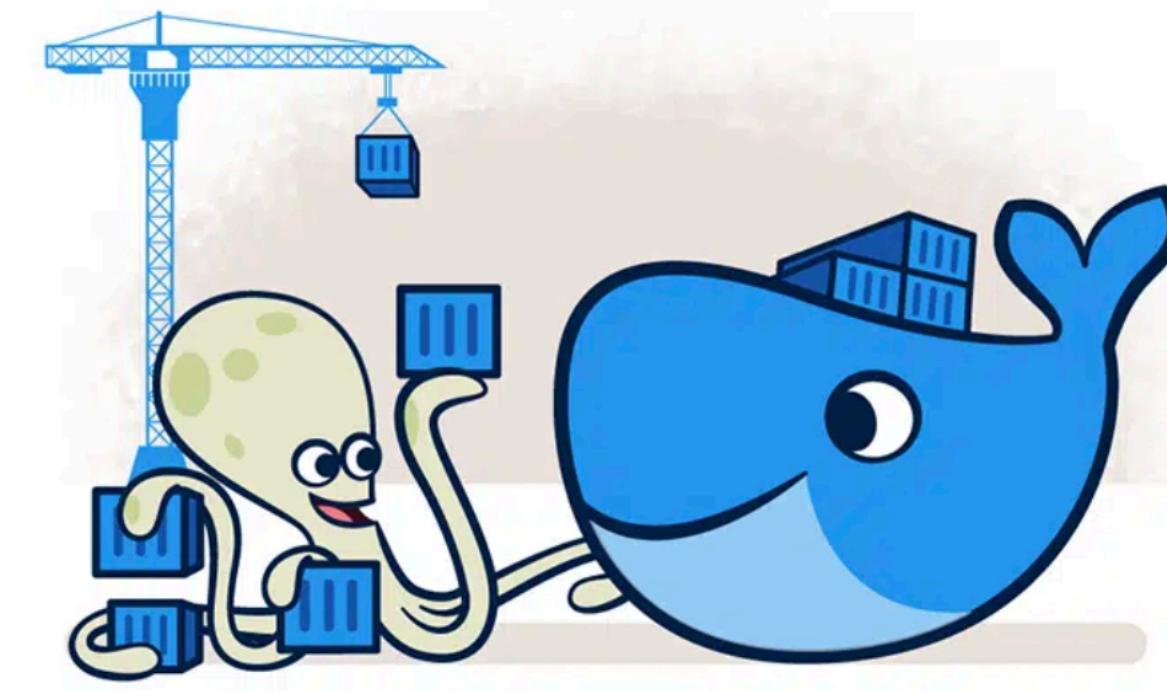
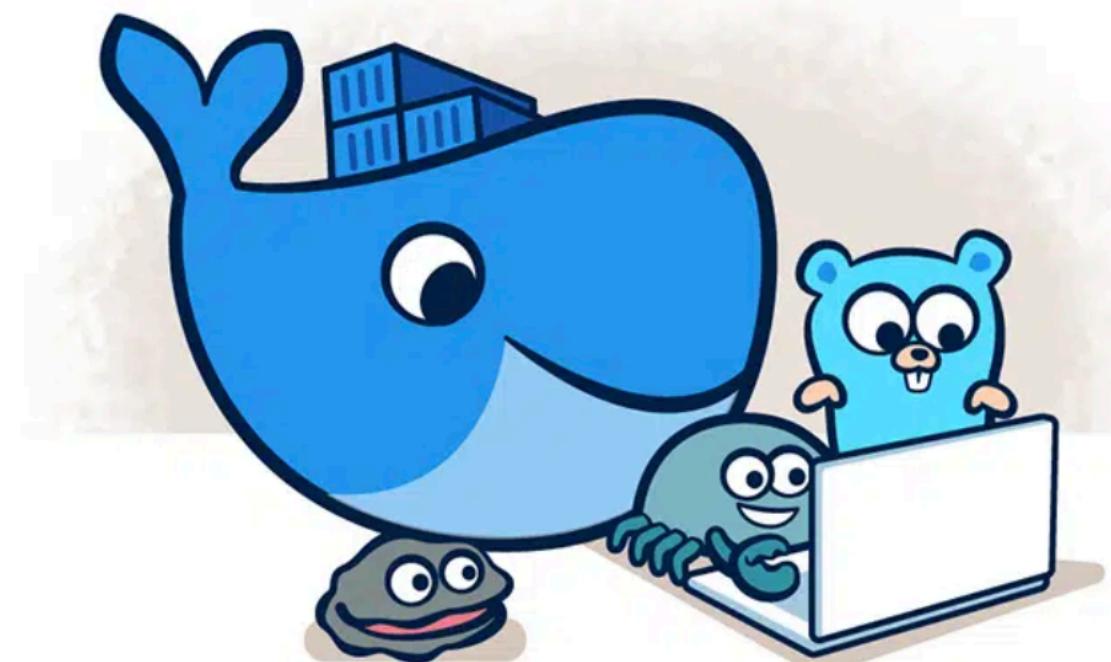


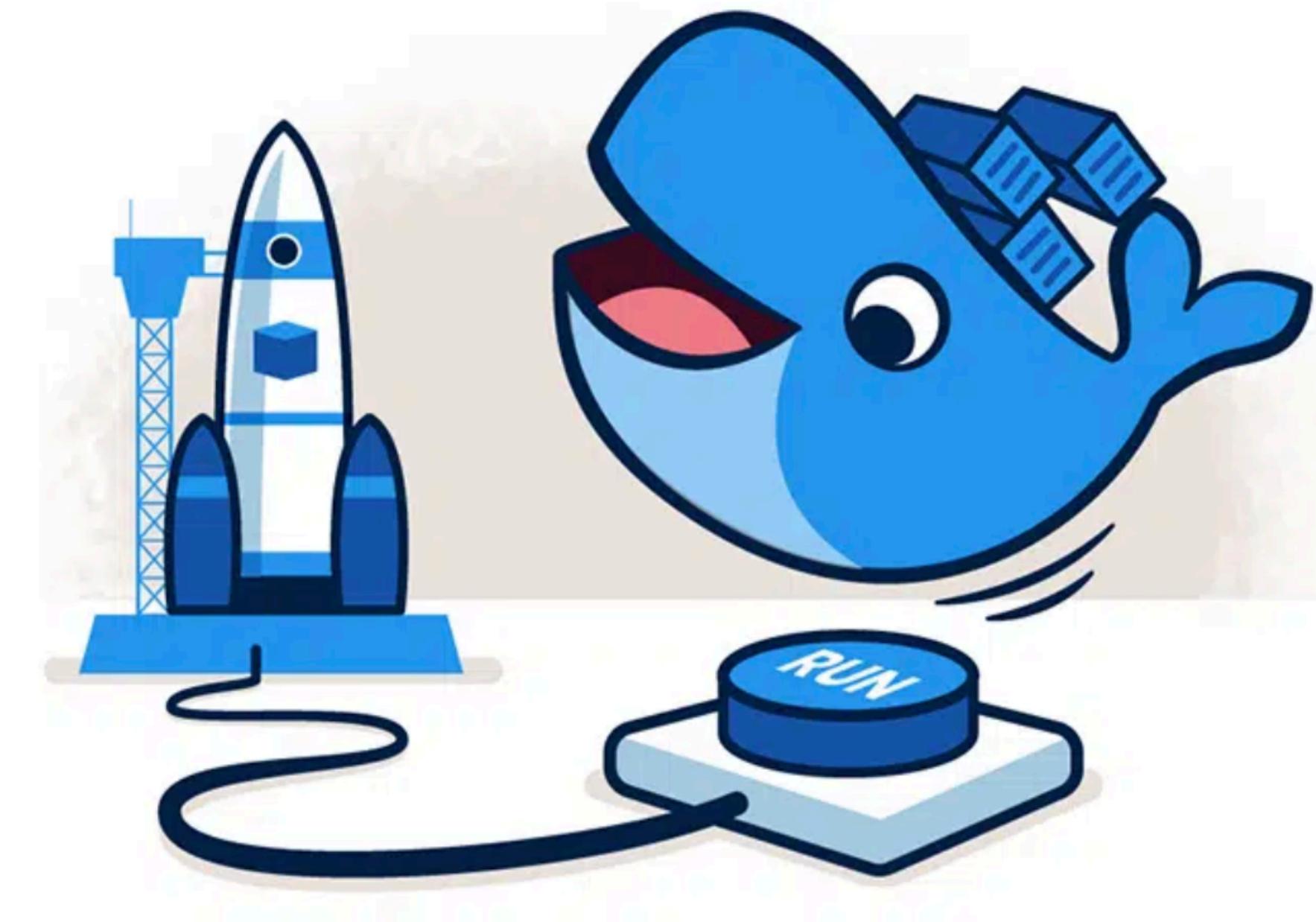
Criando e Gerenciando Containers com o Docker

