PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

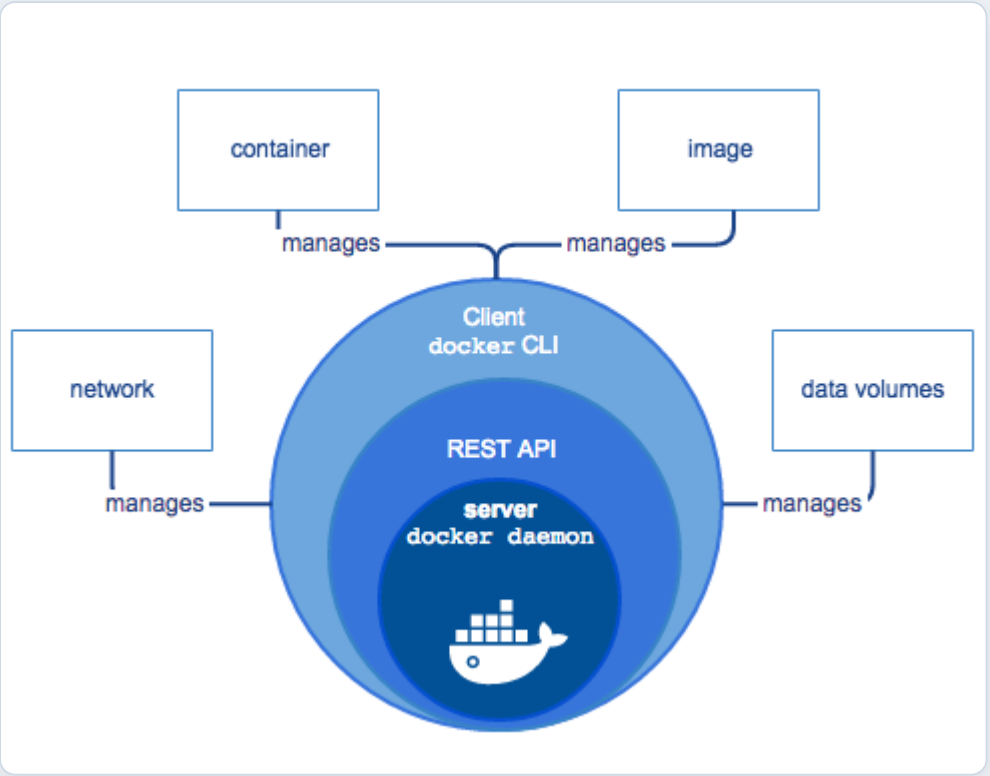
Arquitetura do Docker

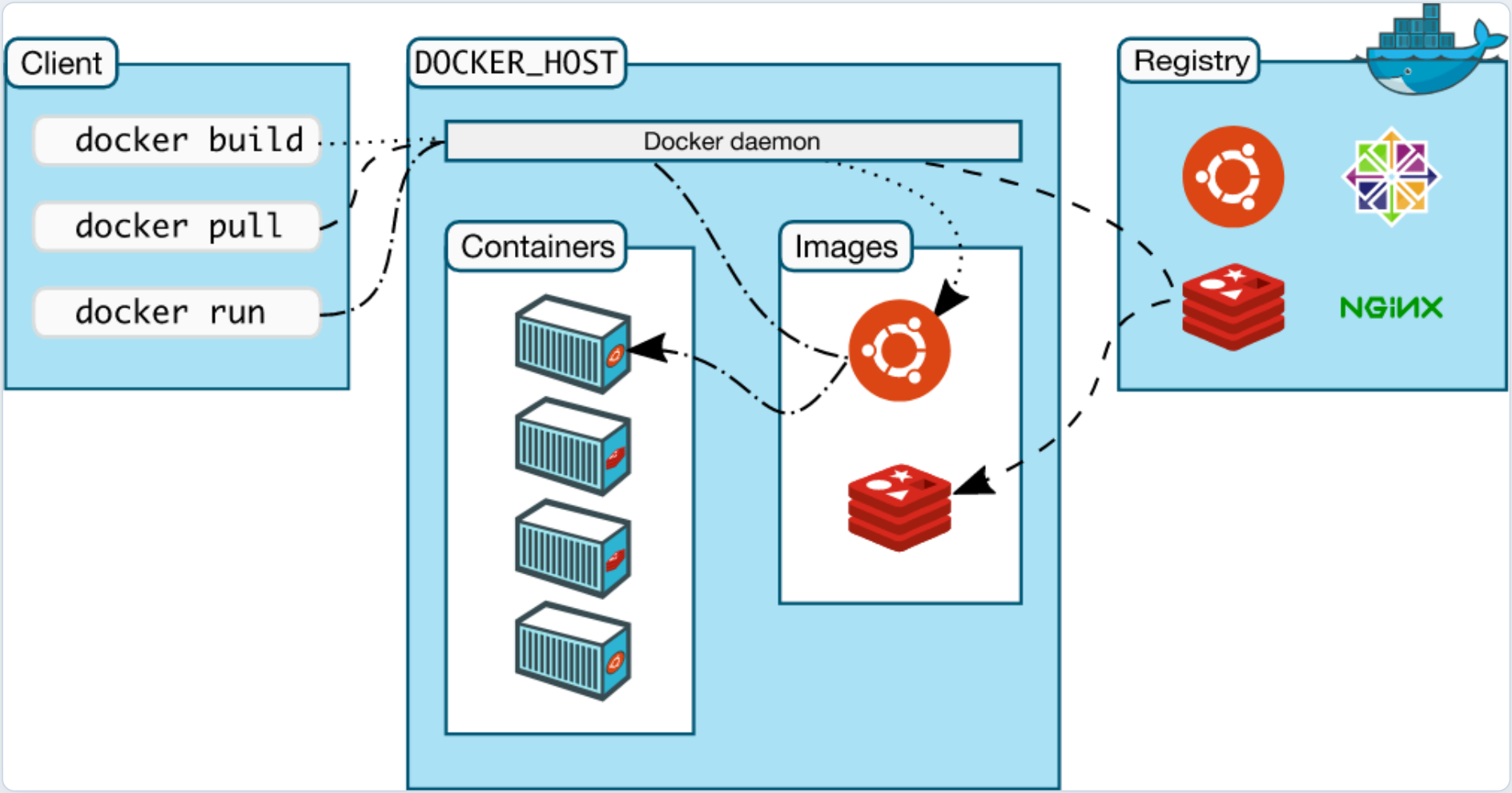
Aula 04 — Componentes, objetos e Dockerfile

Prof. Newton S. B. Miyoshi
newton.miyoshi@baraodemaua.br
Centro Universitário Barão de Mauá — Ribeirão Preto

Arquitetura cliente-servidor

- O cliente `docker` comunica-se com o `docker daemon`:
  - É possível conectar remotamente
  - Comunicação pela REST API
  - Via sockets UNIX ou pela rede (internet)


container
image
manages
manages
Client
docker CLI
network
REST API
data volumes
manages
server
docker daemon
manages


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

Daemon Docker — `dockerd`

- Daemon: processo que executa em segundo plano.
  - Também chamado de serviço, task ou ghost process
  - Tradicionalmente o nome termina com "d" — `syslogd`, `sshd`, `systemd`, `crond` ...
- Aguarda requisições da API Docker e executa de fato os comandos.
- Pode se comunicar com outros daemons para gerenciar serviços Docker.

