

FIAP GRADUAÇÃO

TECNOLOGIA EM DESENVOLVIMENTO DE SISTEMAS

DevOps Tools & Cloud Computing

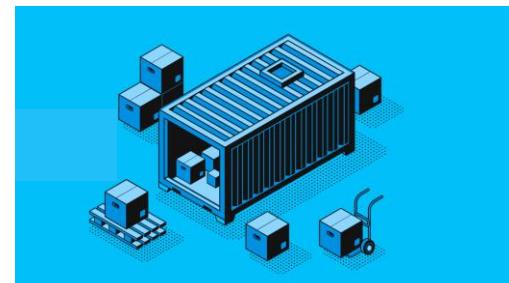
Docker

PROF. João Menk	profjoao.menk@fiap.com.br
PROF. Sálvio Padlipskas	salvio@fiap.com.br
PROF. Antonio Figueiredo	profantonio.figueiredo@fiap.com.br
PROF. Marcus Leite	profmarcus.leite@fiap.com.br
PROF. Thiago Rocha	profthiago.rocha@fiap.com.br
PROF. Thiago Moraes	proftiago.moraes@fiap.com.br

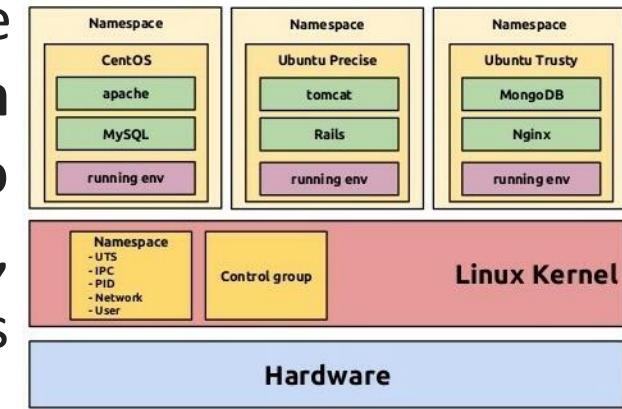
- No mundo real Containers são aquelas caixas enormes de metal que carregam cargas de todos os tipos. Imagine transportar tantos itens aleatórios sem os Containers? Organizar, manter seguro e agrupado
- No contexto da área de tecnologia, um Container parte da mesma ideia do mundo real: Padronizar, isolar e transportar



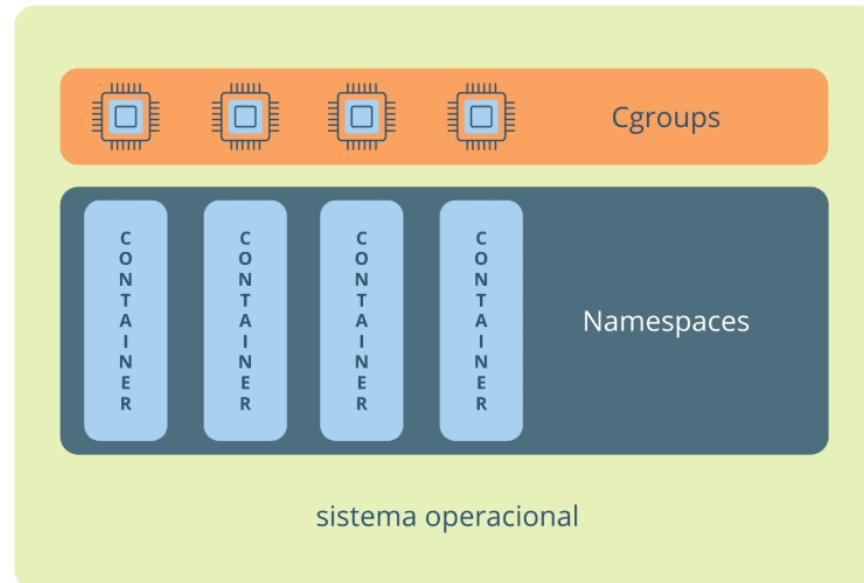
- Containers são tecnologias que nos permitem empacotar e isolar aplicações com seus ambientes de execução por completo, somado a todos os arquivos necessários para funcionar. Aplicações “Conteinerizadas” facilitam sua movimentação entre ambientes (desenvolvimento, teste, produção, etc) enquanto mantêm o funcionamento completo *Fonte: RedHat
- Basicamente é você pegar tudo o que sua aplicação precisa para executar e incluir em um único arquivo. Dessa forma, você garante que a aplicação vai rodar, não importa a versão da tecnologia que você está usando. Se a aplicação está Conteinerizada, qualquer ambiente que suporte container Docker, por exemplo, vai conseguir rodá-la



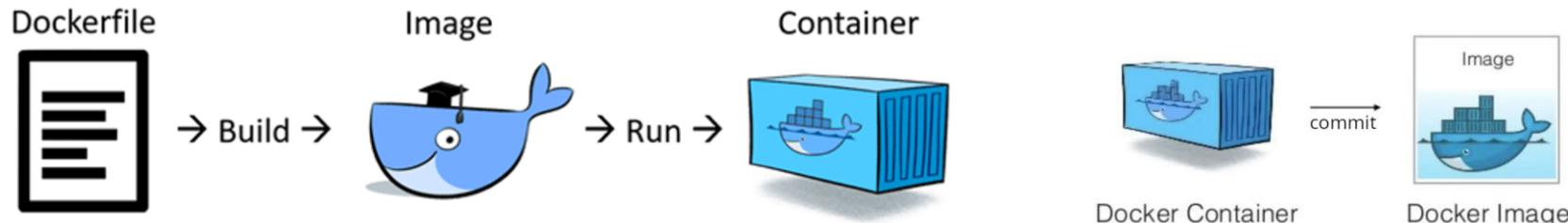
- Um Container não é uma única tecnologia, como o Docker por exemplo, mas sim um conjunto de tecnologias presentes no **Kernel Linux** que quando unidas nos permitem obter todas essas vantagens
- Os Containers fornecem ambientes de trabalho isolados e **são executados em cima da Infraestrutura do Host e do Sistema Operacional**. Assim como as VMs, uma máquina Host pode executar vários containers isolados uns dos outros
- Esses Containers compartilham o Kernel do Sistema Operacional Host, binários, bibliotecas, etc. Cada container tem seu próprio espaço de usuário, mas compartilham o mesmo espaço do Kernel



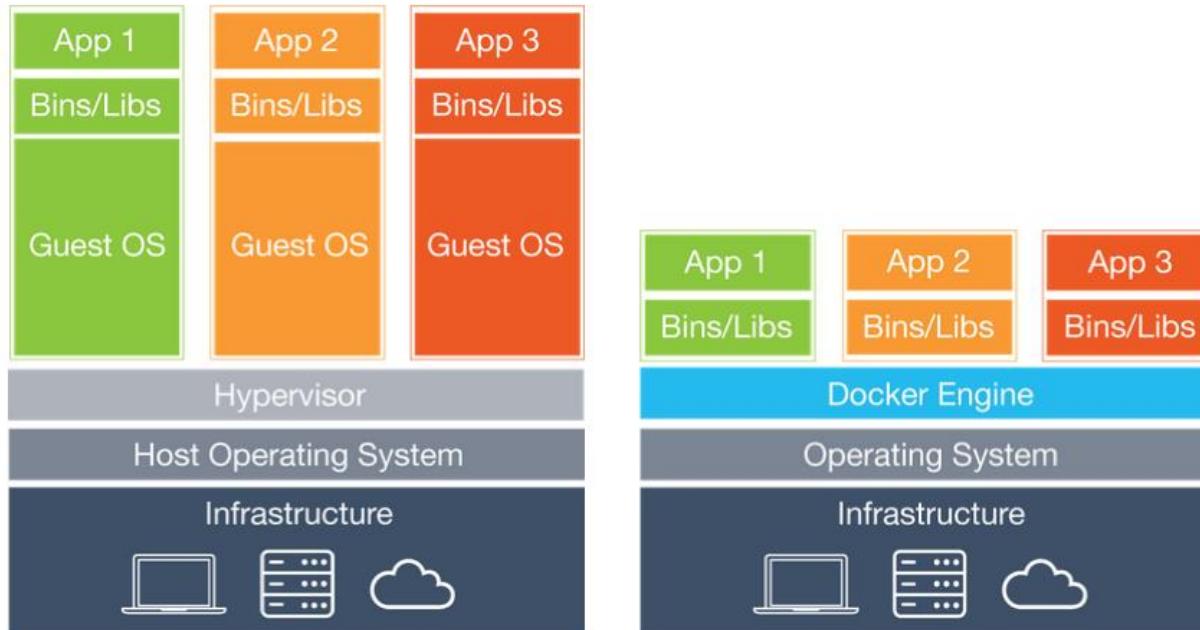
- Os Containers funcionam como processos no Host
- O Docker usa o Kernel do Linux e seus recursos para criar processos, esses recursos são: **Cgroups, Namespaces**
 - ✓ Containers atingem o isolamento através de **Namespaces**
 - ✓ Compartilhamento e gerenciamento de Memória e CPU através do **Cgroups**



- Um conceito importante ao se falar de Containers são as **Imagens**
Sendo o container um processo isolado, a imagem é o que dá a esse processo seu Sistema de Arquivos (File System)
Por exemplo, comparando com uma Máquina Virtual, nós temos a ISO do Sistema Operacional que desejamos e usamos esse arquivo no VirtualBox ou VMware para subir nosso Sistema Operacional
No caso do Docker utilizamos as **Imagens** para esse fim
- **As Imagens são imutáveis**, como uma foto, contêm todo o ambiente de execução de um container, suas dependências e binários. Tudo isso junto em um único pacote, e a este pacote damos o nome de **Image**



- Vamos iniciar com algumas coisas em comum:
 - a) Ambas utilizam a Tecnologia de Virtualização
 - b) Uma máquina hospedeira (é a máquina pela qual as máquinas virtuais e/ou os containers executam)
- O que muda é a forma como é feita a Virtualização, os **Containers** não utilizam Hypervisor como as Máquinas Virtuais, e sim os recursos do Sistema e Processos de Kernel para criar os ambientes



- Máquinas Virtuais e Containers têm suas próprias vantagens, desvantagens e casos de uso. Não existe uma regra específica, rígida ou rápida que você precise utilizar. Depende totalmente do seu caso de uso e requisitos

Cenários mais comuns em que a **Máquina Virtual** é utilizada

- ✓ Usadas quando você precisa dar prioridade à segurança mais do que à utilização de recursos (VMs fornecem ferramentas de gerenciamento e segurança)
- ✓ Quando você precisa gerenciar uma ampla variedade de Sistemas Operacionais ou precisa usar todas as funcionalidades e recursos do Sistema Operacional do Host
- ✓ Quando a aplicação for antiga (legada) e sua forma de trabalho é completamente monolítica
- ✓ Tarefas que possuem riscos de alta segurança, como manipulação de dados infectados por vírus
- ✓ Criação de Backups de SO e testes de SO

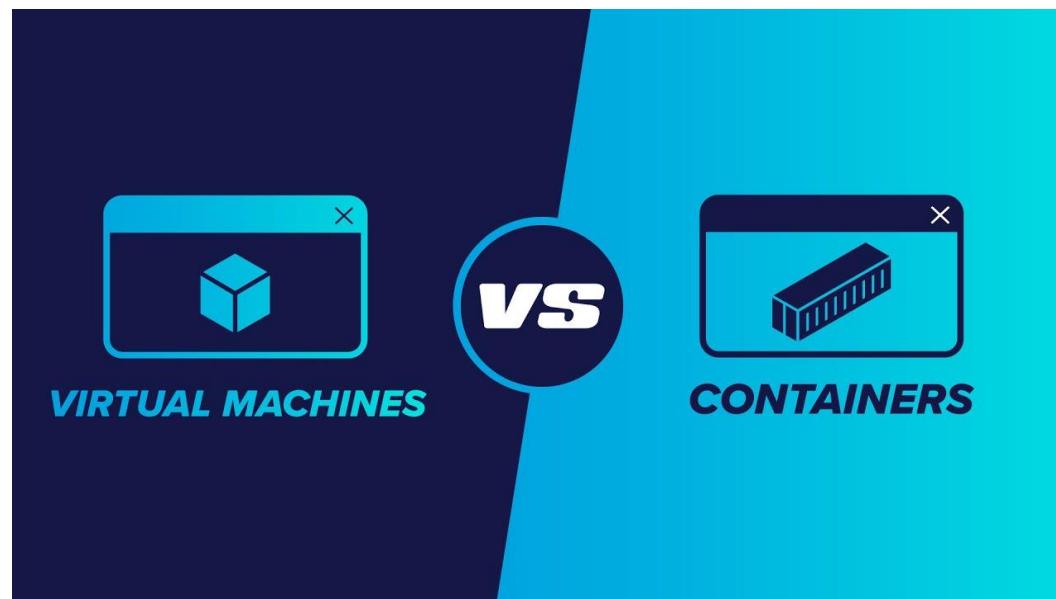


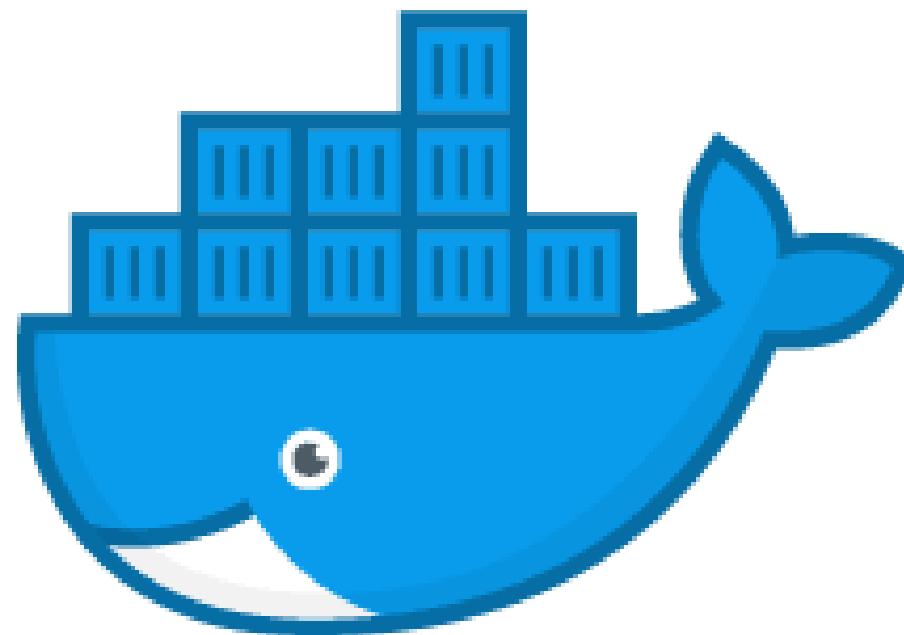
- ✓ **Os Containers** são utilizados quando construirmos aplicações voltadas para Cloud
- ✓ Quando estamos criando Microsserviços
- ✓ Quando queremos aplicar práticas DevOps ou de CI/CD de forma mais agressiva
- ✓ Quando o projeto é escalável e pode se espalhar em uma infraestrutura que compartilha o mesmo Sistema Operacional
- ✓ Para promover o gerenciamento reduzido de recursos, permitindo que os desenvolvedores criem facilmente aplicativos em ambientes de trabalho pré-construídos, empacotados e isolados





