



Fundação Universidade Federal de Rondônia - UNIR
Departamento Acadêmico de Ciência da Computação - DACC
Bacharelado em Ciência da Computação

Sistemas Distribuídos

DOCKER

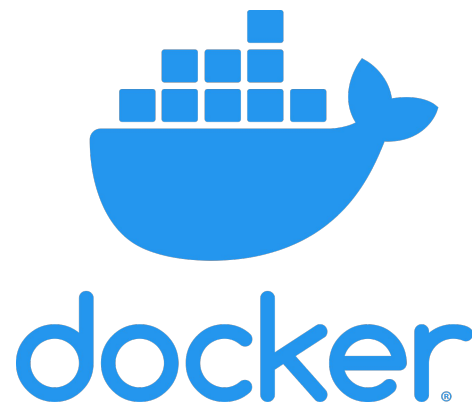
Docente: Dr. Marcelo

Bruno Bordin
Gabriel Russo
Marcus Martins

Porto Velho 2022

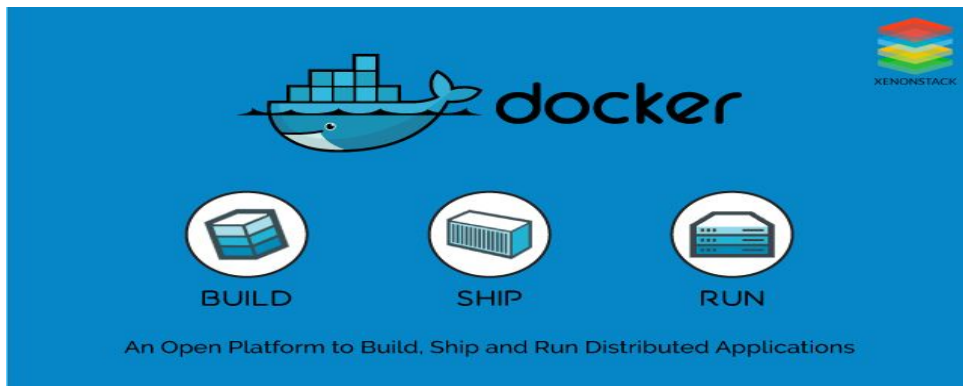
O que é o Docker ?

Docker é uma plataforma de containerização de código aberto. Ele permite que os desenvolvedores empacotam aplicativos em contêineres — componentes executáveis padronizados que combinam o código-fonte do aplicativo com as bibliotecas do sistema operacional (SO) e as dependências necessárias para executar esse código em qualquer ambiente.



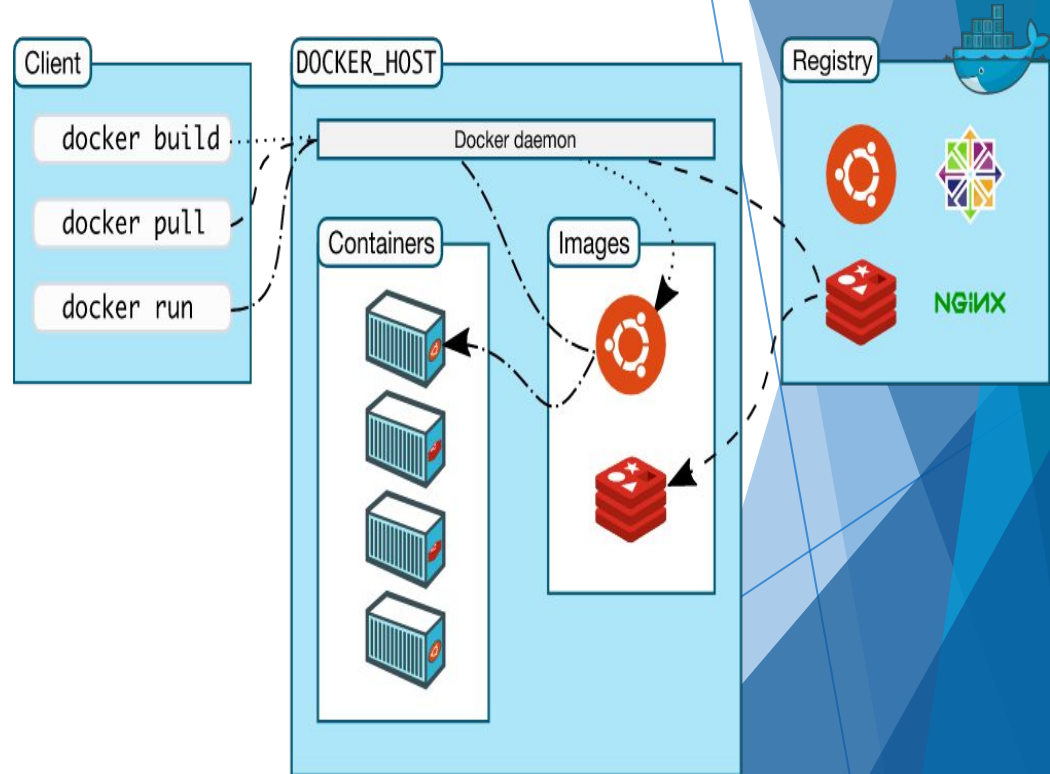
A Plataforma Docker

O Docker fornece a capacidade de empacotar e executar um aplicativo em um ambiente levemente isolado chamado contêiner. O isolamento e a segurança permitem que você execute vários contêineres simultaneamente em um determinado host. Os contêineres são leves e contém tudo o que é necessário para executar o aplicativo, portanto, você não precisa depender do que está instalado atualmente no host