Marcus Adriano



Objetivos

- Conhecer Docker e sua arquitetura;
- Conhecer alguns comandos do Docker;
- Executar containers;
- Criar imagens customizadas;
- Publicar imagens customizadas.



Material de apoio: <https://github.com/MarcusAdriano/docker-mini-curso>

Visão Geral: plataforma

- Plataforma open source para desenvolver, entregar e executar apps;
- Separação da infraestrutura da app;
- Unidade básica: **container** (contém tudo que a app precisa);
- Distribuição é simples e fácil (e funciona em qualquer lugar);
- *Containers* dividem os recursos do mesmo ***host***;

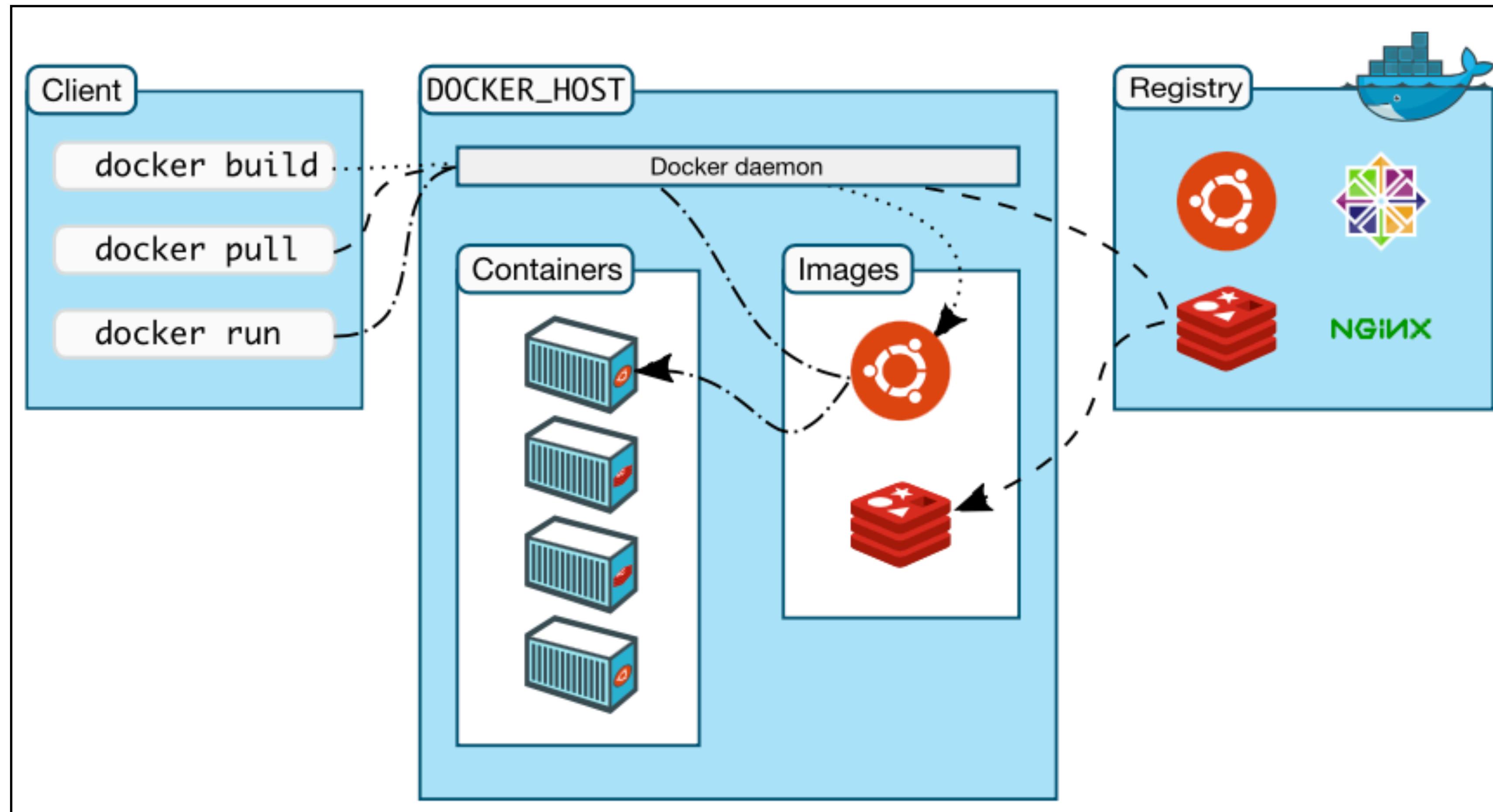
Por que Docker?

- Distribuição de containers é consistente;
 - Ambientes de desenvolvimento, testes e produtivo;
- Portabilidade: máquina local, máquina virtual, *cloud*;
- Execução de vários *containers* no mesmo *host*.

