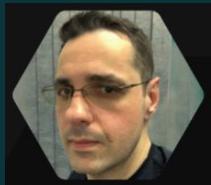


DOCKER APLICADO AO PROTHEUS

Renato Ferreira Campos





Renato Ferreira Campos



LinkedIn

<https://bityli.com/fZWDwXu>

Eu sou SRE Protheus (Disponibilização do ERP em nuvens públicas utilizando Kubernetes, HELM, ARGOCD e NodeJs | Linux | Shell | Shellsript | Bash | Docker | KVM | Kubernetes | Git | Ubuntu | RHEL | AWS | GCP | DevOps | GitHub | GitLab | Raspberry Pi | HELM | ArgoCD | Protheus | Adv/PL | Clipper | Python | NodeJs | Angular)

Formado em Sistemas de Informações e Administração, ambas pela Universidade de Mogi das Cruzes e pós-graduado em administração pela FGV.

Estou na Totvs desde 2006, onde desempenhei as funções:

- Analista de sistemas Protheus com foco em controladoria e finanças
- Liderança do time de desenvolvimento de controladoria e finanças
- Liderança do time de controladoria e logística
- Coordenação do suporte e desenvolvimento do PRIME (grandes contas)
- Coordenação do suporte de controladoria e finanças
- Coordenação do suporte de TAF eSocial e Product Owner do Smart eSocial TOTVS (SaaS).

Conhecimento em regra de negócio de controladoria e finanças, fiscal, e-social. Atuação com as linguagens TOTVS Protheus, Devops e fuçador nas demais linguagens.



MOTIVAÇÃO

Imagina que o(a) Dev está desenvolvendo uma TASK para correção de um programa que faz a leitura/acesso de dados em disco. Ao terminar a implementação, realiza os testes locais em sua máquina WINDOWS e envia para o(a) *cliente*, que possui um ambiente LINUX. Pronto já é motivo para dar erro no processo.

A utilização do Docker tende a resolver isso, pois o(a) Dev poderá utilizar um ambiente configurado parecido com o do cliente, tanto para desenvolver quanto para testar, minimizando as chances de erros.

Resumindo: Aumentar a produtividade e cobertura de testes e ajudar a resolver o famoso problema: "Ah, na minha máquina funciona", uma vez que o ambiente de desenvolvimento será o mesmo utilizado nos testes e também na produção.



IMPORTANTE: Este processo não é homologado para ambientes de produção! Serve para auxiliar os times de desenvolvimento e suporte na montagem e testes em ambientes próximos aos dos clientes reais.



O QUE É DOCKER?

O *Docker* é uma plataforma de código aberto para criar, implantar e gerenciar aplicativos em contêiner. Ele permite que os desenvolvedores empacotem aplicativos em subpartes controladas — componentes executáveis padronizados que combinam o código-fonte do aplicativo com as bibliotecas do sistema operacional e as dependências necessárias para executar o código em qualquer ambiente.



O QUE SÃO CONTÊINERES

Os contêineres são como unidades independentes de *software* que podem ser deslocados de um servidor para outro servidor, sendo executado da mesma forma, porque são isolados no nível do processo e possuem seu próprio sistema de arquivos.

Ao simplificar essa operação, o *Docker* rapidamente se aproximou de um padrão de fato no setor. O programa permite que os desenvolvedores implementem, repliquem, movam e façam *backup* de uma carga de trabalho de maneira simplificada. A base está no uso de um conjunto de imagens reutilizáveis para torná-las mais portáteis e flexíveis do que os métodos anteriores.

