Macoratti.net Docker - Uma introdução básica - III

Neste artigo vou apresentar os conceitos básicos relativos ao Docker sob o ponto de vista de um desenvolvedor .NET.

Hoje vamos continuar criando contêineres a partir de imagens existentes no Docker Hub usando o modo interativo do Docker.

Lembrando que estou no ambiente Linux usando o Ubuntu 18.04.1.(artigo anterior)

Criando contêineres a partir de imagens existentes

Para criar um contêiner usamos o comando: **docker container run** *<nome_da_imagem>* ***<comando>***

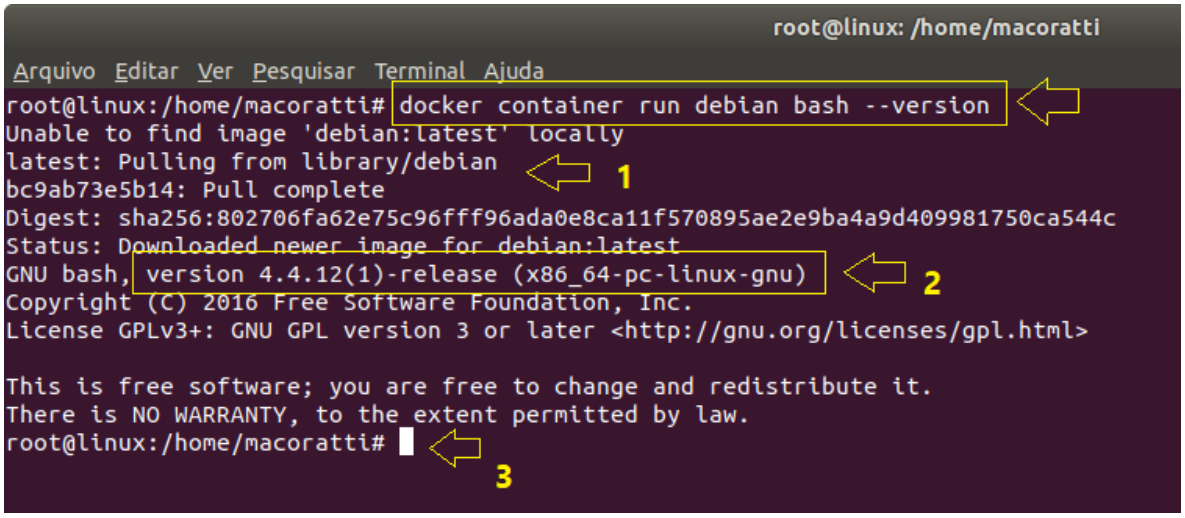
A novidade aqui é que podemos criar um contêiner a partir de uma imagem e executar um comando.

Vamos criar um contêiner digitando o comando abaixo em um terminal:

```
docker container run debian bash --version
```

Onde:

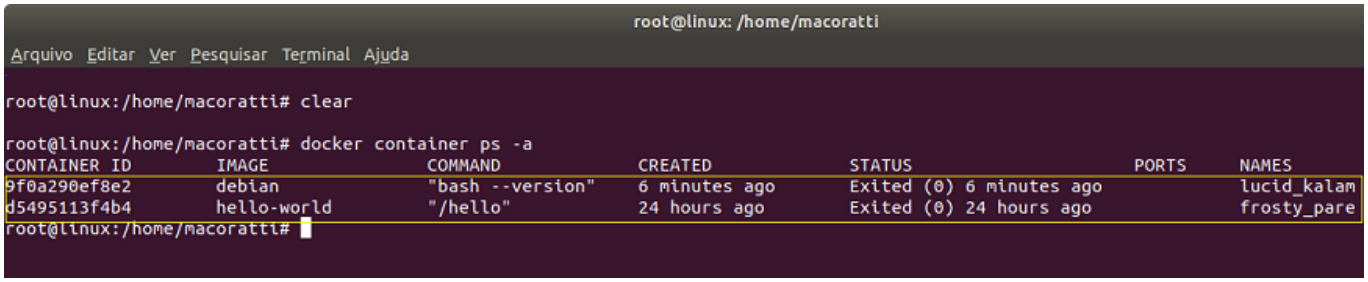
docker container run -> comando para criar um contêiner
debian -> nome da imagem *(será obtida do docker hub)*
bash --version -> comando a ser executado



Na execução do comando temos que :

1 - A imagem não foi localizada localmente e foi baixada do repositório;
2 - O comando bash --version foi executado;
3 - O execução do contêiner foi encerrada e retornamos ao prompt do linux;

