PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Docker Hub e Boas Práticas

Aula 06 — Publicando imagens e construindo com múltiplos estágios

Prof. Newton S. B. Miyoshi

newton.miyoshi@baraodemaua.br

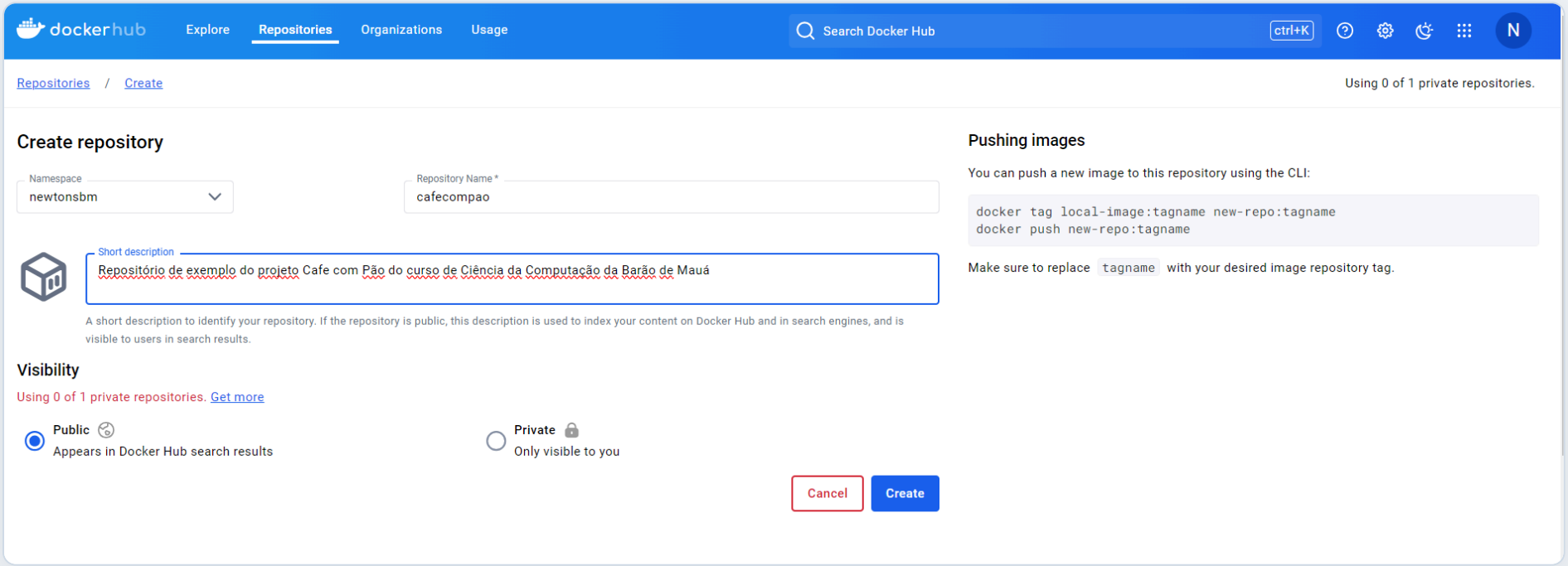
Centro Universitário Barão de Mauá — Ribeirão Preto

