



VER PLANOS

PROGRAMAÇÃO _

DATA SCIENCE _

DEVOPS _

MOBILE _

FRONT-END _

INTELIGÊNCIA ARTIFICIAL _

UX & DESIGN _

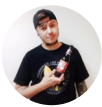
INOVAÇÃO & GESTÃO _

Artigos > **DevOps**

Oi! Posso indicar os melhores artigos para tirar suas dúvidas!

Começando com Docker

```
er pull ubuntu
ault tag: latest
ulling from library/ubuntu
43b: Pull complete
b04: Pull complete
c94: Pull complete
d30: Pull complete
6aa: Pull complete
ha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1a
ownloaded newer image for ubuntu:latest
```



Fernando Furtado

20/06/2017

COMPARTILHE



Esse artigo faz parte da
Formação DevOps

Quando falamos em desenvolvimento de software é comum ter diversos ambientes, por exemplo:

- *Desenvolvimento*
- *Teste*
- *Homologação*
- *Produção*

E outra coisa comum no mundo de desenvolvimento é ter divergências entre estes ambientes.

Confira neste artigo:

- [O que é o Docker?](#)
- [Compreendendo o conceito de containers](#)
- [Ganhos ao usar containers do Docker](#)
- [Como o Docker faz isso?](#)
- [Mão na massa](#)

Quem nunca ouviu a frase:

//

"Na minha máquina funciona!"?

Nesse post vamos abordar o **Docker** como uma alternativa para minimizar essa divergência.

O que é o Docker?

O **Docker é um sistema de virtualização não convencional**. Mas o que isso quer dizer? Em virtualizações convencionais temos um software instalado na máquina *Host* que irá gerenciar as máquinas virtuais (ex.: VirtualBox, VMWare, Parallels e etc...).

Para cada máquina virtual temos uma instalação completa do S.O. que queremos virtualizar, além de ter o próprio hardware virtualizado.

Se por exemplo eu precisar de uma biblioteca comum para todas as máquinas virtuais, preciso instalar em cada uma delas.

O Docker usa uma abordagem diferente, ele utiliza o conceito de container. Como assim **container**?



Matricule-se na escola de DEV OPS

Junte-se a uma comunidade de **+500 mil** estudantes

- Acesso a **TODOS** os cursos em uma única assinatura
- Novos lançamentos a cada semana
- Desafios práticos

SAIBA MAIS

a

Compreendendo o conceito de containers

Se pensarmos em transporte de cargas, container foi uma revolução nessa área. Pois antes deles o tempo de carregar e descarregar um navio era gigantesco e o trabalho era feito manualmente. Sem contar perdas (devido a quebras ou deterioração), desvio e outros problemas.

Com a chegada dos containers foi possível transportar mercadorias de uma forma segura, de fácil manipulação e com pouco, ou nenhum, trabalho braçal no carregamento ou descarregamento. E é justamente isso que o Docker tenta fazer com nossos softwares.

Ganhos ao usar containers do Docker

Imagine nosso software como uma mercadoria a ser transportar como por exemplo, do ambiente de *Desenvolvimento* para *Produção*.

Para fazer isso precisamos garantir que nosso ambiente de *Produção* tenha todos os pré-requisitos instalados, de preferência uma versão do S.O. parecida com a do ambiente de *Desenvolvimento* entre outros cuidados que devem ser tomados (relacionados a permissionamento, serviços dependentes e etc...).

Com o Docker temos um container com nosso software. Esse container é levado inteiro para o outro ambiente.

Com isso não precisamos nos preocupar com pré-requisitos instalados no outro ambiente, versão do S.O., permissionamento e se quisermos podemos ter containers para os serviços dependentes também. Dessa forma minimizamos muito a divergência entre os ambientes.

Como o Docker faz isso?

Essa ideia de container já é bem antiga e a princípio o Docker usava internamente um projeto chamado [LXC](#) (Linux Container).

O projeto LXC usa por baixo dos panos diversas funcionalidades presentes no Kernel do Linux. Abaixo vou listar algumas dessas funcionalidades:

- **chroot** - Responsável por mapear os diretórios do S.O. e criar o ponto de montagem (`/`, `/etc`, `/dev`, `/proc` entre outros).
- **cgroup** - Responsável por controlar os recursos por processo. Com ele podemos por exemplo limitar o uso de memória e/ou processador para um processo específico.
- **kernel namespace** - Com ele podemos isolar processos, ponto de montagem entre outras coisas. Com esse isolamento, conseguimos a sensação de estar usando uma máquina diferente da máquina host. Pois enxergamos somente o ponto de montagem específico e processos específicos, inclusive nossos processos começam com PID baixo.
- **kernel capabilities** - Entre outras coisas, conseguimos rodar alguns comandos de forma privilegiada.

Mão na massa

Agora que temos uma noção do que é e para que serve o Docker, vamos fazer o download da ferramenta e começar nesse mundo de containers.

Instalando o Docker

Atualmente Docker está disponível em duas versões [Docker Community Edition\(CE\)](#) e [Docker Enterprise Edition\(EE\)](#).