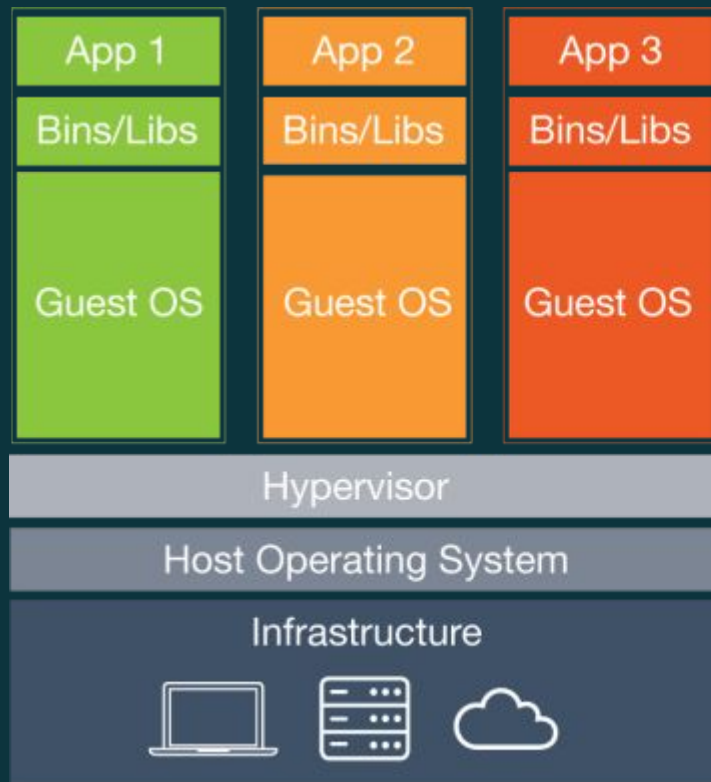
Vantagens

- Os contêineres do *Docker* fornecem uma maneira de criar aplicativos que são mais fáceis de construir e alterar do que os métodos anteriores. Isso oferece várias vantagens para os desenvolvedores de *software*.
- O *Docker* permite que os aplicativos e seus ambientes sejam mantidos limpos e mínimos, isolando-os, o que concede um controle mais granular (em pequenas partes) e maior portabilidade.
- Os módulos distintos da ferramenta permitem a composição. Os contêineres facilitam para os desenvolvedores compor os blocos de construção de um aplicativo em uma unidade modular com partes facilmente intercambiáveis. Esse fato acelera os ciclos de desenvolvimento, lançamentos de recursos e correções de *bugs*.

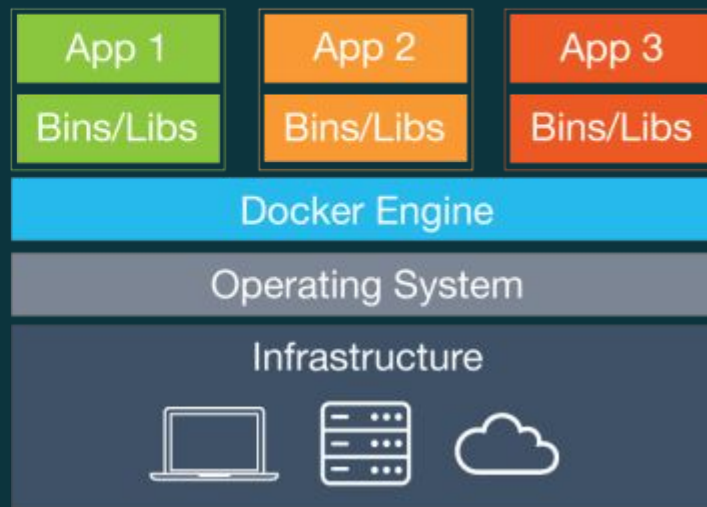
Pontos de Atenção

- Resolve muitos problemas, mas não resolve todos os problemas do desenvolvedor. Não são máquinas virtuais. Ao contrário delas, os contêineres usam porções controladas dos recursos do sistema operacional do *host*, o que significa que os elementos não são tão estritamente isolados quanto seriam em uma VM.
- Não fornecem velocidade “*bare-metal*”. Eles são significativamente mais leves e mais próximos do *metal* do que as máquinas virtuais, mas resultam em alguma sobrecarga de desempenho. Se a carga de trabalho exigir velocidade “*bare-metal*”, um contêiner o aproximará do resultado, mas não o levará até a conclusão do objetivo.
- Os contêineres do *Docker* são imutáveis. São inicializados e executados a partir de uma imagem que descreve seu conteúdo.
- Essa imagem é imutável por padrão. Mas, uma instância de contêiner é transitória. Quando removido da memória do sistema, desaparece para sempre. Se o usuário deseja que seus contêineres persistam no estado entre sessões, como faria uma máquina virtual, é preciso projetar para essa persistência.

