

FIAP GRADUAÇÃO

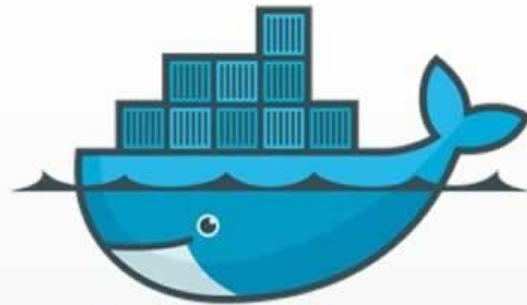
TECNOLOGIA EM DESENVOLVIMENTO DE SISTEMAS

DevOps Tools & Cloud Computing

Dockerfile

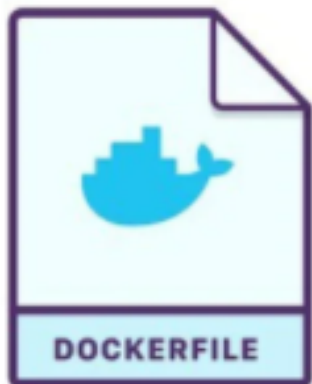
PROF. João Menk
PROF. Rafael Pereira

profjoao.menk@fiap.com.br
profrafael.pereira@fiap.com.br



docker
DOCKERFILE

- A interface de linha de comando (CLI) é uma forma manual de realizar a administração. Visto em nossos exemplos não é complexo realizar comandos como: pull, run, ps, stop etc.
- Porém podemos automatizar o processo utilizando Dockerfiles. Esses arquivos nada mais são do que listas de instruções utilizados para automatizar a criação e configuração de Containers
- Em outras palavras, ele serve como **uma receita para construir uma Imagem**, permitindo definir um ambiente personalizado



No exemplo abaixo temos o fonte de um Dockerfile simples, que realiza alguns dos passos que já executamos em nossos exemplos

```
FROM ubuntu
```

Obter uma imagem do Ubuntu

```
RUN apt-get -y update
```

Atualizar os pacotes do Sistema Operacional da imagem

```
RUN apt-get -y install python
```

Instalar o Python

Instrução FROM

É obrigatória e define qual será o ponto de partida da Imagem que criaremos com o nosso Dockerfile

Instrução RUN

Pode ser executada uma ou mais vezes e, com ela, podemos definir quais serão os comandos executados na **etapa de criação de camadas da Imagem**

Crie um arquivo com o fonte do exemplo anterior em sua Home, no Sistema Operacional, e salve como **Dockerfile** (sem extensão)



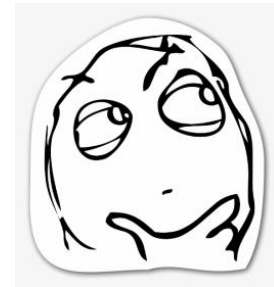
```
Dockerfile X

Users > Menk > Dockerfile > ...
1  FROM ubuntu
2
3  RUN apt-get -y update
4  RUN apt-get -y install python3
```

Agora que escrevemos um Dockerfile, **iremos construir uma imagem** a partir desse arquivo, executando o comando **docker build**, e, por fim, criar e rodar um Container com o comando **docker container run**

“O Container é o fim enquanto a Imagem é o meio”

*#fica
a
dica*



Caso queira criar uma imagem do zero, sem a preocupação de utilizar imagem alguma, utilize a imagem **scratch**

FROM scratch

DOCKER FILE

Para criar uma Imagem a partir desse arquivo usamos o comando **docker build**. Por padrão esse comando procura um arquivo com o nome **Dockerfile**