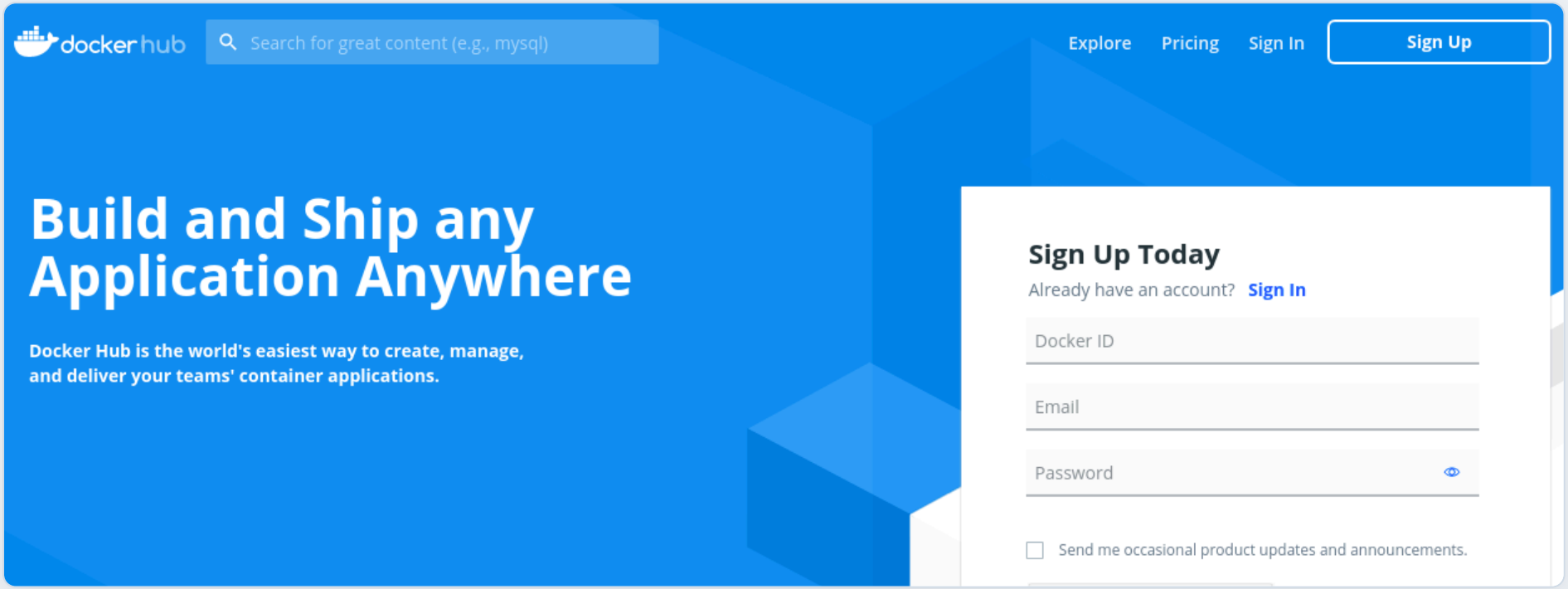
Cliente Docker — `docker`

- É a principal maneira como os usuários interagem com o Docker.
- Comandos como `docker run` enviam, via API Docker, o comando correspondente ao `dockerd`.
- Um cliente pode se comunicar com mais de um daemon.

Docker Registries

- Responsável pelo armazenamento e distribuição das imagens (Docker Image Repository).
- Docker Hub é um registro público que qualquer um pode usar.
  - Por padrão, o Docker procura imagens no Docker Hub
  - É possível ter um registro particular
- `docker pull` e `docker push` baixam e enviam imagens do/para o registro.

Docker Hub


docker hub
Search for great content (e.g., mysql)
Explore
Pricing
Sign In
Sign Up
Build and Ship any Application Anywhere
Docker Hub is the world's easiest way to create, manage, and deliver your teams' container applications.
Sign Up Today
Already have an account? Sign In
Docker ID
Email
Password
Send me occasional product updates and announcements.

OBJETOS DOCKER

Imagens, Containers e Serviços

Os principais objetos do Docker

Imagens

Templates read-only para criar containers.

Containers

Instâncias executáveis de uma imagem.

Serviços

Containers escalados em uma swarm.