- Cuidado ao comparar o Docker com as Máquinas Virtuais, pois são ambos destinados a usos diferentes
- Na verdade, muitas implantações de Containers usam VMs como o Sistema Operacional do Host em vez da execução direta no Hardware, especialmente ao executar Containers na Nuvem

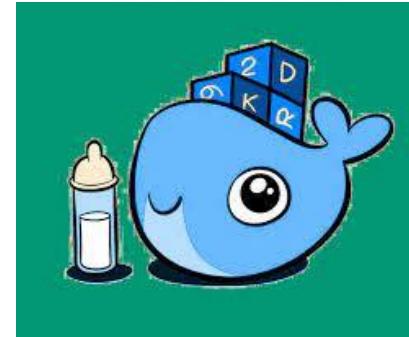




docker

Conceito

- Em 2008 a empresa dotCloud nascia com a ideia de desenvolver soluções PaaS para o mercado. Posteriormente passou a se chamar Docker, Inc e foi a responsável por popularizar os chamados Containers
- A ideia de Containers já vinha sendo usado a bastante tempo nas plataformas Linux/Unix/Solaris, porém ficou popular através da reformulação da ideia que a Docker Inc trouxe, e pela adoção dessa tecnologia por grandes Players do mercado como: RedHat, Google e AWS
- Projetos como o LXC (Linux Containers) já existem faz muitos anos (2008), porém o seu uso é complexo e a curva de aprendizado é bastante grande



The diagram illustrates the differences between three types of containers:

- LXC (represented by a grey cube)
- LXD (represented by a grey cube)
- Docker (represented by a blue whale icon)

Below the containers, the text reads: "What is the Difference Between LXC, LXD and Docker Containers".

<https://linuxways.net/centos/what-is-the-difference-between-lxc-lxd-and-docker-containers/>

<https://www.educba.com/lxc-vs-docker/>

<https://www.upguard.com/blog/docker-vs-lxc>

<https://medium.com/@harsh.manvar111/lxc-vs-docker-lxc-101-bd49db95933a>

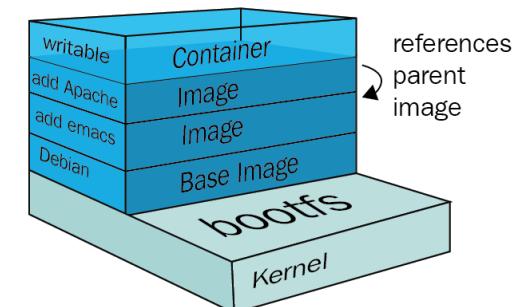
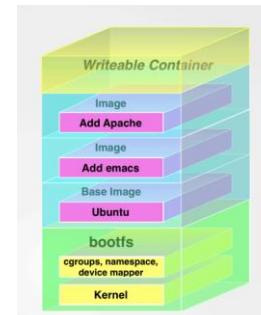
- Docker é uma plataforma de código aberto, desenvolvida na linguagem Go e criada pelo próprio Docker Inc. Por ser de alto desempenho, o software garante maior facilidade na criação e administração de ambientes isolados, garantindo a rápida disponibilização de programas para o usuário final
- O lançamento inicial do Docker foi em março de 2013 e, desde então, tornou-se a palavra de ordem para o desenvolvimento do mundo moderno, especialmente diante de projetos baseados em abordagem Ágil
- Uma das razões para a preferência pelo Docker é o fato de que ele não só simplificou o uso dos Containers, mas também introduziu o seu fornecimento através da nuvem



Imagens - Camadas e Controle de Versão

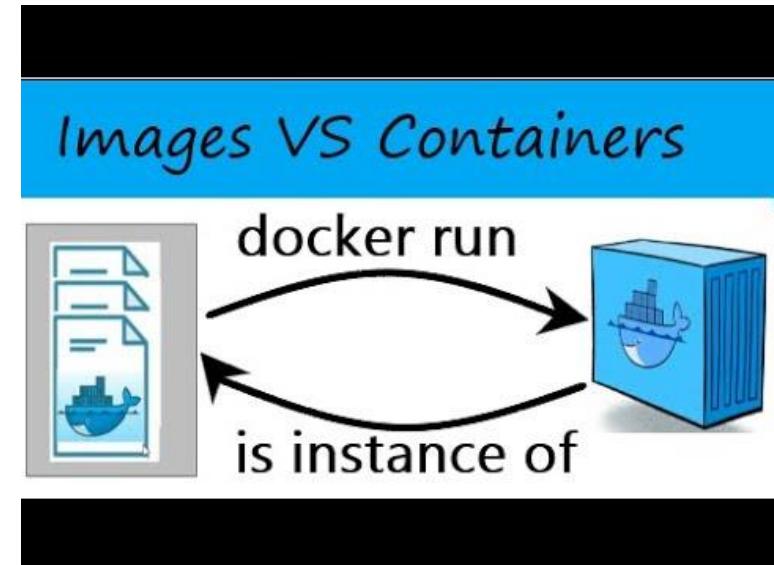
- Um arquivo Docker pode ser formado por diversas camadas diferentes, onde se dividem em dois grupos:
 - ✓ **Imagens:** São formadas por diferentes camadas. Com a sua utilização, o usuário pode facilmente compartilhar um aplicativo ou um conjunto de serviços em diversos ambientes. Quando há alguma alteração na imagem, ou uso de um comando como executar ou copiar, é criada uma nova camada
 - ✓ **Containers:** São formados na reutilização das camadas. **Um container é o local onde estão as modificações da aplicação que está em execução.** É por meio dele que o usuário pode trabalhar em uma Imagem (Camada adicional de Read/Write)

```
FROM ubuntu          ← 1
MAINTAINER sofija ← 2
RUN apt-get update ← 3
CMD ["echo", "Hello World"] ← 4
```



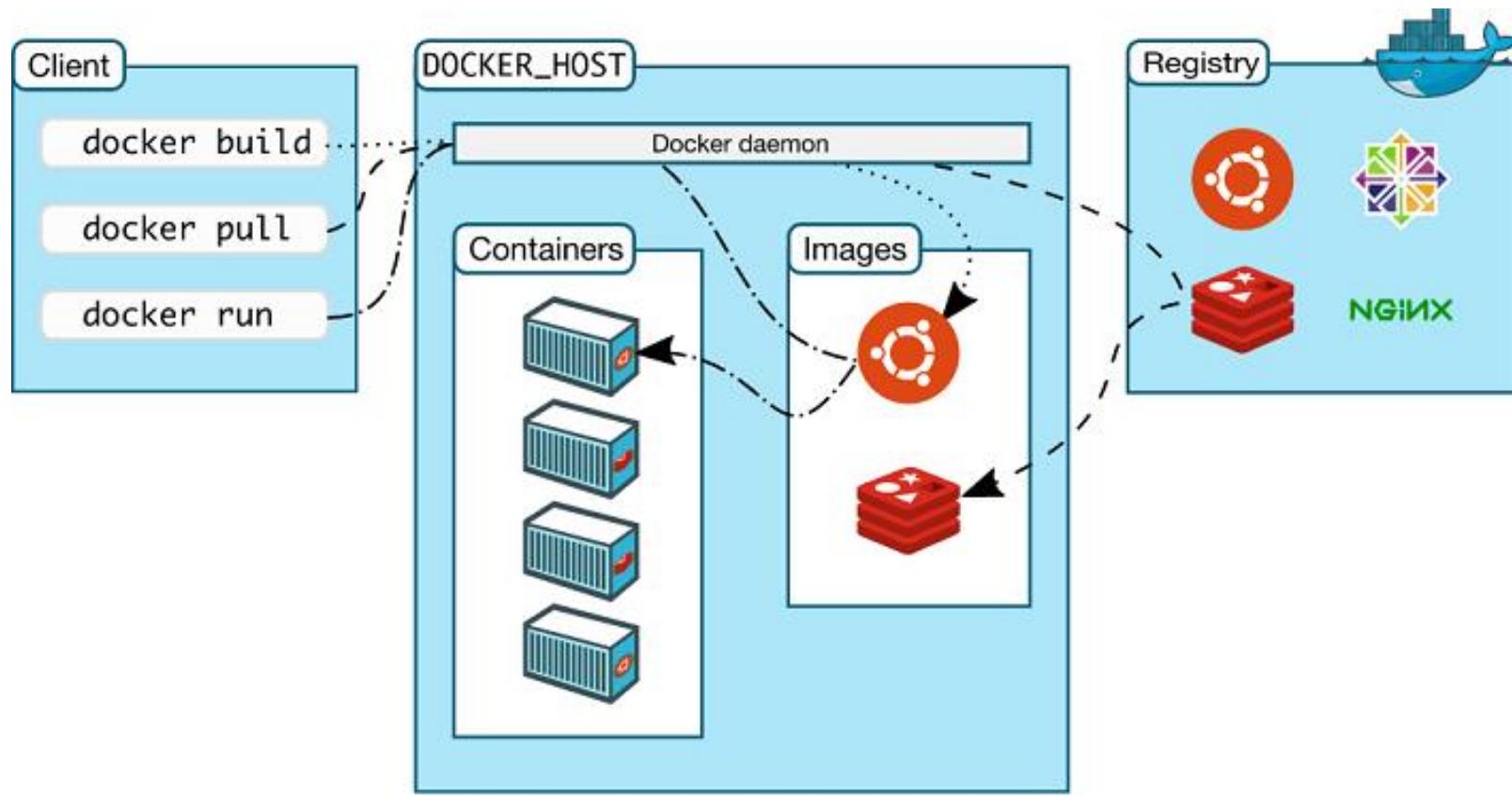
Relação Containers x Imagens

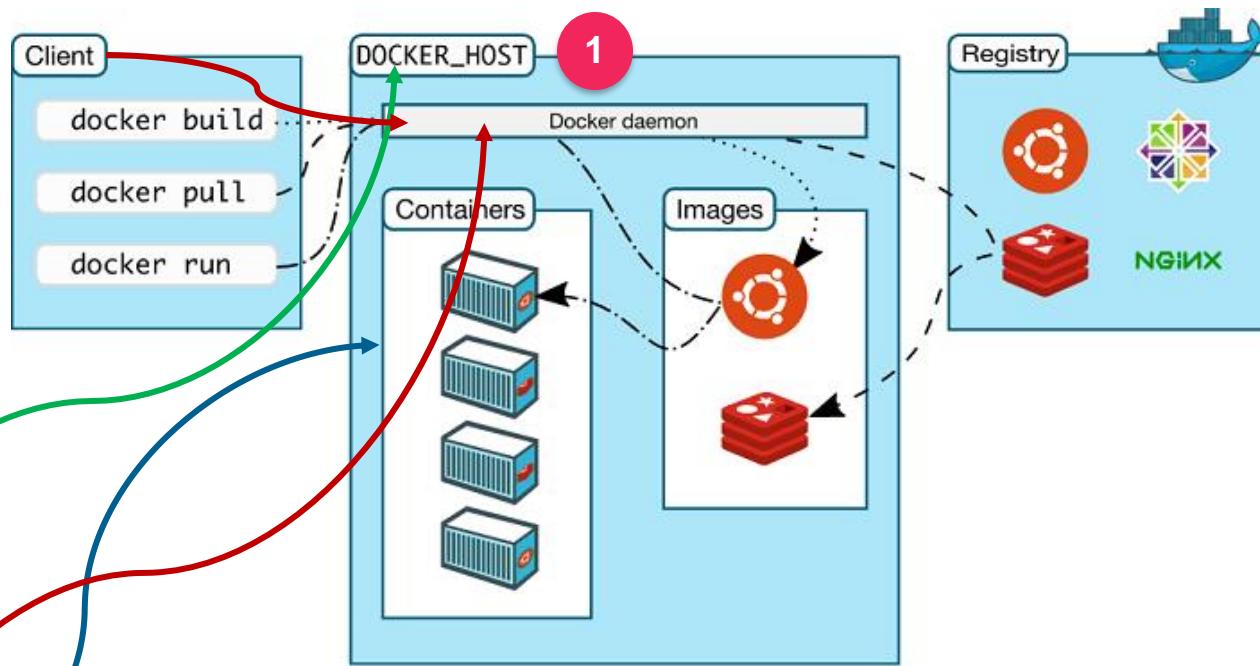
- As Imagens são como receitas para criar o Container
- Um Conjunto de Camadas formam uma Imagem
- A Imagem é imutável
- Um Container tem uma camada adicional de Read/Write na Imagem



Através de cada imagem, é possível criar Containers. Dentro de cada Container, existirá um Sistema Operacional que compartilhará o Kernel com a Máquina Hospedeira (Host)

