizada localmente para acessar a aplicação

--rm deleta o contêiner após ele ser “stoppado”

–it roda o contêiner em modo interativo.

--name dá um nome ao contêiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker start [FLAGS] [CONTAINER\_NAME]: Inicia um contêiner já existente, diferente do docker run, **que cria um novo**.

Flags:

–i inicia o contêiner em modo interativo

–a “attacha-se” ao contêiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker stop [CONTAINER\_NAME]: Para o contêiner informado

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker ps: Lista todos os contêineres e/ou processos rodando (ps = processes)

Flags:

–a lista todos os processos, e não somente aqueles rodando

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker attach [CONTAINER\_NAME]: “Attacha-se” a um contêiner rodando, para que possa-se visualizar o output do contêiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker logs [CONTAINER\_NAME]: Printa os logs do contêiner informado

docker inspect [CONTAINER\_NAME]: Inspeciona as informações de um contêiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker container prune: Deleta todos os contêineres

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker image prune: Deleta todas as imagens que estão “untagged”

Flags:

-a remove **TODAS** as imagens

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker volume prune: Deleta todos os volumes

Volumes

Volumes são pastas compartilhadas entre o contêiner e o computador local. São listados pelo comando docker volume ls

**Anonymous volumes:** Não são nomeados pelo desenvolvedor, e de tal forma, desaparecem assim que o contêiner é deletado. Quando utilizados juntamente aos “bind mounts”, eles podem “apontar” para uma pasta que não será sobreescrita por ele. Quando fazemos um “bind mount” da pasta-raiz do projeto no WORKDIR do contêiner com a pasta na máquina local, as pastas que não existem localmente somem do contêiner, **mesmo que rodemos um “npm install” no Dockerfile para gerar o node\_modules**. Neste cenário, a utilização dos “Anonymous volumes” se demonstra útil. Sua utilização é melhor debatida nas aulas 51 e 53 do curso.

* (No Dockerfile) VOLUME [ “[PATH\_INSIDE\_THE\_CONTAINER\_]” ]
* (Ao criar um contêiner) docker run –v [VOLUME\_PATH\_INSIDE\_CONTAINER]

**Named Volumes:** Utilizados quando é necessário que o volume não desapareça ao deletar o contêiner. Não estão fortemente atrelados a somente um contêiner.

* -v [VOLUME\_NAME]:[VOLUME\_PATH\_INSIDE\_CONTAINER] cria um “named volume”, usado para que os dados não se percam ao finalizar um contêiner

**Bind Mounts:** Utilizados quando precisamos que uma pasta dentro do contêiner possa ser modificada de fora dele, ou vice-versa. Por exemplo, quando não queremos re-buildar a imagem sempre que uma modificação for feita no código.

* -v [FOLDER\_PATH\_TO\_BE\_BOUND\_MYPC]:[ FOLDER\_PATH\_TO\_BE\_BOUND\_CONTEINER] cria um “bind mount”, usado para compartilhar arquivos e mudanças em uma pasta específica entre o PC e contêiner

Comunicação Local-Contêiner

Deve-se adicionar a keyword ‘host.docker.internal’ no lugar de ‘localhost’, segue exemplo:

* mongodb://localhost:27017/swfavorites -> mongodb://host.docker.internal:27017/swfavorites

Comunicação Contêiner–Contêiner

É possível usar o comando docker inspect [CONTAINER\_NAME], encontrar o IP do contêiner em questão, e inserí-lo na URL de comunicação, como segue o exemplo, visando comunicar-se com um MongoDb hospedado em outro contêiner:

* mongodb://localhost:27017/swfavorites -> mongodb://[IP\_EXEMPLO]:27017/swfavorites

Entretanto, essa não é a melhor forma de fazê-lo.

Como alternativa, **podemos criar uma rede interna do docker**, que possibilita a comunicação entre contêineres que são incluídos nela.

Para criar a rede, usamos o comando:

* docker network create [NETWORK\_NAME]

Logo em seguida, podemos incluir contêineres nela. Como exemplo, usarei um contêiner contendo uma imagem do MongoDb, para que ele se comunique com outro contêiner de uma aplicação Nodejs:

* docker network create favorites
* docker run --network favorites –d --rm mongo
* docker run --network favorites –d --rm mynodejscontainer

Após isso, **a URL de conexão na aplicação Nodejs poderá usar, como host, o nome do contêiner que hospeda o banco**, como segue o exemplo:

* mongodb://[IP\_EXEMPLO]:27017/swfavorites -> mongodb://mongo:27017/swfavorites

Environment variables

* (No Dockerfile) ENV [VAR\_NAME] [VAR\_VALUE] cria uma variável de ambiente

Com a utilização de uma **ENV**, podemos referenciá-la dentro do Dockerfile com um ‘$[VAR\_NAME]’, como segue o exemplo:

ENV PORT 80

EXPOSE $PORT

O valor setado no Dockerfile é o padrão, podendo ser alterado no comando docker run

docker run –p 3000:8000 –-env PORT 8000

Também é possível criar um arquivo ‘.env’, que contém as variáveis. Consultar curso, Aula 61, min. 7:10.