Imagens Docker

- Containers são, basicamente, um processo em execução.
- São construídos a partir de imagens.
- Imagens incluem tudo o que é necessário para rodar a aplicação de modo isolado:
  - Sistema de arquivos, binários, runtime, dependências e bibliotecas

Imagens Docker

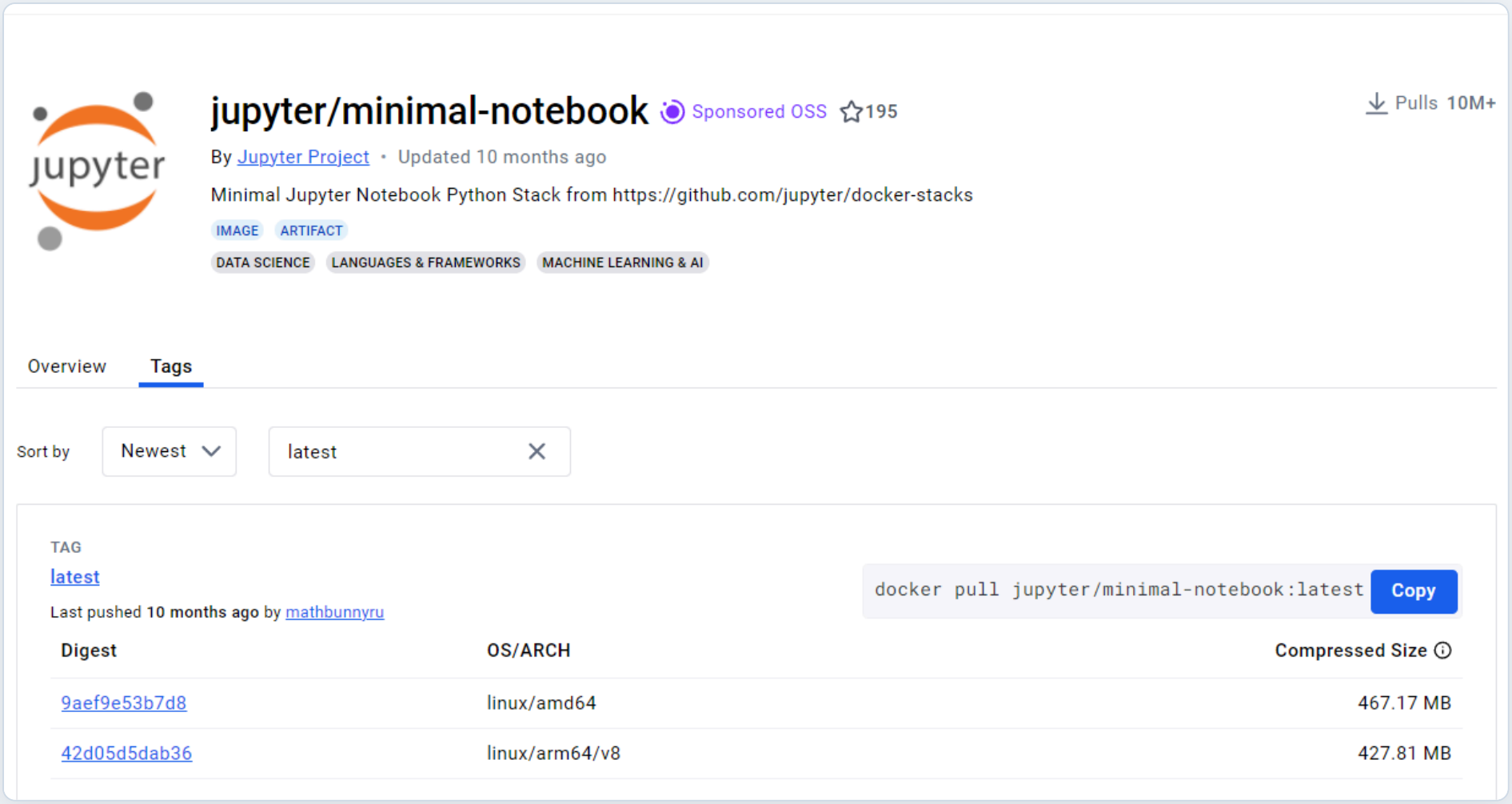
- Uma imagem é um template read-only com instruções para criar um container.
- Uma imagem é baseada em outra (customização) — ex.: uma imagem LAMP baseada na imagem `ubuntu`.
- Você pode criar suas próprias imagens e publicá-las no registro.
- A criação é feita por um arquivo Dockerfile:
  - Passo a passo para construir a imagem
  - Ao alterá-lo, só as partes modificadas são reconstruídas

Containers Docker

- Instância executável de uma imagem. A partir dele pode-se:
  - Conectar a uma ou várias redes
  - Adicionar armazenamento
  - Criar novas imagens a partir do estado atual
- Mantém-se isolado dos outros containers e do host.
- Ao ser removido, as mudanças não persistidas são perdidas.

