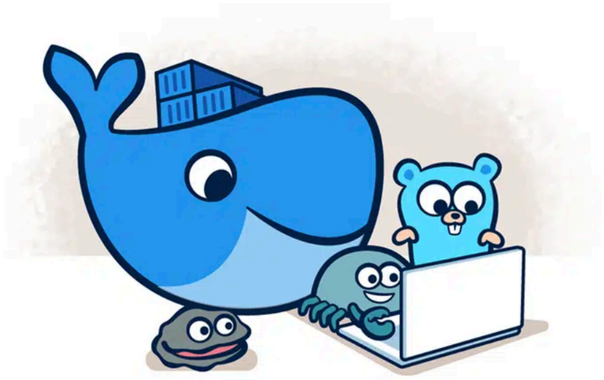
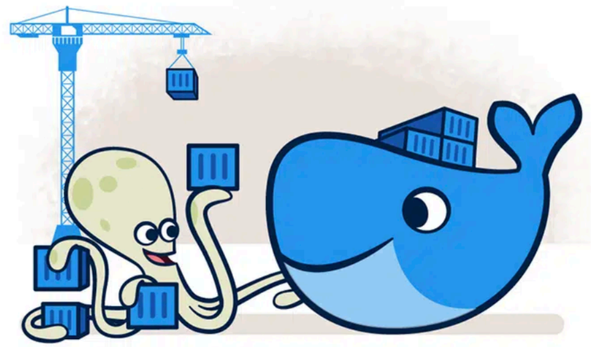
Criando e Gerenciando Containers com o Docker

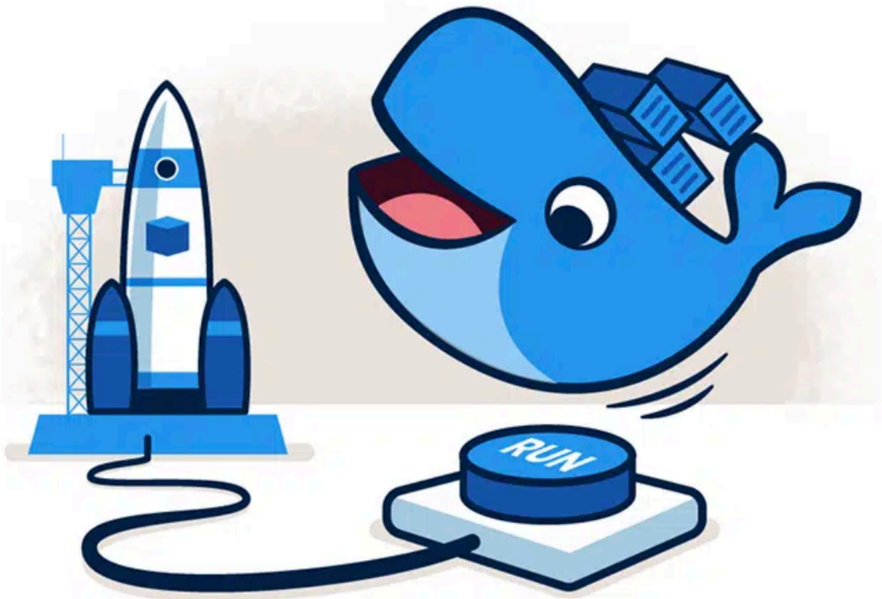


Marcus Adriano



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

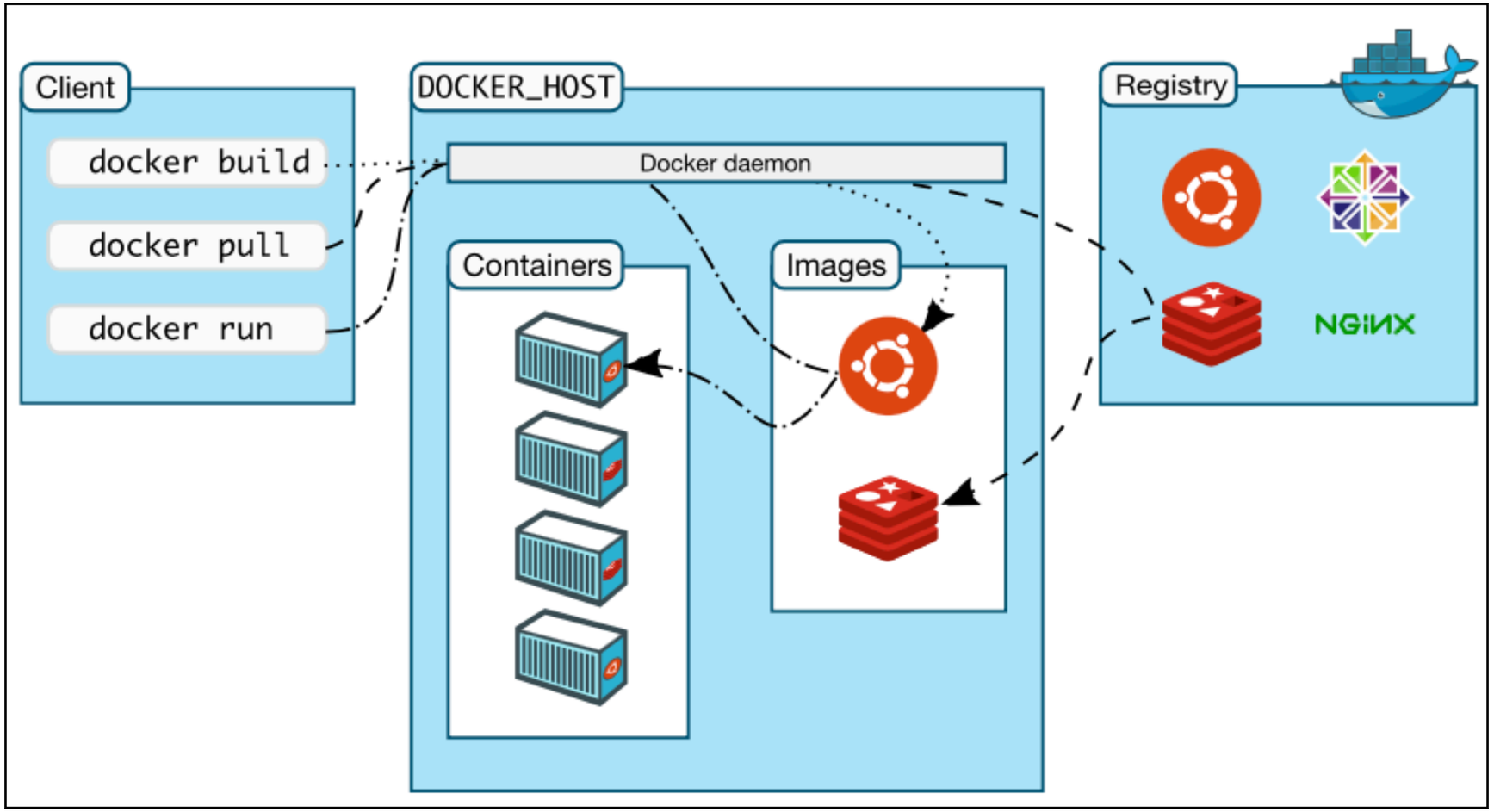
- Plataforma open source para desenvolver, entregar e executar apps;
- Separação da infraestrutura da app;
- Unidade básica: ***container*** (contém tudo que a app precisa);
- Distribuição é simples e fácil (e funciona em qualquer lugar);
- *Containers* dividem os recursos do mesmo ***host;***