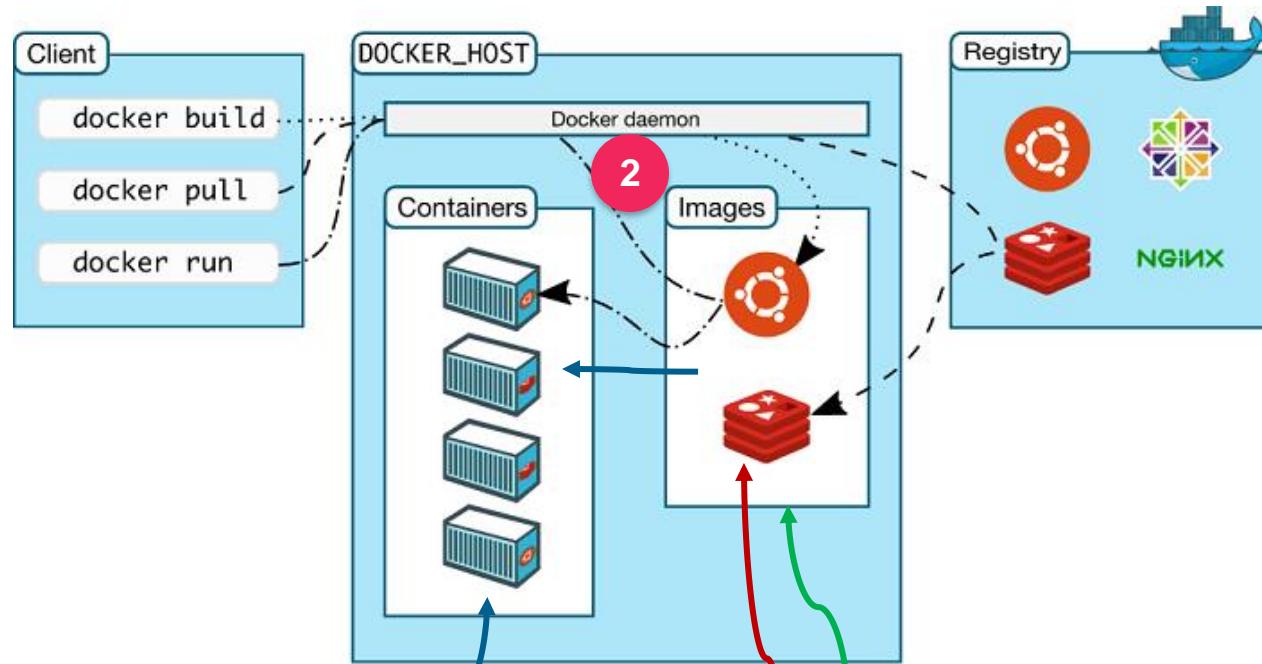
O sistema do Docker é dividido basicamente em quatro partes: **Docker Server e Daemon, Imagens, Containers e Registros**





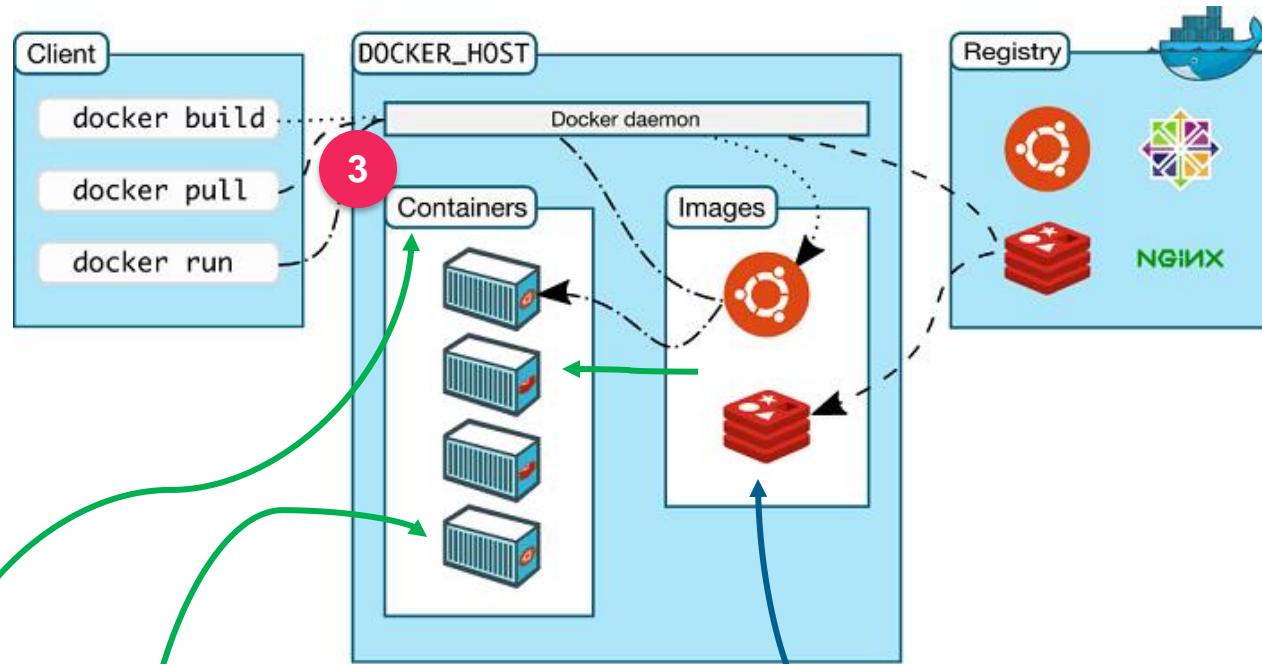
- ✓ O Docker possui uma arquitetura de cliente-servidor, os Containers são armazenados em um Servidor, chamado de Docker Host ou Docker Server
- ✓ O servidor do Docker é responsável por todas as ações relacionadas aos Containers e Imagens
- ✓ O Docker Daemon (serviço) recebe comandos do Cliente a partir de CLI ou API's REST

O Docker Host pode ser local ou remoto, e tem a capacidade de criar, iniciar, desligar e excluir vários containers, e pode oferecer para cada cliente um ou mais Containers



- ✓ Imagens são arquivos que contém todo o conteúdo e estrutura de Sistemas Operacionais
- ✓ São a base de construção de Containers no Docker
- ✓ O servidor do Docker utiliza o AuFS (Advanced multi-layered Unification FileSystem), um Sistema de Arquivos em camadas que permite a separação do espaço do sistema em uma parte Read-only e outra parte Read/write

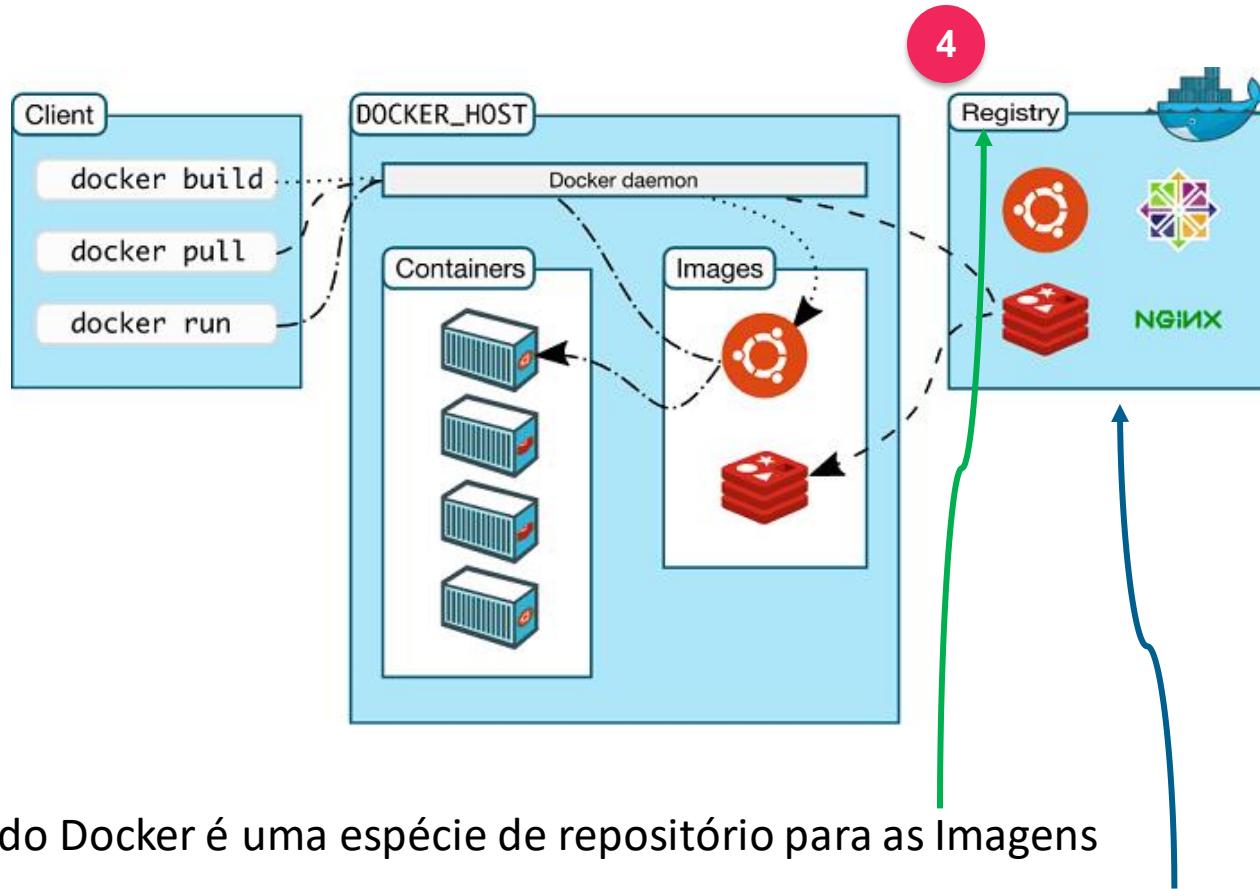
Ao executar um container de um cliente, o Docker utiliza a imagem de Sistema Operacional escolhida como base, e numa camada superior executa as configurações escolhidas pelo cliente



- ✓ Containers são os ambientes de execução do Docker, criados a partir de Imagens
- ✓ De forma simplificada, é uma sandbox para processos (um ambiente de teste, que isola o ambiente de produção e o repositório de mudanças)
- ✓ Uma Imagem é como um Template de classe, e Containers seriam instâncias dessas classes (instanciados na uma camada que permite gravação da Imagem)

O que é o Docker

FIAP



- ✓ O Registro do Docker é uma espécie de repositório para as Imagens
- ✓ Com esse registro, um usuário pode construir, armazenar e distribuir Imagens

Esse registro pode ser on-premises: Hosted Docker Registry, Docker Trusted Registry etc

Ou em Nuvem: Docker Hub Registry, Azure Container Registry, AWS Container Registry, Google Container Registry etc

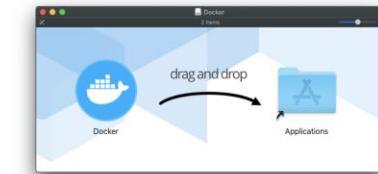
Componentes do Docker

Docker for Mac

Permite executar Containers do Docker no Mac OS



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



Docker for Linux

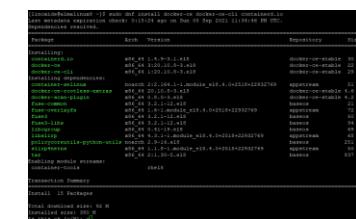
Permite executar Containers Docker no Sistema Operacional Linux



<https://docs.docker.com/engine/install/rhel/>

<https://docs.docker.com/engine/install/ubuntu/>
...

<https://docs.docker.com/engine/install/>



Docker for Windows

Permite executar Containers do Docker no Sistema Operacional Windows



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

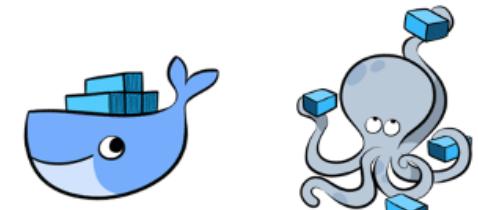


✓ **Docker Engine**

Usado para construir Imagens e criar Containers no Docker

Docker Engine

Docker Compose



✓ **Docker Compose**

É utilizado para definir e compartilhar aplicativos de vários Containers (Orquestrador de containers da Docker)