VM vs DOCKER



Virtualização



Docker



INSTALANDO O DOCKER



Instalando o Docker

O Docker Desktop está disponível para Mac, Linux e Windows. Para obter informações de download, requisitos do sistema e instruções de instalação, consulte:

- [Instale o Docker Desktop no Linux](#)
- [Instale o Docker Desktop no Mac](#)
- [Instale o Docker Desktop no Windows](#)



Requisitos

Aqui vem a parte mais chata do Docker..... os requisitos.

Microsoft Windows

- Windows 11 de 64 bits: Home ou Pro versão 21H2 ou superior, ou Enterprise ou Education versão 21H2 ou superior.
- Windows 10 de 64 bits: Home ou Pro 21H1 (build 19043) ou superior, ou Enterprise ou Education 20H2 (build 19042) ou superior.
- Habilite o recurso WSL 2 no Windows. Para obter instruções detalhadas, consulte a documentação da Microsoft .
- Os seguintes pré-requisitos de hardware são necessários para executar com êxito o WSL 2 no Windows 10 ou Windows 11:
 - Processador de 64 bits (Dual Core no mínimo)
 - 4GB de RAM do sistema (Recomendo 8GB)
 - O suporte à virtualização de hardware no nível do BIOS deve ser ativado nas configurações do BIOS. Para obter mais informações, consulte a Virtualização .
- Baixe e instale o pacote de atualização do kernel Linux .
- Muito espaço em DISCO.



Instalando o Docker

<https://docs.docker.com/desktop/install/windows-install/>

1. Entre no link <https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>
2. Clique duas vezes em Docker Desktop Installer.exe para executar o instalador.
3. Quando solicitado, verifique se a opção Usar WSL 2 em vez de Hyper-V na página Configuração está selecionada ou não, dependendo de sua escolha de back-end. Se o seu sistema suportar apenas uma das duas opções, você não poderá selecionar qual back-end usar.
4. Siga as instruções no assistente de instalação para autorizar o instalador e prosseguir com a instalação.
5. Quando a instalação for bem-sucedida, clique em Fechar para concluir o processo de instalação.
6. Se sua conta de administrador for diferente da sua conta de usuário, você deverá adicionar o usuário ao grupo docker-users . Execute o Gerenciamento do Computador como administrador e navegue até Usuários e Grupos Locais > Grupos > usuários docker . Clique com o botão direito do mouse para adicionar o usuário ao grupo. Saia e faça login novamente para que as alterações entrem em vigor.

IMPORTANTE: Durante a instalação do Docker para Windows, você será questionado sobre utilizar contêineres Windows ao invés de Linux. Fique à vontade para testar esta opção, mas recomendamos que NÃO DEIXE-A MARCADA.