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 utilizam 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.

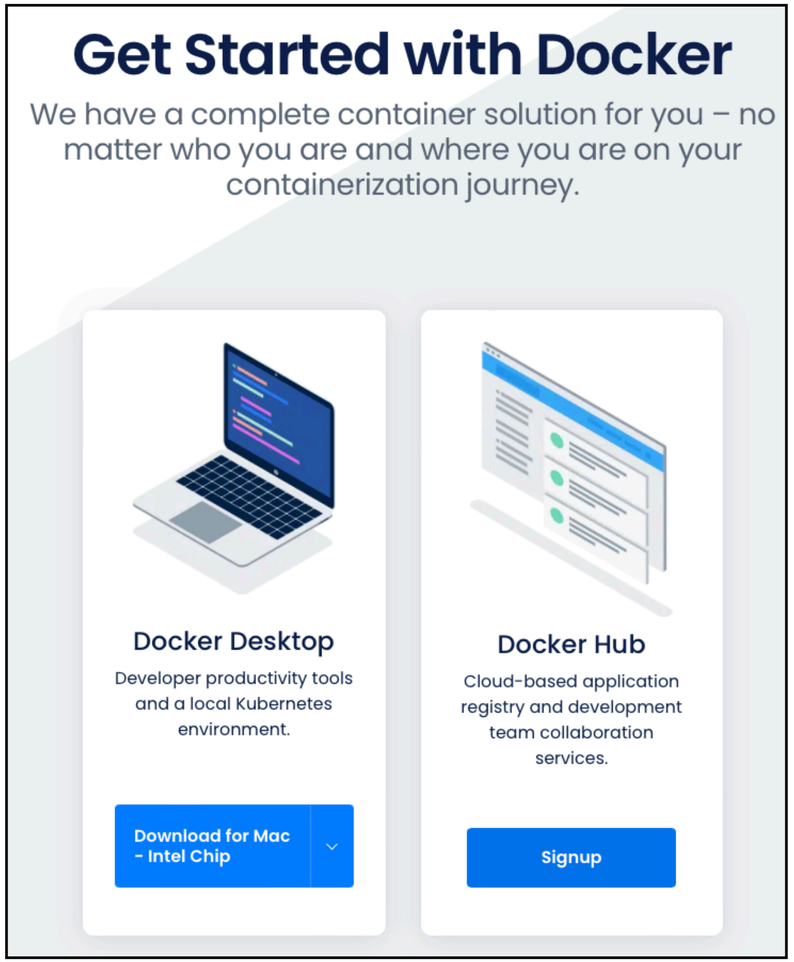
Docker: Objetos

- 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.