PARTE 1

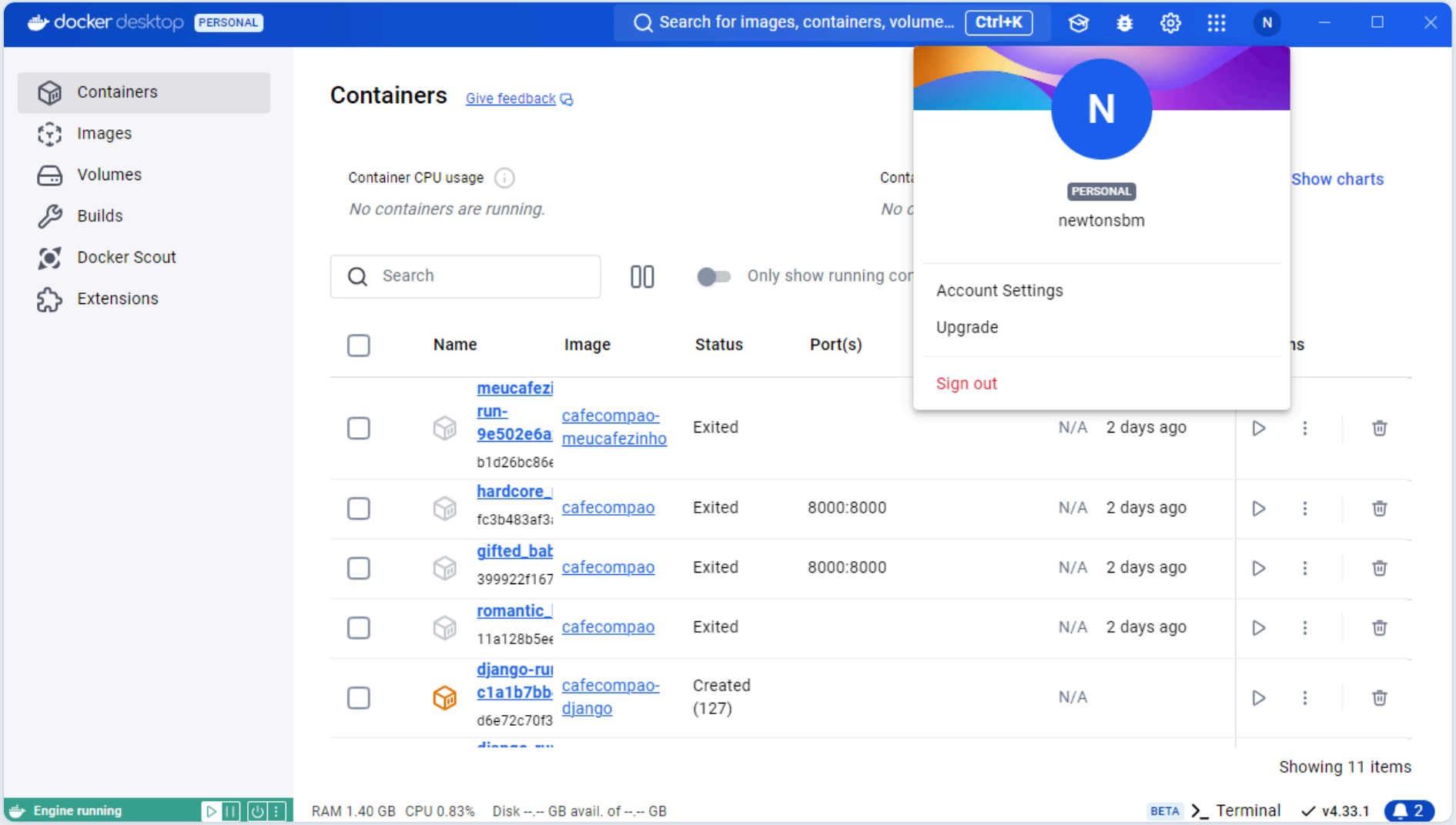
Docker Hub

Publicando uma imagem com `docker compose`.

Enviando com Docker Compose — criar o repositório



Esteja logado no Docker Desktop



Enviando com Docker Compose — propriedade `image`

```yaml
services:
  meucafezinho:
    build: .
    image: newtonsbm/cafecompao_aula
    ports:
      - "8000:8000"
    command: /bin/sh -c "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"
```

No serviço que será enviado, `image` segue o padrão `nome_usuario/repositorio` referente ao Docker Hub.