Em nosso exemplo o comando ficará

docker build . → **Diretório corrente** (não esqueça do espaço antes do ponto)

```
iMac~ Menk$ docker build .
[+] Building 45.1s (7/7) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 108B                                              0.0s
=> [internal] load .dockerignore                                                  0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest                 1.6s
=> [1/3] FROM docker.io/library/ubuntu@sha256:4b1d0c4a2d2aaf63b37111f34eb9     8.0s
=> => resolve docker.io/library/ubuntu@sha256:4b1d0c4a2d2aaf63b37111f34eb9    0.0s
=> => sha256:4b1d0c4a2d2aaf63b37111f34eb9fa89fa1bf53dd6e4c 1.42kB / 1.42kB    0.0s
=> => sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab3291 529B / 529B  0.0s
=> => sha256:a8780b506fa4eeb1d0779a3c92c8d5d3e6a656c758135 1.46kB / 1.46kB  0.0s
=> => sha256:e96e057aae67380a4ddb16c337c5c3669d97fdff69e 30.43MB / 30.43MB  5.7s
=> => extracting sha256:e96e057aae67380a4ddb16c337c5c3669d97fdff69ec537f02  1.5s
=> [2/3] RUN apt-get -y update                                                  22.3s
=> [3/3] RUN apt-get -y install python3                                         12.6s
=> exporting to image                                                            0.4s
=> => exporting layers                                                            0.4s
=> => writing image sha256:967ab74e1a522c005f30a5aea254dff474038e494ee7d6c    0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
iMac~ Menk$
```

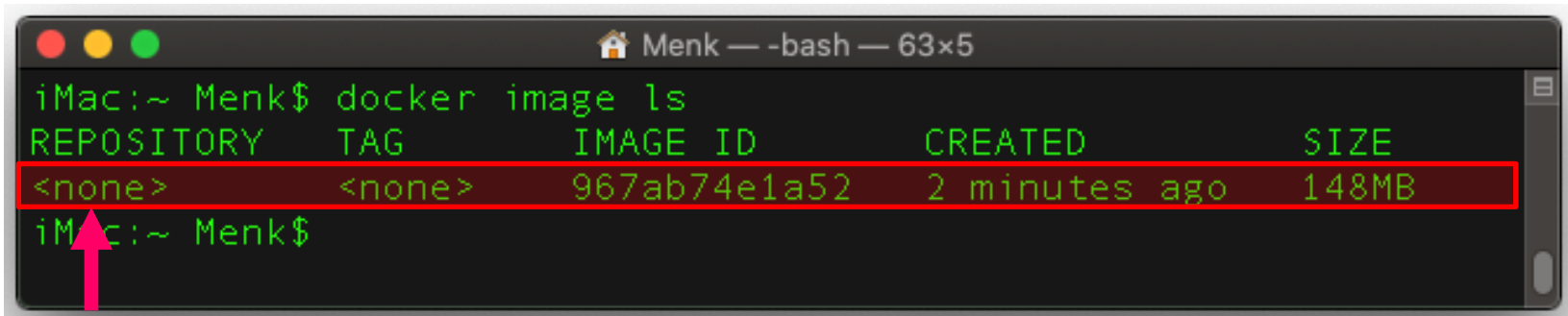
Caso necessite, especifique o nome do seu Dockerfile com a opção **-f**

Exemplo: **docker build -f dockerfile-ubuntu-python .**

DOCKER FILE

Para verificar a imagem criada a partir desse arquivo usamos o comando **docker image ls**

docker image ls

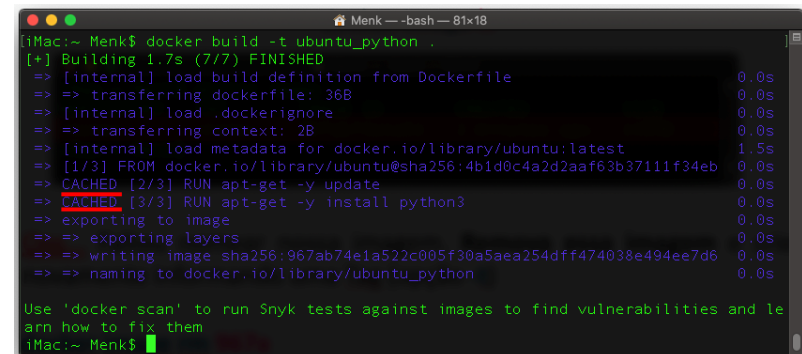


```
iMac:~ Menk$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
<none>        <none>    967ab74e1a52   2 minutes ago  148MB
iMac:~ Menk$
```

Ops... Faltou nomear nossa imagem. **Remova** essa imagem e crie novamente informando um Nome (opção **-t**)

docker image rm 967a

docker build -t ubuntu_python .