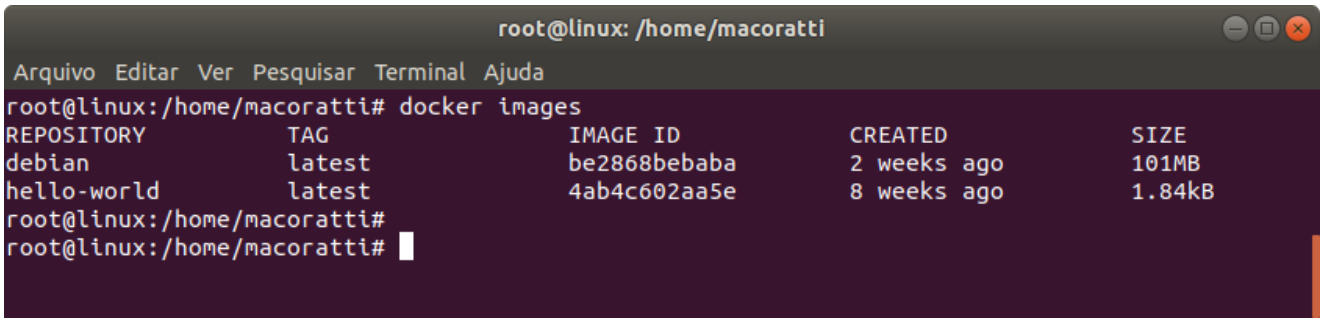
Para ver todos processos dos contêineres digite: docker container ps -a



Note que temos dois contêineres criados e com as seguintes informações:

CONTAINER_ID	O identificador do contêiner
IMAGE	o nome da imagem usada
COMMAND	o comando executado no momento
CREATED	a data de criação do contêiner
STATUS	o status do contêiner (tempo de execução)
PORTS	a porta do contêiner aberta
NAMES	o nome do contêiner atribuído pelo docker

Para visualizar as imagens digite o comando : docker images



Temos duas imagens com as seguintes informações:

REPOSITORY	o nome da imagem no repositório
TAG	a versão da imagem
IMAGE ID	o identificador da imagem
CREATED	a data de criação da imagem
SIZE	o tamanho da imagem

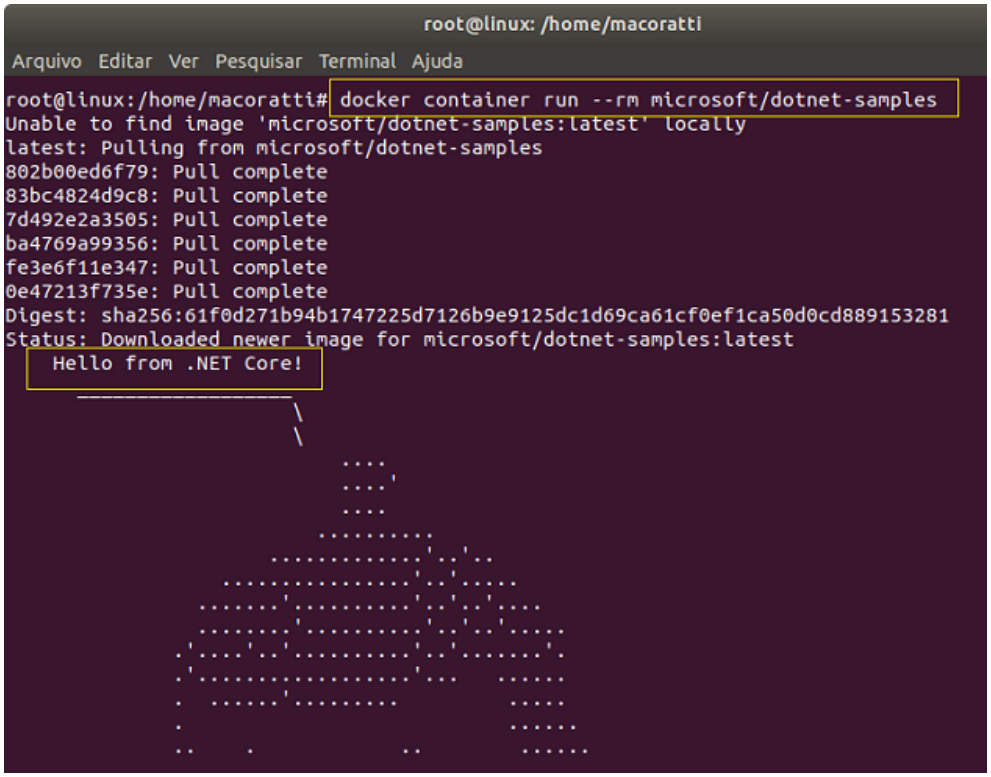
Se você executar novamente o comando, a execução vai ser mais rápida, pois a imagem debian agora existe localmente na sua máquina.