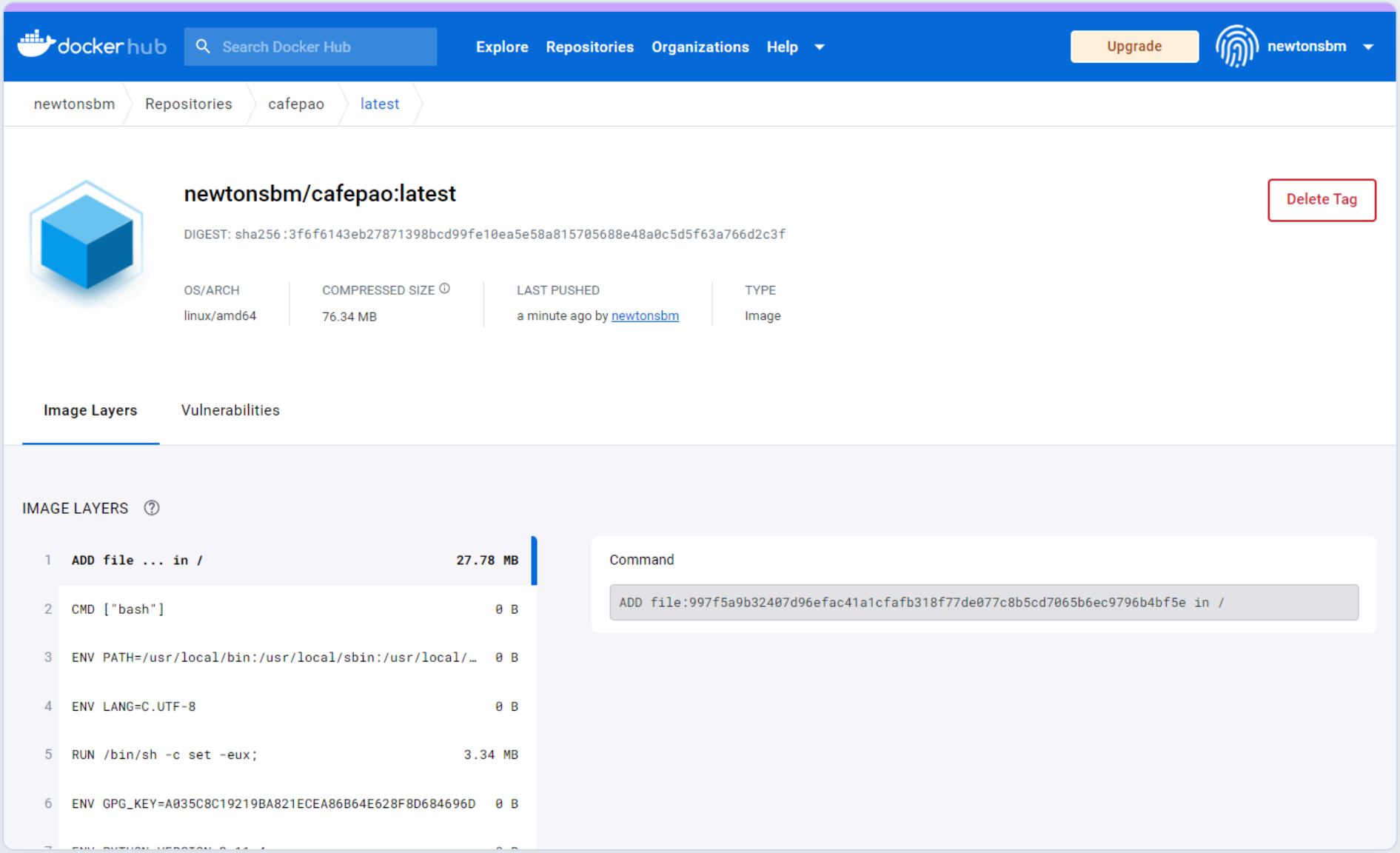
Garanta que a imagem foi buildada

```
$ docker compose build
[+] Building 12.5s (8/10)
 => [meucafezinho 1/5] FROM docker.io/library/python:3-alpine
 => [meucafezinho 2/5] COPY . /app
 => [meucafezinho 3/5] WORKDIR /app
 => [meucafezinho 4/5] RUN pip install -r requirements.txt
 => [meucafezinho 5/5] ...
```

Envie para o Docker Hub

```
$ docker compose push meucafezinho
[+] Pushing 9/9
 - Pushing newtonsbm/cafecompao_aula: 2f44f485586a Pushed
 - Pushing newtonsbm/cafecompao_aula: 699a8ba52c91 Pushed
 - Pushing newtonsbm/cafecompao_aula: 5f70bf18a086 Mounted from newtonsbm/cafecompao_exemplo
```