Em ambas as versões temos acesso a toda a API, basicamente a diferença entre as duas versões é o perfil desejado de aplicações. No EE temos um ambiente homologado pela Docker com toda infraestrutura certificada, segura pensada para o mundo enterprise. Já na versão CE podemos chegar ao mesmo nível que EE porém de uma forma manual.

Nesse [link](#) você pode encontrar as distribuições para downloads em cada sistema operacional disponível e os passos para instalação.

Feito a instalação, execute esse comando no terminal `docker --version`. Se a instalação ocorreu com sucesso deve ser impresso algo semelhante a isso

```
Docker version 17.03.1-ce, build c6d412e .
```

Imagens

O Docker trabalha com o conceito de imagens, ou seja, para colocar um container em funcionamento o Docker precisa ter a imagem no host.

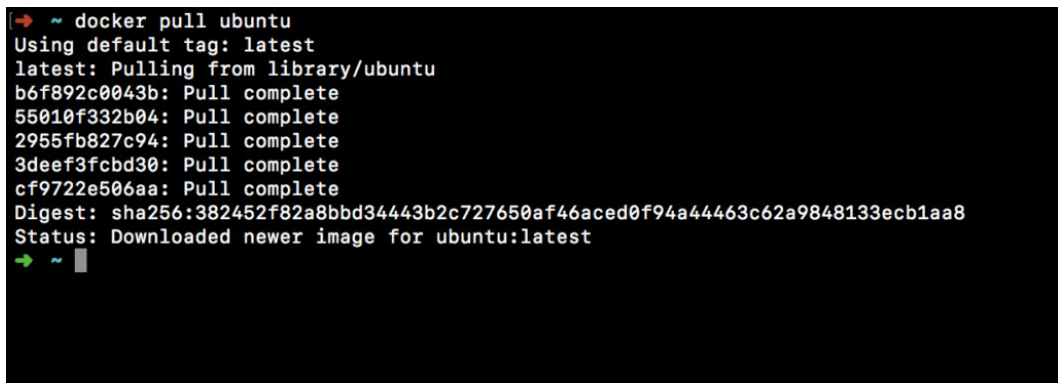
Essas imagens podem ser baixadas de um repositório (a nomenclatura para esse repositório é *registry*) ou criadas localmente e compiladas. Esse é o [link](#) para o registry do Docker.

Nesse registry podemos ter imagens oficiais e não oficiais. Além de podermos criar nossas próprias imagens, também é possível fazer upload dela em um registry.

Baixando Imagens

Para baixar uma imagem podemos usar o comando `docker pull` e o nome da imagem que queremos baixar. Vamos baixar a imagem do *Ubuntu*, para isso execute o seguinte comando no terminal: `docker pull ubuntu`.

[caption id="attachment_7091" align="aligncenter" width="750"]



```
➔ ~ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
b6f892c0043b: Pull complete
55010f332b04: Pull complete
2955fb827c94: Pull complete
3deef3fcbd30: Pull complete
cf9722e506aa: Pull complete
Digest: sha256:382452f82a8bbd34443b2c727650af46aced0f94a44463c62a9848133ecb1aa8
Status: Downloaded newer image for ubuntu:latest
➔ ~
```

Resultado `docker pull ubuntu`[/caption]

Aqui o Docker baixou nossa imagem. Percebam que uma imagem é composta de várias camadas, por esse motivo teve que fazer vários Downloads/Pulls.

Listando imagens baixadas

Para listar todas as imagens podemos usar o comando `docker images`. O retorno desse comando é algo semelhante a isso:

```
➜ ~ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    ebed9d4fca88   46 hours ago   118 MB
```

O nome da imagem é exibido na coluna `REPOSITORY`, cada imagem tem um identificador único que é exibido na coluna `IMAGE ID`. A coluna `TAG` indica a "versão" da imagem do `ubuntu`. O `latest` quer dizer que é a última "versão" da imagem (a mais recente).

Executando Containers

A partir da imagem podemos iniciar quantos containers quisermos através do comando `docker run`.

Para acessarmos um terminal do *Ubuntu* podemos usar o comando `docker run -i -t ubuntu` OU `docker run -it ubuntu`. O parâmetro `-i` indica que queremos um container interativo, o `-t` indica que queremos anexar o terminal virtual `tty` do container ao nosso host.

Listando containers em execução

Para ver os containers em execução podemos usar o comando `docker ps` (em outro terminal ou aba), e ele exibirá um retorno parecido com esse:

```
➜ ~ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS       NAMES
a6697ed945d5   ubuntu   "/bin/bash"             12 hours ago   Up 4 seconds   -           dreamy_bassi
```

Aqui temos informações sobre os containers em execução, como id, imagem base, comando inicial, há quanto tempo foi criado, status, quais portas estão disponíveis e/ou mapeadas para acesso e o nome do mesmo. Quando não especificamos um nome ao iniciá-lo, será gerado um nome aleatoriamente.