Nome de uma imagem

namespace / nome_da_imagem : tag



Baixando uma imagem — `docker pull`

```
$ docker pull jupyter/minimal-notebook:latest
latest: Pulling from jupyter/minimal-notebook
aece8493d397: Pull complete
fd92c719666c: Pull complete
088f11eb1e74: Pull complete
4f4fb700ef54: Pull complete
... (várias camadas) ...
Digest: sha256:1c4c8b6c7c27059c353d4e80523c2696e34723fde67d27418873ebeb42032551
Status: Downloaded newer image for jupyter/minimal-notebook:latest
docker.io/jupyter/minimal-notebook:latest
```

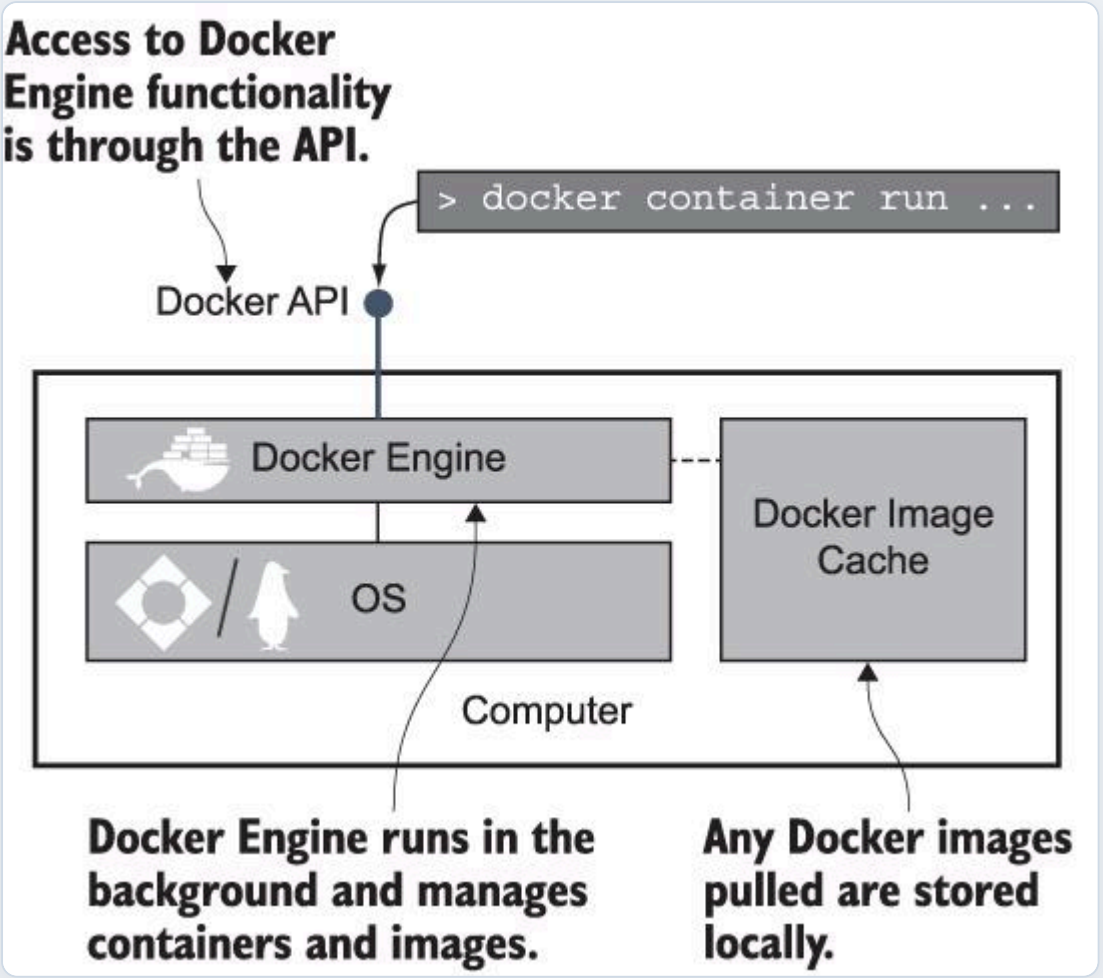
Listar e executar

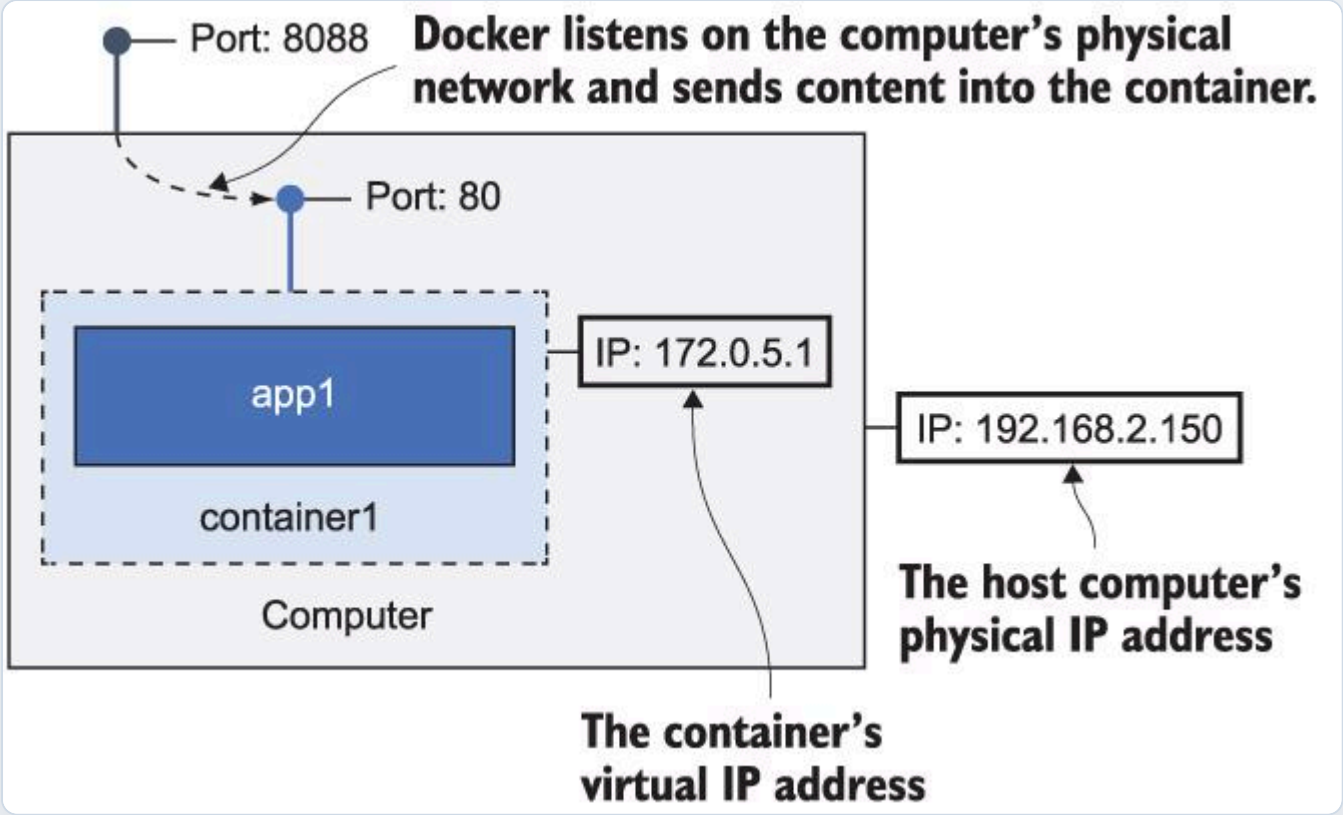
```
$ docker image ls        # lista as imagens baixadas
REPOSITORY                TAG      IMAGE ID       CREATED         SIZE
welcome-to-docker         latest   7a138ca1b41b   3 days ago      229MB
jupyter/minimal-notebook  latest   e75d34b8f32b   10 months ago   1.56GB
hello-world               latest   d2c94e258dcb   16 months ago   13.3kB

$ docker run jupyter/minimal-notebook:latest    # CTRL+C para parar
```