Arquitetura do Docker

O Docker usa uma arquitetura cliente-servidor. O cliente Docker conversa com o daemon do Docker, que faz o trabalho pesado. O cliente e o daemon do Docker podem ser executados no mesmo sistema ou conectar a um daemon remoto do Docker. O cliente Docker e o daemon se comunicam usando uma API REST, em soquetes UNIX ou uma interface de rede.



Docker Daemon

O daemon do Docker (dockerd) escuta as solicitações da API do Docker e gerencia objetos do Docker, como imagens, contêineres, redes e volumes. Um daemon também pode se comunicar com outros daemons para gerenciar os serviços do Docker.

Imagem

Uma imagem do Docker é um modelo read-only que contém um conjunto de instruções para criar um contêiner que pode ser executado na plataforma Docker. Ele fornece uma maneira conveniente de empacotar aplicativos e ambientes de servidor pré-configurados, que você pode usar para seu próprio uso privado ou compartilhar publicamente com outros usuários do Docker

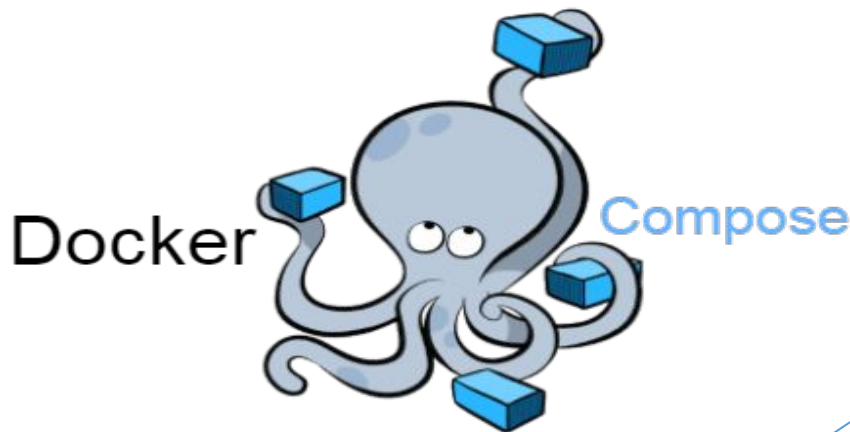
Container

Um contêiner é uma instância executável de uma imagem. Você pode criar, iniciar, parar, mover ou excluir um contêiner usando a API ou CLI do Docker. Você pode conectar um contêiner a uma ou mais redes, anexar armazenamento a ele ou até mesmo criar uma nova imagem com base em seu estado atual.

Por padrão, um contêiner é relativamente bem isolado de outros contêineres e de sua máquina host.

Docker Compose

O Docker Compose é uma ferramenta desenvolvida para ajudar a definir e compartilhar aplicativos de vários contêineres. Com o Compose, podemos criar um arquivo YAML para definir os serviços e com um único comando, podemos girar tudo para cima ou para baixo.



Containers x VMs

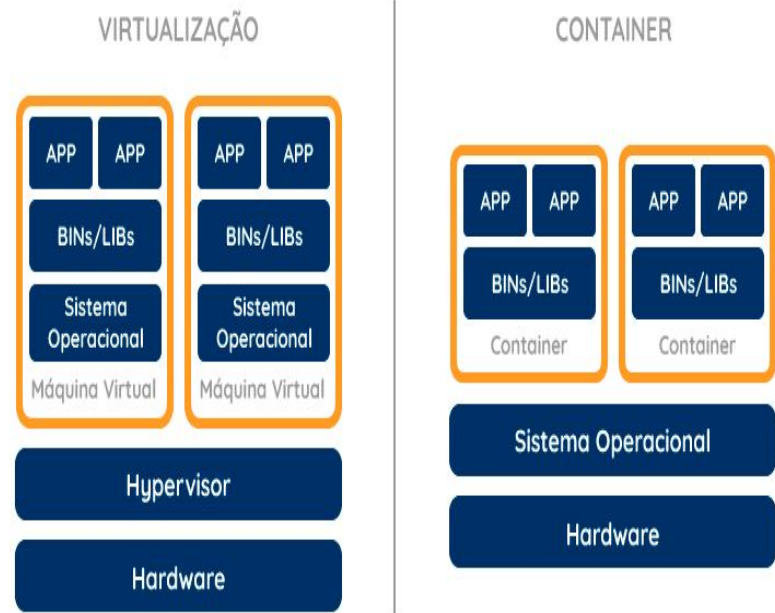
A virtualização permite o isolamento total do ambiente da sua aplicação, já que ela virtualiza a máquina por completo.