Empresas que utilizan Docker



Docker: arquitetura



Docker: arquitetura

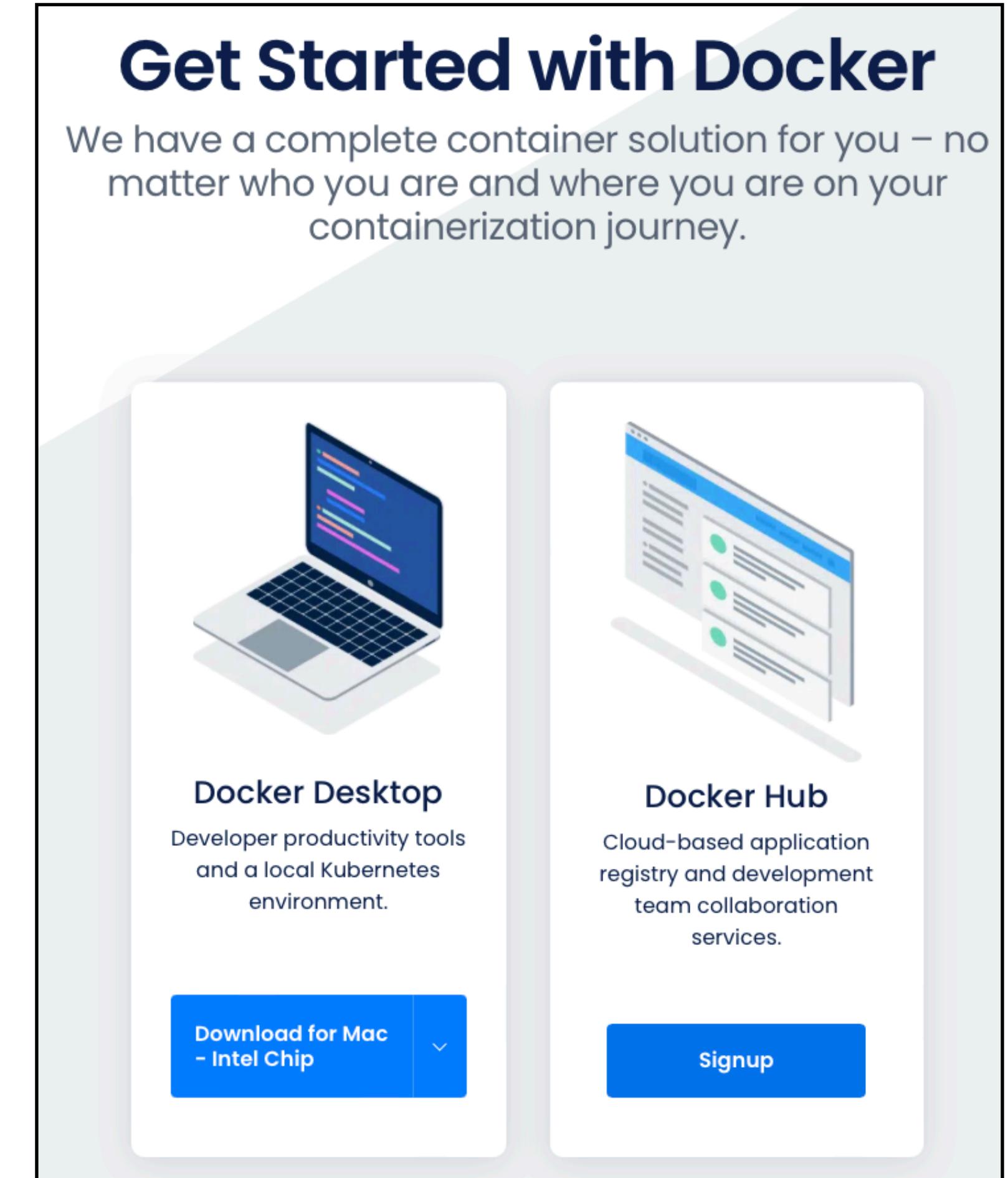
- Docker Client: forma primária de comunicação com o ***dockerd***;
- Docker Daemon (***dockerd***): (onde a mágica acontece) controla e gerencia a execução dos objetos Docker (containers, images, network, volumes, etc);
- Docker Registry: Repositório de imagens Docker.

Docker: Objetos

- Images: template para criação de containers; usuários criam suas próprias imagens baseando-se em imagens disponíveis no Registry;
- Containers: instância executável de uma imagem; Docker disponibiliza comandos para iniciar, parar, remover um container a partir de uma imagem.

Primeiros Passos

- Download Docker: <https://www.docker.com/get-started/>
- Cadastro Docker-Hub: <https://hub.docker.com/signup>



Verificando a instalação



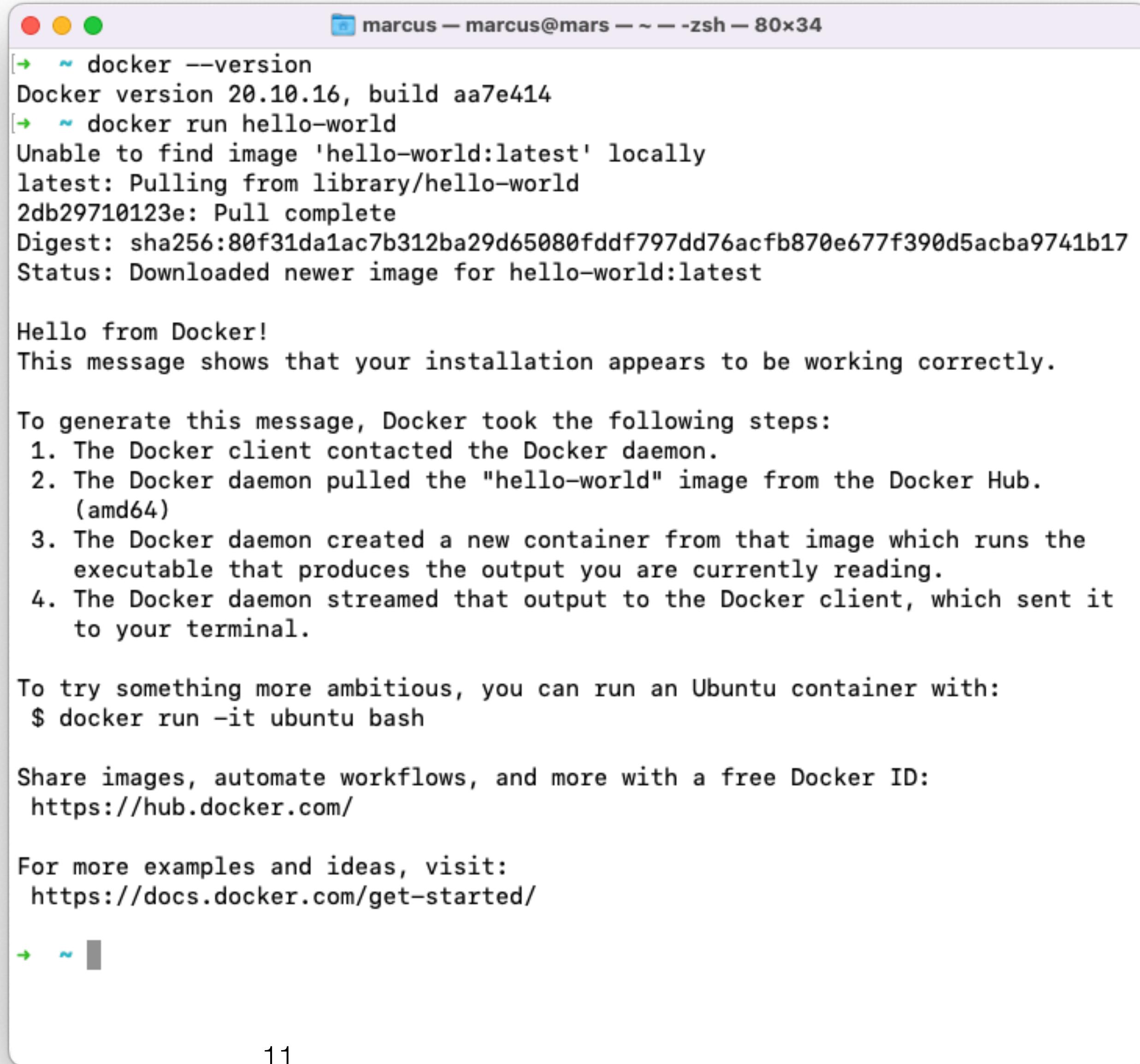
A screenshot of a macOS terminal window titled "marcus — marcus@mars — ~ — -zsh — 80x34". The window shows the command "docker --version" being run, with the output "Docker version 20.10.16, build aa7e414". The terminal has a light gray background and standard macOS-style window controls.

```
[~] docker --version
Docker version 20.10.16, build aa7e414
[~]
```

