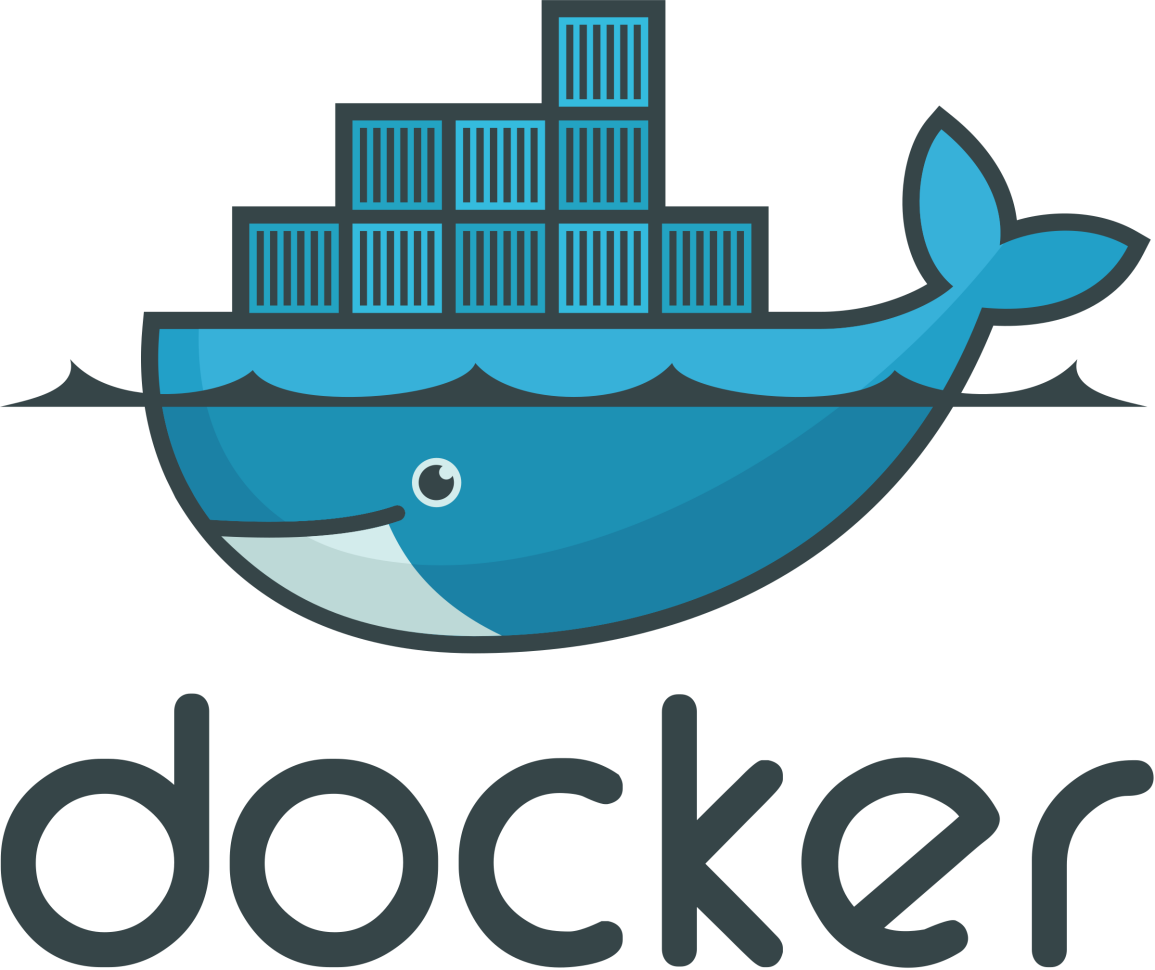
**INTRODUÇÃO AO DOCKER**



**Autor: Vamberto Rocha Jr**

**Contato: vambertojr@gmail.com**

# INTRODUÇÃO

O Docker é uma plataforma open source, mantido pela Docker Inc, que facilita a criação e administração de contêineres. Podemos definir um contêiner como sendo um ambiente isolado que contém um conjunto de processos que são executados a partir de uma imagem. Eles compartilham o mesmo *kernel* e isolam os processos da aplicação do restante do *host* onde ele está instalado. Apesar de não ser a única tecnologia de contêineres, o Docker é a mais utilizada.

Dentre as principais vantagens de se utilizar Docker, podemos citar:

* Padronização
* Portabilidade
* Escalabilidade
* Agilidade
* Economia de recursos

Este documento não foi concebido com o objetivo de detalhar o funcionamento do Docker, bem como todos os componentes do seu ambiente, e sim servir como material de consulta.

## Instalando o Docker

O processo de instalação do Docker é simples, neste documento é apresentado de duas formas, a saber: Instalação a partir do repositório oficial e executando *script* de instalação automática.