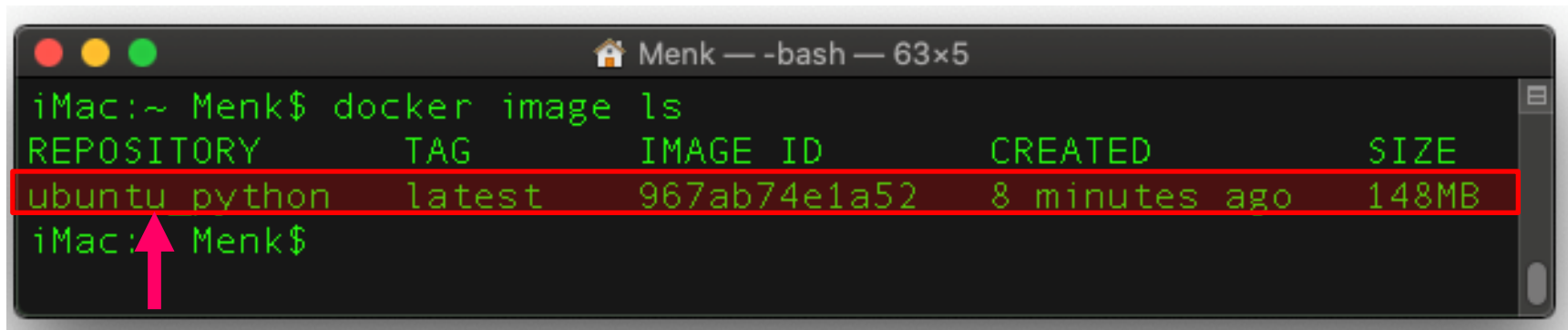
```
iMac:~ Menk$ docker build -t ubuntu_python .
[+] Building 1.7s (7/7) FINISHED
=> [internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 36B                      0.0s
=> [internal] load .dockerignore                       0.0s
=> => transferring context: 2B                          0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 1.5s
=> [1/3] FROM docker.io/library/ubuntu@sha256:4b1d0c4a2d2aaf63b3711f34eb 0.0s
=> CACHED [2/3] RUN apt-get -y update                   0.0s
=> CACHED [3/3] RUN apt-get -y install python3          0.0s
=> exporting to image                                  0.0s
=> => exporting layers                                   0.0s
=> => writing image sha256:967ab74e1a522c005f30a5aea254dff474038e494ee7d6 0.0s
=> => naming to docker.io/library/ubuntu_python        0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
iMac:~ Menk$
```