- docker –version

Verificando a instalação

- docker run hello-world



```
[~] docker --version
Docker version 20.10.16, build aa7e414
[~] docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:80f31da1ac7b312ba29d65080fddf797dd76acfb870e677f390d5acba9741b17
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Executando um container

- docker run -d -p 8080:80 docker/getting-started
- -d: execução em segundo plano
- -p: mapeia a porta 8080 do host para a porta 80 do container
- docker/getting-started: imagem

Conferindo a execução

- Abrir no navegador: <http://localhost:8080>

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost
- Page Title:** dockerLabs - Getting Started
- Header:** dockerLabs Getting Started Search docker/getting-started 2.0k Stars · 5.1k Forks
- Left Sidebar (Getting Started):**
 - Getting Started
 - Getting Started
 - Our Application
 - Updating our App
 - Sharing our App
 - Persisting our DB
 - Using Bind Mounts
 - Multi-Container Apps
 - Using Docker Compose
 - Image Building Best Practices
 - What Next?
- Main Content (Getting Started):**

Getting Started

The command you just ran

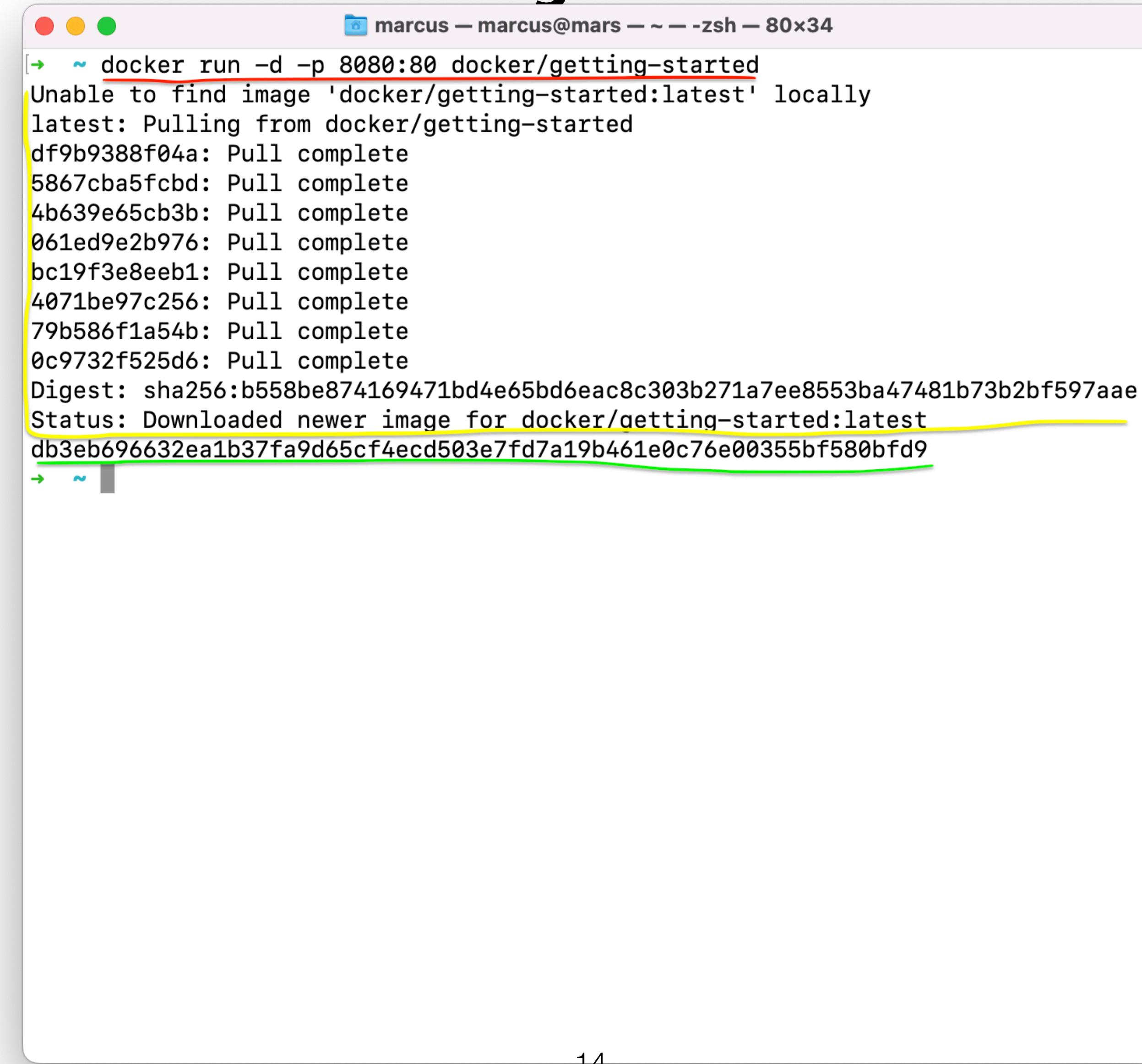
Congratulations! You have started the container for this tutorial! Let's first explain the command that you just ran. In case you forgot, here's the command:

```
docker run -d -p 80:80 docker/getting-started
```

You'll notice a few flags being used. Here's some more info on them:

 - `-d` - run the container in detached mode (in the background)
 - `-p 80:80` - map port 80 of the host to port 80 in the container
- Right Sidebar (Table of contents):**
 - Table of contents
 - The command you just ran
 - The Docker Dashboard
 - What is a container?
 - What is a container image?

Sobre a execução do container

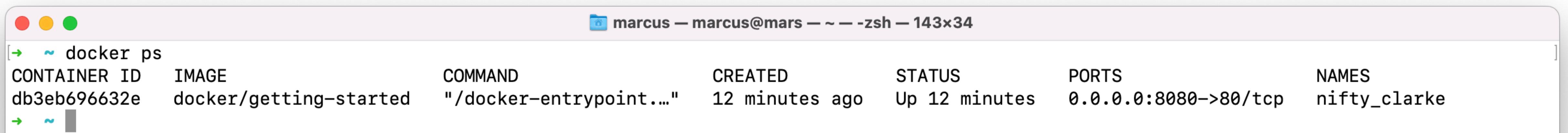


A screenshot of a terminal window titled "marcus — marcus@mars — ~ — zsh — 80x34". The window shows the command "docker run -d -p 8080:80 docker/getting-started" being executed. The output indicates that the image 'docker/getting-started:latest' was not found locally and was pulled from the Docker registry. The pull process completed successfully, and the status message "Status: Downloaded newer image for docker/getting-started:latest" is displayed. The terminal window has a yellow border around the command and its output.

```
[~] ~ docker run -d -p 8080:80 docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
df9b9388f04a: Pull complete
5867cba5fcbd: Pull complete
4b639e65cb3b: Pull complete
061ed9e2b976: Pull complete
bc19f3e8eeb1: Pull complete
4071be97c256: Pull complete
79b586f1a54b: Pull complete
0c9732f525d6: Pull complete
Digest: sha256:b558be874169471bd4e65bd6eac8c303b271a7ee8553ba47481b73b2bf597aae
Status: Downloaded newer image for docker/getting-started:latest
db3eb696632ea1b37fa9d65cf4ecd503e7fd7a19b461e0c76e00355bf580bfd9
```

Listando os containers ativos

- docker ps



A screenshot of a macOS terminal window titled "marcus — marcus@mars — ~ — zsh — 143x34". The window shows the command "docker ps" being run, followed by a table of container information. The table has columns: CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES. One container is listed:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
db3eb696632e	docker/getting-started	"/docker-entrypoint..."	12 minutes ago	Up 12 minutes	0.0.0.0:8080->80/tcp	nifty_clarke

Listando os container ativos e inativos

- docker ps -a

