**PREVIEW DO CURSO – Iniciando com Docker | School Of Net**

**- como realizar a instalação do Docker;**

**- como criar containers;**

**- como expor a porta para poder ter acesso ao container;**

**- conectar ao container através da máquina local;**

**- como linkar containers, exemplo um de wordpress, outro de mysql;**

**- compartilhamento de volumes entre a máquina local e o container;**

**- 2 formas de automatizar o containers**

**-> docker file (criar a sua própria imagem)**

**-> automatizar a criação de vários containers (ex.: 1 PHP, 1 Mysql, 1 AngularCLI), com o Docker Compose.**

**Digamos que por algum motivo seu SO não suportaria PHP7, não suportaria o angularCLI, você usaria o docker e ainda poderia garantir o mesmo enviroment para todo o time.**

**Vale muito a pena se eu tiver 30 desenvolvedores, por exemplo**

**Trabalha com containers em paralelo, cada um com sua responsabilidade, ex. em um angular\_cli outro php, etc**

**Mesmo ambiente pra todo o setor de produção**

**Possui um arquivo de configuração que prepara o ambiente em segundos...**

**O desnecessário é descartado, ex. para rodar o php ele nao usa um sistema operacional inteiro, usa apenas o estritamente necessário para rodar o PHP**

**Diferente da virtualização, que tem o SO inteiro**

**na virtual machine: hypervisor**

**no docker: docker engine (usa kernel)**

**Foi feito basicamente para rodar no Linux**

**No Windows a partir do 10 Professional or Enterprise 64-bit: roda no hyper-v nativo do SO**

**Abaixo disso: Baixa o Docker Toolbox, precisa ativar hyper-V, e precisa ter uma virtual box instalada** [**www.virtualbox.org**](www.virtualbox.org)

**INSTALAÇÃO INICIAL DO DOCKER**

**- baixou/ instalou o Virtual Box;**

**- baixou/ instalou o Docker;**

**\* A máquina virtual do Docker recebe um IP**

**COMANDOS BÁSICOS**

**$ docker version**

**vai exibir um client e um server**

**$ docker run hello-world (hello-world é uma imagem que o docker tem de pronta para teste)**

**$ docker images (vai exibir todas as imagens baixadas que meu pc tem)**

**$ docker ps (lista todos os containers ativos, ou seja, rodando no momento)**

**$ docker ps -a (lista todos os containers criados)**

**$ docker rm nome\_do\_container (remove o container, mas permanece a imagem, a imagem serve para eu criá-lo novamente com base nela)**

**LINKANDO CONTAINERS COM WORDPRESS**

**Vamos imaginar que temos um Wordpress rodando em um container e um Mysql rodando em outro.**

**Criando container mysql**

**Para criar um container, rode o seguinte comando:**

**$ docker run -d --name dbserver -e "MYSQL\_ROOT\_PASSWORD=root" -e "MYSQL\_DATABASE=wordpress" mysql**

**Note que -e é um parâmetro que ainda não vimos. O -e ou --env são variáveis de ambiente que, em certos casos, são necessárias.**

**Neste caso, é necessária para configurar o mysql durante o carregamento.**

**Durante a criação do container, o Docker já irá setar o password e criar o banco de dados wordpress.**

**O mais importante é entendermos que existem containers que dependem de variáveis de ambiente, para que eles consigam rodar suas configurações iniciais.**

**Estamos criando nosso container baseado na imagem oficial do Mysql.**

**Nota: Existem outras variáveis de ambiente, como:**

**MYSQL\_USER e MYSQL\_PASSWORD, caso queiram criar um usuário e uma password para seu ambiente, além do root.**

**Criando container Wordpress**

**Para criar o container, rode o seguinte comando:**

**$ docker run -d --name wordpress -p 8085:80 wordpress**

**Rodando o comando acima, o container wordpress não reconhecerá nosso container Mysql. O Wordpress necessita do mysql para sua instalação.**

**Para resolver esta questão, devemos inserir mais um parâmetro importante do comando run, que segue abaixo.**

**$ docker run -d --name wordpress --link dbserver:mysql -p 8085:80 wordpress**

**O parâmetro --link dbserver:mysql cria o vínculo entre nosso container mysql e o container wordpress.**

**O comando informa o Docker: "Gostaria que meu container dbserver fosse reconhecido dentro do container wordpress como mysql. Isso quer dizer que, o que vem após o : será o registro no container wordpress, podendo assim escolher o nome que quiser. Em nosso caso utilizamos o mysql.**

**CRIANDO CONTAINER Nginx E EXPONDO PORTA**

**$ docker run nginx**

**\* em docker hub (hub.docker.com) você tem um repositório com todas as imagens de enviroment disponíveis, inclusive nginx**