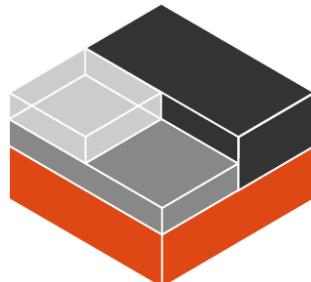
✓ **Docker Hub / Azure, AWS, GCP etc**

É o registro que é usado para hospedar várias Imagens do Docker





podman
Ganhando bastante terreno



LXD
Código aberto



Kaniko
Google



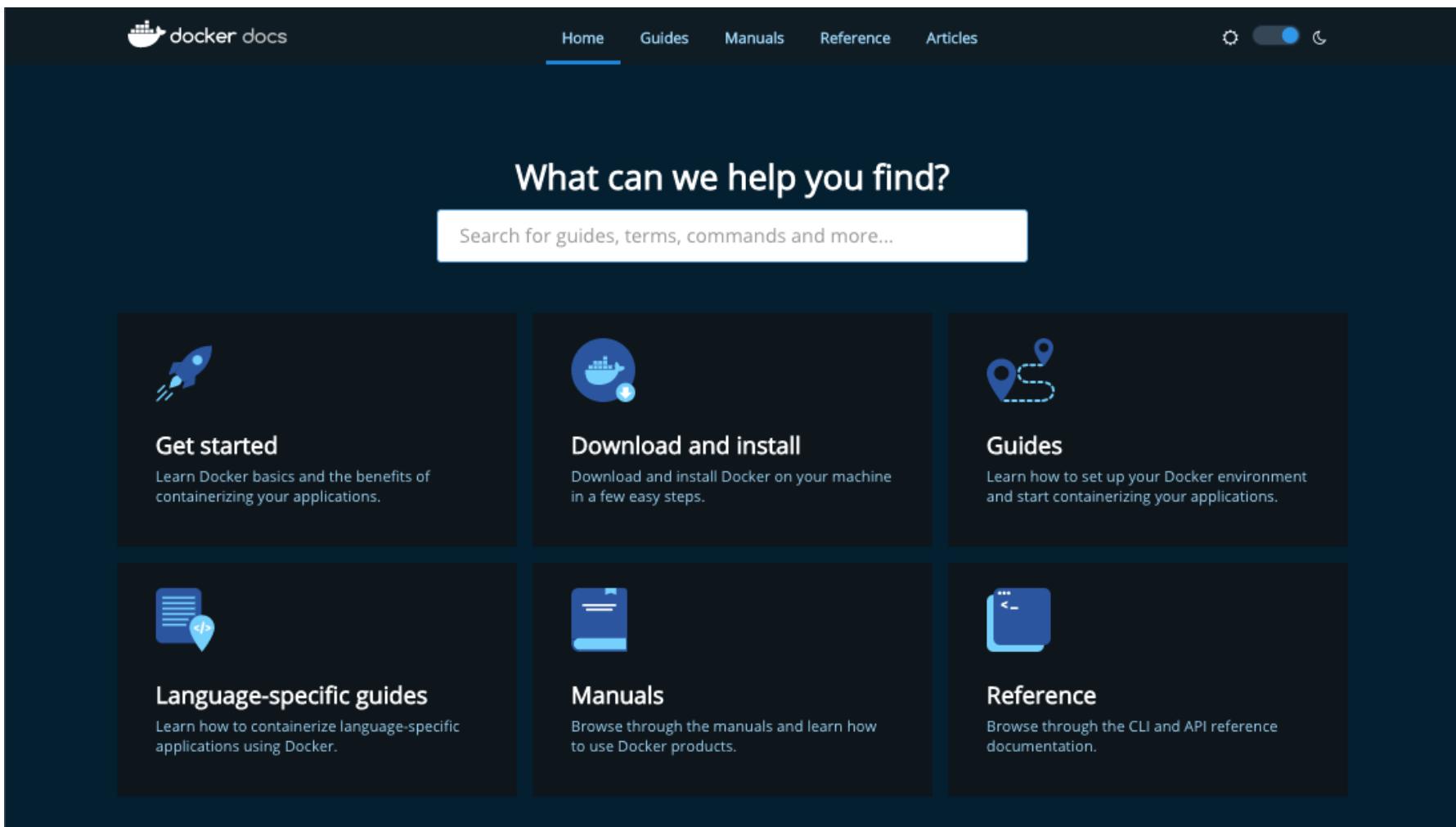
buildah
Red Hat

e muitos outros...

USANDO O DOCKER



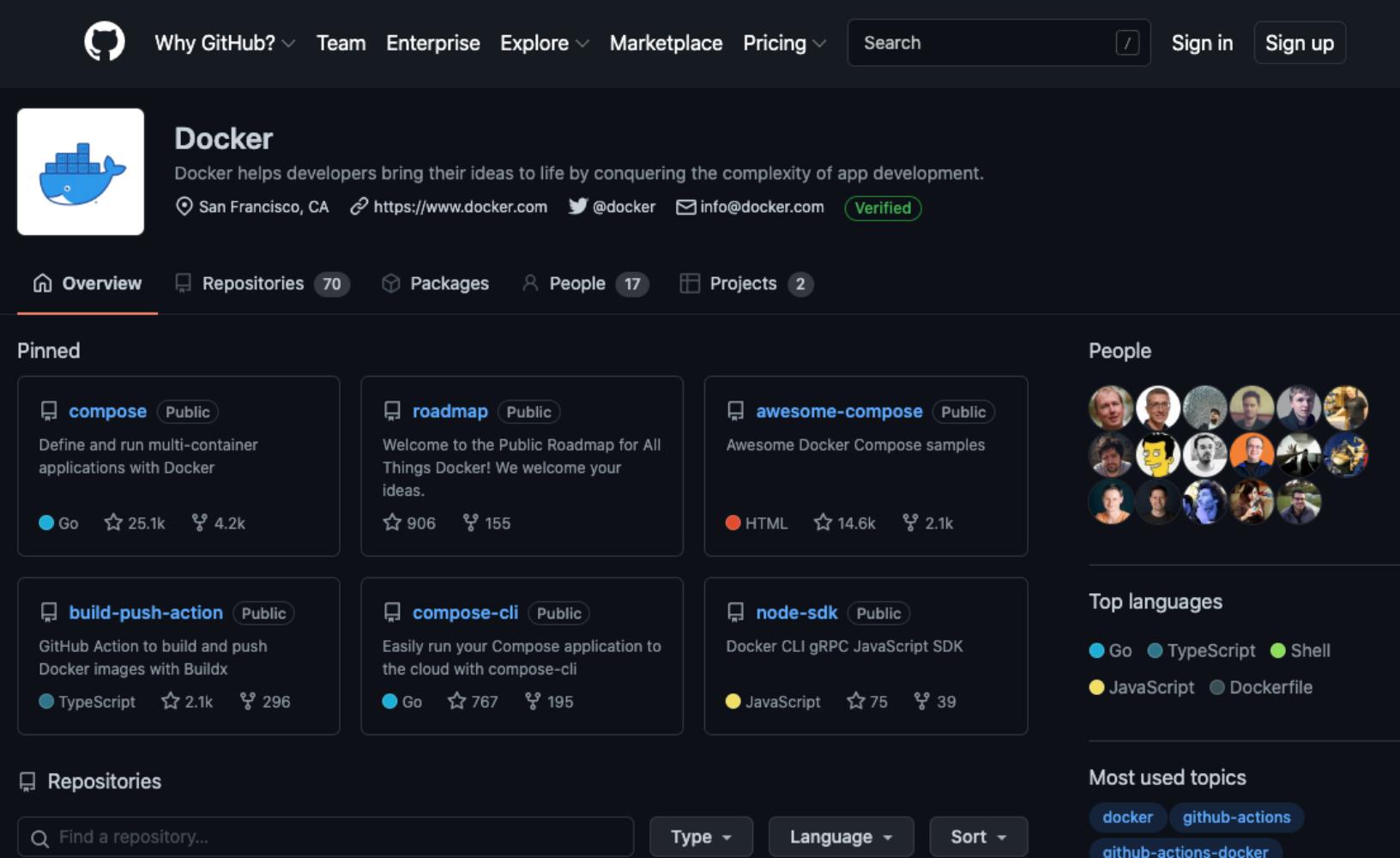
<https://docs.docker.com>



The screenshot shows the official Docker documentation website at <https://docs.docker.com>. The page has a dark blue header with the Docker logo and navigation links for Home, Guides, Manuals, Reference, and Articles. A search bar is centered above a grid of six cards. The cards are arranged in two rows of three. The top row contains: 'Get started' (with a rocket icon), 'Download and install' (with a Docker ship icon), and 'Guides' (with a location pin icon). The bottom row contains: 'Language-specific guides' (with a document icon), 'Manuals' (with a book icon), and 'Reference' (with a clipboard icon). Each card has a brief description below its title.

- Get started**
Learn Docker basics and the benefits of containerizing your applications.
- Download and install**
Download and install Docker on your machine in a few easy steps.
- Guides**
Learn how to set up your Docker environment and start containerizing your applications.
- Language-specific guides**
Learn how to containerize language-specific applications using Docker.
- Manuals**
Browse through the manuals and learn how to use Docker products.
- Reference**
Browse through the CLI and API reference documentation.

<https://github.com/docker>



The screenshot shows the GitHub profile for the Docker organization. The header includes the GitHub logo, navigation links (Why GitHub?, Team, Enterprise, Explore, Marketplace, Pricing), a search bar, and sign-in links. The main profile area features a blue whale icon, the word "Docker", a brief description, location (San Francisco, CA), website (https://www.docker.com), social links (@docker on Twitter, info@docker.com), and a green "Verified" badge. Below this are sections for Overview (with 70 repositories), Packages, People (17), and Projects (2). The "Pinned" section contains cards for "compose" (multi-container applications), "roadmap" (Public Roadmap), "awesome-compose" (samples), "build-push-action" (GitHub Action for Docker images), "compose-cli" (CLI for compose), and "node-sdk" (Docker CLI gRPC JavaScript SDK). The "People" section shows a grid of 17 profile pictures. The "Top languages" section lists Go, TypeScript, Shell, JavaScript, and Dockerfile. The "Most used topics" section lists docker, github-actions, and github-actions-docker. A search bar at the bottom allows users to find specific repositories.

Why GitHub? ▼ Team Enterprise Explore ▼ Marketplace Pricing ▼

Search / Sign in Sign up

 Docker

Docker helps developers bring their ideas to life by conquering the complexity of app development.

San Francisco, CA <https://www.docker.com> [@docker](#) info@docker.com Verified

[Overview](#) [Repositories 70](#) [Packages](#) [People 17](#) [Projects 2](#)

Pinned

 [compose](#) Public

Define and run multi-container applications with Docker

Go 25.1k Stars 4.2k

 [roadmap](#) Public

Welcome to the Public Roadmap for All Things Docker! We welcome your ideas.

Stars 906 Forks 155

 [awesome-compose](#) Public

Awesome Docker Compose samples

HTML 14.6k Stars 2.1k

 [build-push-action](#) Public

GitHub Action to build and push Docker images with Buildx

TypeScript 2.1k Forks 296

 [compose-cli](#) Public

Easily run your Compose application to the cloud with compose-cli

Go 767 Forks 195

 [node-sdk](#) Public

Docker CLI gRPC JavaScript SDK

JavaScript 75 Forks 39

People



Top languages

Go TypeScript Shell
JavaScript Dockerfile

Most used topics

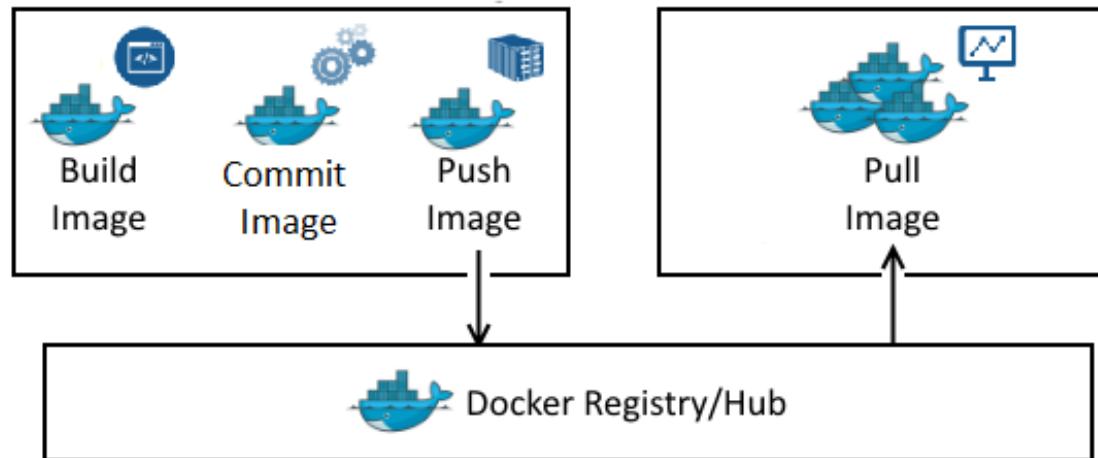
docker github-actions
github-actions-docker

Find a repository... Type Language Sort

- O Docker Hub é um serviço de registro na nuvem que permite baixar Imagens do Docker criadas por outras comunidades. Você também pode carregar suas próprias Imagens criadas do Docker para o Docker Hub

Página Inicial do Docker Hub

<https://www.docker.com/products/docker-hub>



Criar id Docker Hub

Primeiro, você precisa fazer uma inscrição simples no Docker Hub. Caso seja o seu primeiro acesso, pressione o botão **Go to Docker Hub**

1

The screenshot shows the Docker Hub homepage at docker.com/products/docker-hub. At the top, there's a navigation bar with links for Products, Developers, Pricing, Blog, About us, and Partners. On the right side of the header, there are 'Sign In' and 'Get Started' buttons. The main content area features a large heading 'Docker Hub' and a subtext: 'The world's leading service for finding and sharing container images with your team and the Docker community.' Below this, there are two blue buttons: 'Go to Docker Hub' (which is highlighted with a red box) and 'Secure, Private Repo Pricing'. Further down, there's a section titled 'Share and Collaborate with Docker Hub' with a subtext about Docker Hub being the largest repository of container images.

2

The screenshot shows the 'Get Started Today for Free' sign-up form. It includes fields for 'Email' (spadlipskas) and 'Password' (spadlipskas@gmail.com). There are also fields for 'First Name' (.....) and 'Last Name' (.....). Below the fields are two checkboxes: one for product updates and another for agreeing to the Subscription Service Agreement, Privacy Policy, and Data Processing Terms. A reCAPTCHA field is also present. At the bottom, a large blue 'Sign Up' button is highlighted with a red box.

Cadastre seus dados e crie sua conta com login no Docker. Depois pressione o botão **Sign Up**

Informe o nome do **ID Docker** e pressione o botão **Continue**

Welcome

Log in to Docker to continue to Docker Hub.