Quando encerramos um container ele não será mais exibido na saída do comando `docker ps`, porém isso não significa que o container não existe mais. Para verificar os containers existentes que foram encerrados podemos usar o comando `docker ps -a` e teremos uma saída parecida com essa:

```
~ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a6697ed945d5	ubuntu	"/bin/bash"	12 hours ago	Exited (0) 4 seconds ago		dreamy_bassi

Como o próprio `status` do container informa, o mesmo já saiu de execução e no nosso caso saiu com status `0` (ou seja saiu normalmente).

Removendo containers

Para remover o container podemos usar o comando `docker rm` e informar o id do container ou o nome dele. Para nosso caso poderíamos executar o comando `docker rm 43aac92b4c99` OU `docker rm dreamy_bassi` para remover o container por completo.

Caso tenhamos a necessidade de remover todos os container (em execução ou encerrados) podemos usar o comando `docker rm $(docker ps -qa)`. A opção `-q` do comando `docker ps` tem como saída somente os ids dos containers, essa lista de ids é passado para o `docker rm` e com isso será removido todos os containers.

Só será possível remover um container caso o mesmo não esteja em execução, do contrário temos que encerrar o container para removê-lo.

Como são feitas as imagens?

Nesse momento podemos pensar que o Docker é meio mágico (e é...kkk). Dado uma imagem ele pode rodar um ou mais containers com pouco esforço, mas como são feitas as imagens?

Uma imagem pode ser criada a partir de um arquivo de definição chamado de `Dockerfile`, nesse arquivo usamos algumas diretivas para declarar o que teremos na nossa imagem. Por exemplo se olharmos a definição da imagem do *Ubuntu* podemos ver algo semelhante a isso:


```
FROM scratch ADD ubuntu-xenial-core-cloudimg-amd64-root.tar.gz / .  
CMD \["bin/bash"]
```

Para ver o Dockerfile completo consulte [aqui](#).

Com um arquivo `Dockerfile` podemos compilá-lo com o comando `docker build .`. Ao compilar um arquivo `Dockerfile` temos uma imagem. Mas isso é um assunto para um próximo post.


◀  ▶
Não deixe de conferir nosso curso de [docker](#) na **Alura**.

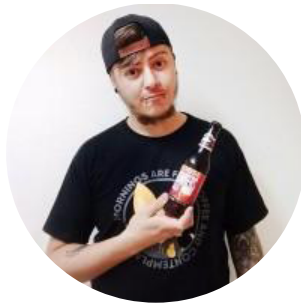


Matricule-se 
na escola de
DEV OPS

Junte-se a uma comunidade de
+ 500 mil pessoas

- Acesse **TODOS** os cursos em uma única assinatura
- Novos lançamentos a cada semana
- Desafios práticos


SAIBA MAIS 



Fernando Furtado

Fernando é desenvolvedor com foco em Java, contudo, já fez projetos para mobile (iOS), onde atuou com as linguagens Objective-C e Swift. No seu trabalho, sempre consulta boas práticas como testes ou design de código (Design Pattern e SOLID, entre outros).

[Artigo Anterior](#)

[Linux: compactando e descompactando arquivos com o tar](#)

[Próximo Artigo](#)

[SSH: o acesso remoto aos servidores](#)



Veja outros artigos sobre
[DevOps](#)

Quer mergulhar em tecnologia e aprendizagem?

Receba conteúdos, dicas, notícias, inovações e tendências sobre o mercado tech diretamente na sua caixa de entrada.

bins.br@gmail.com

ENVIAR

Nossas redes e apps



Institucional

Sobre nós

Carreiras Alura

Para Empresas

Para Sua Escola

Política de Privacidade

Compromisso de Integridade

Termos de Uso

A Alura

Como Funciona

Formações

Plataforma

Depoimentos

Instrutores(as)

Dev em <T>

Luri, a inteligência artificial da Alura

Documentos Institucionais

IA Conference 2025

Status

Cursos imersivos

Certificações

Uma empresa do grupo Alun

Conteúdos

Fale Conosco

Alura Cases

Email e telefone

Imersões

Perguntas frequentes

Artigos

Podcasts

Artigos de educação
corporativa

Imersão Cloud Devops

Novidades e Lançamentos

ENVIAR

CURSOS

Cursos de Programação

Lógica | Python | PHP | Java | .NET | Node JS | C | Computação | Jogos | IoT

Cursos de Front-end

HTML, CSS | React | Angular | JavaScript | jQuery

Cursos de Data Science

Ciência de dados | BI | SQL e Banco de Dados | Excel | Machine Learning | NoSQL | Estatística

Cursos de Inteligência Artificial

IA para Programação | IA para Dados

Cursos de DevOps

AWS | Azure | Docker | Segurança | IaC | Linux

Cursos de UX & Design

Usabilidade e UX | Vídeo e Motion | 3D

Cursos de Mobile

Flutter | iOS e Swift | Android, Kotlin | Jogos

Cursos de Inovação & Gestão

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas

CURSOS UNIVERSITÁRIOS FIAP

Graduação | Pós-graduação | MBA