O container, compartilha um mesmo kernel do sistema operacional, traz apenas isolamento parcial.

A virtualização dá garantia de recursos para sua aplicação em nível de hardware.

O container possui a dependência do Sistema operacional que ele está rodando.

Os containers necessitam de muito menos recursos. Este fator também implica em maior rapidez de resposta na inicialização de um container.



Algumas Vantagens do Docker

Ágil implantação: os containers podem ser transportados de forma simples entre os diversos ambientes

Maior disponibilidade do sistema: mais espaço livre, uma vez que há o compartilhamento do sistema operacional e de outros componentes

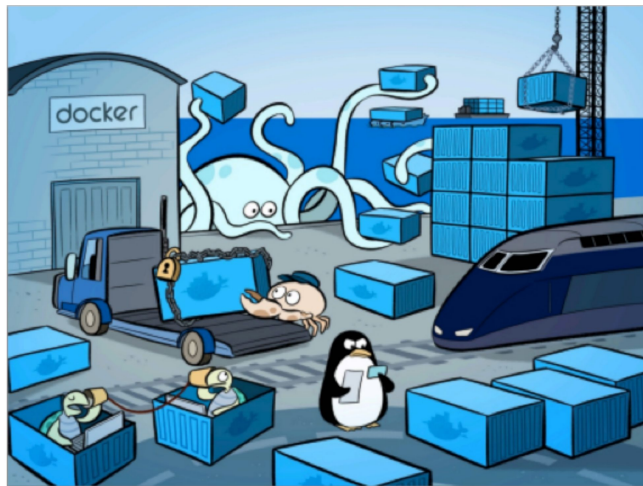
Reversibilidade dos containers: caso as mudanças efetuadas não saírem como o esperado, é só retornar o container para sua versão prévia.

Abordagem baseada em microsserviços: a possibilidade de interromper somente uma parte do software em execução.

Camadas de controle para as versões de imagem: cada imagem Docker é composta por um conjunto de camadas. Sempre que há modificações nesse arquivo, uma nova camada é criada.


Docker na prática

Docker Hub, Docker CLI, Dockerfile e Docker compose



Docker hub

www.hub.docker.com



[Explore](#) [Pricing](#) [Sign In](#) [Sign Up](#)

[Docker](#) [Containers](#) [Plugins](#)

Filters

Images

☐ Verified Publisher ⓘ


☐ Official Images ⓘ
Official Images Published By Docker


Categories ⓘ


- ☐ Analytics
- ☐ Application Frameworks
- ☐ Application Infrastructure
- ☐ Application Services
- ☐ Base Images
- ☐ Databases
- ☐ DevOps Tools
- ☐ Featured Images
- ☐ Messaging Services
- ☐ Monitoring
- ☐ Operating Systems
- ☐ Programming Languages
- ☐ Security
- ☐ Storage

1 - 25 of 8,841,483 available Images.


Suggested

**ubuntu** Official Image
Updated 16 days ago
1B+ Downloads 10K+ Stars
Ubuntu is a Debian-based Linux operating system based on free software.
Container Linux x86-64 ARM ARM 64 PowerPC 64 LE riscv64 IBM Z 386 Base Images Operating Systems

**redis** Official Image
Updated 13 days ago
1B+ Downloads 10K+ Stars
Redis is an open source key-value store that functions as a data structure server.
Container Windows Linux ARM 64 IBM Z PowerPC 64 LE ARM 386 mips64le x86-64 Databases


**node** Official Image
Updated 7 days ago
1B+ Downloads 10K+ Stars
Node.js is a JavaScript-based platform for server-side and networking applications.
Container Linux IBM Z 386 x86-64 ARM ARM 64 PowerPC 64 LE Application Infrastructure

Docker hub - Tags

 dockerhub

Explore Pricing Sign In [Sign Up](#)

Explore Official Images python

 **python** Official Image ☆

Python is an interpreted, interactive, object-oriented, open-source programming language.

1B+

Container Windows Linux 386 PowerPC 64 LE IBM Z mips64le x86-64 ARM ARM 64

Programming Languages Official Image

Copy and paste to pull this image

`docker pull python`

[View Available Tags](#)