```
Last login: Sat Jun 11 17:11:14 on ttys000
[~] ~ docker ps -a
CONTAINER ID  IMAGE
690a358e7b5a  hello-world
db3eb696632e  docker/getting-started
6c874c99d6b7  hello-world
66e005a8e5f9  docker.elastic.co/kibana/kibana:8.2.2
cc3022c547e0  docker.elastic.co/elasticsearch/elasticsearch:8.2.2
[~] ~ [~]
```

COMMAND	CREATED	STATUS	PORTS	NAMES
"/hello"	9 minutes ago	Exited (0) 9 minutes ago		infallible_gould
"/docker-entrypoint...."	21 minutes ago	Up 21 minutes	0.0.0.0:8080->80/tcp	nifty_clarke
"/hello"	27 minutes ago	Exited (0) 27 minutes ago		magical_lumiere
"/bin/tini -- /usr/l..."	43 hours ago	Exited (0) 42 hours ago		kib-01
"/bin/tini -- /usr/l..."	44 hours ago	Exited (143) 42 hours ago		es01

Ajuda sobre os comandos

- docker [comando] –help

```
[→ ~ docker ps --help

Usage: docker ps [OPTIONS]

List containers

Options:
  -a, --all            Show all containers (default shows just running)
  -f, --filter filter  Filter output based on conditions provided
  --format string       Pretty-print containers using a Go template
  -n, --last int        Show n last created containers (includes all states) (default -1)
  -l, --latest          Show the latest created container (includes all states)
  --no-trunc            Don't truncate output
  -q, --quiet           Only display container IDs
  -s, --size             Display total file sizes
[→ ~ ]
```

Mais um exemplo de ajuda

- docker –help

```
[→ ~ docker --help

Usage: docker [OPTIONS] COMMAND
       A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
      -c, --context string    Name of the context to use to connect to
      -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"
      --tls                 Use TLS; implied by --tlsverify
      --tlscacert string    Trust certs signed only by this CA (def
      --tlscert string      Path to TLS certificate file (default "
      --tlskey string       Path to TLS key file (default "/Users/m
      --tlsverify            Use TLS and verify the remote
      -v, --version          Print version information and quit

Management Commands:
  builder      Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.8.2)
  compose*    Docker Compose (Docker Inc., v2.6.0)
  config       Manage Docker configs
  container    Manage containers
  context      Manage contexts
  image        Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  ...
```

Informações sobre um container

CONTAINER ID	IMAGE	COMMAND
16d9d8ff03d1	docker/getting-started	"/docker-entrypoint...."

```
~ docker logs 16d9d8ff03d1
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/06/11 20:32:18 [notice] 1#1: using the "epoll" event method
2022/06/11 20:32:18 [notice] 1#1: nginx/1.21.6
```

Outros comandos: stop, start, pause, restart, rename, etc.

Exemplos de mais comandos

```
→ ~ docker stop 16d  
16d  
→ ~ docker start 16d9  
16d9  
→ ~ docker restart 16d9d8ff03d1  
16d9d8ff03d1  
→ ~
```

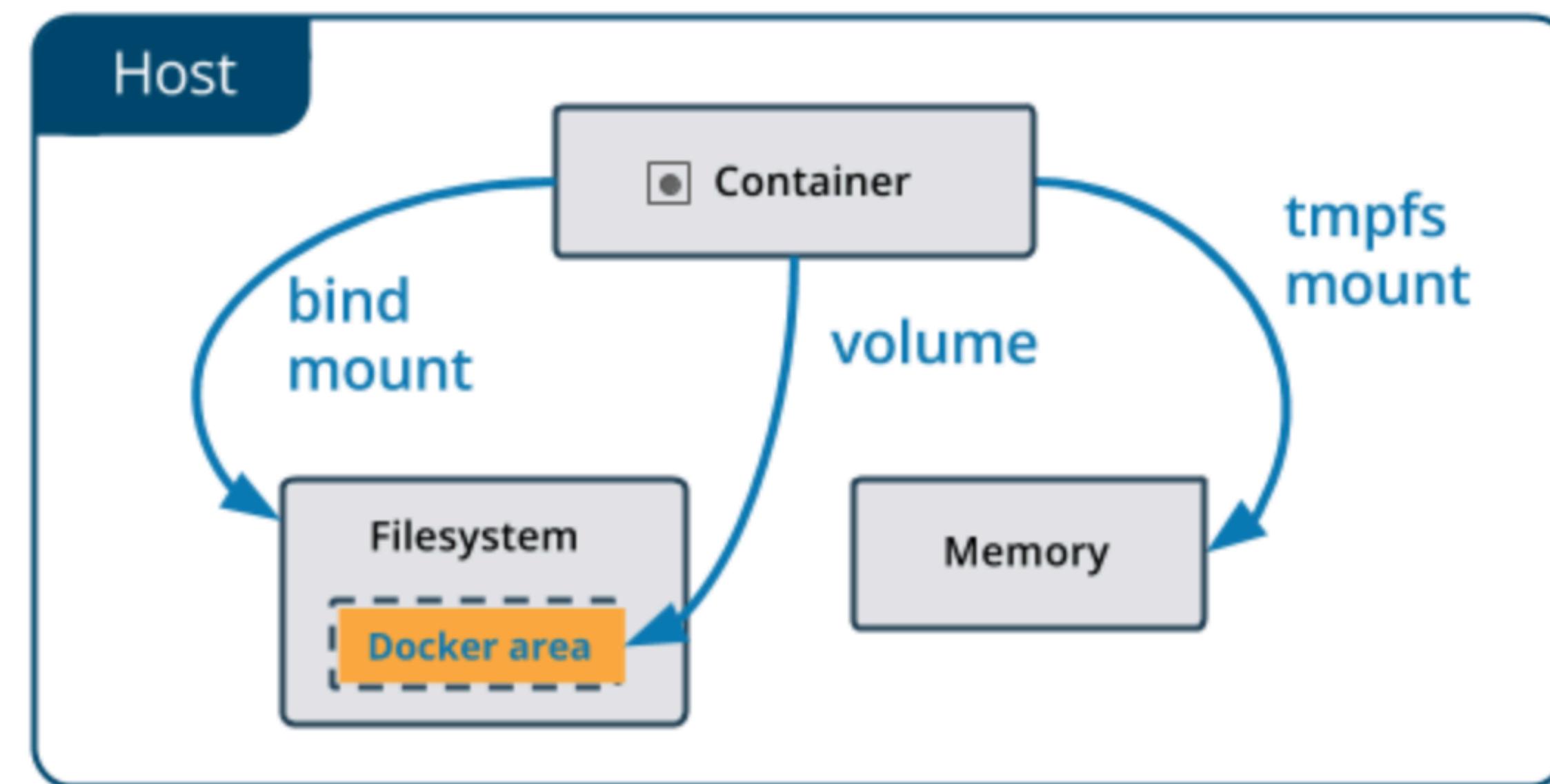
```
→ ~ docker stop 16d && docker rm 16d  
16d  
16d
```

```
→ ~ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
~						

Volumes

- Conectando o sistema de arquivos do Host com o Container



Volume exemplo: MySQL

- docker run --name some-mysql -v {host_dir}:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql
 - –name nome do container
 - -v mapa de volumes para o container {host_dir}:/var/lib/mysql
 - -e variável de ambiente chave=valor
 - -d execução em segundo plano
- Os dados do MySQL serão salvos em {host_dir}.