Agora vamos executar o comando: docker container run --rm debian bash --version

Neste comando temos o parâmetro --rm que vai remover o contêiner após sua execução. Assim após a execução o processo do contêiner na memória será removido.

Para mostrar isso vamos criar um contêiner e após sua execução removê-lo. Digite:

docker container run --rm microsoft/dotnet-samples



Nota: *Aqui estou usando uma imagem do .NET Core.*

Após a execução do comando se procurarmos pelo contêiner digitando: docker ps -a

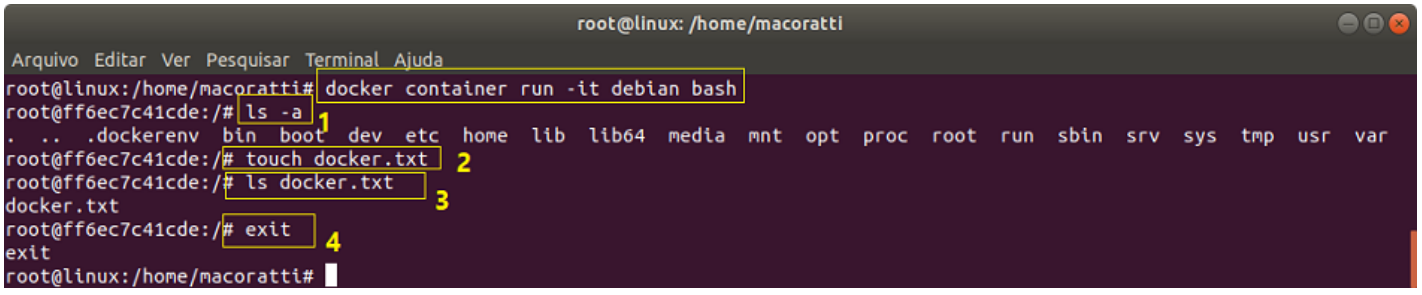
Não veremos o processo usado para esse contêiner pois ele foi removido.

Podemos criar um contêiner e ter acesso iterativo ao contêiner. Vamos ver isso digitando o comando:

docker container run -it debian bash

Neste comando estamos usando os parâmetros -it onde i significa o modo interativo e o t para que possamos ter acesso ao terminal bash:

- -i é o parâmetro para interagir(*vai manter o STDIN aberto*).
- -t é para alocar um TTY(*Talk to you*) que é um terminal.



Na execução entramos no terminal do contêiner e executamos os comandos:

1- ls -a -> exibe os arquivos
2- touch docker.txt -> cria o arquivo docker.txt
3- ls docker.txt -> exibe o arquivo
4- exit -> sai do contêiner

Nota: *Para sair de dentro do container, digite exit. Isso vai matar o container e ele não vai existir mais. O mesmo vai acontecer se você fizer um CTRL+C.*

Assim criamos um arquivo texto no contêiner criado.

Se você executar novamente o último comando e procurar pelo arquivo docker.txt não vai encontrar...

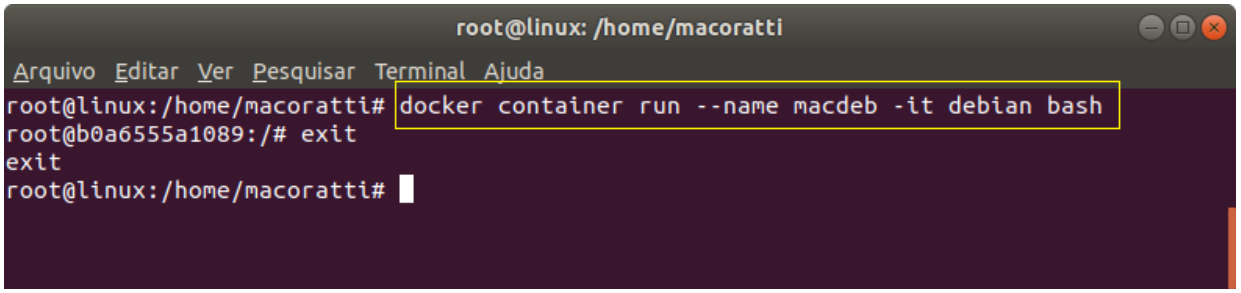
Por quê ???