Os passos a seguir foram realizados para instar o Docker no Sistema Operacional Linux Ubuntu 18.04 LTS. Os mesmos procedimentos podem ser realizados caso a instalação seja em sistemas Debian. Para instalação em sistemas baseados no Linux Red Hat, a documentação oficial deverá ser consultada.

### Instalação a partir do Repositório Oficial

1. Removendo versões antigas

$ sudo apt-get remove docker docker-engine docker.io containerd runc

1. Atualizando o repositório de pacotes

$ sudo apt-get update

1. Instalando pacotes que permitem o apt baixar pacote sobre HTTPS

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

1. Adicionando a chave GPG oficial do Docker

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

1. Adicionando o repositório oficial estável do Docker

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

1. Atualizando o repositório de pacotes

$ sudo apt-get update

1. Instalando a ultima versão do Docker Community Engine

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

### *Script* de Instalação Automática

Para instalar o Docker utilizando o *script* de instalação automática, basta executar, como usuário root, o comando abaixo:

wget -qO- https://get.docker.com/ | sh

## Instalando o Docker Compose

curl -L https://github.com/docker/compose/releases/download/1.24.1/docker-compose--$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

### Gerenciar Docker sem fazer uso do comando sudo

1. **Crie o grupo Docker (caso não exista)**

$sudo groupadd docker

1. **Adicione seu usuário no grupo Docker**

$sudo usermod -aG docker $USER

WARNING: Error loading config file: /home/user/.docker/config.json -

stat /home/user/.docker/config.json: permission denied

Caso apresente o erro:

1. **Execute:**

$sudo chown "$USER":"$USER" /home/"$USER"/.docker -R

$sudo chmod g+rwx "/home/$USER/.docker" -R

# Docker Images

Imagens Docker são utilizadas como base para criação dos contêineres. Elas são criadas a partir de uma sequencia de comandos escritos em um arquivo chamado de Dockerfile e compostas por sistemas de arquivos de camadas. Imagens oficiais ou criadas por outros usuários podem ser baixadas no site Docker Hub ([https://hub.docker.com](https://hub.docker.com/)).

Abaixo seguem os principais comandos para manipular imagens no Docker:

## Principais comandos

1. **Listar imagens**

docker image ls

1. **Baixar imagem do Docker Registry**

docker image pull <imagem>

1. **Exibir o histórico da imagem**

docker image history <imagem>

1. **Exibir informações detalhadas de uma ou mais imagens**

docker image inspect <imagem>

1. **Realizar o backup de uma imagem**

docker image save <imagem> -o <arquivo.tar.gz>

1. **Restaurar uma imagem**

docker image load -i [arquivo.tar.gz]

1. **Criar/Alterar tag da imagem**

docker image tag <imagem>

1. **Apagar imagem**

docker image rm <imagem>

1. **Apagar todas as imagens**

docker image rm $(docker image ls -aq)

1. **Apagar todas as imagens não utilizadas**

docker image prune

# Docker Contêineres

Podemos definir um contêiner como sendo uma instância de uma Imagem em execução. É um ambiente isolado que compartilha algumas partes do kernel onde o Docker está instalado. A partir de uma imagem podemos criar vários contêineres.

Abaixo seguem os principais comandos para manipular contêineres no Docker:

## Principais Comandos

1. **Listar os contêineres em execução e parados**

docker container ls -a

1. **Criar uma nova imagem a partir de um contêiner**

docker container commit <container> [repositório[:TAG]]

1. **Executar um comando em um contêiner em execução**

docker container exec [opções] <container> comando

1. **Acessar o bash de um contêiner**

docker container exec -it <container> /bin/bash

1. **Apresentar informações detalhadas de um ou mais contêineres**

docker container inspect <container>

1. **Exibir os logs do contêiner**

docker container logs <container>

1. **Exibir informações sobre os processor em execução de um contêiner**

docker container top <container>

1. **Exibir o consumo de recursos de um contêiner**

docker container stats <container>

1. **Apagar um contêiner**

docker container rm <container>

1. **Apagar todos os contêineres parados**

docker container prune

1. **Apagar todos os contêineres**

docker container rm $(docker container ls -aq)

1. **Principal comando para executar um contêiner**

docker container run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

O comando acima executa uma série de outros comandos, a saber:

1. Download automático das imagens não encontradas: docker image pull
2. Criação do contêiner: docker container create
3. Execução do contêiner: docker container start
4. Uso do modo interativo: docker container exec

As opções mais utilizadas com o comando docker container run são:

|  |  |
| --- | --- |
| **OOpção** | **Descrição** |
| --d | Execução do container em background |
| --i | Modo interativo. |
| --t | Aloca uma pseudo TTY |
| --e | Especificar variável (-e MYVAR1=foo) |
| --rm | Automaticamente remove um container após finalização |
| --name | Nomear um container |
| --restart | Altera política de reinicialização do container |
| --link | Serve para linkar um conteiner à outro <nome>:<alias> |
| --v | Mapeamento de volume <host>:<container> |
| --p | Mapeamento de porta <host>:<container> |
| --m | Limitar o uso de memória RAM |
| --c | Balancear o uso de CPU (proporcional) <1 a 1024> |

## Persistência dos dados

Por padrão, todos os arquivos criados dentro de um contêiner são armazenados em uma camada de contêiner gravável. Isso significa que: os dados não persistem quando esse contêiner não existir mais.