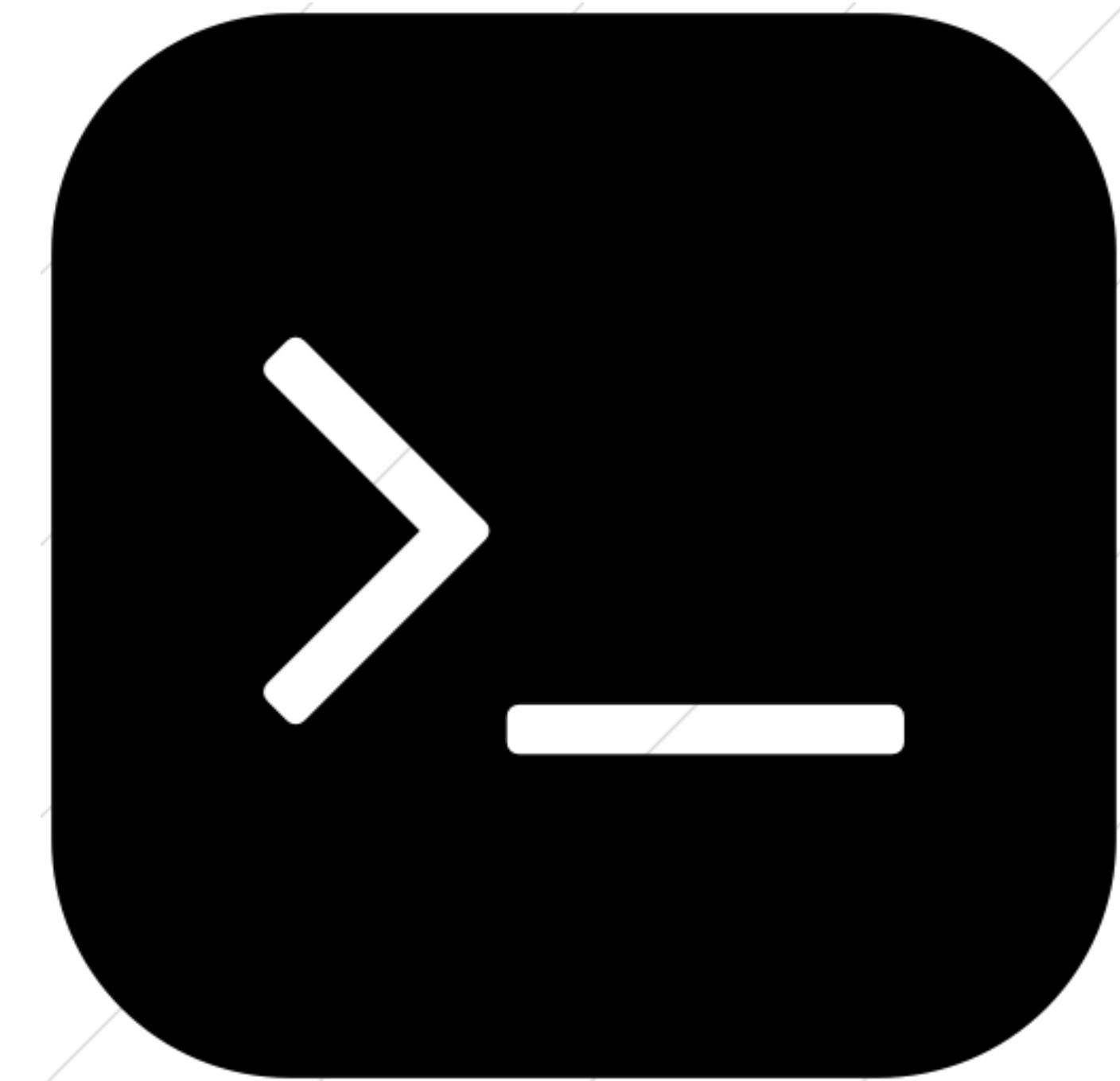
Sobre a imagem do MySQL:

- https://hub.docker.com/_/mysql



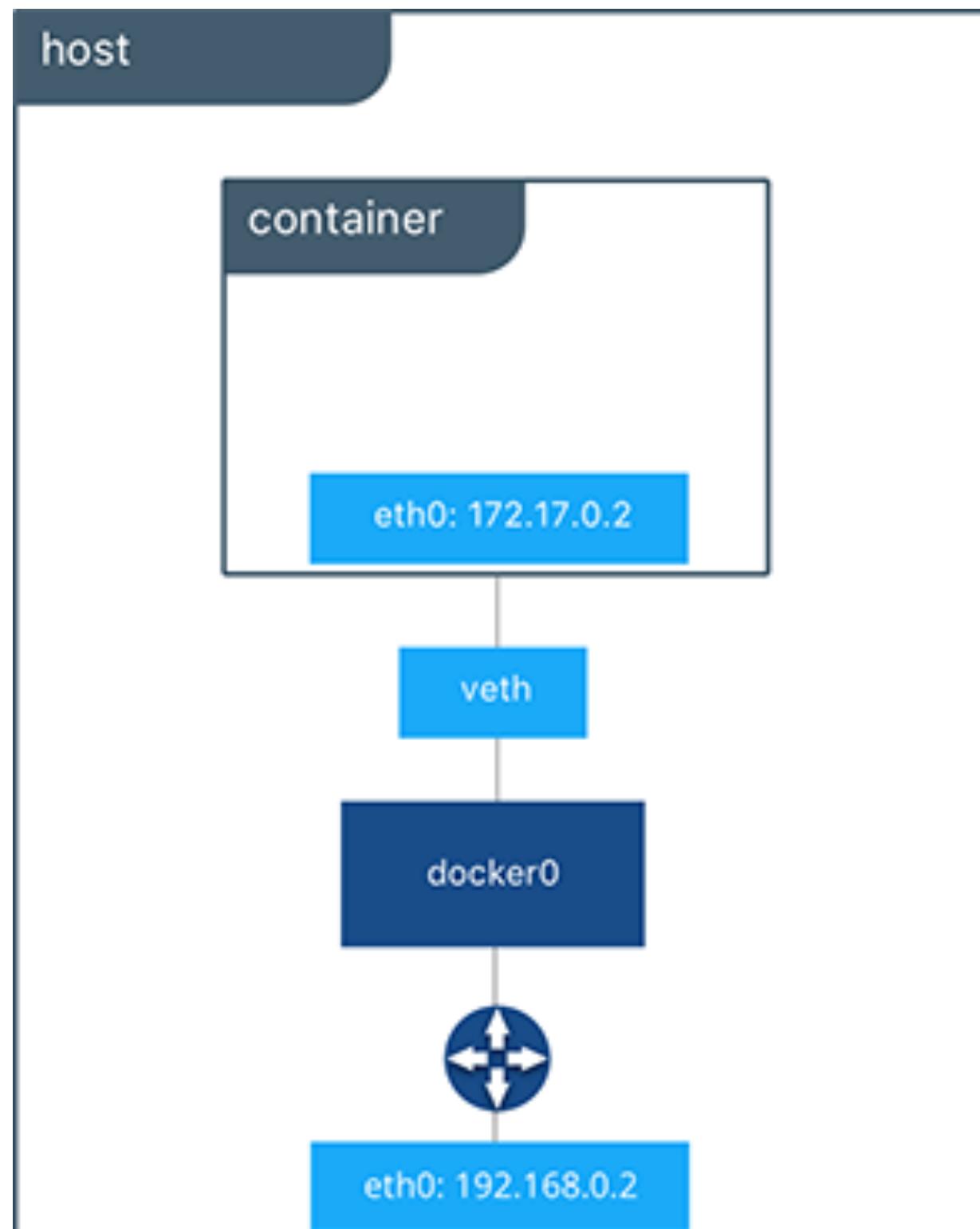
Acessando um Container

- docker exec -it {comando}
- docker exec -it ls /dir
- docker exec -it mysql -u root -p
- docker exec -it ping 172.2.0.1