Porque um comando docker container run sempre cria um novo contêiner.

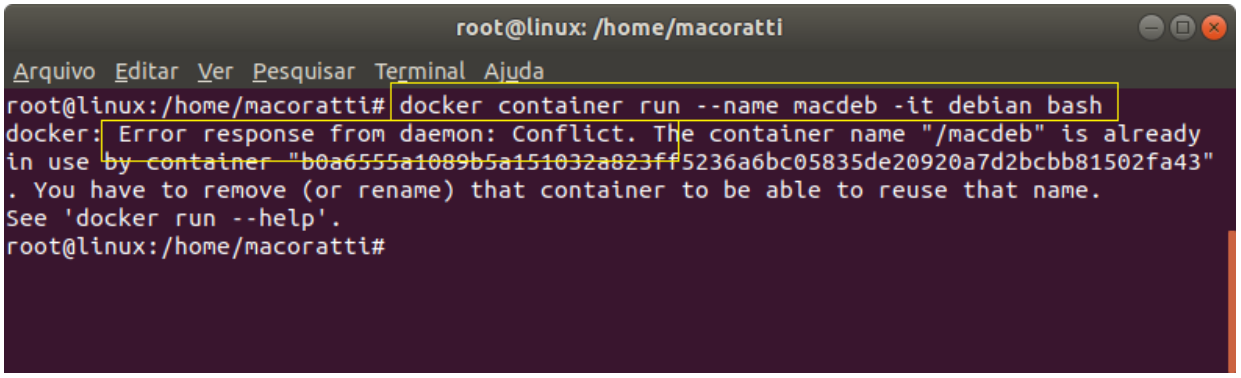
Vamos agora criar um contêiner e dar um nome a ele, pois como você já percebeu, o Docker dá nome aleatório aos contêineres, mas esses nomes não são fácies de lembrar.

Digite no terminal: docker container run --name macdeb -it debian bash

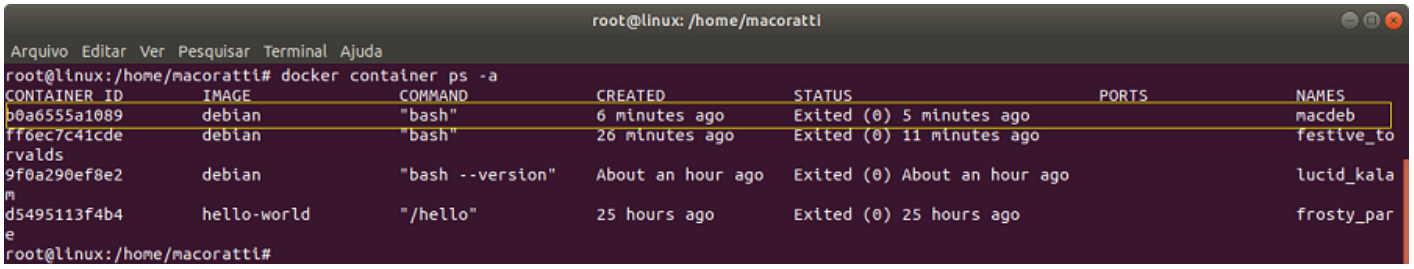
Aqui estamos usando o parâmetro --name e dando o nome de macdeb para o contêiner:



Se tentarmos executar novamente o mesmo comando teremos um erro pois já existe um contêiner com o nome macdeb:



Visualizando todos os contêineres: docker container ps -a

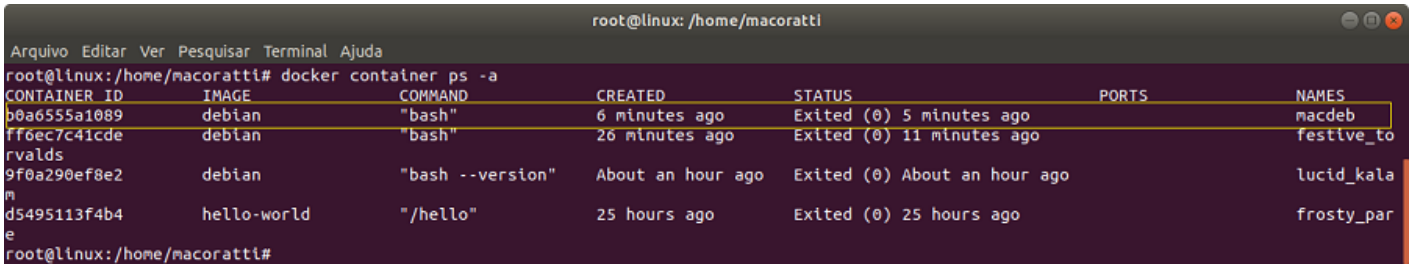


Vemos o nosso contêiner criado com o nome **macdeb** atribuído.