Verifique no Docker Hub

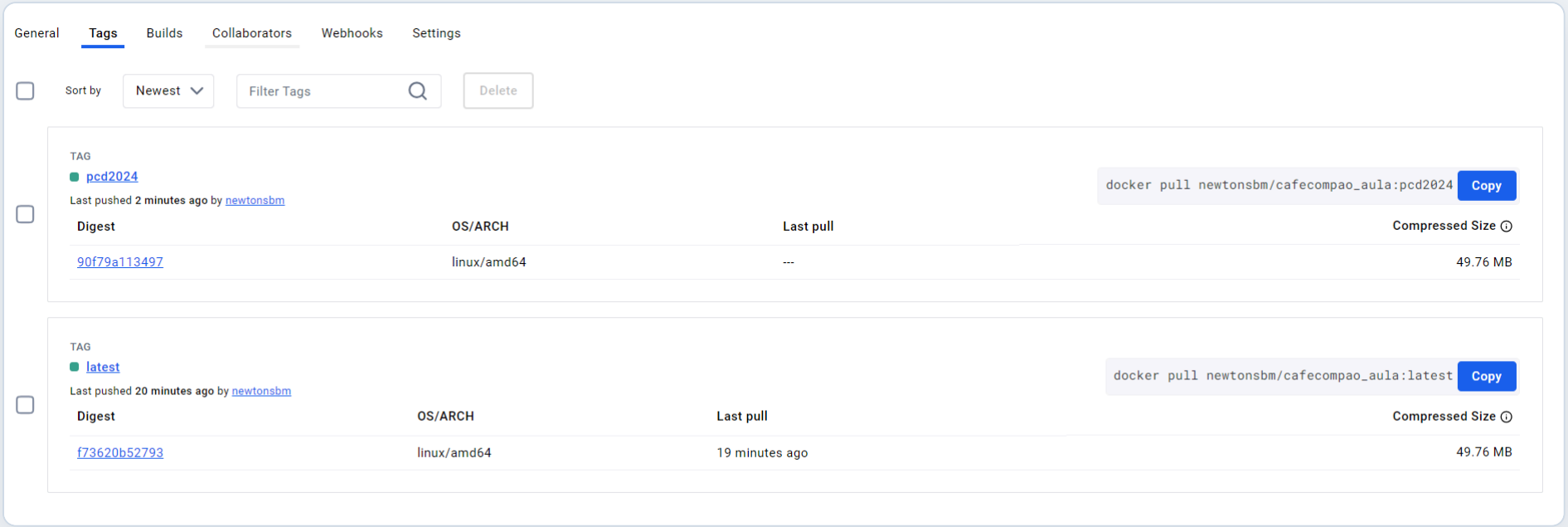


Opcional: `tag`

```yaml
services:
  meucafezinho:
    build: .
    image: newtonsbm/cafecompao_aula:pcd2024
    ports:
      - "8000:8000"
```

A tag é opcional — quando omitida, o padrão é `latest`.

Tags no Docker Hub



PARTE 2

Boas Práticas — Multi-stage builds

Imagens menores, separando build de execução.

Use builds de múltiplos estágios

- Muitas vezes precisamos de mais de uma imagem para uma aplicação.
- Uma imagem para fazer o build, outra para fazer o deploy
  - ex.: Maven para Java, Tomcat para deploy.
- Vamos ver um exemplo: aplicação em Go, com build no `golang` e deploy no `alpine`.

Usando um estágio anterior para construir outro

```
1  FROM alpine:latest as builder
2  RUN apk --no-cache add build-base
3
4  FROM builder as build1
5  COPY source1.cpp source.cpp
6  RUN g++ -o /binary source.cpp
7
8  FROM builder as build2
9  COPY source2.cpp source.cpp
10 RUN g++ -o /binary source.cpp
```

Os estágios `build1` e `build2` reaproveitam o estágio `builder` — sem repetir a instalação das dependências.

EXEMPLO PRÁTICO

Passo a passo — Pelican SSG

- Criar `Dockerfile`.
- Criar `compose.yaml`.
- Gerar o projeto Pelican.
- Configurar.

Gerando o projeto — `pelican-quickstart`

```
$ docker compose run -it cafeblog pelican-quickstart
Welcome to pelican-quickstart v4.9.1.

> Where do you want to create your new web site? [.]
> What will be the title of this web site? Cafezando
> Who will be the author of this web site? Newton Miyoshi
> Do you want to specify a URL prefix? (Y/n) y
  blog.cafecompao.com.br
Done. Your new project is available at /app
```

Docker Multi Stage — Pelican: `Dockerfile` (single / dev)

```
FROM python:3-slim
RUN mkdir /app
COPY . /app
WORKDIR /app
RUN apt-get update
RUN apt-get install -y git
RUN pip install pelican[markdown] git+https://github.com/arulrajnet/attila.git@master
```

Uma única imagem: Python + Git + todas as dependências de build, também usada para servir o site em desenvolvimento.

Docker Multi Stage — Pelican: `DockerfileProd` (multi / prod)

```dockerfile
FROM python:3-slim AS builder
RUN mkdir /app
COPY . /app
WORKDIR /app
RUN apt-get update
RUN apt-get install -y git
RUN pip install pelican[markdown] git+https://github.com/arulrajnet/attila.git@master
RUN pelican content -o output -s publishconf.py

FROM nginx:1-alpine
COPY --from=builder /app/output/ /usr/share/nginx/html
```

O estágio `builder` gera o site estático; só o resultado (`output/`) vai para a imagem final do nginx — Python e Git ficam de fora.