**\* não passando a versão (na line command), vai pegar a latest**

**\* o terminal vai ficar parado, mas nao se assuste, o servidor está rodando**

**\* ctrl + c mata o processo**

**$ docker run -d nginx (com o -d a tela não trava...)**

**Se vc estiver usando Windows, utilizando o Docker Toolbox, vai ter um ip da maquina virtual do docker**

**Se estiver usando o windows ou mac de forma nativa, com o Hyper -V, não precisa usar IP da maquina virtual (até pq não existe), vc vai acessar o seu browser novamente.**

**Para saber o IP da maquina virtual:**

**$ docker -machine (para ver uma lista de comandos, inclusive o comando abaixo)**

**$docker machine ls (vai exibir algumas informações, como abaixo)**

**NAME (default)| ACTIVE (\*) | DRIVER (virtualbox) | STATE (running) | URL (tcp://192.168.99.100:2376) | SWARM () | DOCKER (v1.12.3) | ERRORS**

**Acessou 192.168.99.100 e não funcionou**

**Mas quando rodei $ docker ps, ele mostrou que a porta 80 está liberada: PORTS (80/tcp, 443/tcp)**

**Então, ele matou esta instância, para rodar novamente expondo a porta, então echo que vc não consegue expor a porta depois que você instanciou sem expor a porta**

**$ docker stop 89f9785618a2 (parou a maquina, ref ao container IP)**

**$ docker rm 89f9785618a2 (removeu a instância)**

**$ docker run -d -p 8080:80 nginx (o "-p" é que mapeia a porta que vamos utilizar)**

**estou expondo a porta 80 do serv docker para a porta 8080 da minha máquina**

**automaticamente, quando acessar a porta 8080 do browser, estarei acessando a porta 80 do nginx**

**agora mudou quando rodei $ docker ps, está mostrando PORTS (443/tcp, 0.0.0.0:8080)**

**agora acessou 192.168.99.100:8080**

**depois deu $ docker stop clever\_hooper**

**recaptulando**

**"-d" para rodar no mesmo terminal,**

**"-p" para expor a porta**

**"--name nome\_minha\_imagem\_de\_servidor" para dar um nome e nao ficar usando os nomes esquisitos que o Docker dá automaticamente**

**melhor exemplo:**

**$ docker run -d -p 8080:80 nome\_meu\_webserver nginx**

**$ docker ps (vai listar, os baixados e os que estão rodando)**

**$ docker stop nome\_meu\_webserver (parou)**

**$ docker start nome\_meu\_webserver (rodou novamente)**

**TRABALHANDO DE FORMA INTERATIVA COM exec**

**$ docker exec nome\_meu\_webserver (+ comandos)**

**$ docker exec nome\_meu\_webserver uname -a (comando linux p exibir versao do kernel)**

**$ docker exec nome\_meu\_webserver bash (nao vai rodar de forma interativa, o cmd vai ficar estático)**

**$ docker exec -it nome\_meu\_webserver bash (acessa o container por linha de comando e consegue rodar comandos)**

**depois instalou o vim**

**depois acessou o index.html do projeto com o vim**

**> apt-get update**

**> apt-get install vim**

**> cd /usr/share/nginx/html/**

**> vim index.html**

**e teve acesso ao arquivo, alterou o arquivo e salvou**

**os arquivos ficam na maquina**

**COMPARTILHANDO VOLUMES**

**Qual objetivo de compartilhar volumes?**

**O objetivo é, basicamente, que o Nginx, quando for carregado, deixe de ler o diretório padrão que vem configurado. Ele passará a ler o nosso diretório como base de inicialização.**

**Em vez de ler o arquivo index.html, padrão, que está em /usr/share/nginx/html do servidor, passará a ler o nosso arquivo index.html, presente em nossa máquina.**

**Desta forma, se fizermos uma alteração local, em qualquer arquivo de nosso projeto, terá efeito direto no servidor, porque o container está criando um apontamento ou compartilhamento com nossa pasta local.**

**Para conseguirmos este apontamento, basta rodar o seguinte comando:**

**$ docker run -d --name webserver2 -p 8081:80 -v /c/Users/wesley/docker-project/html:/usr/share/nginx/html nginx**

**Caso queira, ou goste de economizar código, podemos otimizar ainda mais o comando anterior:**

**$ docker run -d --name webserver2 -p 8081:80 -v $(pwd)/html:/usr/share/nginx/html nginx**

**Desta forma o $(pwd) pega, automaticamente, o caminho da pasta do seu projeto, desde que, você esteja rodando o comando de dentro dela.**