Description Reviews **Tags**

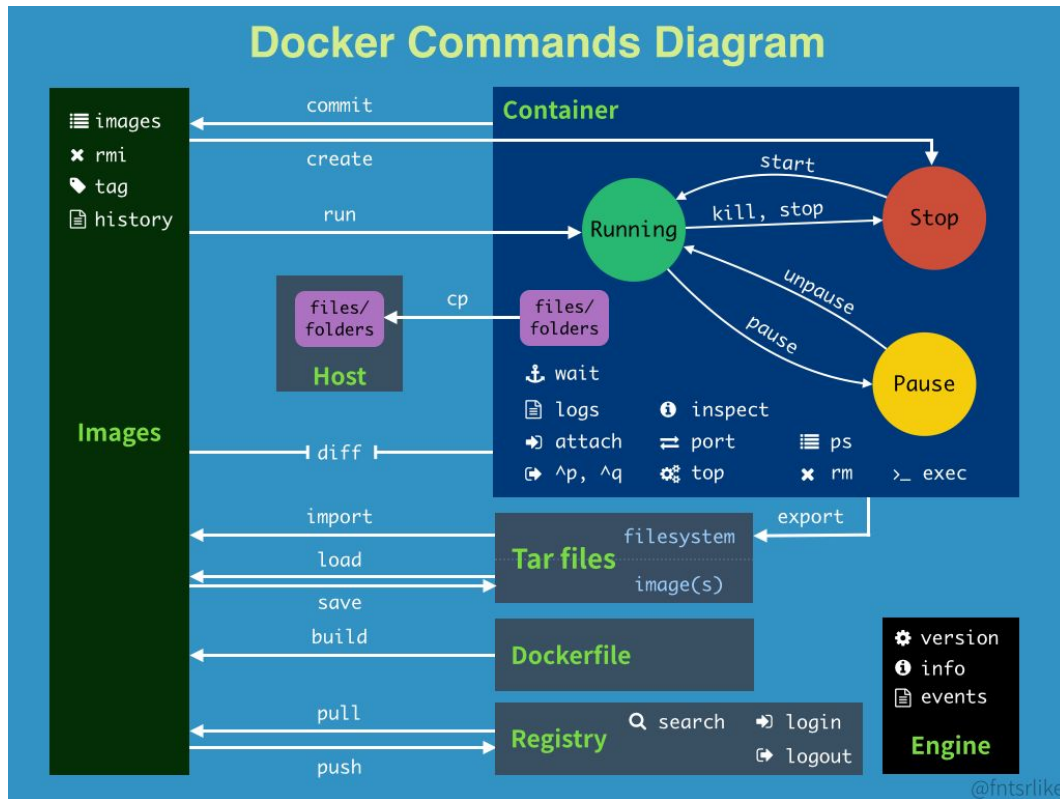
TAG

Sort by Newest

TAG	OS/ARCH	COMPRESSED SIZE
latest Log4Shell CVE not detected Last pushed 8 days ago by dojanky		
d6bfb1542f21	linux/386	338.85 MB
d9bc15281f70	windows/amd64	2.58 GB
3f7e5127646a	windows/amd64	2.11 GB
cd150f52893e	linux/amd64	333.65 MB
076fca909d16	linux/arm/v5	306.89 MB
ebaf27334287	linux/arm/v7	294.31 MB
519ebab04dda	linux/arm64/v8	324.98 MB
cd66dad7d301	linux/ppc64le	342.54 MB
17489f149132	linux/s390x	308.2 MB

`docker pull python:latest`

Docker Command Line Interface



Usando o docker na mão!

Atenção!: O seu usuário precisa estar no grupo docker, caso contrário, apenas o administrador/root pode usar a CLI

Obs: o til (~) não pertence ao comando

Estrutura da docker CLI :

\$ docker [Opções] comando [Argumentos...]

Opções úteis:

-D ou **--debug** ~ Ativa o modo debug

-l ou **--log-level** ~ Muda o nível de log ("debug"|"info"|"warn"|"error"|"fatal")
(default "info")

Comandos: pull e images

O comando **pull** faz o download de uma imagem do docker hub.

O comando **images** lista as imagens baixadas.

Estrutura do comando:

`docker pull [Opções] <nome da imagem>:[tag ou @digest]`

`docker images [Opções]`

Exemplo:

```
[gabriel@void ~]$ docker images
REPOSITORY    TAG                IMAGE ID           CREATED            SIZE
php            8.1-apache        a3e1df81efc8      3 weeks ago       477MB
postgres      latest            e94a3bb61224      2 months ago      374MB
[gabriel@void ~]$ docker pull python:latest
```


Comando: run

O comando **run** cria um novo container.

Estrutura do comando:

`docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`

Opções úteis:

-i -t ~ mantem o STDIN do container aberto e abre um TTY

--name [nome] ~ nome do novo container

-p [porta do host]:[porta do container] ~ Def. Porta host -> container

Exemplo:

```
[gabriel@void ~]$ docker run -it --name exemplo_pgsql -p 5432:5432 postgres:12.10-alpine
```

Comando: ps

O comando **ps** lista os containers em background.

Estrutura do comando:

docker ps [OPTIONS]

Opções úteis:

-a ou **--all** ~ Exibe todos os containers, tanto “Up” quanto “Exited”

Exemplo:

```
[gabriel@void ~]$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
[gabriel@void ~]$ docker ps --all
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
9ece6fb235c4   postgres:latest  "docker-entrypoint.s..."  2 months ago  Exited (0) 5 minutes ago    bd aula
```

Obs: O comando : docker container ls -a faz a mesma coisa!

Comando: start e stop

O comando **start** inicia um container parado.

O comando **stop** para um container em execução.