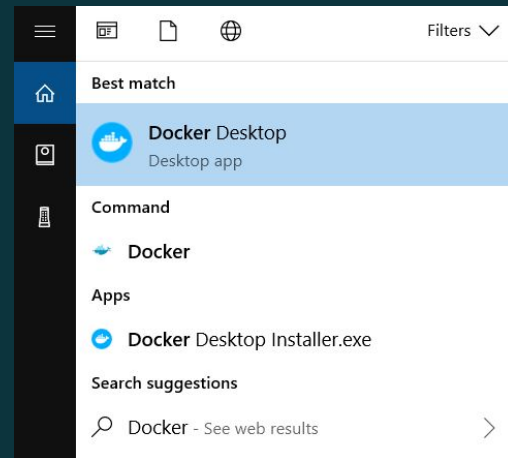
UTILIZANDO O DOCKER

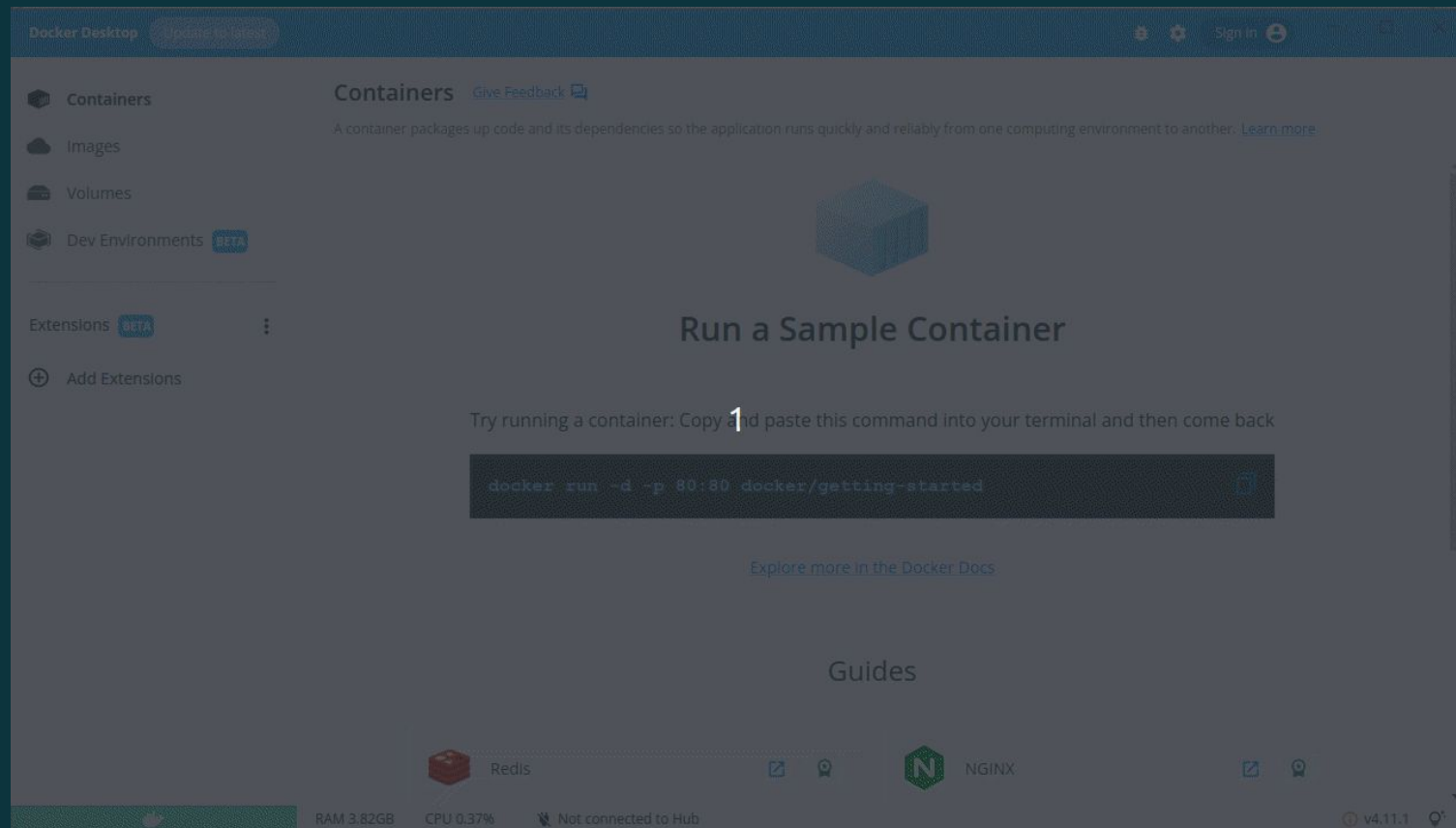


Inicie o Docker Desktop

O Docker Desktop não inicia automaticamente após a instalação. Para iniciar o Docker Desktop:

1. Pesquise Docker e selecione Docker Desktop nos resultados da pesquisa.
2. O menu Docker () exibe a janela Docker Subscription Service Agreement. Aqui está um resumo dos pontos principais:
 - O Docker Desktop é gratuito para pequenas empresas (menos de 250 funcionários e menos de US\$ 10 milhões em receita anual), uso pessoal, educação e projetos de código aberto não comerciais.
 - Caso contrário, requer uma assinatura paga para uso profissional.
 - Assinaturas pagas também são necessárias para entidades governamentais.
 - As assinaturas Docker Pro, Team e Business incluem o uso comercial do Docker Desktop.
3. Selecione Aceitar para continuar. O Docker Desktop é iniciado depois que você aceita os termos.







EXECUTANDO O PROTHEUS NO DOCKER



Passo 1 - Baixar os binários do site da TOTVS

Passo 2 - Criar uma pasta que será compartilhada entre o seu Sistema Operacional e a imagem docker

Passo 3 - Descompactar os binários dentro da sua pasta local (Se possível, já criar o appserver.ini apontando para estas pastas)

Passo 4 - Baixar a imagem base para utilização do protheus (Vamos utilizar o ORACLELINUX:8.5)

Passo 5 - Executar a imagem base, mapeando os volumes dentro do seu docker (docker run -d --name protheus_linux --net host -v /totvs/docker/appserver:/protheus -v /totvs/docker/volume:/volume -it oraclelinux:8.5)

Passo 6 - Executar o protheus dentro do container (docker exec -it protheus_linux bash)



<https://drive.google.com/file/d/1Kug3NmL-gWlka06NPg0ljpMg4IM8x9vY/view?usp=sharing>



CRIANDO UMA IMAGEM DO PROTUEUS NO DOCKER



Antes de explicarmos como criar sua imagem, vale a pena tocarmos em uma questão que normalmente confunde iniciantes do docker:

Qual a diferença entre Imagem e Container?

Traçando um paralelo com o conceito de orientação a objeto, a imagem é a classe e o container o objeto. A imagem é a abstração da infraestrutura em estado somente leitura, de onde será instanciado o container.

Todo container é iniciado a partir de uma imagem, dessa forma podemos concluir que nunca teremos uma imagem em execução.

Um container só pode ser iniciado a partir de uma única imagem. Caso deseje um comportamento diferente, será necessário customizar a imagem.



Como criar imagens

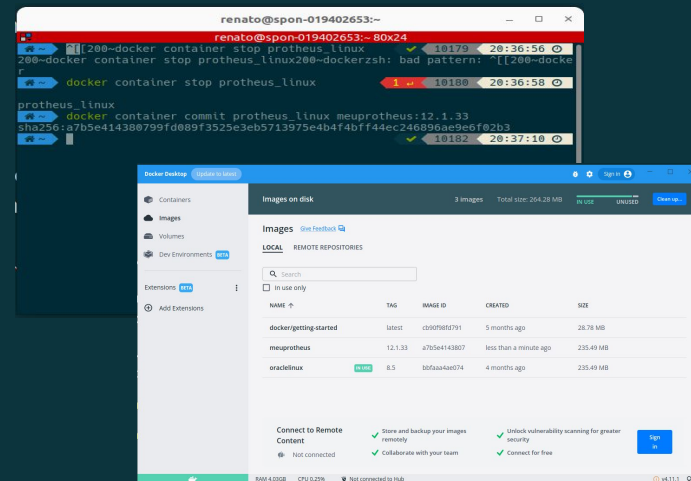
Há duas formas de criar imagens customizadas: com **commit** e com **Dockerfile**.

Utilizando o COMMIT

É possível criar imagens executando o comando commit, relacionado a um container. Esse comando usa o status atual do container escolhido e cria a imagem com base nele.