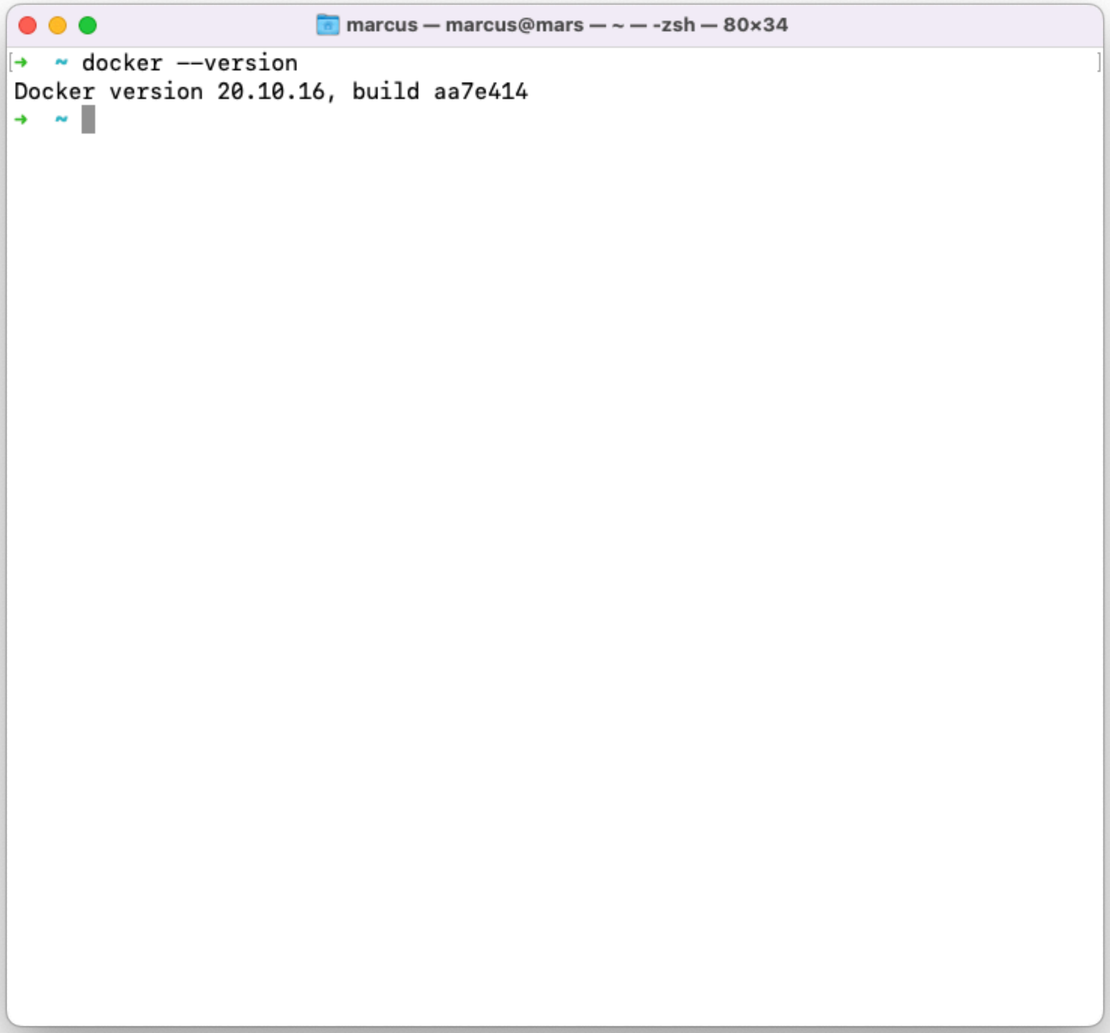
Primeiros Passos

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



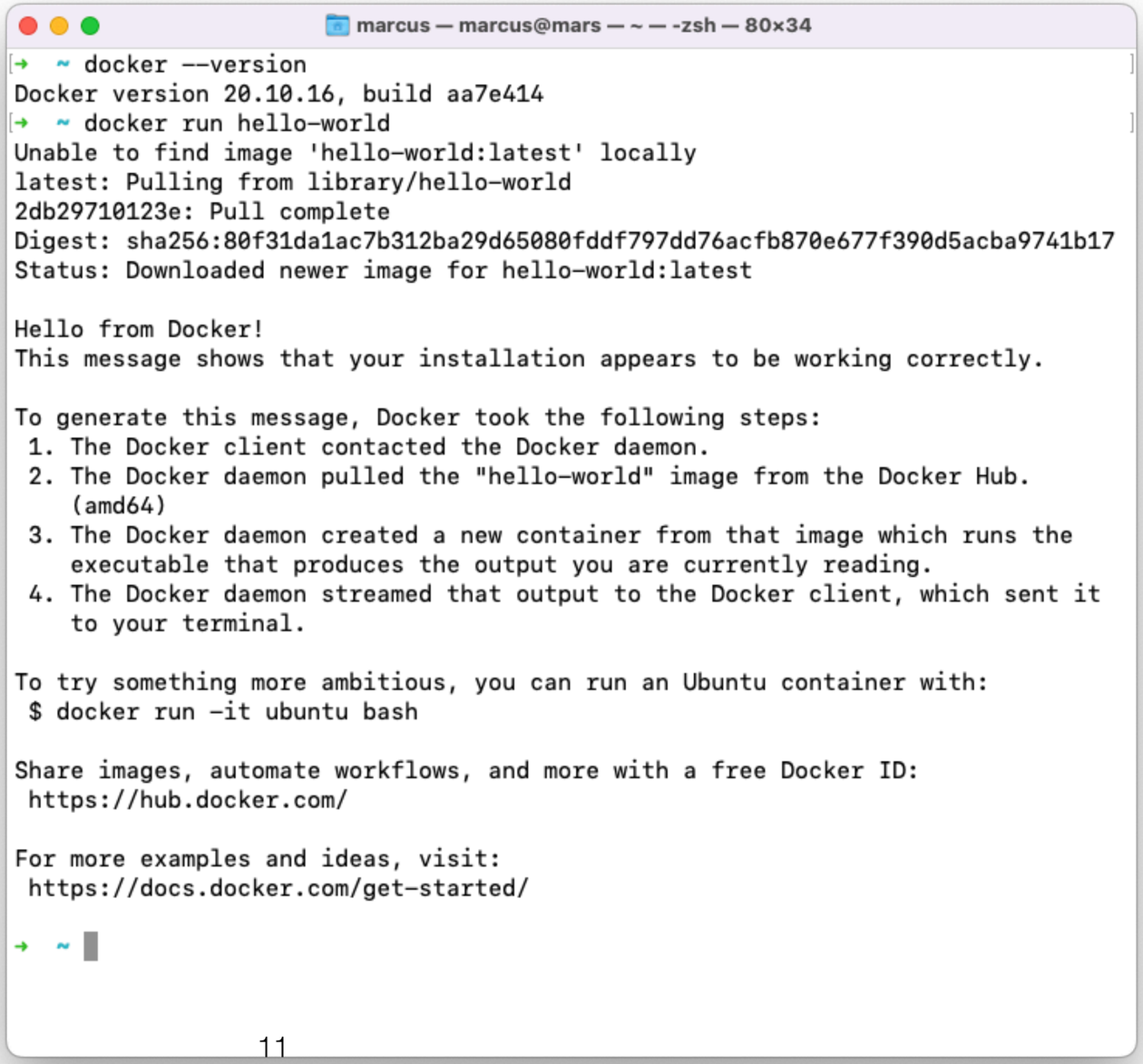
Verificando a instação