Estrutura do comando:

`docker start [Opções] nome container ou id container`

`docker stop [Opções] nome container ou id container`

Exemplo:

```
[gabriel@void ~ ]$ docker ps --all
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
9ece6fb235c4   postgres:latest "docker-entrypoint.s..." 2 months ago   Exited (0) 45 minutes ago          bd_aula
[gabriel@void ~ ]$ docker start bd_aula
```

Action	Shortcut
Start	Ctrl+Shift+S

```
[gabriel@void ~ ]$ docker start 9ece6fb235c4
```

Comando: rm e rmi

O comando **rm** apaga um container.

O comando **rmi** apaga uma imagem.

Estrutura do comando:

`docker rm [Opções] nome container ou id container`

`docker rmi [Opções] nome container ou id container`

Exemplo:

```
[gabriel@void ~]$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
php 8.1-apache a3e1df81efc8 3 weeks ago 477MB
postgres latest e94a3bb61224 2 months ago 374MB
[gabriel@void ~]$ docker rmi postgres
```

```
[gabriel@void ~]$ docker ps --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
9ece6fb235c4 postgres:latest "docker-entrypoint.s..." 2 months ago Exited (0) 50 minutes ago bd_aula
[gabriel@void ~]$ docker rm bd_aula
```

Mais comandos? Use o comando de ajuda!

Comandos de ajuda do docker CLI:

`docker --help` ~ Lista TODOS os comandos e mais uma descrição.

`docker [comando] --help` ~ Lista a descrição, opções e Arg do comando.

Dockerfile

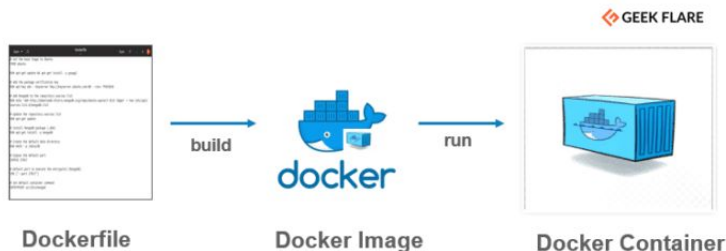


Receita para criar imagens!

Docker **gera uma imagem** a partir da execução de várias instruções escritas em um Dockerfile.

Um Dockerfile é um documento de texto que contém todos os comandos que um usuário usaria caso fosse configurar um container “na mão”.

Usando o comando **docker build** o docker irá executar cada comando em sequência dentro do Dockerfile para gerar a imagem.



Como criar um Dockerfile?

Cria-se um arquivo chamado “Dockerfile” - com **D maiúsculo e sem extensão!**



Como fazer um Dockerfile?

Estrutura de um Dockerfile:

Dockerfile

```
# Comentário  
INSTRUÇÃO argumentos
```

Dockerfile

```
# Hello world!  
RUN echo 'Sistemas distribuídos é top'
```

A **instrução** não é case-sensitive. Porém é esperado que seja escrita **toda em MAIÚSCULA** para se diferenciar de um argumento e ser lido mais facilmente. Resumindo: Boa prática.

Como fazer um Dockerfile?

Formatações permitidas no Dockerfile:

Dockerfile

Comentário

```
RUN echo "Boa \
      Noite\
      Bruno"
```



Dockerfile

Não comece com espaço em branco!!

```
RUN echo Sistemas
RUN echo Distribuidos
```



Quais são as instruções?

FROM : É uma instrução que especifica a “imagem pai” que você está se baseando. E → **SEMPRE** ← deve ser a primeira coisa que você escreve em um Dockerfile. Ex: **FROM ubuntu:latest**

MAINTAINER : é uma instrução para representar o autor do Dockerfile.

Ex: **MAINTAINER boa.noite@bruno.com.br**

RUN : É uma instrução para executar qualquer comando shell. O comando shell terá como alvo a imagem especificada no FROM.

WORKDIR : É uma instrução equivalente ao “cd”, redefine um diretório como ponto de início e todos os comandos abaixo serão executados a partir desse diretório. Ex: **WORKDIR /var/www/html**

COPY : A instrução irá copiar os arquivos especificados do host para o container. Ex: **COPY comandos.sql /usr/local/meus-sql**

EXPOSE : Define uma porta que o container irá “escutar”.

Ex: **EXPOSE 80**

Quais são as instruções?

CMD : Essa instrução tem a função de executar um comando/processo/launcher ao final da criação do container. Obs: Apenas 1 CMD deve existir no Dockerfile. Ex: `java -jar aplicativo.jar`

ENV : Essa instrução é usada para modificar/criar variáveis de ambiente no container.

ENTRYPOINT : Essa instrução é executada antes do CMD (caso seja usada) durante a criação do container .É usada para executar passos antes de iniciar de fato a aplicação. Na prática, entrypoint é usado com um Shell script para fazer as configurações iniciais da aplicação utilizando as variáveis de ambiente do container. Um exemplo:

```
ENTRYPOINT ["/meu_inicializador.sh"]
```

```
CMD ["iniciar"]
```

Os comandos acima irão ser executados como: `meu_inicializador.sh` `iniciar`. o CMD irá fornecer um argumento ao entrypoint e executar.