Vamos ao exemplo. Pegando o contêiner em execução PROTHEUS_LINUX, já com todo o processo configurado, posso simplesmente executar o comando:

- **docker container stop protheus_linux**
- **docker container commit protheus_linux meuprotheus:12.1.33**





Utilizando o DockerFile

Podemos resumir que o arquivo Dockerfile, na verdade, representa a exata diferença entre uma determinada imagem, que aqui chamamos de base, e a imagem que se deseja criar. Nesse modelo temos total rastreabilidade sobre o que será modificado na nova imagem.

Criando a imagem

Crie um arquivo chamado Dockerfile e dentro dele o seguinte conteúdo:

<pre>FROM oraclelinux:8.5</pre> <pre>RUN mkdir -p /protheus/</pre> <pre>ENV LD_LIBRARY_PATH=/protheus/bin:</pre> <pre>COPY /appserver /protheus/</pre> <pre>COPY /volume /volume</pre> <pre>WORKDIR /protheus/bin</pre> <pre>ENTRYPOINT /protheus/bin/appsrvlinux</pre>	<p><i>No exemplo ao lado utilizamos cinco instruções:</i></p> <p>FROM para informar qual imagem usaremos como base, nesse caso foi oraclelinux:8.5</p> <p>RUN para informar quais comandos serão executados nesse ambiente para efetuar as mudanças necessárias na infraestrutura do sistema. São como comandos executados no shell do ambiente, igual ao modelo por commit, mas nesse caso foi efetuado automaticamente e, é completamente rastreável, já que esse Dockerfile será armazenado no sistema de controle de versão.</p> <p>ENV para informar as variáveis de ambiente que desejamos executar dentro do contêiner.</p> <p>COPY é usado para copiar arquivos da estação onde está executando a construção para dentro da imagem. Essa instrução é muito utilizada para enviar arquivos de configuração de ambiente e códigos para serem executados em serviços de aplicação.</p> <p>WORKDIR é usado setar a pasta padrão de execução do container</p> <p>ENTRYPOINT para informar o executável a ser executado na subida do contêiner.</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



Processando o arquivo Dockerfile

Muito bem já temos o arquivo Dockerfile que descreve as etapas que serão executadas para criar a imagem. Para processar o arquivo Dockerfile usamos o comando : `docker build -t <imagem>` .

Vamos supor que quero criar a imagem com o nome : `harpia:20.1.0.11`

Para fazer isso usamos o comando `build` e informamos o nome da imagem, a tag e um ponto(.). O comando fica assim:

`docker build -t harpia:20.1.0.11 .`

onde:

- `docker build` -> O comando
- `-t` -> Parâmetro usado para informar que a imagem pertence ao meu usuário
- `harpia:20.1.0.11` -> O nome da imagem e a tag atribuída à imagem
- `.` -> significa o diretório atual (*pois dei o build dentro da pasta do Dockerfile*)

Nota: O build não trabalha com o caminho do arquivo, apenas com o seu diretório, então é preciso informar o caminho do diretório ou, no nosso caso, como estamos no diretório onde se localiza o Dockerfile, apenas um ponto(.) para identificar que o Dockerfile está no diretório atual.

O comando `docker build` constrói uma imagem a partir de um Dockerfile e de um contexto. O contexto do build é o conjunto de arquivos na localização especificada `PATH` ou `URL`. O `PATH` é o diretório no seu sistema de arquivos local e a `URL` é a localização do repositório GIT.



Verificando a execução

Muito bem já temos a imagem gerada e agora precisamos colocá-la em execução, para isso executamos o comando: **docker run -d --name harpia_20.1.0.11--net host -it harpia:20.1.0.11**

onde:

- `docker run -d` -> O comando
- `--name` -> Parâmetro usado para informar o nome para o contêiner, `harpia_20.1.0.11`
- `--net host` -> Parâmetro para compartilhar sua rede com o contêiner;
- `-it harpia:20.1.0.11` -> Qual a imagem que quero executar

*Se tudo estiver correto, seu protheus deve estar no ar, para conferir, basta executar o comando: **docker logs -f harpia_20.1.0.11***



Comandos para o DOCKER

Comandos relacionados às informações

- `docker version` - exibe a versão do docker que está instalada.
- `docker inspect ID_CONTAINER` - retorna diversas informações sobre o container.
- `docker ps` - exibe todos os containers em execução no momento.
- `docker ps -a` - exibe todos os containers, independentemente de estarem em execução ou não.