Mapeando portas e volumes

```
$ docker run -p porta_host:porta_container namespace/imagem:tag

# exemplos
$ docker run -p 8888:8888 --name lab jupyter/minimal-notebook:latest
$ docker run -p 8888:8888 -v .\lab_data\:/home/jovyan/work jupyter/minimal-notebook:latest
```

Outros parâmetros: `--name nome_container` · `-v path_host:path_container`


Access to Docker Engine functionality is through the API.
> docker container run ...
Docker API
Docker Engine
OS
Docker Image Cache
Computer
Docker Engine runs in the background and manages containers and images.
Any Docker images pulled are stored locally.


Port: 8088
Docker listens on the computer's physical network and sends content into the container.
Port: 80
app1
container1
IP: 172.0.5.1
IP: 192.168.2.150
Computer
The host computer's physical IP address
The container's virtual IP address

Serviços Docker

- Permitem escalar containers entre vários daemons que trabalham juntos em uma swarm (enxame), com managers e workers.
- Cada membro da swarm é um daemon e usa a API Docker para se comunicar.
- Define-se um estado desejado para a colmeia:
  - Quantidade mínima de réplicas em execução
  - Balanceamento de carga entre os nós
- Quem consome o serviço aparenta falar com uma única aplicação.

MÃO NA MASSA

Dockerfile — criando um container

Dockerfile

- Arquivo texto com os comandos para construir uma imagem.
  - Imagens são construídas a partir de outras imagens
  - Sintaxe e comandos específicos (ver a reference)
- Cada linha é um comando que cria uma camada (layer).
  - Imagens são construídas por camadas read-only

Exemplo — imagem baseada em Ubuntu

```
1 FROM ubuntu:18.04
2 COPY . /app
3 RUN make /app
4 CMD python /app/app.py
```

Exemplo — imagem baseada em Node

```
1 FROM node:20-alpine
2 RUN mkdir /code
3 COPY . /code
4 WORKDIR /code
5 RUN yarn install
6 EXPOSE 8000
7 CMD [ "yarn", "dev", "-p", "8000" ]
```

Buildando a imagem

```
$ docker build -t hello-world-docker .
Sending build context to Docker daemon  7.825MB
Step 1/7 : FROM node:20-alpine
 ---> bbb0e90a28e0
Step 2/7 : RUN mkdir /code
 ---> Using cache
 ---> cf16256b0f20
Step 3/7 : COPY . /code
 ---> 3669a4ec88f6
...
Step 7/7 : CMD [ "yarn", "dev", "-p", "8000" ]
Successfully built 3d68dcc2bd62
Successfully tagged hello-world-docker:latest
```

Criando o container

```
$ docker run -d -it -p 8000:8000 hello-world-docker
789f28254ac8a0c9bc3ec4f977acc9687e014b0c7de8f9b183d0a15d124c6498
```

`-d` executa em segundo plano (detached); a saída é o ID do container.

Comando de build

```
$ docker build -t nome_tag caminho_para_dockerfile

# exemplo
$ docker build -t fastapi_exemplo .
```

ATIVIDADE

A04 — Dockerfile

Crie o seu próprio Dockerfile, faça o build da imagem e suba o container.

Veja o link para o GitHub Classroom.

Referências

- Dockerfile Reference — docs.docker.com/engine/reference/builder
- Dockerfile Best Practices — docs.docker.com/.../dockerfile_best-practices