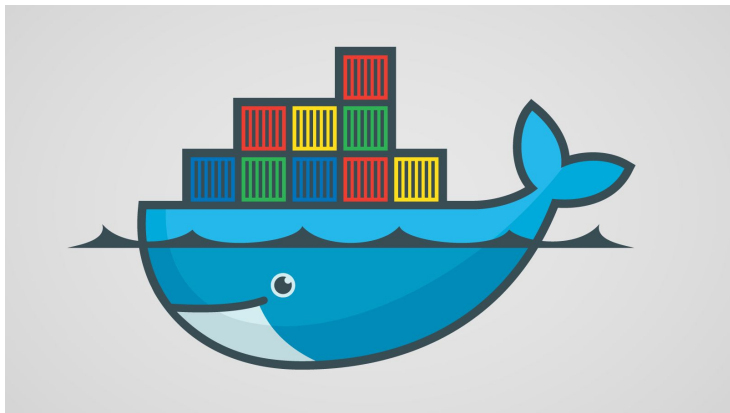
Quais são as instruções?

VOLUME : Essa instrução cria um ponto de montagem em um determinado diretório. Ex: **VOLUME** /var/db

USER : Essa instrução cria um usuário no sistema. Ex: **USER** bruno[:users]

ARG : Essa instrução cria “variáveis” com argumentos recebidos durante o processo de build usando a flag --build-arg <variavel>=<valor>.

Ex: **ARG** nome1



Como buildar uma imagem?

O comando build gera uma imagem modificada a partir de um Dockerfile.

A build é “sensível ao contexto”, dependendo de onde o Dockerfile estiver, ela irá interagir com os arquivos do Diretório.

Um exemplo é usar a instrução COPY, caso estiver em um local não apropriado, pode copiar todos os seus arquivos para dentro da imagem.

É recomendado manter o Dockerfile em uma pasta, e ir adicionando os arquivos necessários para a build.

Como usar o comando build?

Com o terminal dentro da pasta do Dockerfile, execute o comando:

```
[gabriel@void ~/Meu Docker]$ docker build .
```

Caso queira dar um nome personalizado à imagem, use a opção **-t** :

```
[gabriel@void ~/Meu Docker]$ docker build -t minha_imagem .
```

Nome com tag? não tem problema!:

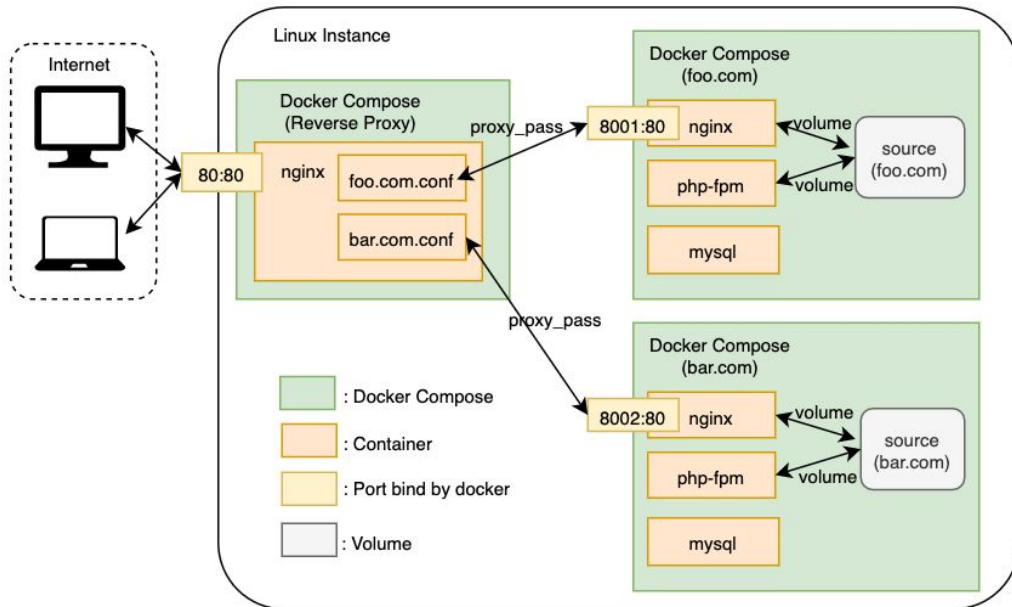
```
[gabriel@void ~/Meu Docker]$ docker build -t minha_imagem:1.0-alpha .|
```

Docker compose



Docker compose... ou simplesmente microsserviços?

Docker compose possibilita criar e executar multi-containers em uma aplicação Docker. Em poucas palavras, o agrupamento de containers criam serviços independentes em uma rede.



Como fazer um docker-compose.yml

Cria-se um arquivo chamado docker-compose.yml

A extensão .yml é conhecida como YAML, é um formato de serialização de dados legíveis por humanos inspirado em linguagens como XML, C, Python, Perl...



Como fazer um docker-compose.yml

Estrutura de um yml:

docker-compose.yml

```
chave: valor # eu sou um comentário
```

docker-compose.yml

```
chave: # lembra um arquivo json e a ling. Python!
```

```
  valor
```

```
    chave:
```

```
      valor
```

```
      valor
```

Obs: yml é sensível à indentação!!