Comandos relacionados à execução

- `docker run NOME_DA_IMAGEM` - cria um container com a respectiva imagem passada como parâmetro.
- `docker run -it NOME_DA_IMAGEM` - conecta o terminal que estamos utilizando com o do container.
- `docker run -d -P --name NOME dockersamples/static-site` - ao executar, dá um nome ao container.
- `docker run -d -p 12345:80 dockersamples/static-site` - define uma porta específica para ser atribuída à porta 80 do container, neste caso 12345.
- `docker run -v "CAMINHO_VOLUME" NOME_DA_IMAGEM` - cria um volume no respectivo caminho do container.
- `docker run -it --name NOME_CONTAINER --network NOME_DA_REDE NOME_IMAGEM` - cria um container especificando seu nome e qual rede deverá ser usada.

Comandos relacionados à inicialização/interrupção

- `docker start ID_CONTAINER` - inicia o container com id em questão.
- `docker start -a -i ID_CONTAINER` - inicia o container com id em questão e integra os terminais, além de permitir interação entre ambos.
- `docker stop ID_CONTAINER` - interrompe o container com id em questão.

Comandos relacionados à remoção

- `docker rm ID_CONTAINER` - remove o container com id em questão.
- `docker container prune` - remove todos os containers que estão parados.
- `docker rmi NOME_DA_IMAGEM` - remove a imagem passada como parâmetro.



Comandos para o DOCKER

Comandos relacionados à construção de Dockerfile

- `docker build -f Dockerfile` - cria uma imagem a partir de um Dockerfile.
- `docker build -f Dockerfile -t NOME_USUARIO/NOME_IMAGEM` - constrói e nomeia uma imagem não-oficial.
- `docker build -f Dockerfile -t NOME_USUARIO/NOME_IMAGEM CAMINHO_DOCKERFILE` - constrói e nomeia uma imagem não-oficial informando o caminho para o Dockerfile.

Comandos relacionados ao Docker Hub

- `docker login` - inicia o processo de login no Docker Hub.
- `docker push NOME_USUARIO/NOME_IMAGEM` - envia a imagem criada para o Docker Hub.
- `docker pull NOME_USUARIO/NOME_IMAGEM` - baixa a imagem desejada do Docker Hub.

Comandos relacionados à rede

- `hostname -i` - mostra o ip atribuído ao container pelo docker (funciona apenas dentro do container).
- `docker network create --driver bridge NOME_DA_REDE` - cria uma rede especificando o driver desejado

Segue também uma breve lista dos novos comandos utilizados:

- `docker-compose up` - sobe os serviços criados
- `docker-compose down` - para os serviços criados.
- `docker-compose ps` - lista os serviços que estão rodando.



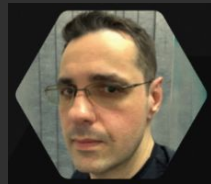
Referências

- Docker: <https://docs.docker.com/>
- 4Linux: <https://blog.4linux.com.br/>
- Alura: <https://www.alura.com.br/>
- Microsoft Docs: <https://docs.microsoft.com/en-us/windows/wsl/install>
- Renato Campos: github.com/renatofcampos



IMPORTANTE: Este processo não é homologado para ambientes de produção! Serve para auxiliar os times de desenvolvimento e suporte na montagem e testes em ambientes próximos aos dos clientes reais.

Obrigado!



Renato Ferreira Campos

renato.campos@totvs.com.br



LinkedIn

<https://bityli.com/fZWDwXu>



Github

<https://github.com/renatofcampos>