Username or email address

Continue

Informe sua senha do id Docker e pressione o botão Continue

Enter Your Password

spadlipskas [Edit](#)

Password [@](#)

[Forgot password?](#)

Continue

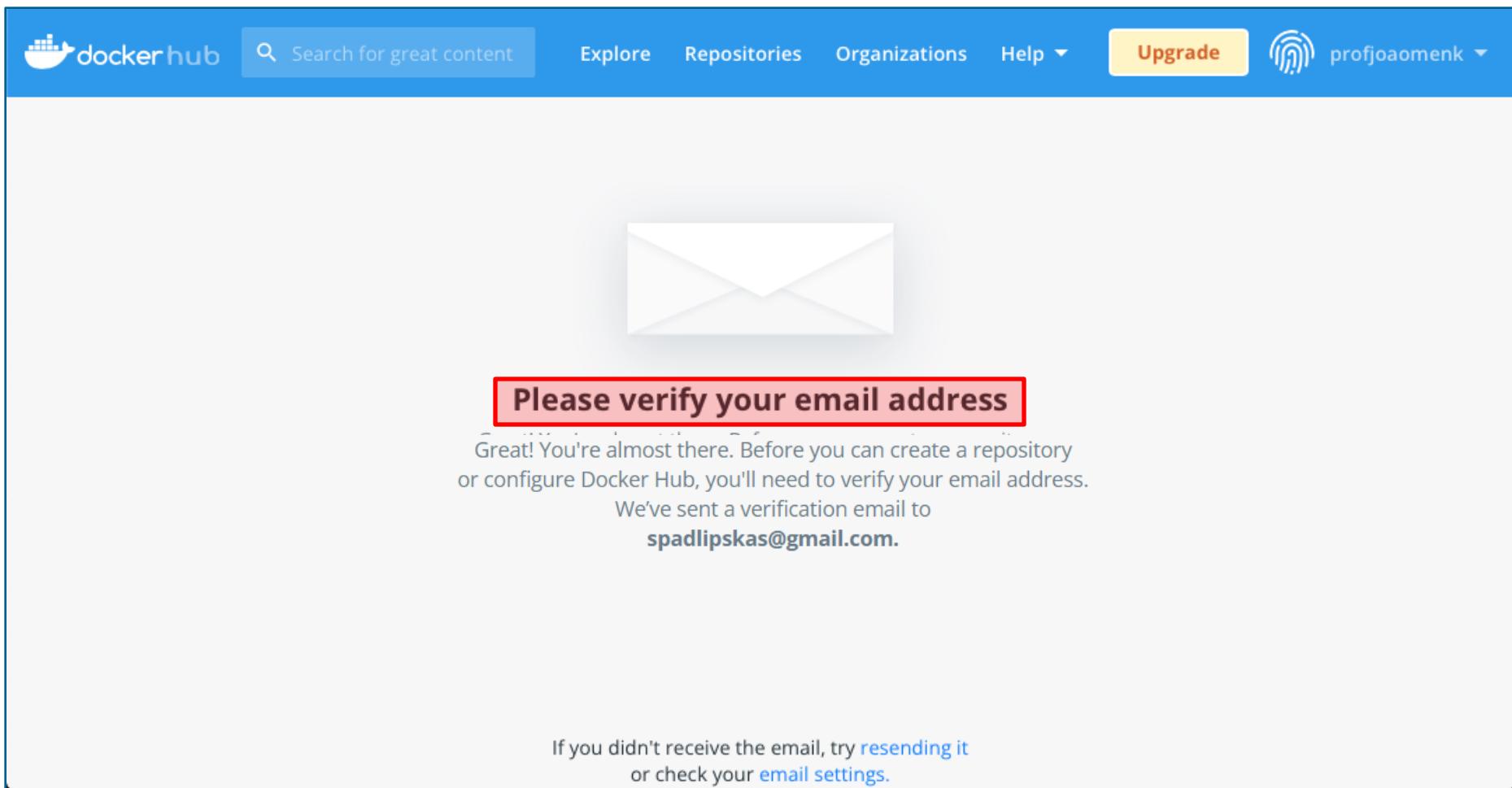
Escolha o Plano que deseja

The screenshot shows the Docker Hub pricing page with four plan options:

- Personal** (\$0): Ideal for individual developers, education, open source communities, and small businesses. Includes Docker Desktop, unlimited public repositories, Docker Engine + Kubernetes, and limited image pulls per day. A red box highlights this plan.
- Pro** (\$5/month): Extends Docker capabilities for individual developers, including pro tools for accelerating productivity.
- Team** (\$7 /user/month, minimum 5 seats): A developer favorite, ideal for teams, offering enhanced collaboration, productivity, and security. This plan is highlighted with a blue box.
- Business** (\$21 /user/month, annual subscription): Ideal for medium and large businesses needing centralized management and advanced security.

Each plan includes a 'Buy Now' button and a note about billing frequency. The 'Team' plan is described as a 'Developer Favorite'.

Verifique seu e-mail informado para ativar a conta



The screenshot shows the Docker Hub homepage with a verification message. At the top, there's a blue header bar with the Docker Hub logo, a search bar, and navigation links for Explore, Repositories, Organizations, Help, Upgrade, and a user profile icon for 'profjoaomenk'. The main content area features a large envelope icon and a red-bordered box containing the text 'Please verify your email address'. Below this, a message says: 'Great! You're almost there. Before you can create a repository or configure Docker Hub, you'll need to verify your email address.' It then states: 'We've sent a verification email to' followed by the email address 'spadlipskas@gmail.com.'. At the bottom, there's a note: 'If you didn't receive the email, try [resending it](#) or check your [email settings](#)'.

Verifique seu e-mail informado para ativar a conta

The screenshot shows an email from Docker in a inbox. The subject is "[Docker] Please confirm your email address". The email is from "Docker <no-reply@notify.docker.com>" and was sent "19:55 (há 3 minutos)". The message body includes a Docker logo, a greeting to "profjoaomenk", and a message asking to verify the email address by clicking a blue button labeled "Verify email address". This button is highlighted with a red rectangle. The message concludes with "- The Docker Team". At the bottom of the email view, there are buttons for "Responder" and "Encaminhar".

[Docker] Please confirm your email address

Docker <no-reply@notify.docker.com>
para mim

19:55 (há 3 minutos)

Caixa de entrada

⋮

profjoaomenk,

Thanks for creating a Docker ID. Please verify your email address by clicking the button below.

[Verify email address](#)

- The Docker Team

Responder Encaminhar

<https://www.docker.com/products/docker-hub>

Se logue em sua conta



Welcome

Log in to Docker to continue to Docker Hub.

Username or email address

Continue



Enter Your Password

profjoao.menk@gmail.com [Edit](#)

Password [Edit](#)

[Forgot password?](#)

Continue

Navegue no Docker Hub

A screenshot of the Docker Hub homepage. At the top, there is a blue header bar with the Docker Hub logo, a search bar containing "Search for great content (e.g., mysql)", and navigation links for "Explore", "Repositories", "Organizations", and "Help". A red arrow points down to the search bar. To the right of the header are buttons for "Upgrade" and a user profile with the name "profjoaomenk". Below the header, the main content area shows a user profile for "profjoaomenk" with a repository named "profjoaomenk / docker101tutorial". The repository was last pushed 3 months ago and has 0 stars, 6 downloads, and is public. There is also a tip message about switching namespaces. To the right, there is a promotional box for creating an organization and managing repositories.

profjoaomenk

Search for great content (e.g., mysql)

Explore Repositories Organizations Help ▾

Upgrade profjoaomenk ▾

profjoaomenk / **docker101tutorial**
Last pushed: 3 months ago

Not Scanned 0 ⚡ 6 Public

Tip: Not finding your repository? Try switching namespace via the top left dropdown.

Create Repository

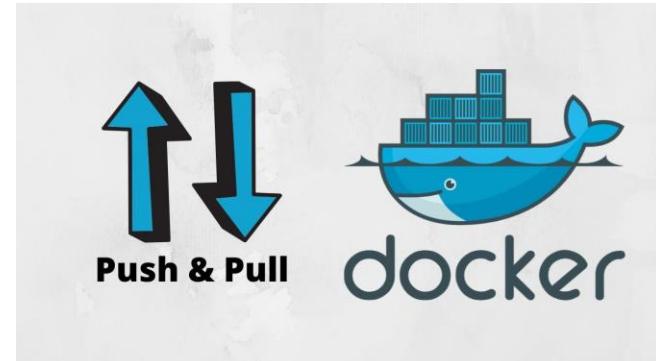
Create an Organization
Manage Docker Hub repositories
with your team

- A utilização do Docker é feita através de CLI (Command Line Interface): Shell, Bash ou CMD do Windows
- Existe a interface gráfica, porém é pouco utilizada
- Para utilizar uma Imagem específica, por exemplo do Sistema Operacional Ubuntu, precisamos primeiro obtê-la, e isso é feito com o comando **docker pull**
- Vamos dar uma olhada no Docker Hub para a Imagem do Ubuntu

```
$ docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Documentação:

<https://docs.docker.com/engine/reference/commandline/pull/>



Comandos Docker

FIAP

The screenshot shows the Docker Hub interface. A red circle labeled '1' highlights the search bar containing the text 'ubuntu'. A red arrow labeled '2' points from the search bar down to the search results. The results page displays '1 - 25 of 117.306 results for ubuntu'. The top result is 'ubuntu' (Official Image), which is a Debian-based Linux operating system. It has 1B+ Downloads and 10K+ Stars. The image card includes a red icon, the name 'ubuntu', a 'Verified Publisher' badge, and a 'Base Images' badge. Below the image card, there are tabs for Container, Linux, PowerPC 64 LE, riscv64, IBM Z, 386, x86-64, ARM, and ARM 64. The second result listed is 'websphere-liberty' (Official Image), which is updated 3 days ago and based on Ubuntu 18.04.

1

2

ubuntu

Explore Repositories Organizations Help Upgrade profjoao

Docker Containers Plugins

Filters

Images

Verified Publisher i

Official Images i
Official Images Published By Docker

Categories i

Analytics

Application Frameworks

Application Infrastructure

Application Services

1 - 25 of 117.306 results for **ubuntu**. [Clear search](#)

Best Match

ubuntu Updated a month ago

Ubuntu is a Debian-based Linux operating system based on free software.

Container Linux PowerPC 64 LE riscv64 IBM Z 386 x86-64 ARM ARM 64

Base Images Operating Systems

websphere-liberty Updated 3 days ago

WebSphere Liberty multi-architecture images based on Ubuntu 18.04

10M+ 283 Downloads Stars

Navegue na tela da imagem

The screenshot shows the Docker Hub interface for the 'ubuntu' image. At the top, there's a search bar with 'ubuntu', navigation links for 'Explore', 'Repositories', 'Organizations', 'Help', and an 'Upgrade' button. Below the header, the image card for 'ubuntu' is displayed, featuring a ship icon, the name 'ubuntu', an 'Official Image' badge, and a star icon. A brief description states: 'Ubuntu is a Debian-based Linux operating system based on free software.' Below the image card, there's a download count of '1B+' and a list of supported architectures: Container, Linux, PowerPC 64 LE, riscv64, IBM Z, 386, x86-64, ARM, and ARM 64. Underneath these, there are categories: Base Images, Operating Systems, and Official Image. On the right side of the image card, there's a red box highlighting a command-line interface (CLI) input field containing 'docker pull ubuntu'. Above this field is a placeholder text: 'Copy and paste to pull this image'. Below the CLI field is a link 'View Available Tags'.

Quick reference

- Maintained by: Canonical
- Where to get help: the

Supported tags and respective Dockerfile links

- 18.04, bionic-20220128, bionic
- 20.04, focal-20220113, focal, latest
- 21.10, impish-20220128, impish, rolling
- 22.04, jammy-20220130, jammy, dev
- 14.04, trusty-20191217, trusty
- 16.04, xenial-20210804, xenial

How to use this image

What is Ubuntu?

License



Examine as várias sessões na página

- Vamos executar o comando **docker pull** para "pegar" uma imagem do Ubuntu do Docker Hub
- Com o serviço do Docker iniciado, abra um terminal e digite o comando abaixo

docker pull ubuntu

```
iMac:~ Menk$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
08c01a0ec47e: Extracting 28.56MB/28.56MB
08c01a0ec47e: Pull complete
Digest: sha256:669e010b58baf5beb2836b253c1fd5768333f0d1dbcb834f7c07a4dc93f474be
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Usando Tags para recuperar outras imagens do Ubuntu:

docker pull ubuntu:20.04

docker pull ubuntu:16.04

Supported tags and respective Dockerfile links

- 18.04, bionic-20220128, bionic
- 20.04, focal-20220113, focal, latest
- 21.10, impish-20220128, impish, rolling
- 22.04, jammy-20220130, jammy, devel
- 14.04, trusty-20191217, trusty
- 16.04, xenial-20210804, xenial

A imagem na Interface Gráfica

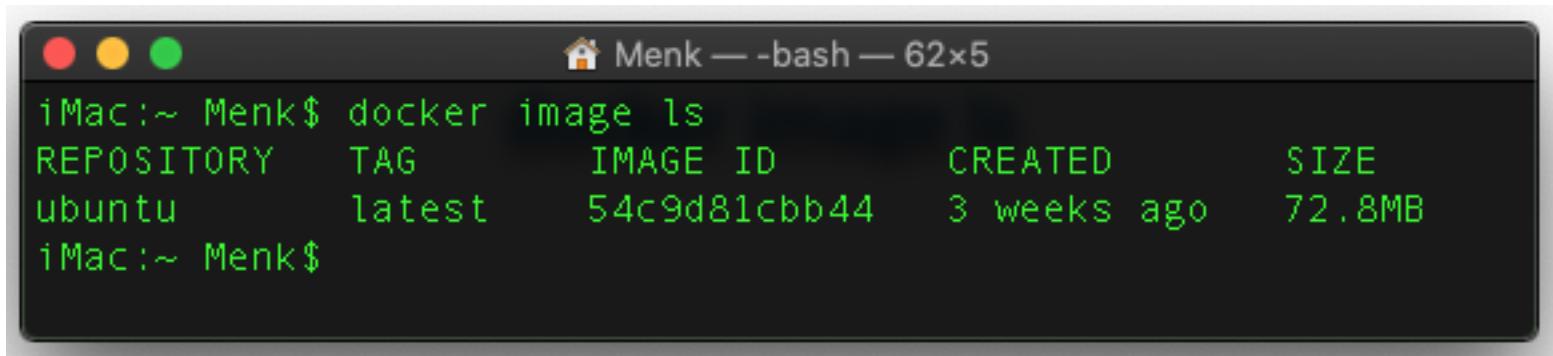
The screenshot shows the Docker desktop application window. On the left, there's a sidebar with icons for Containers / Apps, Images (which is selected and highlighted with a red box), Volumes, and Dev Environments (with a PREVIEW button). The main area is titled "Images on disk" and shows "1 images" with a total size of "72.78 MB". It has tabs for LOCAL and REMOTE REPOSITORIES, with a yellow arrow pointing from the LOCAL tab to the text "Imagens que possui no Docker Hub etc". There's also a "Search" bar and an "In Use only" checkbox. A table lists the single image: Name is "ubuntu", Tag is "latest", Image ID is "54c9d81cbb44", Created is "27 days ago", and Size is "72.78 MB".

