**Curso Docker e Docker Compose Alura**

- No Windows uso o power shell, no Linux o bash normal

Comandos:

* **docker run hello-world** : procura a imagem localmente para criar um container, se não acha a imagem vai no repositório ([www.hub.docker.com](http://www.hub.docker.com)) e procura a imagem e faz o download (pull). Cria o container e starta o container.
* **docker ps** : lista todos os containers ativos no momento.
* **docker ps -a** : exibe todos os containers inclusive os parados. (ou docker ps --all)
* **docker run -it ubuntu** : cria um novo container e atrela o meu terminal ao container
  + -it : atrela o terminal ao container
  + control + D ou exit sai do terminal do container
* **docker start [parte do id do container]** : inicia o container sem criar outro novo, inicia o container que o id foi passado
* **docker stop [parte do id do container]** : para o container de id especificado
* **docker start -a -i [parte do id do container]**: inicia o container e já atrela o terminal ao container
  + -a : flag de attach
  + -i : flag de interação com o terminal (input e output)
* **docker exec -it [nome do container] bash** : apos ter usado o cmdo start ou run esse comando da o terminal do container de volta para você.
* **docker --help** : traz toda a lista de comandos do docker
* **docker [comando] --help** : traz os comandos e flags atreladas ao comando passado por parâmetro
* **docker rm [parte do id do container]** : remove o container que eu especifiquei
  + -f : forca a parada
* **docker container prune** : remove todos os containers inativos
* **docker images** : exibe todas as imagens que tenho localmente
* **docker rmi [nome da imagem]** : remove a imagem especificada
* **docker run -d [nome da imagem]** : a flag -d permite que o comando run seja executado e o terminal não fique travado, ou seja, posso usar o terminal (detached)
* **docker stop -t [qtd de segundos] [nome do container]** : mata o container em X segundos (o padrão é 10 segundos)
* **docker run -d -P [nome da imagem]**: a flag –P (p maiúsculo) atribui uma porta do container para o host poder se conectar a aplicação que esta dentro do container
  + **flags**:
    - --name : possibilita que eu de um nome ao meu container. Ex: docker run -d -P --name [nome que se quer dar] [nome da imagem]
    - -p : escolher a porta que do host que sera mapeada. Ex: docker run -d -p [porta host]:[porta container] [nome da imagem]; ex2: docker run -d -p 12345:80 dockersamples/static-site
    - -e : seta uma variável de ambiente. Ex: docker run -d -P –e AUTHOR=”Joao Carlos” dockersamples/static-site
    - -w : seta o diretório de trabalho. Por exemplo: docker –w “var/www” node npm start (ira criar um container com node e executar o comando npm start no diretório var/www)
    - --network : atrela uma rede a esse container. Ex: docker run --network [nome da rede] [nome da imagem]
* **docker port [id do container] :** mostra as portas que estao sendo utilizadas e mapeadas pelo container
* **docker-machine ip** : se estiver usando Docker toolbox esse comando serve para que seja exibido o IP que o Oracle Virtual Box esta usando, para que seja possível com esse ip se conectar as aplicações dentro dos containers
* **docker ps -q** : retorna apenas os ids dos containers que estão rodando
* **$([comando docker])** : vai executar o comando e por o output. Exemplo: parar todos os containers: docker stop $(docker ps -q)
* **docker pull [nome da imagem e versão] :** faz o download da imagem do repositório docker hub

**Volumes**

* **docker run -v “var/www” ubuntu** : cria um container e um volume no caminho var/www. Para ver onde no host esse volume esta sendo refletido posso usar o comando abaixo
* **docker inspect [id do container]** : a saida Mounts demonstra onde foi montada a pasta no host para o volume do container
* **docker run -v “[nome da pasta no host]”:”[nome do diretório no container]” [nome da imagem]** : cria um volume no container direcionando para a pasta que especifiquei do meu host.
* **docker run [flags] [imagem] [comando] :** após o nome da imagem posso passar algum comando que já quero que seja executado quando esse container for iniciado. Por exemplo: docker run -d node npm start (inicia o node js no container)

**Dockerfile**

Serve para orientar o build da sua própria imagem. Para criar uma imagem minha devo introduzir um arquivo chamado Dockerfile ou qualquercoisa.dockerfile, geralmente na pasta raiz do meu código.

* **docker build** : faz o build da sua imagem
  + -f : [nome do arquivo dockerfile]. Ex: Dockerfile
  + -t : [nome da tag, geralmente nome/tipo\_imagem]. Ex: costalopes/node
  + Qual o contexto para procurar o dockerfile. Ex: .
* **Comandos dockerfile:**
  + **FROM :** a imagem base a partir da qual minha imagem sera montada. Ex: node:6, node:latest
  + **MAINTAINER :** pessoa que criou e esta cuidando da imagem. Ex: Joao Lopes
  + **COPY :** copia o código fonte da aplicação para dentro da imagem. COPY [path do código] [path do destino na imagem]. Ex: COPY . /var/www
  + **WORKDIR :** diretório onde os comandos serão executados assim que o container for carregado. Ex: WORKDIR /var/www
  + **RUN :** comando a ser executado quando a imagem estiver sendo construída. Ex: RUN npm install
  + **ENTRYPOINT :** comando de entrada assim que o container for iniciado. Ex: ENTRYPOINT npm start
  + **EXPOSE :** diz qual a porta do container sera exposta
  + **ENV :** seta uma variável de ambiente. Ex: NODE\_ENV=production, ENV PORT=3000. Posso usar essas variáveis, por exemplo, EXPOSE $PORT

**Docker Hub**

* **docker login :** faz o login no docker hub
* **docker push [nome da imagem] :** manda a imagem para o docker hub
* **docker pull [nome da imagem] :** baixa a imagem
* **docker pull run [nome da imagem] :** baixa e executa a imagem

**Network**

Para que os containers possam comunicar entre si, o docker poe todos eles em uma rede default que ele mesmo cria. Para comunicar nessa rede default basta utilizar o ip que o docker ira atribuir para aquele container.

O problema dessa abordagem por exemplo, é que você não saberá de antemão o endereco ip para poder configurar na sua aplicação, pois o docker atribui um ip diferente a cada hora que você sobe um novo container. Por isso, o docker permite que você crie sua própria rede e ao criar sua própria rede posso conectar os containers através do nome que dei a eles.

* **docker network ls** : lista todas as redes existentes
* **docker network create --driver [o driver que quero criar, 99% dos casos é brigde] [nome da rede]**
* **docker run --network [nome da rede que deseja atrelar] [nome da imagens]**
* **docker network inspect [rede a inspecionar] :** inspeciona a rede que eu determinar

**Docker-compose**

* **docker-compose build:** irá buildar todas as imagens envolvidas no docker-compose.yml. Ex: docker-compose build (lembrar de estar na pasta raiz do projeto onde esta o docker-compose.yml)
* **docker-compose up:** executa o docker-compose.yml fazendo todo o passo a passo descrito nele
* **docker-compose up -d :** executa a mesma coisa que o comando de cima mas em modo detached, ou seja, não prende o terminal
* **docker-compose ps:** exibe os containers que estão rodando
* **docker-compose down:** para todos os containers e remove
* **docker exec -it [nome do container] ping [nome do container OU nome do serviço]:** pinga o IP do serviço ou container passado como parâmetro
* **docker-compose stop:** para os serviços/containers
* **docker-compose restart:** restarta os serviços/containers