- docker —version



Verificando a instalação

- docker run hello-world

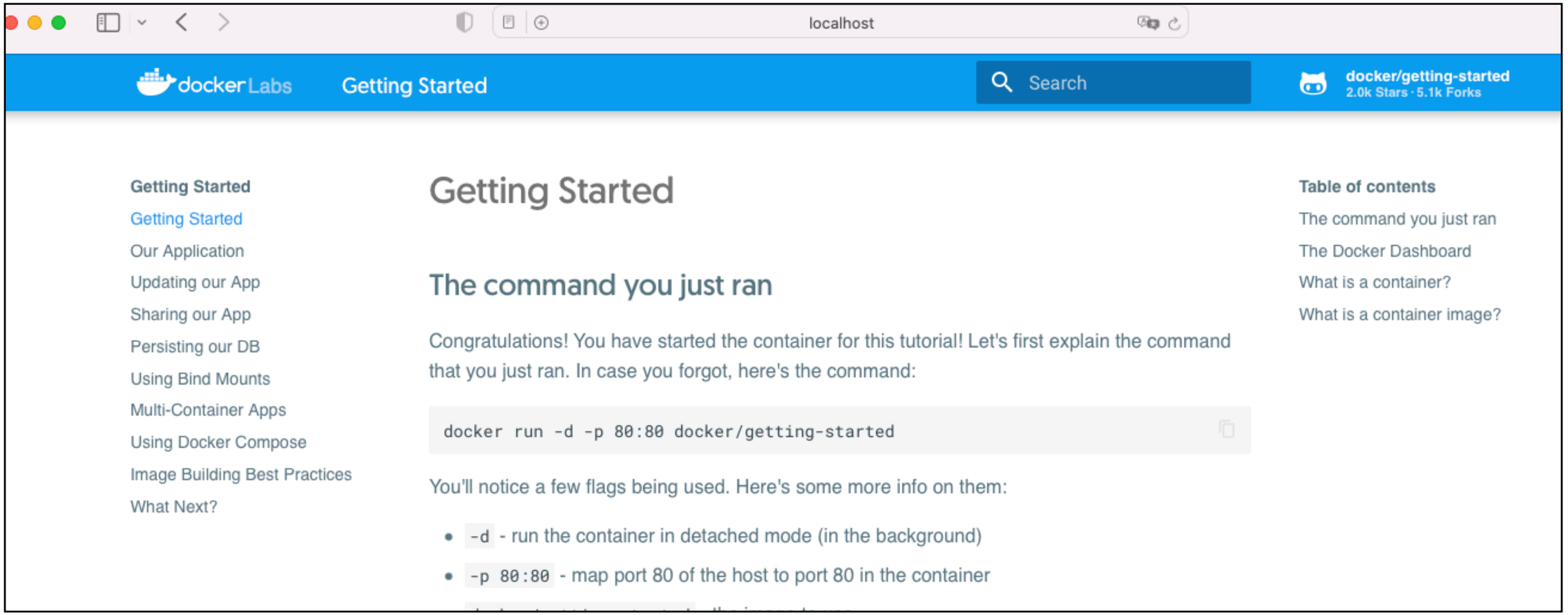


Executando um container

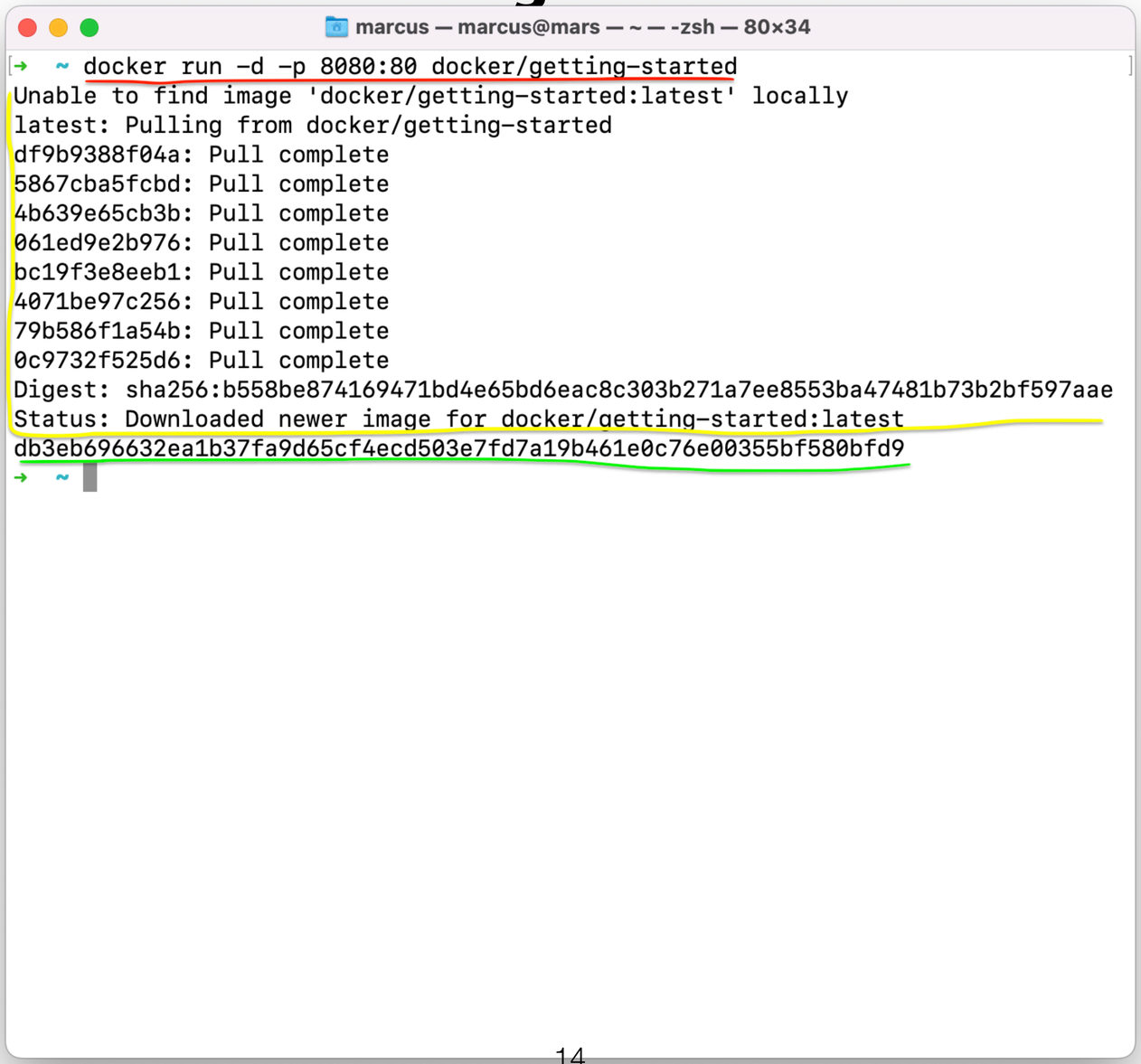
- docker run -d -p 8080:80 docker/getting-started
- -d: execução em segundo plano
- -p: mapeia a porta 8080 do host para a porta 80 do container
- docker/getting-started: imagem