Exemplo: docker-compose.yml

```
1 # Docker "docker-compose"
2
3 version: '3.3'
4
5 services:
6
7   db:
8     image: mysql:8.0.19
9     command: --default-authentication-plugin=mysql_native_password
10    volumes:
11      - db_data:/var/lib/mysql
12    restart: "no"
13    environment:
14      MYSQL_ROOT_PASSWORD: somewordpress
15      MYSQL_DATABASE: wordpress
16      MYSQL_USER: wordpress
17      MYSQL_PASSWORD: wordpress
18
19   wordpress:
20     depends_on:
21       - db
22     image: wordpress:latest
23     ports:
24       - "8000:80"
25     restart: "no"
26     environment:
27       WORDPRESS_DB_HOST: db:3306
28       WORDPRESS_DB_USER: wordpress
29       WORDPRESS_DB_PASSWORD: wordpress
30       WORDPRESS_DB_NAME: wordpress
31
32 volumes:
33   db_data: {}
34
```

Chaves e valores de um docker compose: version

Compose file format	Docker Engine release
3.8	19.03.0+
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

Chaves e valores de um docker compose: services

A chave **services** é onde irá declarar todos os containers e suas configurações que irão ser usados.

Os valores irão ser usadas como chaves para identificar cada “nó” de container. O nome do container é você que escolhe! e será usado como identificador na “rede do docker”, como se fosse um DNS.

docker-compose.yml

```
services:
  servico1:
    ...
  servico2:
    ...
```

Chaves e valores de um docker compose: image e build

Ambas são responsáveis por identificar qual a imagem que o container será baseado. No caso de **image** ele irá baixar a imagem direto do **docker hub**. E **build** irá buscar um **Dockerfile** para gerar uma imagem personalizada e usa-la.

docker-compose.yml

```
services:
  bancoDados:
    image: postgres:latest
    ou
    build: . //Dockerfile no mesmo dir.
```

Chaves e valores de um docker compose: ports

A chave **ports** é usada para especificar a porta de entrada do host e do container separados por dois pontos. É passado uma lista de string para a chave, usando hífen no início.

docker-compose.yml

```
services:
  bancoDados:
    image: postgres:latest
    ports:
      - "5432:5432"
```

Declaração de um item da lista
reconhecido pelo yaml

Chaves e valores de um docker compose: volumes

A chave **volumes** é usada para montar um diretório do projeto dentro do container, permitindo fazer alterações de arquivos sem precisar reconstruir a imagem. É passado uma lista no formato de:

<dir host> : <dir dentro do container>

docker-compose.yml

```
services:
  bancoDados:
    image: postgres:latest
    ports:
      - "5432:5432"
    volumes:
      - ./source:/var/projeto
```

Chaves e valores de um docker compose: environment

A chave **environment** é usada para definir variáveis de ambiente. Geralmente na documentação da imagem é especificado o nome da variável e o valor esperado para ser usado nas configurações iniciais do aplicativo, como por exemplo usuário e senha do admin. Os valores esperados são chave : valor

docker-compose.yml

```
services:
  bancoDados:
    ...
    volumes:
      - ./source:/var/projeto
    environment:
      POSTGRES_USER: bruno
      POSTGRES_PASSWORD: boanoite123
```

Um mar de possibilidades...

DOCKER COMPOSE CHEAT SHEET

File

structure

```
services:  
  container1:  
    properties: values  
  
  container2:  
    properties: values
```

```
networks:  
  network:
```

```
volumes:  
  volume:
```

Types

value

```
key: value
```

array

```
key:  
  - value  
  - value
```

dictionary

```
master:  
  key: value  
  key: value
```

Properties

build

build image from dockerfile
in specified directory

```
container:  
  build: ./path  
  image: image-name
```

image

use specified image
`image: image-name`

container_name

define container name to access
it later

```
container_name: name
```

volumes

define container volumes to
persist data

```
volumes:  
  - /path:/path
```

command

override start command for the
container

```
command: execute
```

environment

define env variables for the
container

```
environment:  
  KEY: VALUE
```

```
environment:  
  - KEY=VALUE
```

env_file

define a env file for the
container to set and override
env variables

```
env_file: .env
```

```
env_file:  
  - .env
```

restart

define restart rule
(no, always, on-failure, unless-
stopped)

```
restart:  
  - "9999"
```

networks

define all networks for the
container

```
networks:  
  - network-name
```

ports

define ports to expose to other
containers and host

```
ports:  
  - "9999:9999"
```

expose

define ports to expose only to
other containers

```
expose:  
  - "9999"
```

network_mode

define network driver
(bridge, host, none, etc.)

```
network_mode: host
```

depends_on

define build, start and stop
order of container

```
depends_on:  
  - container-name
```

Other

idle container

send container to idle state
> container will not stop

```
command: tail -f /dev/null
```

named volumes

create volumes that can be used in
the volumes property

```
services:  
  container:  
    image: image-name  
    volumes:  
      - data-  
volume:/path/to/dir
```

```
volumes:  
  data-volume:
```

networks

create networks that can be used
in the networks property

```
networks:  
  frontend:  
    driver: bridge
```