NAME	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	54c9d81cbb44	27 days ago	72.78 MB

- A imagem foi importada com sucesso
- Para verificar as imagens que temos, abra um terminal e digite o comando abaixo

docker image ls

docker images



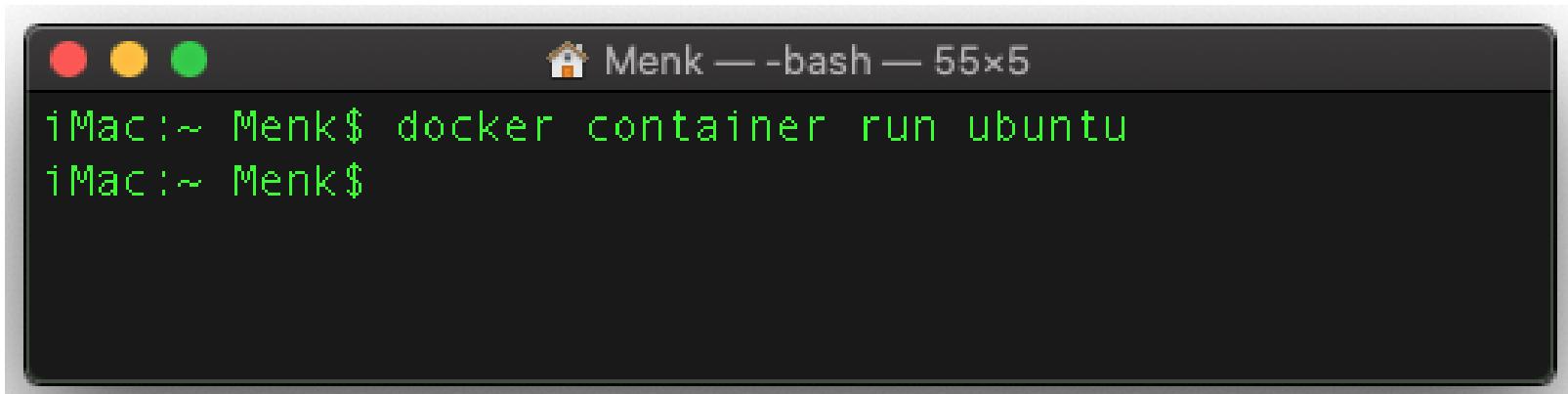
A screenshot of a macOS terminal window titled "Menk — -bash — 62x5". The window shows the command "docker image ls" being run, followed by a table of images. The table has columns: REPOSITORY, TAG, IMAGE ID, CREATED, and SIZE. One row is visible for the "ubuntu" repository, which has a TAG of "latest", an IMAGE ID of "54c9d81cbb44", was created "3 weeks ago", and is 72.8MB in size.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	54c9d81cbb44	3 weeks ago	72.8MB

Agora que temos a Imagem do Ubuntu podemos **criar** e **executar** um **Container** com base nessa Imagem com os comandos abaixo:

docker container run ubuntu

docker run ubuntu



A screenshot of a macOS terminal window titled "Menk — -bash — 55x5". The window has red, yellow, and green close buttons. The terminal prompt is "iMac:~ Menk\$". The user has entered the command "docker container run ubuntu" and pressed enter. The output shows the command being processed and then the prompt returning to "iMac:~ Menk\$".

```
iMac:~ Menk$ docker container run ubuntu
iMac:~ Menk$
```

- O Container foi executado? Porque não “fez nada”?
- O Container foi executado, porém nenhuma instrução adicional foi passada para a execução ou interação



Para podermos interagir com o Container temos a opção **-it**

Significa **Modo Interativo**, e com essa opção podemos ter interação com o Container através de um Terminal

docker container run ubuntu echo "Rodando Ubuntu"

docker container run -it ubuntu /bin/bash

```
[iMac:~ Menk$ docker container run -it ubuntu /bin/bash
[root@16e8d6c09cb7:/]# com.docker.cli - docker container run -it ubuntu /b...
[root@16e8d6c09cb7:/]# pwd
/
[root@16e8d6c09cb7:/]# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot etc  lib   lib64  media  opt  root  sbin  sys  usr
[root@16e8d6c09cb7:/]#
[root@16e8d6c09cb7:/]#
```



O comando **run** funciona mesmo que a imagem não tenha sido obtida previamente. Nas situações onde a imagem não é encontrada, ela é baixada e executada em seguida. Por exemplo, utilizar o comando abaixo (mesmo sem ter realizado o pull da imagem) faz com que a imagem do **Alpine** seja baixada, logo depois executada e liberada para acesso

docker container run -it alpine /bin/sh

```
[iMac:~ Menk$ docker container run -it alpine /bin/sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c158987b0551: Full complete
Digest: sha256:9b48fd548633d7ddeaa61b7ec2ed8280804f267e1dcfabff63aeecc13ba05094eb
Status: Downloaded newer image for alpine:latest
[/ # ls
bin      etc      lib      mnt      proc      run      srv      tmp      var
dev      home     media     opt      root      sbin     sys      usr
/ #
```

Comandos Docker

FIAP

Até o momento na Interface Gráfica...

The screenshot shows the Docker Desktop application window. The left sidebar has a red box around the 'Images' icon. The main area is titled 'Images' with a 'LOCAL' tab selected. It shows a table with two images: 'alpine' and 'ubuntu'. The 'alpine' row has a red box around it, and two arrows point from it to the commands: `docker container run -it alpine /bin/sh` and `docker pull ubuntu`.

	NAME	TAG	STATUS	CREATED	SIZE	ACTIONS
<input type="checkbox"/>	alpine 49176f190c7e	latest	In use	about 1 hour ago	7.05 MB	
<input type="checkbox"/>	ubuntu a8780b506fa4	latest	In use	20 days ago	77.79 MB	

The screenshot shows the Docker Desktop application window. The left sidebar has a red box around the 'Containers' icon. The main area is titled 'Containers' with a 'Only show running containers' toggle switch. It shows a table with three containers: 'charming_pike', 'strange_borg', and 'busy_tharp'. The first two rows have red boxes around them, and two arrows point from them to the commands: `docker container run ubuntu` and `docker container run -it ubuntu /bin/bash`. The third row also has an arrow pointing to the command `docker container run -it alpine /bin/sh`.

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	charming_pike 8fe720317d0e	ubuntu:latest	Exited			
<input type="checkbox"/>	strange_borg 16e8d6c09cb7	ubuntu:latest	Exited			
<input type="checkbox"/>	busy_tharp 5321f9ca0633	alpine:latest	Exited			

Visualização por linha de comando

```
[iMac:~ Menk$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
alpine      latest  49176f190c7e About an hour ago 7.05MB
ubuntu      latest  a8780b506fa4 2 weeks ago   77.8MB
[iMac:~ Menk$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[iMac:~ Menk$ docker container ls -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
5321f9ca0633 alpine "/bin/sh" 13 minutes ago Exited (0) 12 minutes ago busy_tharp
16e8d6c09cb7 ubuntu "/bin/bash" 18 minutes ago Exited (0) 16 minutes ago strange_borg
8fe720317d0e ubuntu "bash"    19 minutes ago Exited (0) 19 minutes ago charming_pike
[iMac:~ Menk$ iMac:~ Menk$
```

Comandos Docker

FIAP

Alguns comandos iniciais do Docker

docker info

Verificar as informações do nosso Docker Host

```
iMac:~ Menk$ docker info
Client:
  Context:    default
  Debug Mode: false
  Plugins:
    buildx: Docker Buildx (Docker Inc., v0.9.1)
    compose: Docker Compose (Docker Inc., v2.12.2)
    dev: Docker Dev Environments (Docker Inc., v0.0.3)
    extension: Manages Docker extensions (Docker Inc., v0.2.13)
    sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
    scan: Docker Scan (Docker Inc., v0.21.0)

Server:
  Containers: 3
    Running: 0
    Paused: 0
    Stopped: 3
  Images: 2
  Server Version: 20.10.21
```

docker version

Verificar a versão do Client

```
iMac:~ Menk$ docker version
Server: Docker Desktop 4.14.0 (91374)
Engine:
  Version:      20.10.21
  API version:  1.41 (minimum version 1.12)
  Go version:   go1.18.7
  Git commit:   3056208
  Built:        Tue Oct 25 18:00:19 2022
  OS/Arch:      linux/amd64
  Experimental: false
  containerd:
    Version:     1.6.9
    GitCommit:   1c90a442489720eec95342e1789ee8a5e1b9536f
  runc:
    Version:     1.1.4
    GitCommit:   v1.1.4-0-g5fd4c4d
  docker-init:
    Version:     0.19.0
    GitCommit:   de40ad0
iMac:~ Menk$
```

docker --help

Verificar todos os comandos que o Docker possui

```
iMac:~ Menk$ docker --help
Usage: docker [OPTIONS] COMMAND
A self-sufficient runtime for containers

Options:
  --config string       Location of client config files (default "/Users/Menk/.docker")
  -c, --context string  Name of the context to use to connect to the daemon (overrides
                        DOCKER_HOST env var and default context set with "docker
                        context use")
  -D, --debug           Enable debug mode
  -H, --host list        Daemon socket(s) to connect to
  -l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal")
                        (default "info")
  --tls                Use TLS; implied by --tlscacert
  --tlscacert string    Trust certs signed only by this CA (default
                        "/Users/Menk/.docker/ca.pem")
  --tlscert string      Path to TLS certificate file (default
                        "/Users/Menk/.docker/cert.pem")
```

docker search

Verificar imagens no Docker Hub

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	15255	[OK]	
websphere-liberty	WebSphere Liberty multi-architecture images ...	290	[OK]	
ubuntu-upstart	DEPRECATED, as is upstart (find other proces...	112	[OK]	
neurodebian	NeuroDebian provides neuroscience research s...	97	[OK]	
ubuntu/nginx	Nginx, a high-performance reverse proxy & we...	67		
open-liberty	Open Liberty multi-architecture images based...	56	[OK]	
ubuntu-debootstrap	DEPRECATED; use "ubuntu" instead	49	[OK]	
ubuntu/apache2	Apache, a secure & extensible open-source HT...	47		
ubuntu/squid	Squid is a caching proxy for the Web. Long-t...	44		
ubuntu/mysql	MySQL open source fast, stable, multi-thread...	38		
ubuntu/prometheus	Prometheus is a systems and service monitori...	33		
kasmweb/ubuntu-bionic-desktop	Ubuntu productivity desktop for Kasm Workspa...	33		
ubuntu/bind9	Bind 9 is a very flexible, full-featured DNS...	32		
ubuntu/postgres	PostgreSQL is an open source object-relational...	22		
ubuntu/kafka	Apache Kafka, a distributed event streaming ...	15		
ubuntu/redis	Redis, an open source key-value store. Long-t...	15		
ubuntu/prometheus-alertmanager	Alertmanager handles client alerts from Prom...	8		
ubuntu/grafana	Grafana, a feature rich metrics dashboard & ...	6		

HANDS ON: 1º EXERCÍCIO PRÁTICO UTILIZANDO



- Vamos explorar o conceito de Containers para ser aplicado muito breve na DimDim
- Como 1º desafio, vamos criar um container chamado `linux_mobile` que irá conter o Sistema Operacional  **CentOS**, utilizado por milhares de empresas no mundo corporativo
- Dentro desse poderoso Sistema Operacional, crie um diretório chamado `area_transf` e crie um arquivo texto chamado `primeiro_arq.txt`. Insira algumas linhas dentro desse arquivo
- Após isso, finalize a sua sessão externa, saia de seu Container e depois verifique o status atual que ele se encontra



HANDS ON: 1º EXERCÍCIO PRÁTICO UTILIZANDO



Primeiro abra um terminal no seu ambiente Host

E como 1º passo, busque uma imagem do  CentOS a partir do servidor de registro do Docker. No seu terminal digite:

docker pull centos

```
iMac:~ Menk$ docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Extracting [=====] 83.52MB/83a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbb9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
iMac:~ Menk$
```

docker image ls

```
iMac:~ Menk$ docker image ls
REPOSITORY    TAG      IMAGE ID      CREATED       SIZE
alpine        latest   49176f190c7e  2 hours ago   7.05MB
ubuntu         latest   a8780b506fa4  2 weeks ago   77.8MB
centos        latest   5d0da3dc9764  14 months ago  231MB
iMac:~ Menk$
```

HANDS ON: 1º EXERCÍCIO PRÁTICO UTILIZANDO



Agora já temos a imagem do CentOS disponibilizada para uso, digite o comando abaixo, que irá criar um Container chamado `linux_mobile`, incorporando o Timezone de São Paulo, dado que a necessidade é para o local em que estamos localizado. Perceba a velocidade de criação do serviço

`docker container run --name rmXXXX_linux_mobile -it -e TZ=America/Sao_Paulo centos /bin/bash`

date
pwd
whoami
ls

Por padrão o Docker não tem Standard Input, ou seja, ele não recebe os comando do Terminal sem que seja especificado que ele receba, por isso, para obter um Shell dentro do container indicamos o caminho dele. Em nosso exemplo temos uma imagem Centos, o terminal do Centos é o bash, então `/bin/bash`