O Docker possui duas opções para tratar a persistência dos dados na máquina *host*, a saber: volumes e *bind mounts.* Os volumes são armazenados em uma parte dos sistema de arquivos da máquina *host* que é gerenciada pelo Docker, e estão localizados em: /var/lib/docker/volumes. Os bind mounts podem ser armazenados em qualquer lugar da máquina *host*, um arquivo ou diretório da máquina *host* é montado diretamente no contêiner.

### Criando e Gerenciando Volumes

1. Criando um volume

docker volume create <nome>

1. Listando volumes

docker volume ls

1. Removendo um volume

docker volume rm <nome>

1. Iniciando um container com um volume

docker container run --name teste **-v <nome>:/app** nginx:latest

### Iniciando um contêiner com bind mount

docker run --name teste -v <pasta/arquivo\_host>:<pasta/arquivo\_container> nginx:latest

## Redes Docker

Um dos motivos pelos quais contêineres Docker serem tão poderosos é que você pode conecta-los entre si ou a outros serviços que não são Docker. O subsistema de rede do Docker é conectável utilizando *drivers*. Abaixo são listados os *drivers* de rede do Docker bem como uma breve descrição deles:

*Drivers* de rede Docker:

* Bridge
  + Driver de rede padrão, é usado quando seus aplicativos são executados em contêineres independentes que precisam se comunicar.
* Host
  + Para standalone contêineres, apenas diponível para Docker Swarm
* Overlay
  + Conecta vários daemons do Docker juntos e permite que os serviços do Swarm se comuniquem
* Macvlan
  + Permite atribuir um endereço MAC a um contêiner, fazendo com que ele apareça como um dispositivo físico na sua rede
* None
  + Desativa todas as redes

### Criando e Gerenciando Redes Docker

1. Criando uma rede bridge (default)

docker network create <nome>

1. Criando uma rede com outro driver

docker network create -d <driver> <nome>

1. Listando as redes

docker network ls

1. Apagando uma ou mais redes

docker network rm <nome>

1. Conectar contêiner a uma rede

docker network connect <rede> <container>

1. Exibindo informações detalhadas de uma ou mais redes

docker network inspect <nome>

# Dockerfile

Como informado anteriormente, o Dockerhub é um repositório de imagens oficiais e criadas pelos usuários da comunidade, caso nenhuma das imagens contidas no Dockerhub sirva para seu propósito, ou se você quiser criar a sua própria imagem, deve-se fazer uso do Dockerfile

O Dockerfile é um arquivo texto com uma sintaxe simples, em YML ou YAML, que serve para criação de imagens Docker, ele contem todos os comandos que um usuário executa para montar uma imagem.

A seguir, seguem os principais comandos utilizados no Dockerfile para construir uma imagem Docker.

* **FROM**
  + Especifica a “Imagem Base” que será utilizada para construção da nova imagem.
* **RUN**
  + Irá executar qualquer comando em uma nova camada no topo da imagem corrente
* **EXPOSE**
  + Essa instrução informa ao Docker que o container irá escutar uma determinada porta em tempo de execução.
* **ENV**
  + Essa Instrução define uma variável de ambiente.
* **COPY**
  + Essa instrução copia arquivos e diretórios de um <src> localizado no hostpara um <dest> no filesystem da imagem.
* **ADD**
  + Essa instrução copia arquivos, diretórios ou URLs de um <src> localizado no hostpara um <dest> no filesystem da imagem. A vantagem do comando add sobre o copy é que , além de copiar URLs, se <src> for um arquivo compactado, em um formato de compactação reconhecido, ele será descompactado automaticamente como um diretório para a imagem do Docker.
* **LABEL**
  + Essa instrução adiciona metadados a uma imagem e informações adicionais que servirão para identificar versão, tipo de licença, ou *host*. A cada nova instrução LABEL é criada uma nova layer.
* **VOLUME**
  + Essa instrução mapeia um diretório do *host* para ser acessível pelo container.
* **WORKDIR**
  + Essa instrução define o diretório de trabalho para qualquer instrução RUN, CMD, ENTRYPOINT, COPY e ADD que se segue no Dockerfile.
* **ONBUILD**
  + Define algumas instruções que podem ser realizadas quando alguma determinada ação for executada, é basicamente como uma *trigger*.
* **USER**
  + Define com qual usuário serão executadas as instruções durante a geração da imagem;
* **CMD**
  + Define um comando a ser executado quando um contêiner baseado nessa imagem for iniciado, esse parâmetro pode ser sobrescrito caso o contêiner seja iniciado utilizando alguma informação de comando, como: docker contêiner run -d imagem comando, neste caso o CMD da imagem será sobrescrito pelo comando informado. Se o CMD for usado para fornecer argumentos padrão para a instrução ENTRYPOINT, as instruções CMD e ENTRYPOINT deverão ser especificadas com o formato de matriz JSON.
* **ENTRYPOINT**
  + Informa qual comando será executado quando um contêiner for iniciado utilizando esta imagem, diferentemente do CMD, o ENTRYPOINT não é sobrescrito, isso quer dizer que este comando será sempre executado. Exemplo:

FROM ubuntu

ENTRYPOINT [“top”, “-b”]

CMD [“-c”]

Após a construção do arquivo Dockerfile, é utilizado o comando docker image build para contruir a nova imagem.

1. **Gerar imagem a partir de um Dockerfile**

docker image build -t [tag] .

obs: Deverá existir um arquivo Dockerfile na mesma pasta da execução do comando

1. **Gerar uma imagem especificando o arquivo Dockerfile**

docker image build -t [tag] -f [caminho/Dockerfile]

**FROM** nginx

**RUN** cd / && mkdir Arquivos && chmod 777 -R Arquivos/

**COPY** ./site/index.html /usr/share/nginx/html/

**VOLUME** /Arquivos/

**EXPOSE** 80

**ENV** API\_URL=http://localhost:8000/api/

**ENV** API\_BANCO=meu\_site

**WORKDIR** /usr/share/nginx/html/

**ENTRYPOINT** ["/usr/sbin/nginx"]

**CMD** ["-g", "daemon off;"]

Para maiores informações, consultar o guia oficial de referência do Dockerfile, disponível em:

<https://docs.docker.com/engine/reference/builder/>

# Docker Compose

O Docker-Compose é uma ferramenta para orquestração, ou seja, definir e executar aplicativos Docker de vários contêineres. É utilizado um arquivo YAML para configurar os serviços do aplicativo, em seguida, com um único comando, será criado e iniciado todos os serviços definidos no arquivo YAML.

|  |  |
| --- | --- |
| version | Especificar a versão da sintaxe do arquivo docker-compose.yml |
| services | Contém a configuração aplicada a cada contêiner iniciado para esse serviço |
| image | Especificar a imagem que o container utilizará |
| build | Especificar opções que serão aplicadas no build dos conteineres |
| container\_name | Especificar um nome para o container criado |
| labels | Adicionar metadata a imagem resultante |
| environment | Adicionar variáveis de ambiente |
| ports | Especificar portas que serão mapeadas para máquina host(host:container) |
| links | Linkar contêineres |
| volumes | Montar pastas ou volumes do hostpara o container |
| expose | Expor portas sem publica-las na máquina host |
| restart | Definir a política de restart dos containeres (no (default), always, on-failure, unless-stopped) |
| networks | Criar e ingressar o container a uma rede |

Abaixo seguem os principais comandos utilizados no docker-compose, é importante salientar que **os referidos comandos devem ser executados no mesmo diretório em que se encontrar o arquivo docker-compose.yml**

1. Validar e visualizar o arquivo compose

docker-compose config

1. Iniciar os contêineres em *background* dos serviços especificados no arquivo docker-compose

docker-compose up -d

1. Recriar e iniciar os contêineres, em background, dos serviços especificados no arquivo docker-compose (utilizado caso tenha alguma alteração no arquivo docker-compose).

docker-compose up -d --force-recreate

1. Build

docker-compose build

1. Interromper e remover contêineres, redes, volumes e imagens criados por pelo comando up

docker-compose down

1. Iniciar contêineres existentes para um serviço

docker-compose start

1. Parar contêineres em execução sem remove-los

docker-compose stop

1. Reiniciar todos os serviços parados e em execução

docker-compose restart

Abaixo segue um exemplo de um arquivo docker-compose.yml que cria dois serviços: Drupal e Postgres e define algumas opções para eles.

version: '3.1'

services:

drupal:

image: drupal:8-apache

container\_name: drupal\_BCC

ports:

- 8080:80

volumes:

- /var/www/html/modules

- /var/www/html/profiles

- /var/www/html/themes

- /var/www/html/sites

restart: always

links:

- postgres

postgres:

image: postgres:10

environment:

POSTGRES\_PASSWORD: drupal

POSTGRES\_DB: drupal

restart: always