Conferindo a execução

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



Sobre a execução do container

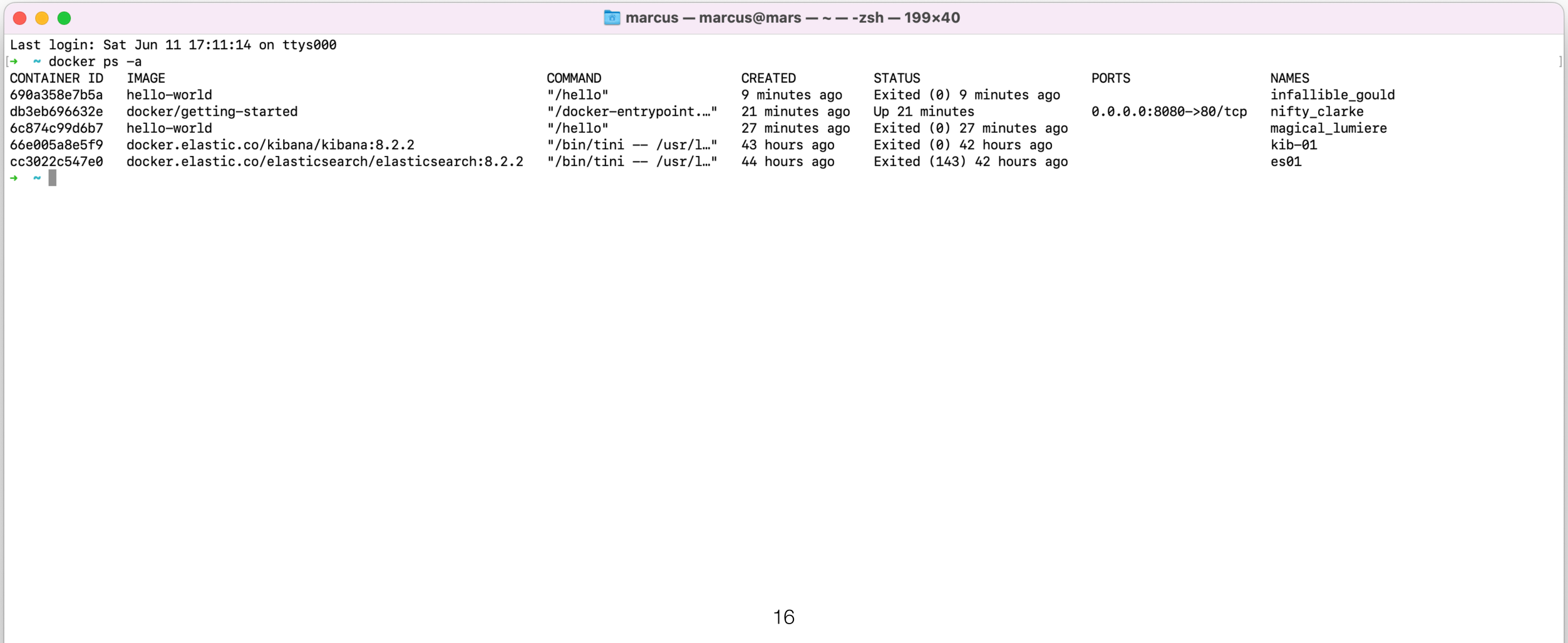


Listando os containers ativos

- docker ps

Listando os container ativos e inativos

- docker ps -a



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 (defaul
  -c, --context string     Name of the context to use to connect t
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"
      --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

```
→  ~ docker ps
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

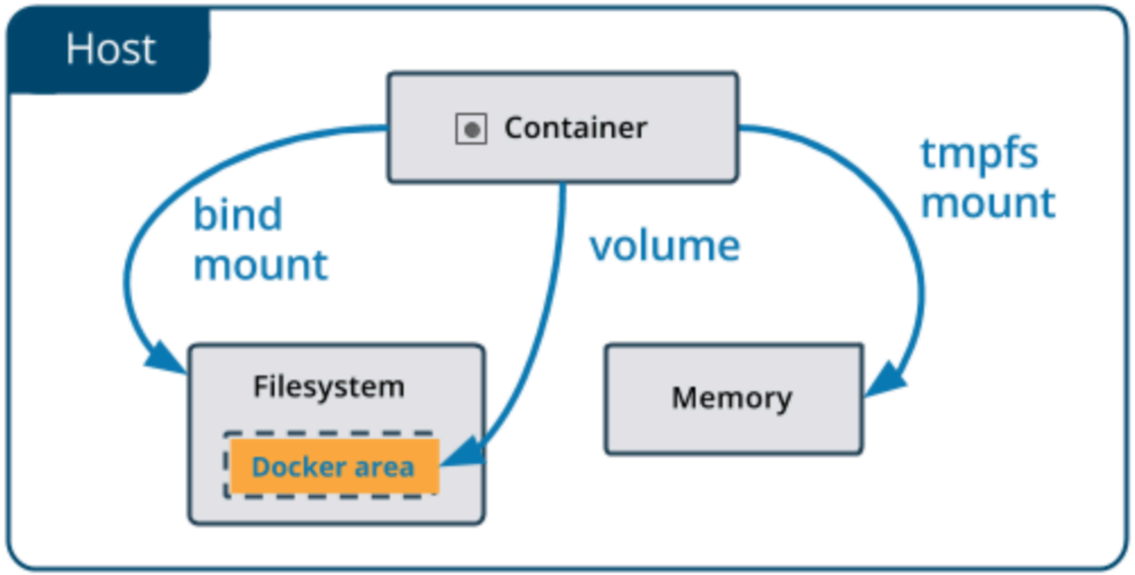
```
→  ~ docker stop 16d
16d
→  ~ docker start 16d9
16d9
→  ~ docker restart 16d9d8ff03d1
16d9d8ff03d1
→  ~
```

```
→  ~ 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

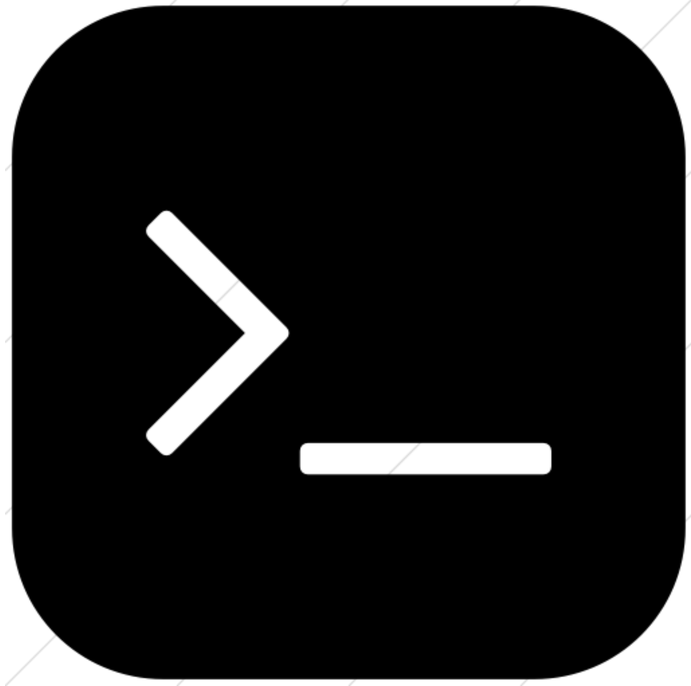
- docker run --name some-mysql -v {host_dir}:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql
    - —name nome do container
    - -v mapa de volumes para o container {host_dir}:/var/lib/mysql
    - -e variável de ambiente chave=valor
    - -d execução em segundo plano
- Os dados do MySQL serão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

Sobre a imagem do MySQL:

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

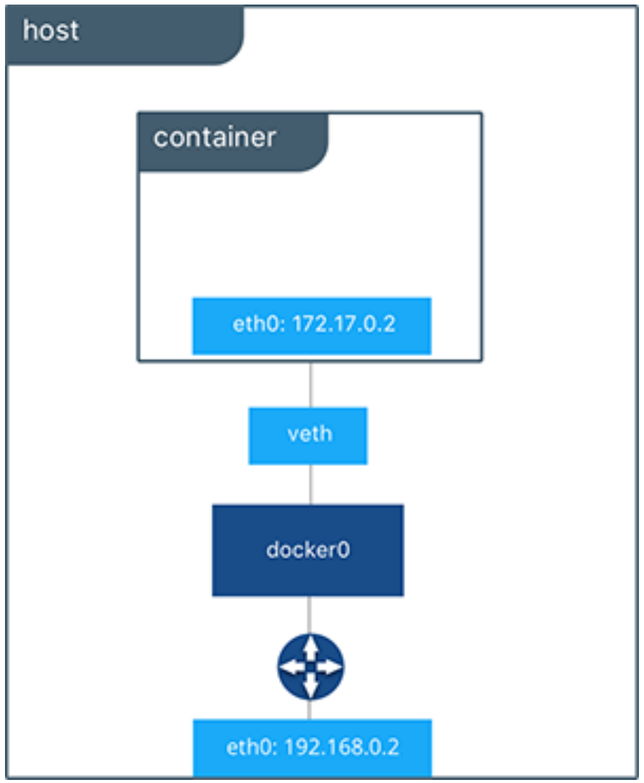
Acessando um Container

- 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

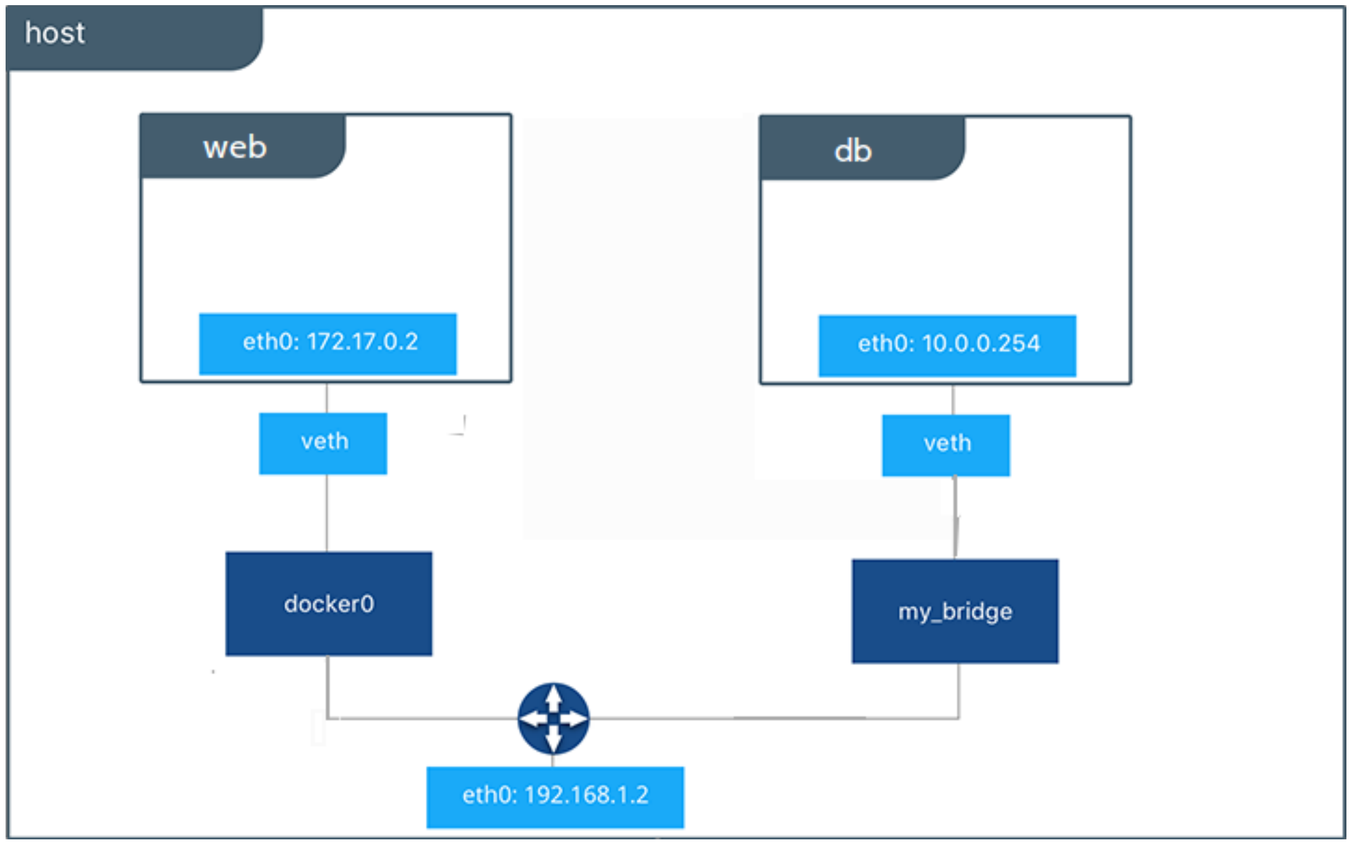
- Containers por padrão são alocados na mesma rede