`compose.yaml` — dev (single-stage)

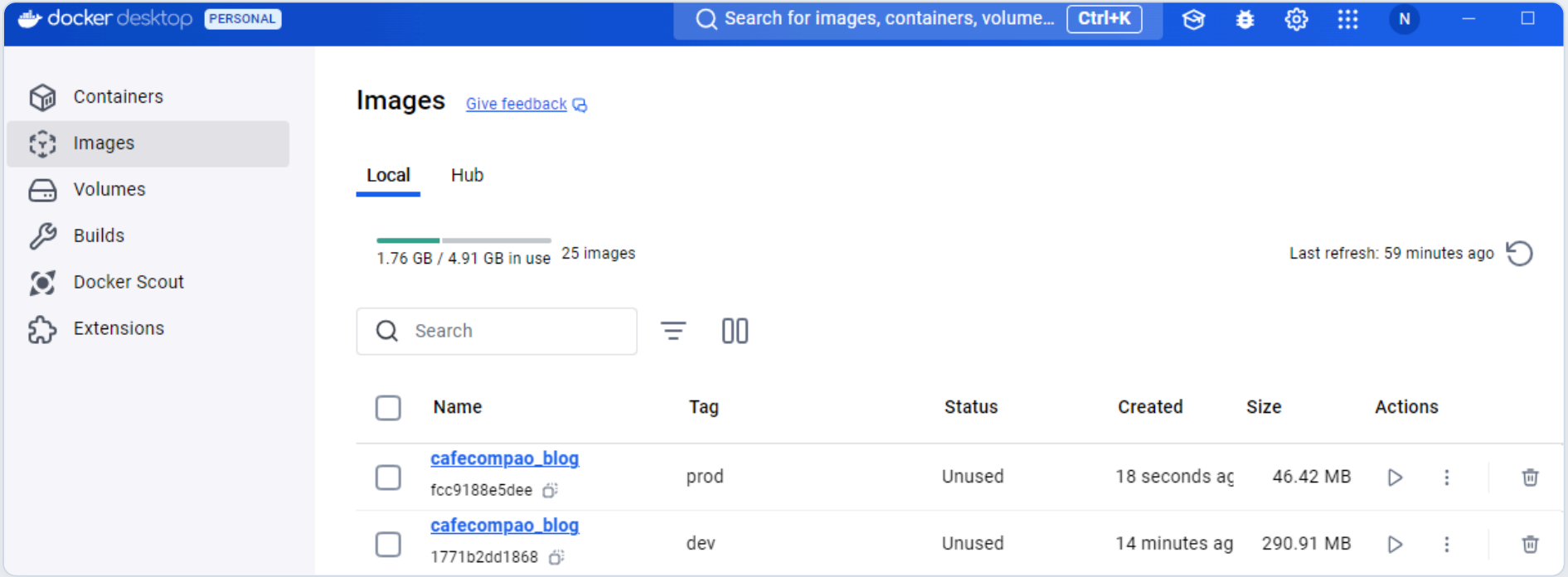
```yaml
services:
  cafeblog:
    build: .
    image: cafecompao_blog:dev
    ports:
      - "3000:3000"
    volumes:
      - .:/app
    command: pelican --autoreload --listen --bind 0.0.0.0 --port 3000
```

`compose.prod.yaml` — prod (multi-stage)

```yaml
services:
  cafeblog:
    image: cafecompao_blog:prod
    build:
      context: .
      dockerfile: DockerfileProd
    ports:
      - "8080:80"
```

`docker compose -f compose.prod.yaml up` usa outro arquivo e outro `dockerfile` — o nginx passa a servir na porta 80.

Comparação


docker desktop
PERSONAL
Search for images, containers, volume...
Ctrl+K
N
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Images
Give feedback
Local
Hub
1.76 GB / 4.91 GB in use
25 images
Last refresh: 59 minutes ago
Search
Name
Tag
Status
Created
Size
Actions
cafecompao_blog
fcc9188e5dee
prod
Unused
18 seconds ag
46.42 MB
cafecompao_blog
1771b2dd1868
dev
Unused
14 minutes ag
290.91 MB

MATERIAL EM VÍDEO

YOUTUBE **Docker Multistage Build — O segredo pra enxugar suas imagens** youtube.com/watch?v=zP4lInIwfXg

Referências bibliográficas

- DOCKER. Docker Hub quickstart. docs.docker.com/docker-hub/quickstart
- DOCKER. Multi-stage builds. docs.docker.com/build/building/multi-stage