Rede em containers

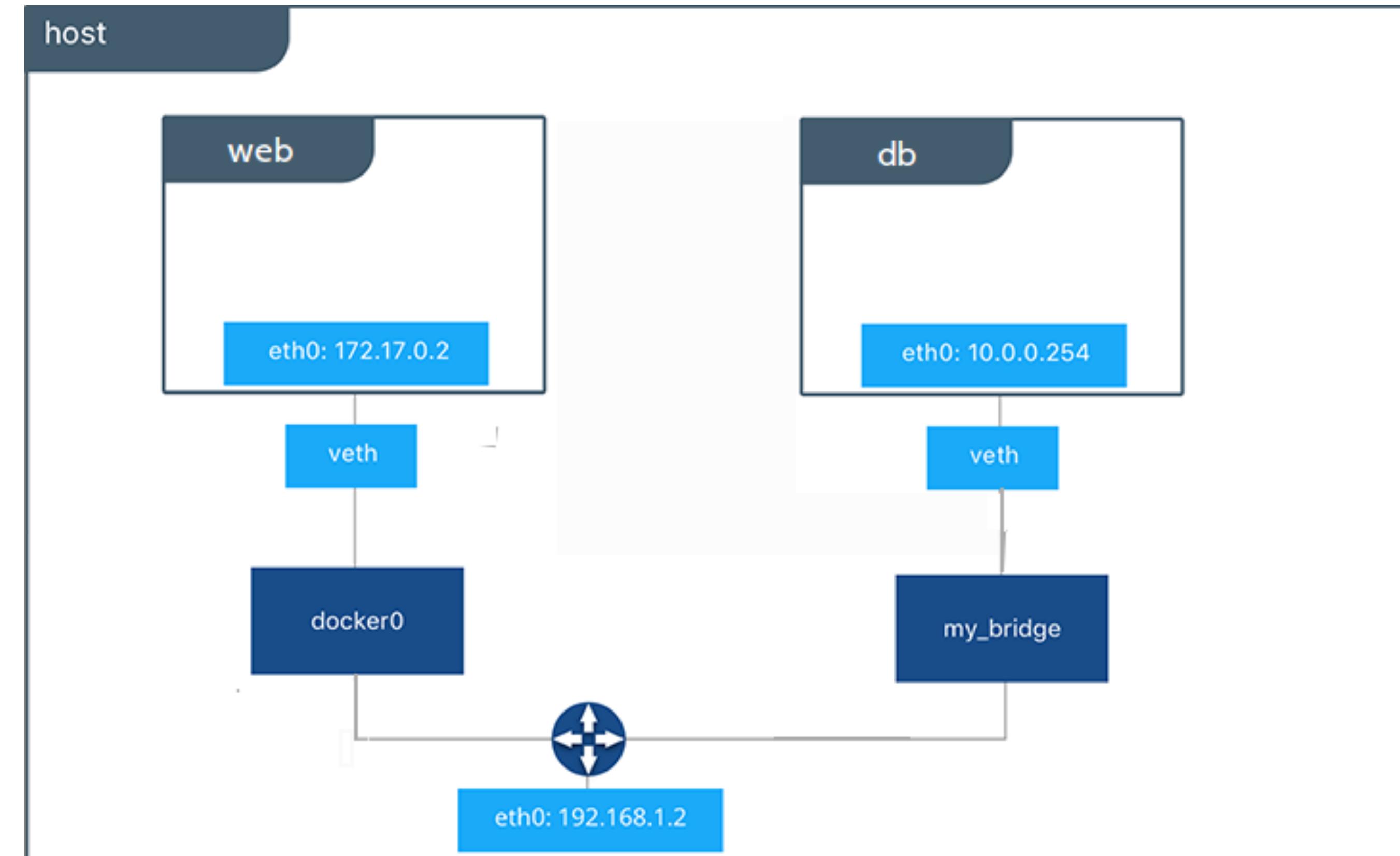
- Containers por padrão são alocados na mesma rede



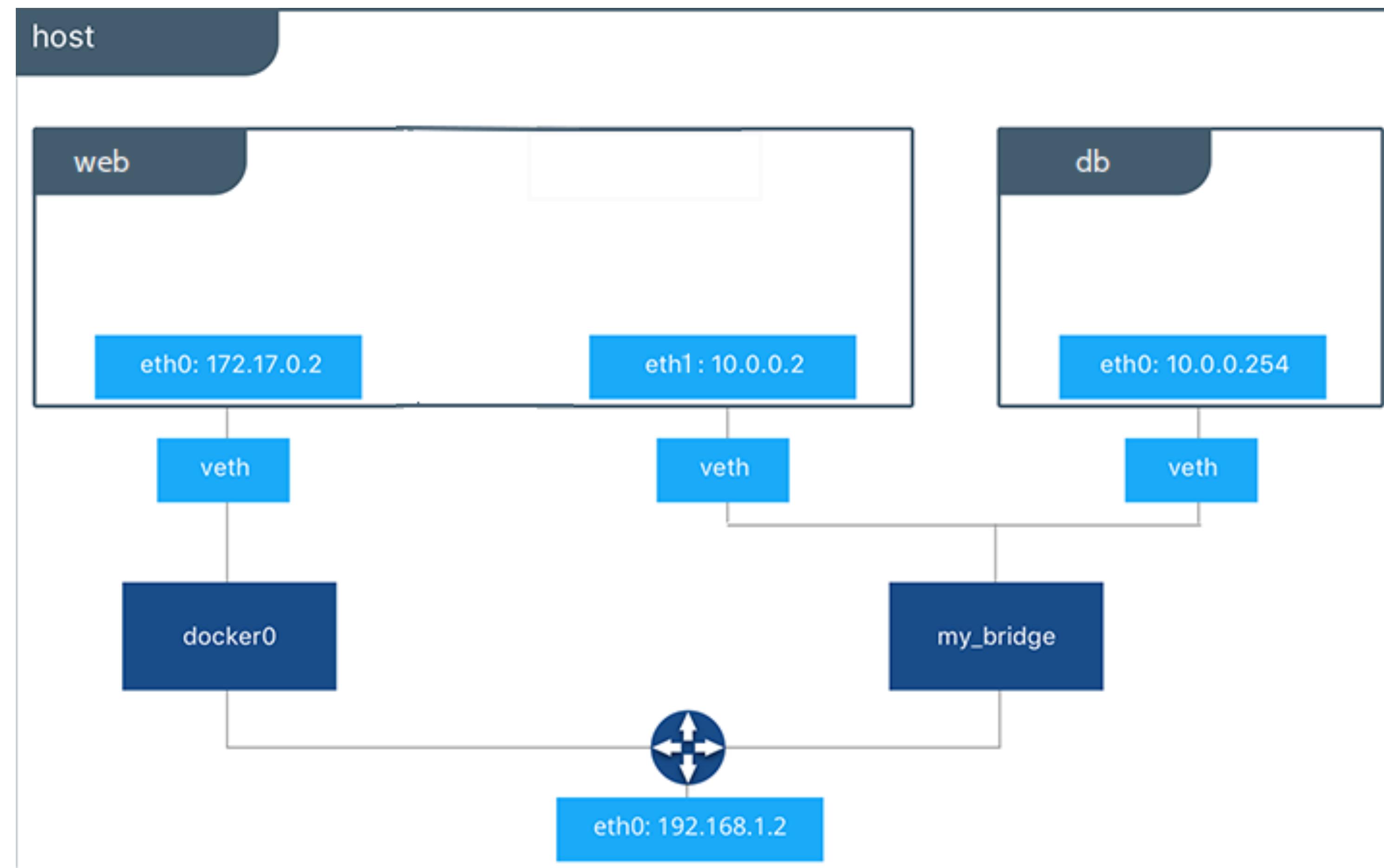
Rede em containers

Criação de redes apartadas:

- docker network create {nome}



Containers que dividem a mesma rede



Rede em containers

Conectando um container em uma rede

- docker run —network {nome da rede}

Ou no caso de um container existir

- docker network connect {rede} {container}
- Exemplos: <https://docs.docker.com/network/network-tutorial-standalone/>

Criação de Containers Customizados

Material de apoio: <https://github.com/MarcusAdriano/docker-mini-curso>

Criação de imagem

- Dockerfile: arquivo com os detalhes da image;
- Build: processo de realizar o empacotamento e construção da imagem



Criando a própria imagem

Dockerfile (backend)

```
FROM openjdk:11

ENV APP_PATH=/opt/myapp/
ENV APP_JAR=todo-0.0.1-SNAPSHOT.jar

RUN mkdir $APP_PATH
COPY target/$APP_JAR $APP_PATH

EXPOSE 8080

WORKDIR $APP_PATH

ENTRYPOINT java -jar $APP_JAR
```

Criando a própria imagem

Dockerfile (frontend)

```
FROM nginx  
  
COPY html/* /usr/share/nginx/html
```

Gerando uma imagem

Comando para criar uma imagem local:

- docker build -t {user docker hub}/{nome imagem}:{versão} -f {Dockerfile}

Exemplo:

- docker build -t marcusadriano/backend-todo .
- docker build -t marcusadriano/frontend-todo .

Execução de Container (frontend)

Listando as imagens locais disponíveis

- docker images

Executando um container a partir da imagem recém criada:

- docker run –rm -it -p 8082:80 –name frontend marcusadriano/frontend-todo
 1. –rm o container será deletado após execução
 2. -i iterativo (stdin <– terminal), -t TTY terminal elegante
 3. –name nome do container

Execução de container (BD)

Executando o MySQL:

- docker run --name mysqltodo --network todoapp -e MYSQL_ROOT_PASSWORD=senha_dificil -d mysql

Parâmetros:

1. –name do container
2. –network todoapp interface de rede
3. -e variável de ambiente no formato chave=valor
4. -d execução em segundo plano

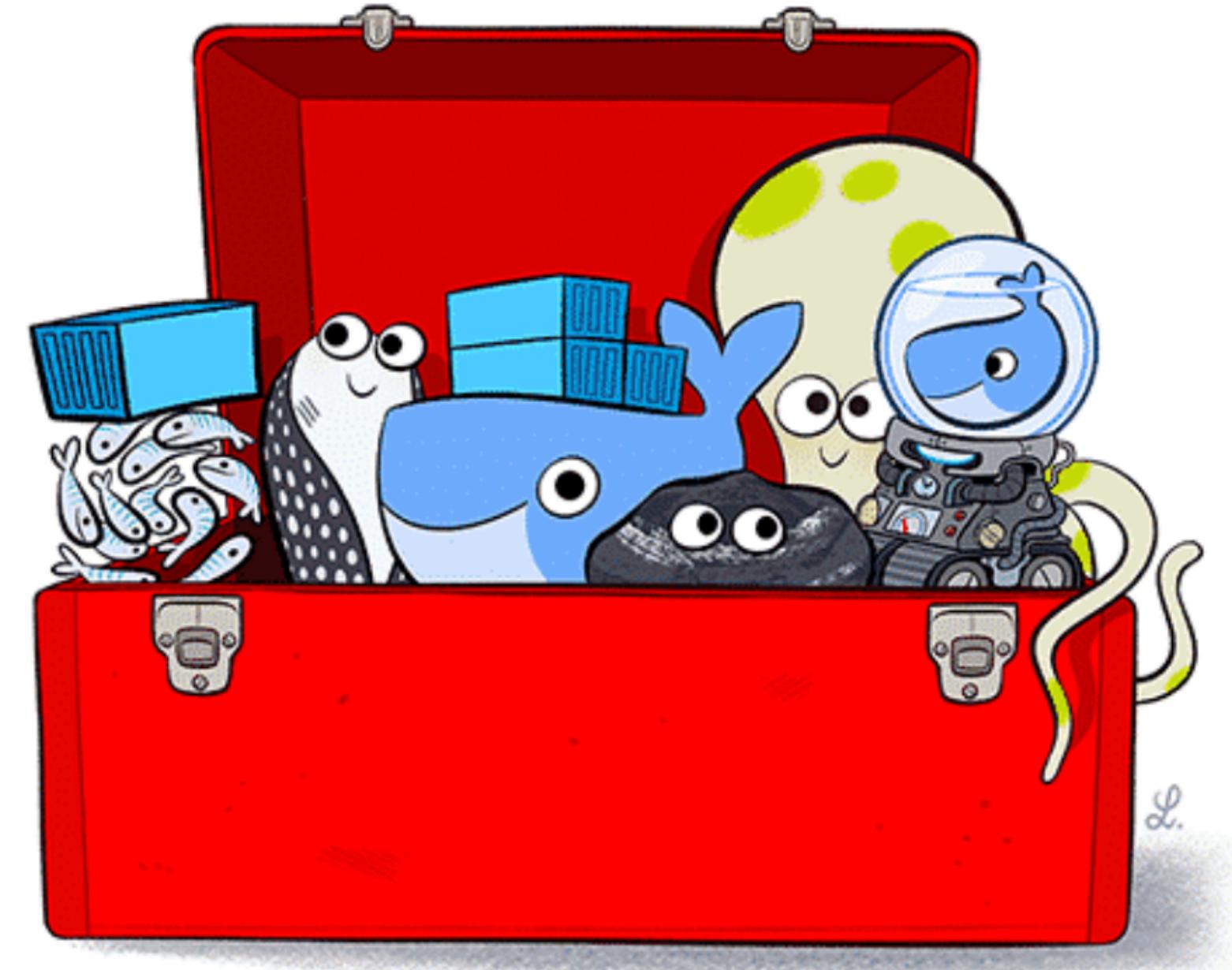
Execução do container (backend)

- docker run -ti -m 256MB --rm -e MYSQL_HOST=mysqltodo -e MYSQL_USR=root -e MYSQL_PWD=senha_difícil -p 8081:8080 --network todoapp {usuario_docker_hub}/backend-todo
 - -m 256MB memória máxima disponível ao container
 - -e MYSQL_HOST (host do BD)
 - -e MYSQL_USR (usuario do BD)
 - -e MYSQL_PWD (senha do BD)
 - -p 8081 (porta do host) para a porta do container (8080)
 - –network todo app

Distribuindo Imagens

Docker hub

- Maior biblioteca e comunidade de imagens para containers;
- Exemplos de imagens:
 - hello-world
 - docker/getting-started
 - openjdk
 - MySQL
 - ... e muitas outras



Passos para distribuição

- Criação de Conta Docker Hub;
- Dockerfile;
- Construção do Dockerfile localmente;
- Envio da imagem construída para o Docker Hub.

Publicando nossas imagens

Publicando uma imagem

- docker push {imagem}:{tag}

Fazendo download da imagem

- docker pull {imagem}:{tag}

Tag ou versão:

- {tag} parâmetro opcional em ambos os passos.
 - {tag} valor padrão: latest

Docker-compose

- Executando apps multi-container de forma simplificada;
- Nosso aplicativo TODO:
 - Backend-todo
 - Frontend-todo
 - BD: Mysql
- Docker-compose proporciona configurar e executar todos os containers;

Curiosidades e Outros Desafios

- Docker foi desenvolvido utilizando a linguagem Go;
- Docker em modo Swarm;
- Kubernetes (K8s).

Dúvidas e Perguntas

Referências

<https://docs.docker.com>

Fale comigo no LinkedIn