Docker compose CLI: comandos básicos

Inicializar o docker-compose:

```
[gabriel@void ~/meu-Docker]$ docker-compose up
```

Parar um docker-compose:

```
[gabriel@void ~/meu-Docker]$ docker-compose stop
```

Parar e **deletar todos os containers** do docker-compose:

```
[gabriel@void ~/meu-Docker]$ docker-compose down|
```

Iniciar um docker-compose parado:

```
[gabriel@void ~/meu-Docker]$ docker-compose start|
```

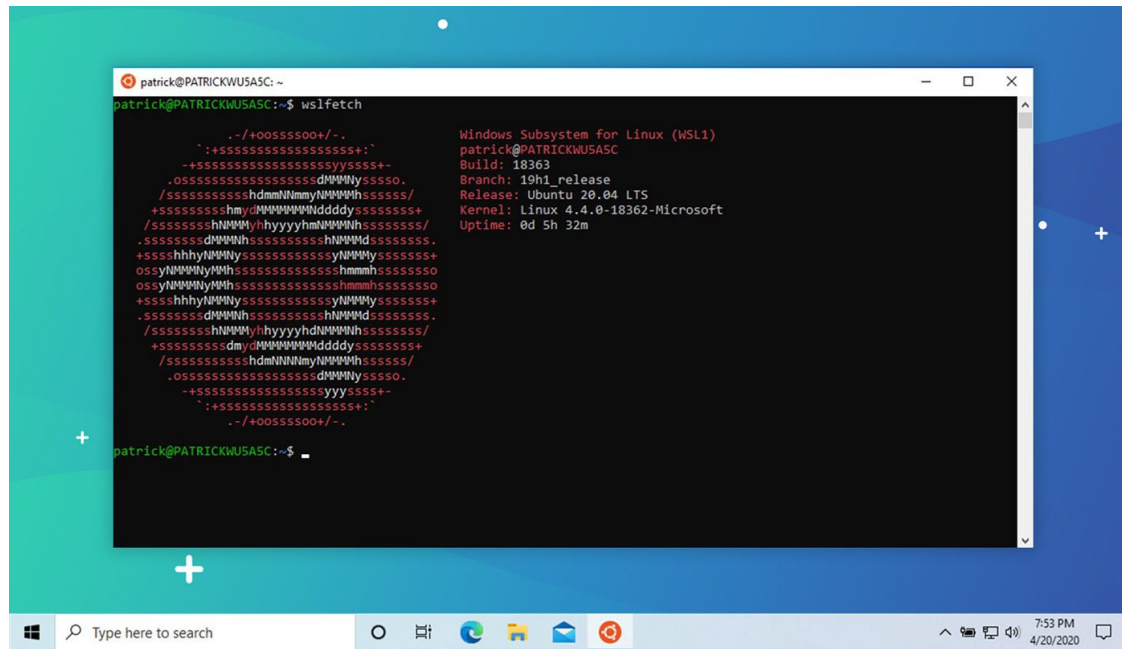
Aplicando a teoria

Instalando o docker, criando um Dockerfile e docker-compose!



Instalando o docker (Windows)

- 1º - Baixar o Instalador no [site do Docker](#)
- 2º - Requisitos: Windows Subsystem for Linux ou Hyper-V Windows Features



Instalando o docker (Linux)

1° - Abra o Terminal

2° - Instale o pacote com: `sudo <package manager> install docker`

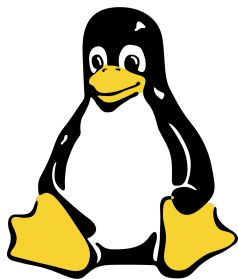
Search Results for docker

Search:	Name	Summary	Files	Provides	Requires	Search Syntax
Filter:	Distros: 28 hidden	Archives: intel	Types: official			
Alpine 3.15	Arch Linux	CentOS 9 Stream	Debian 11 (Bullseye)	Debian Sid	Fedora 35	FreeBSD 13
Magela 8	NetBSD 9.1	OpenMandriva Rolling	openSUSE Leap 15.3	Solus	Ubuntu 20.04 LTS (Focal Fossa)	Ubuntu 18.04 LTS (Bionic Beaver)
Ubuntu 21.10 (Impish Indri)	Void Linux					

Do zero ao deploy!

O objetivo:

Deploy em uma página web usando o stack L.A.P.P com containers docker!



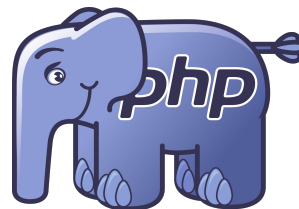
Linux



Apache



Postgres



PHP

OBRIGADO



Fundação Universidade Federal de Rondônia - UNIR
Departamento Acadêmico de Ciência da Computação - DACC
Bacharelado em Ciência da Computação

Sistemas Distribuídos

DOCKER

Docente: Dr. Marcelo

Bruno Bordin
Gabriel Russo
Marcus Martins

Porto Velho 2022