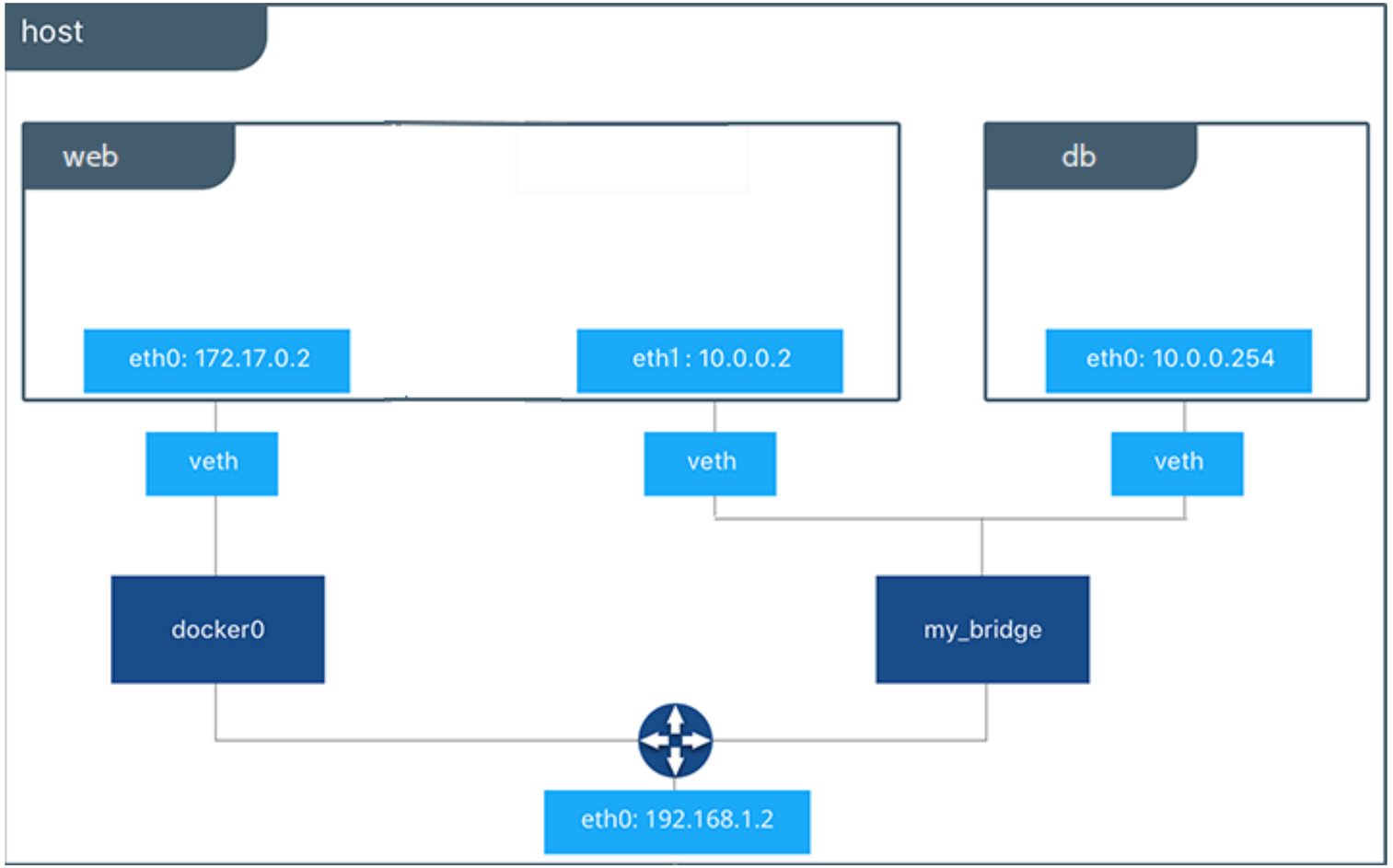
Rede em containers

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

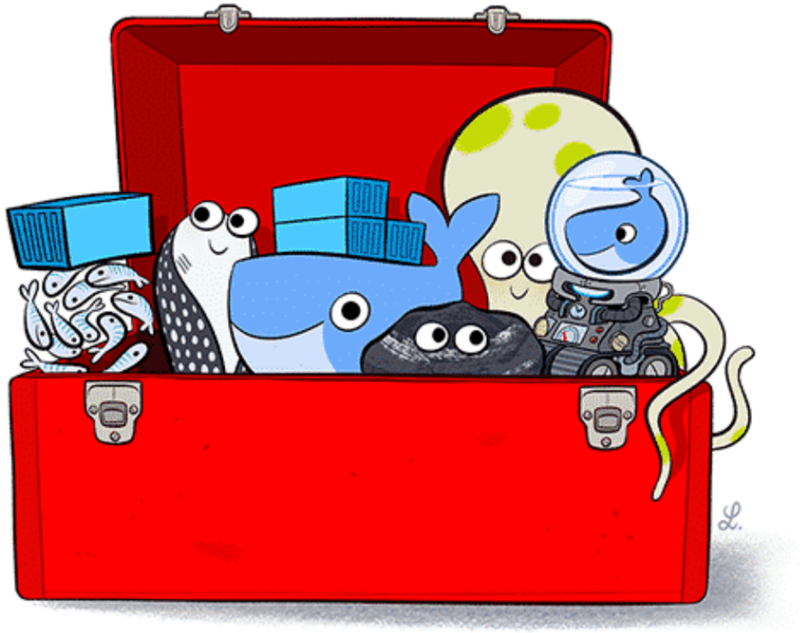
Execução do container (backend)

- docker run -ti -m 256MB --rm -e MYSQL_HOST=mysqltodo -e MYSQL_USR=root -e MYSQL_PWD=senha_dificil -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