Utilizaremos a opção `--name` para especificar o nome desejado ao Container

Utilizaremos as opções `-it` para especificarmos que queremos que ele execute em modo Interativo

Utilizaremos a opção `-e` para especificar uma variável de ambiente com o nome `TZ` e valor `America/Sao_Paulo`

```
iMac:~ Menk$ docker container run --name linux_mobile -it -e TZ=America/Sao_Paulo centos /bin/bash...
[[root@aa92d134556b ~]# date
[Tue Nov 22 21:11:48 -03 2022]
[[root@aa92d134556b ~]# pwd
/
[[root@aa92d134556b ~]# whoami
root
[[root@aa92d134556b ~]# ls
bin  etc  lib    lost+found  mnt  proc  run    srv  tmp  var
dev  home lib64  media       opt  root  sbin  sys  usr
[root@aa92d134556b ~]#
```

HANDS ON: 1º EXERCÍCIO PRÁTICO UTILIZANDO



O próximo passo será criar um diretório chamado `arq_transf` e também criar um arquivo com conteúdo chamado `primeiro_arq.txt`. Vamos realizar essas tarefas:

CTRL + L = Limpa a tela

mkdir arq_transf

cd arq_transf

touch primeiro_arq.txt

vi primeiro_arq.txt

exit

```
Menk — @aa92d134556b:/arq_transf — -bash — 67x19
[[root@aa92d134556b ~]# mkdir arq_transf
[[root@aa92d134556b ~]#
[[root@aa92d134556b ~]# cd arq_transf
[[root@aa92d134556b arq_transf]# 
[[root@aa92d134556b arq_transf]# touch primeiro_arq.txt
[[root@aa92d134556b arq_transf]# 
[[root@aa92d134556b arq_transf]# vi primeiro_arq.txt
[[root@aa92d134556b arq_transf]# 
[[root@aa92d134556b arq_transf]# cat primeiro_arq.txt
Esse é nosso primeiro arquivo texto criado dentro de um Container

Muito TOP

Obrigado Universo pelo conhecimento!
Somos Todos Um!

[[root@aa92d134556b arq_transf]# exit
exit
iMac:~ Menk$
```

HANDS ON: 1º EXERCÍCIO PRÁTICO UTILIZANDO



O container fica disponível para uso sempre que necessário

Docker Desktop Update to latest

Containers Give feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Only show running containers

Search

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	charming_pike 8fe720317d0e	ubuntu:late	Exited			▶ ⋮ ℹ️
<input type="checkbox"/>	strange_borg 16e8d6c09cb7	ubuntu:late	Exited			▶ ⋮ ℹ️
<input type="checkbox"/>	busy_tharp 5321f9ca0633	alpine:late	Exited			▶ ⋮ ℹ️
<input type="checkbox"/>	linux_mobile aa92d134556b	centos:late	Exited			▶ ⋮ ℹ️

`docker container start linux_mobile -i`

HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



- Como 2º desafio, vamos criar um Container chamado `bd_portalweb` que irá conter o famoso banco de dados  , também utilizado por milhares de empresas no mundo corporativo
- Dentro desse poderoso SGBD, vamos criar o Banco de Dados `BD_FIAP`, depois um usuário `FIAP`, criar uma tabela nesse novo Banco de Dados e inserir algumas linhas dentro dessa tabela
- Após isso, finalize a sua sessão externa, saia de seu Container e depois verifique no Docker o status atual que ele se encontra



HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Pensando em fazer de uma maneira mais otimizada, vamos solicitar que seja recuperada a imagem mais atual do MySQL. Para isso digite o comando abaixo, que irá criar um Container chamado `bd_portalweb`

```
docker container run --name bd_portalweb -e MYSQL_ROOT_PASSWORD=@Fiap2tds2023 -d mysql/mysql-server:latest
```

Utilizaremos a opção `--name` para especificar o nome desejado ao Container

Utilizaremos a opção `-e` para especificar uma variável de ambiente com o nome `MYSQL_ROOT_PASSWORD` e valor `@Fiap2tds2023`

Utilizaremos as opções `-d` para especificarmos que queremos que ele execute esse comando em modo Detached (Segundo Plano)

```
[iMac:~ Menk$ docker container run --name bd_portalweb -e MYSQL_ROOT_PASSWORD=@Fiap2tds2023 -d mysql/mysql-server:latest
Unable to find image 'mysql/mysql-server:latest' locally
latest: Pulling from mysql/mysql-server
134439bbc243: Pull complete
24197d57c06a: Pull complete
a8ff14042390: Pull complete
209d472e303b: Pull complete
4158d94acc40: Pull complete
807107bf7d7a: Pull complete
5f5d5a703fe0: Pull complete
Digest: sha256:1b2005199e9dc12d88d5950cd738dfd12172b1224675294646ea9d6031c78408
Status: Downloaded newer image for mysql/mysql-server:latest
417821efc8898ffeb2634e7510a4328a35c3f0ae26cf518d59792225a9b1a6b9
iMac:~ Menk$
```

HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Resultado no Terminal

```
iMac:~ Menk$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
417821efc889 mysql/mysql-server:latest "/entrypoint.sh mysq..." 10 minutes ago Up 9 minutes (healthy) 3306/tcp, 33060-33061/tcp bd_portalweb
iMac:~ Menk$
```

Resultado no Docker Desktop

The screenshot shows the Docker Desktop interface on a Mac OS X desktop. The sidebar on the left includes 'Containers', 'Images', 'Volumes', 'Dev Environments (BETA)', 'Extensions (BETA)', and an 'Add Extensions' button. The main area is titled 'Containers' with a sub-instruction: 'A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another.' Below this is a search bar and a toggle switch for 'Only show running containers'. The table lists five containers:

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	linux_mobile	centos:late	Exited			
<input type="checkbox"/>	busy_tharp	alpine:late	Exited			
<input type="checkbox"/>	strange_borg	ubuntu:late	Exited			
<input type="checkbox"/>	charming_pike	ubuntu:late	Exited			
<input type="checkbox"/>	bd_portalweb	mysql/mys...	Running		12 minutes ago	

At the bottom, status indicators show 'RAM 1.14GB', 'CPU 3.23%', 'Not connected to Hub', and 'v4.14.1'. A pink footer bar contains the number '55'.

HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Agora vamos executar o container `bd_portalweb` e solicitar acesso como root do MySQL. Execute os comandos abaixo e informe a senha do root: `@Fiap2tds2023`



`docker container exec -it bd_portalweb bash`

```
[iMac:~ Menk$ docker container exec -it bd_portalweb bash
bash-4.4#
```

`mysql -u root -p`

```
[bash-4.4# mysql -u root -p
[Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 52
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```



HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Com o usuário root conectado, vamos criar o banco de dados `bd_fiap`, o usuário `fiap` e os devidos privilégios

```
select @@version;
```

```
create database bdःfiap;
```

```
create user 'fiap'@'localhost' IDENTIFIED BY '@Fiap2tds2023';
```

```
grant all on bdःfiap.* TO 'fiap'@'localhost';
```

```
flush privileges;
```

```
show databases;
```



```
[mysql> select @@version;
+-----+
| @@version |
+-----+
| 8.0.31   |
+-----+
1 row in set (0.00 sec)

[mysql> create database bdःfiap;
Query OK, 1 row affected (0.01 sec)

[mysql> create user 'fiap'@'localhost' IDENTIFIED BY '@Fiap2tds2023';
Query OK, 0 rows affected (0.01 sec)

[mysql> grant all on bdःfiap.* TO 'fiap'@'localhost';
Query OK, 0 rows affected, 1 warning (0.01 sec)

[mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)

[mysql> show databases;
+-----+
| Database      |
+-----+
| bdःfiap       |
| information_schema |
| mysql          |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.00 sec)

[mysql>
```

HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Com tudo configurado, vamos realizar o login com o novo usuário e criar os objetos no Banco

exit;

mysql -u fiap -p Senha: @Fiap2tds2023

use bdfiap;

create table t_aluno (rm_aluno varchar(07), nm_aluno varchar(80), dt_nascimento date);

insert into t_aluno (rm_aluno, nm_aluno, dt_nascimento) values ("RM3344", "Maria Linda", "2000-03-19");

select * from t_aluno;

exit;

```
Menk — @aa92d134556b:/ — com.docker.cli + docker container exec -it bd_portalweb bash — 111x37
mysql> exit
Bye
bash-4.4#
bash-4.4# mysql -u fiap -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use bdfiap;
Database changed
mysql>
mysql> create table t_aluno ( rm_aluno varchar(07), nm_aluno varchar(80), dt_nascimento date );
Query OK, 0 rows affected (1.26 sec)

mysql> insert into t_aluno (rm_aluno, nm_aluno, dt_nascimento) values ("RM3344", "Maria Linda", "2000-03-19");
Query OK, 1 row affected (0.01 sec)

mysql> select * from t_aluno;
+-----+-----+-----+
| rm_aluno | nm_aluno | dt_nascimento |
+-----+-----+-----+
| RM3344   | Maria Linda | 2000-03-19  |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> exit
Bye
bash-4.4#
```

HANDS ON: 2º EXERCÍCIO PRÁTICO UTILIZANDO



Verifique os Containers criados. Sucesso!

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	linux_mobile aa92d134556b	centos:late	Exited			
<input type="checkbox"/>	busy_tharp 5321f9ca0633	alpine:late:	Exited			
<input type="checkbox"/>	strange_borg 16e8d6c09cb7	ubuntu:late	Exited			
<input type="checkbox"/>	charming_pike 8fe720317d0e	ubuntu:late	Exited			
<input type="checkbox"/>	bd_portalweb 03e178cbe9fe	mysql/mys	Running		21 minutes ago	

**Para entrar novamente no Container
docker container exec -it bd_portalweb bash**

**Parando o Container
docker container stop bd_portalweb**



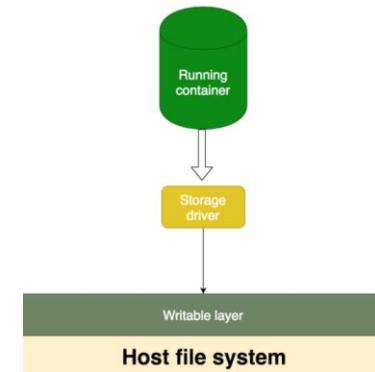
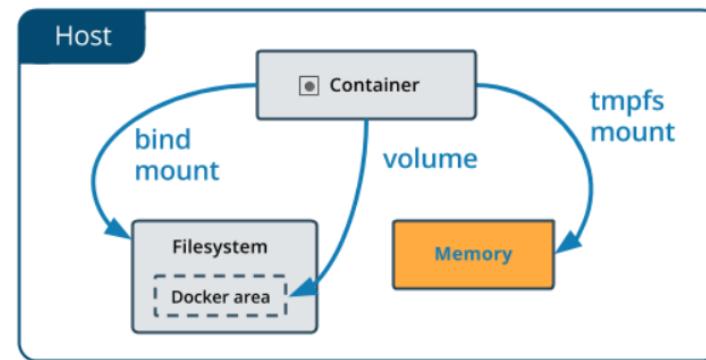
**docker container pause bd_portalweb
docker container unpause bd_portalweb**

HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



- Como 3º desafio, vamos aprender como criar uma área que pode ser compartilhada entre diversos Containers, facilitando a transferência e uso de diversos arquivos gerados pelas aplicações
- Volumes são localizações físicas disponibilizadas no Host Docker e podem ser do tipo: Anônimo ou Nomeado. Nesse nosso exemplo prático, vamos praticar o conceito de **Volume Nomeado**
- O Volume Anônimo recebe um ID para identificação e deve ser utilizado com critério
- O Volume Nomeado permite criar um nome significativo e o Docker é quem gerencia o local em que o volume será criado (indicado para ambientes produtivos)

Vamos praticar!



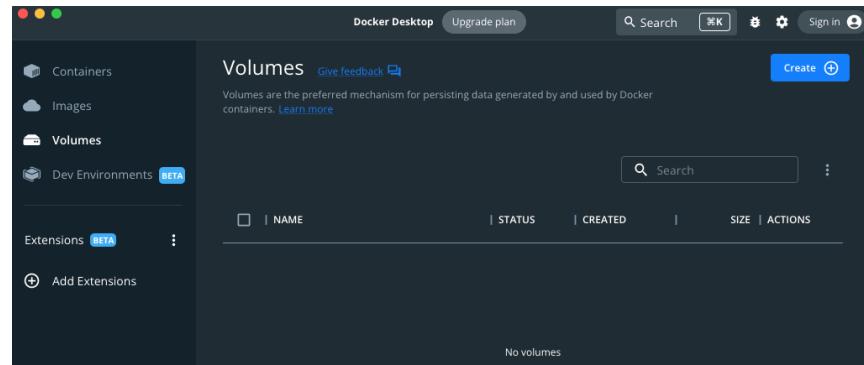
HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



Veja que ainda não temos um volume criado para compartilhar arquivos

```
iMac:~ Menk$ docker volume ls
DRIVER      VOLUME NAME
iMac:~ Menk$
```

docker volume ls



Vamos criar nosso volume compartilhado chamado **logdata**

docker volume create logdata

```
iMac:~ Menk$ docker volume create logdata
logdata
iMac:~ Menk$
```

HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



Agora temos o volume **logdata** disponível

Docker Desktop interface showing the Volumes section. A single volume named 'logdata' is listed, created 2 minutes ago, with a size of 8 kB.

