01

Comandos básicos com containers

Transcrição

Nesta aula, vamos entender melhor o processo de criação de *containers* e execução de imagens.

Primeiramente, devemos verificar se o Docker está rodando. Se sim, no Linux ou no macOs, vamos executar os comandos do Docker através do próprio **terminal nativo do sistema operacional**. No caso do Windows, o próprio Docker recomenda que os seus comandos sejam executados através do **Windows PowerShell**.

> *Lembrando que se o seu Docker foi instalado pelo **Docker Toolbox**, você deve executar os seus comandos através do **Docker Quickstart Terminal**, terminal que foi instalado pelo próprio **Docker Toolbox**.*

A diferença entre imagens e containers

Na aula anterior, para executar a imagem **hello-world**, executamos o comando `docker run hello-world`. Quando executado esse comando, a primeira coisa que o Docker faz é verificar se temos a imagem **hello-world** no nosso computador, e caso não tenhamos, o Docker buscará e baixará essa imagem no **Docker Hub** (https://hub.docker.com/)/**Docker Store** (https://store.docker.com/).

A imagem é como se fosse uma receita de bolo, uma série de instruções que o Docker seguirá para criar um *container*, que irá conter as instruções da imagem, do *hello-world*. Criado o *container*, o Docker executa-o. Então, tudo isso é feito quando executamos o `docker run hello-world`.

Há milhares de imagens na **Docker Store** disponíveis para executarmos a nossa aplicação. Por exemplo, temos a imagem do Ubuntu:

```
docker run ubuntu
```

Ao executar o comando, o download começará. Podemos ver que não é feito somente um download, pois a imagem é dividida em **camadas**, que veremos mais à frente. Terminados os downloads, nenhuma mensagem é exibida, então significa que o *container* não foi criado? Na verdade o *container* foi criado, o que acontece que é a imagem do Ubuntu não executa nada, por isso nenhuma mensagem foi exibida.

Podemos verificar isso vendo os *containers* que estão sendo executados no momento, executando o seguinte comando:

```
docker ps
```

Ao executar esse comando, vemos que não há nenhum *container* ativo, pois quando não há nada para o *container* executar, eles ficam **parados**. Para ver **todos** *containers*, inclusive os parados, adicionamos a *flag* `-a` ao comando acima:

```
docker ps -a
```

```
CONTAINER ID    IMAGE          COMMAND       CREATED          STATUS                      PORTS     NAM
4139842e283a    ubuntu         "/bin/bash"   3 minutes ago    Exited (0) 3 minutes ago              e
```

```
c1a155091114    hello-world    "/hello"    4 days ago    Exited (0) 4 days ago    ni
```

Com esse comando, conseguimos ver o **id** e o **nome** do *container*, valores que são criados pelo próprio Docker. Além disso, temos a outras informações dos *containers*, a **imagem** em que eles são baseados, o **comando inicial** que roda quando ele é executado, quando ele foi criado e qual o seu status.

Então, o *container* do Ubuntu foi executado, mas ele não fez nada pois não pedimos para o *container* executar algo que funcione dentro do Ubuntu. Então, quando executamos o *container* do Ubuntu, precisamos passar para ele um comando que rode dentro dele, por exemplo:

```
docker run ubuntu echo "Ola Mundo"
```

Com isso, o Docker irá executar um *container* com Ubuntu, executar o comando `echo "Ola Mundo"` dentro dele e nos retornar a saída:

```
alura@alura-estudio-03:~$ sudo docker run ubuntu echo "Ola Mundo"
Ola Mundo
```

Só que sabemos que o Ubuntu é um sistema operacional completo, então não queremos ficar somente executando um comando por comando dentro dele, sempre criando um novo *container*. Então, como fazemos para criar um *container* e **interagir** com ele mais do que com um único comando?

Trabalhando dentro de um container

Podemos fazer com que o terminal da nossa máquina seja integrado ao terminal de dentro do *container*, para ficar um terminal interativo. Podemos fazer isso adicionando a *flag* `-it` ao comando, atrelando assim o terminal que estamos utilizando ao terminal do *container*:

```
docker run -it ubuntu
```

Assim que executamos o comando, já podemos perceber que o terminal muda:

```
alura@alura-estudio-03:~$ sudo docker run -it ubuntu
root@05025384675e:/#
```

Com isso, estamos trabalhando **dentro do *container***. E dentro dele, podemos trabalhar como se estivéssemos trabalhando dentro do terminal de um Ubuntu, executando comandos como `ls` , `cat` , etc.

Agora, se abrirmos outro terminal e executar o comando `docker ps` , veremos o *container* que estamos executando. Podemos parar a sua execução, digitando no *container* o comando `exit` ou através do atalho `CTRL + D` .

Executando novamente um container

Paramos a execução do *container*, tanto que o comando `docker ps` não nos retorna mais nada. E se listarmos todos os *containers*, através do comando `docker ps -a` , vemos que ele está lá, parado. Mas agora, para não criar novamente um novo *container*, queremos executá-lo novamente.

Fazemos isso pegando **id** do *container* a ser iniciado, e passando-o ao comando `docker start`

```
docker start 05025384675e
```

Esse comando roda um *container* já criado, mas não atrela o nosso terminal ao terminal dele. Para atrelar os terminais, primeiramente devemos parar o *container*, com o comando `docker stop` mais o seu **id**:

```
docker stop 05025384675e
```

E rodamos novamente o *container*, mas passando duas *flags*: `-a` , de ***attach***, para integrar os terminais, e `-i` , de ***interactive***, para interagirmos com o terminal, para podermos escrever nele:

```
alura@alura-estudio-03:~$ sudo docker start -a -i 05025384675e
root@05025384675e:/#
```

Com isso, conseguimos ver um pouco de como subir um *container*, pará-lo e executá-lo novamente, além de trabalhar dentro dele.