Sabendo o nome do contêiner podemos reusar o contêiner.

Vamos primeiro exibir os contêiners digitando: docker container ls -a

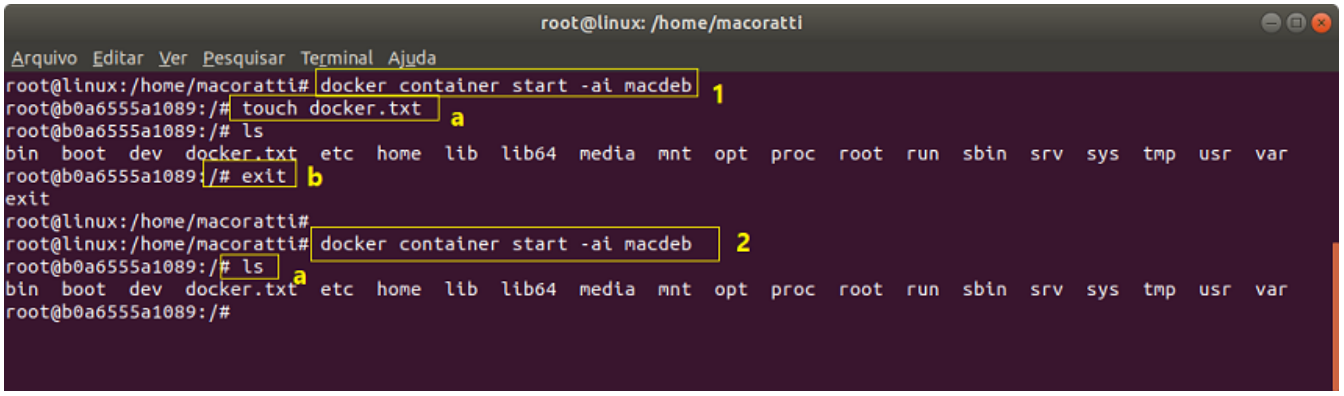
Este comando também lista todos os contêiners:



Para reutilizar o contêiner macdeb digite:

docker container start -ai macdeb

Neste comando estamos usando o método start para iniciar um contêiner e os parâmetros -ai onde a vai anexar*(atachar)* o terminal e i indica o modo interativo:



Na execução fizemos o seguinte:

1- Reutilizamos o contêiner **macdeb**
a - no contêiner criamos o arquivo texto **docker.txt** e listamos os arquivos
b- saímos do contêiner(**exit**)

2- Reutilizamos novamente o contêiner **macdeb**
a- Digitamos **ls** e vemos que o arquivo **docker.txt** criado no contêiner esta presente.

Com isso podemos reutilizar contêineres já existentes. Assim é importante dar um nome ao contêiner para poder reutilizá-lo.

Para concluir vou deixar uma lista dos principais comandos relacionados aos contêineres:

Comando	Descrição
docker container run	Cria e inicia um contêiner
docker container ps	Exibe os processos dos contêineres em execução. Para lista todos inclua -a.
docker container rm	Remove um contêiner. Apagar todos: **docker container rm $(docker ps -a -q)**
docker container create	Cria um novo contêiner com base em uma imagem
docker container stop	Para a execução de um contêiner
docker container logs	Exibe a saída gerada pelo contêiner
docker container exec	Executa um comando em um contêiner ou inicia uma sessão interativa
docker container rename	Renomeia um contêiner existe
docker container start	Inicia a execução de um contêiner existente

Nota: *Você pode até omitir a palavra* ***container*** *dos comandos mas a sintaxe atual recomanda usá-la.*

Os parâmetros mais utilizados na execução de um container são:

Parâmetro	Descrição
-d	Execução do container em background(segundo plano)
-i	Modo interativo. Mantém o STDIN aberto mesmo sem console anexado
-t	Aloca uma pseudo TTY
--rm	Automaticamente remove o container após finalização. *(Não funciona com -d)*
--name	Da um nome ao container
-v	Realiza o mapeamento de um volume
-p	Faz o mapeamento de porta
-m	Limta o uso de memória RAM
-c	Realiza o balanceamento do uso de CPU

Na próxima aula veremos como mapear portas, diretórios e volumes em contêineres.