```
[iMac:~ Menk$ docker volume ls
DRIVER      VOLUME NAME
local      logdata
iMac:~ Menk$ ]
```

docker volume ls

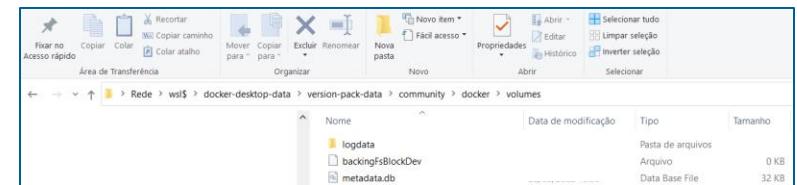
Por default, o Docker **Windows** disponibiliza acesso na seguinte localização:

Docker Engine v19:

`\wsl$\docker-desktop-data\version-pack-data\community\docker\Volumes\`

Docker Engine v20:

`\wsl$\docker-desktop-data\data\docker\Volumes`



Por default, o Docker **Mac* / Linux** disponibiliza acesso na seguinte localização:

`/var/lib/docker/volumes`

```
[iMac:~ Menk$ nc -U ~/Library/Containers/com.docker.docker/Data/debug-shell.sock - 81x8
[iMac:~ Menk$ nc -U ~/Library/Containers/com.docker.docker/Data/debug-shell.sock
~/var/lib/docker/volumes # pwd
~/var/lib/docker/volumes
~/var/lib/docker/volumes # ls
[ls
backingFsBlockDev  logdata
~/var/lib/docker/volumes # metadata.db
[metadata.db
~/var/lib/docker/volumes # ]]
```

HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



Agora vamos criar um container chamado `apl_web_dimdim` e vamos associar o volume `logdata` no Host para o Container com a opção: `-v`

Host Container



`docker container run -it --name apl_web_dimdim -v logdata:/volumedimdim alpine`

```
Menk — @aa92d134556b:/ — com.docker.cli • docker container run -it --name apl_web_dimdim -v logdata:/volumedimdim alpine — 91x6
iMac:~ Menk$ docker container run -it --name apl_web_dimdim -v logdata:/volumedimdim alpine
/ #
```

Digite o comando `ls -l` e veja que o diretório `volumedimdim` se encontra disponível no Container

```
[/ # ls -l
total 60
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 bin
drwxr-xr-x  5 root    root        360 Nov 24 00:17 dev
drwxr-xr-x  1 root    root        4096 Nov 24 00:17 etc
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 home
drwxr-xr-x  7 root    root        4096 Nov 22 13:06 lib
drwxr-xr-x  5 root    root        4096 Nov 22 13:06 media
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 mnt
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 opt
dr-xr-xr-x 191 root    root          0 Nov 24 00:17 proc
drwxr----- 1 root    root        4096 Nov 24 00:18 root
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 run
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 sbin
drwxr-xr-x  2 root    root        4096 Nov 22 13:06 srv
dr-xr-xr-x 13 root    root          0 Nov 24 00:17 sys
drwxrwxrwt  2 root    root        4096 Nov 22 13:06 tmp
drwxr-xr-x  7 root    root        4096 Nov 22 13:06 usr
drwxr-xr-x 12 root    root        4096 Nov 22 13:06 var
drwxr-xr-x  2 root    root        4096 Nov 24 00:03 volumedimdim
/ #
```

HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



O próximo passo é criar um arquivo dentro do Container e depois ver se é possível visualizar no Host

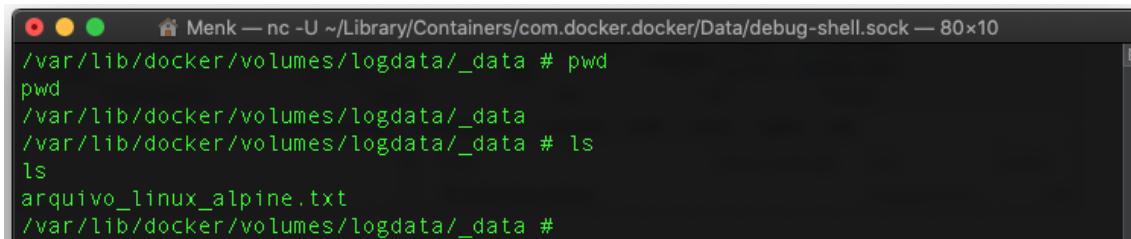
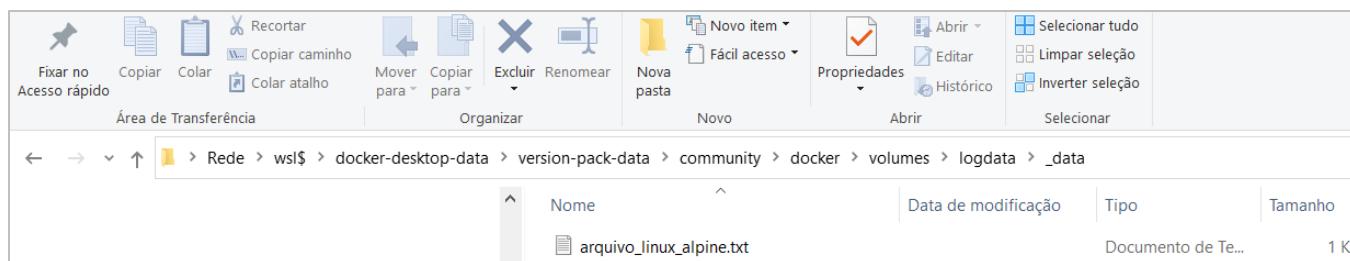


```
Menk — @aa92d134556b:/ — com.docker.cli • docker container run -it --name api_web.dimdim ...
/volumedimdim
/volumedimdim #
/volumedimdim # pwd
/volumedimdim
/volumedimdim #
/volumedimdim # echo Ola do Container > arquivo_linux_alpine.txt
/volumedimdim #
/volumedimdim # ls
arquivo_linux_alpine.txt
/volumedimdim #
/volumedimdim #
```

cd volumedimdim

echo Ola do Container > arquivo_linux_alpine.txt

Veja que o arquivo aparece disponível no Volume do Docker Host

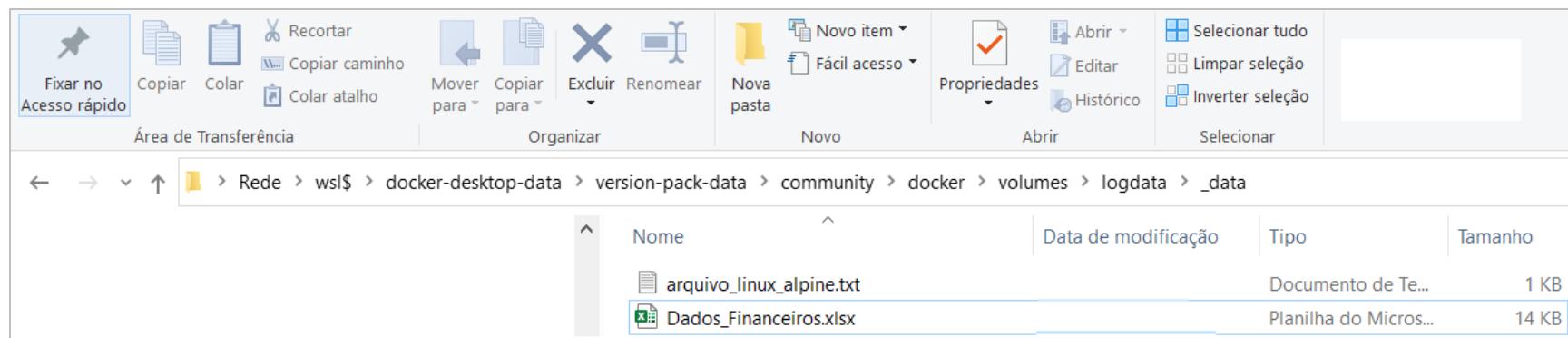


```
Menk — nc -U ~/Library/Containers/com.docker.docker/Data/debug-shell.sock — 80x10
/var/lib/docker/volumes/logdata/_data # pwd
pwd
/var/lib/docker/volumes/logdata/_data
/var/lib/docker/volumes/logdata/_data # ls
ls
arquivo_linux_alpine.txt
/var/lib/docker/volumes/logdata/_data #
```

HANDS ON: 3º EXERCÍCIO PRÁTICO UTILIZANDO



Agora é no sentido contrário. Crie um arquivo no Host e verifique se é possível visualiza-lo no Container



Veja que o arquivo aparece disponível também no Container. Top!

The screenshot shows a terminal window titled "Prompt de Comando - docker run -it --name apl_web_dimdim -v logdata:/volumedimdim alpine". The output of the "ls -l" command is displayed:

```
/volumedimdim # ls -l
total 4
-rw-r--r--    1 root      root           112              arquivo_linux_alpine.txt
/volumedimdim # ls -l
total 20
-rw-r--r--    1 root      root        14259             Dados_Financeiros.xlsx
-rw-r--r--    1 root      root           112              arquivo_linux_alpine.txt
/volumedimdim #
```

- Podemos obter informações específicas e detalhadas sobre um Container utilizando o comando **inspect <nome do container>**
- Esse comando é muito útil para por exemplo exibir o IP do Container, o Mac Address, as portas disponíveis etc

docker container inspect linux_mobile

```
iMac:~ Menk$ docker container inspect linux_mobile
[{"Id": "64158c721d200cef7d950efad12b6b8d476b21ab8de098396a6ff58085c9a39e",
 "Created": "2022-03-19T04:45:35.175871031Z",
 "Path": "/bin/bash",
 "Args": [],
 "State": {
     "Status": "exited",
     "Running": false,
     "Paused": false,
     "Restarting": false,
     "OOMKilled": false,
     "Dead": false,
     "Pid": 0,
     "ExitCode": 1,
     "Error": ""}]
```

Para interromper um container em execução utilizamos o comando **stop** passando como parâmetro o **ID** ou **NAME** do container

Utilize outra janela do Terminal

docker container stop 4de4

Não é necessário informar todo o número do container_id

```
[iMac:~ Menk$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS     NAMES
4de467012ac9        alpine      "/bin/sh"    37 hours ago   Up  5 minutes
iMac:~ Menk$
```

docker container stop apl_web_dimdim

Uma vez que o container já foi executado e parado podemos utilizar o comando **start** para executar novamente

docker container start apl_web_dimdim

```
[iMac:~ Menk$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
[iMac:~ Menk$ docker container start apl_web_dimdim
apl_web_dimdim
[iMac:~ Menk$ docker ps
CONTAINER ID   IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
4de467012ac9   alpine    "/bin/sh"    37 hours ago  Up 2 seconds          apl_web_dimdim
iMac:~ Menk$ ]
```

Também podemos utilizar o comando **restart** para reiniciar a execução de um Container

docker container restart apl_web_dimdim

Para remover completamente um container utilize o comando **rm**

docker container ls -a

```
iMac:~ Menk$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
4de467012ac9        alpine              "/bin/sh"          37 hours ago       Up 3 minutes
03e178cbe9fe        mysql/mysql-server:latest    "/entrypoint.sh mysq..."   38 hours ago       Exited (255) 55 minutes ago   3306/tcp, 33060-33061/tcp
aa92d134556b        centos              "/bin/bash"         2 days ago        Exited (0) 39 hours ago
5321f9ca0633        alpine              "/bin/sh"          2 days ago        Exited (0) 2 days ago
16e8d6c09cb7        ubuntu              "/bin/bash"         2 days ago        Exited (0) 2 days ago
8fe720317d0e        ubuntu              "bash"              2 days ago        Exited (0) 2 days ago
iMac:~ Menk$
```

docker container rm linux_mobile

```
iMac:~ Menk$ docker container rm linux_mobile
linux_mobile
iMac:~ Menk$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
4de467012ac9        alpine              "/bin/sh"          37 hours ago       Up 6 minutes
03e178cbe9fe        mysql/mysql-server:latest    "/entrypoint.sh mysq..."   38 hours ago       Exited (255) 58 minutes ago   3306/tcp, 33060-33061/tcp
5321f9ca0633        alpine              "/bin/sh"          2 days ago        Exited (0) 2 days ago
16e8d6c09cb7        ubuntu              "/bin/bash"         2 days ago        Exited (0) 2 days ago
8fe720317d0e        ubuntu              "bash"              2 days ago        Exited (0) 2 days ago
iMac:~ Menk$
```

O Container não pode estar em execução para ser removido

Inclua a opção **-f** para eliminar um Container que está em execução

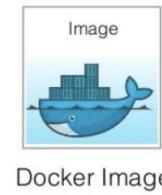
A opção `rm` pode ser usada em outros objetos do Docker como Imagens e Volumes também. Vamos aprender uma forma simples de limpar os nossos recursos criados no Docker

O comando abaixo encerra a execução de todos os Containers no Host
`docker container stop $(docker container ls -q)`

O comando abaixo remove todos os Containers do Host
`docker container rm $(docker container ls -aq)`



O comando abaixo remove todas as Imagens no Host
`docker image rm $(docker image ls -aq)`



O comando abaixo remove todas os Volumes no Host
`docker volume rm $(docker volume ls -q)`



O comando abaixo remove:

- ✓ Todos os Containers parados
- ✓ Todas as Redes não usadas
- ✓ Todos os Volumes não usados
- ✓ Todas as Imagens sem nenhum Container associado
- ✓ Todo o Cache Build pendente
- ✓ Não pede confirmação



docker system prune -a -f --volumes

```
iMac:~ Menk$ docker system prune --all --force --volumes
Deleted build cache objects:
puu9ykhgum7fmd4al8olfenct
t70qo51dz462l28jk50owlztm

Total reclaimed space: 0B
iMac:~ Menk$
```



Rodar um Container Servidor Nginx e alterar a página inicial

1. Crie um Volume Nomeado para compartilhar dados entre o Host e o Container. Nome: **nginx-data**. Ao iniciar o Container, mapeie o Volume criado com a pasta da Imagem: `/usr/share/nginx/html`
2. Exponha a porta 80 do Container para podermos ter acesso ao Servidor Nginx
3. Altere os arquivos abaixo localizados na pasta compartilhada da forma que desejar, e realize os testes rodando o Container do Servidor Nginx

50x.html

index.html

`docker system prune -a -f --volumes`



Copyright © 2023 Prof. João Carlos Menk e Prof. Salvio Padlipskas

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).