DOCKER FILE

Verifique o resultado

docker image ls



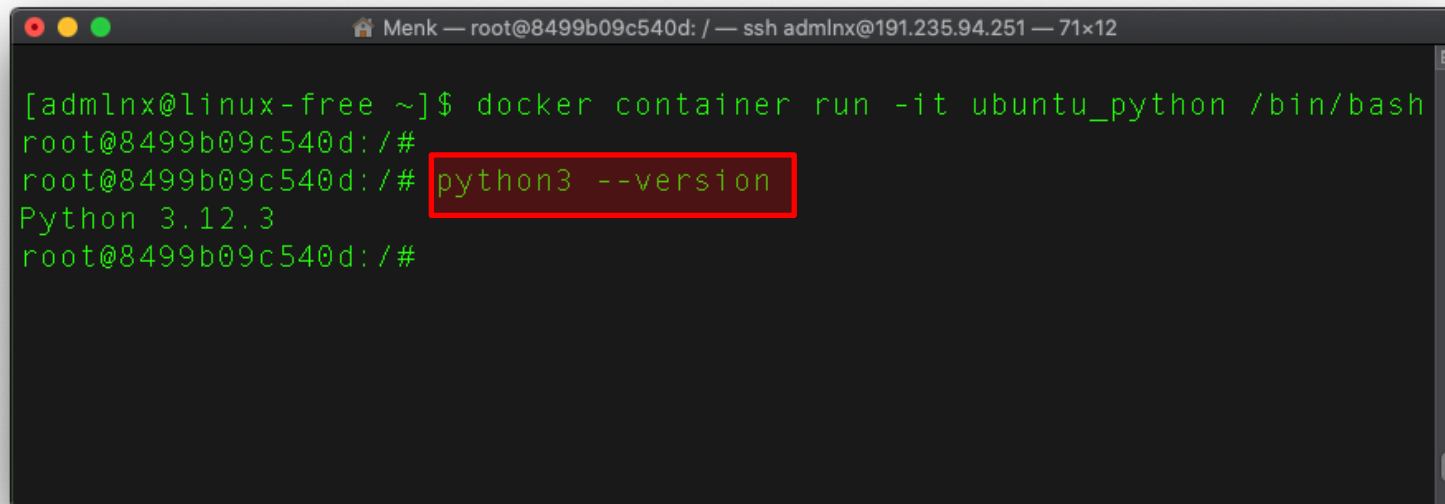
```
iMac:~ Menk$ docker image ls
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
ubuntu python   latest       967ab74e1a52   8 minutes ago 148MB
iMac:~ Menk$
```



Como último passo vamos rodar um Container criado por meio de um Dockerfile

Vamos rodar esse Container em modo Interativo e acessar o terminal para verificar se as tarefas foram concluídas

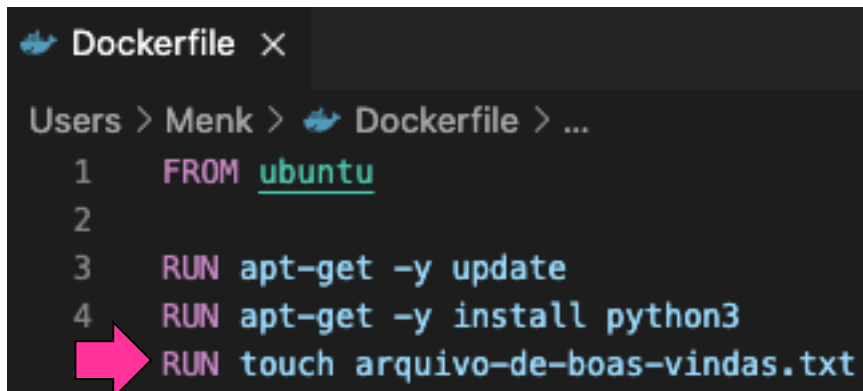
docker container run -it ubuntu_python /bin/bash

A terminal window titled 'Menk — root@8499b09c540d: / — ssh admlnx@191.235.94.251 — 71x12'. The terminal shows the command '[admlnx@linux-free ~]\$ docker container run -it ubuntu_python /bin/bash' being executed. The prompt changes to 'root@8499b09c540d: /#'. Then, the command 'python3 --version' is entered and highlighted with a red box. The output is 'Python 3.12.3'. The prompt returns to 'root@8499b09c540d: /#'.

```
[admlnx@linux-free ~]$ docker container run -it ubuntu_python /bin/bash
root@8499b09c540d: /#
root@8499b09c540d: /# python3 --version
Python 3.12.3
root@8499b09c540d: /#
```

DOCKER FILE

- Agora vamos **recrir** a imagem a partir do Dockerfile alterando com novas solicitações
- Cada RUN criará uma etapa na criação da Imagem
- Cada camada gerada por ele poderá ser reutilizada na criação de outras Imagens
- Altere seu Dockerfile conforme abaixo adicionando mais uma tarefa, salve e execute novamente o Build



```
Dockerfile X
Users > Menk > Dockerfile > ...
1  FROM ubuntu
2
3  RUN apt-get -y update
4  RUN apt-get -y install python3
5  RUN touch arquivo-de-boas-vindas.txt
```

Excluir uma única linha no Editor Nano
Ctrl+K

docker build -t ubuntu_python .

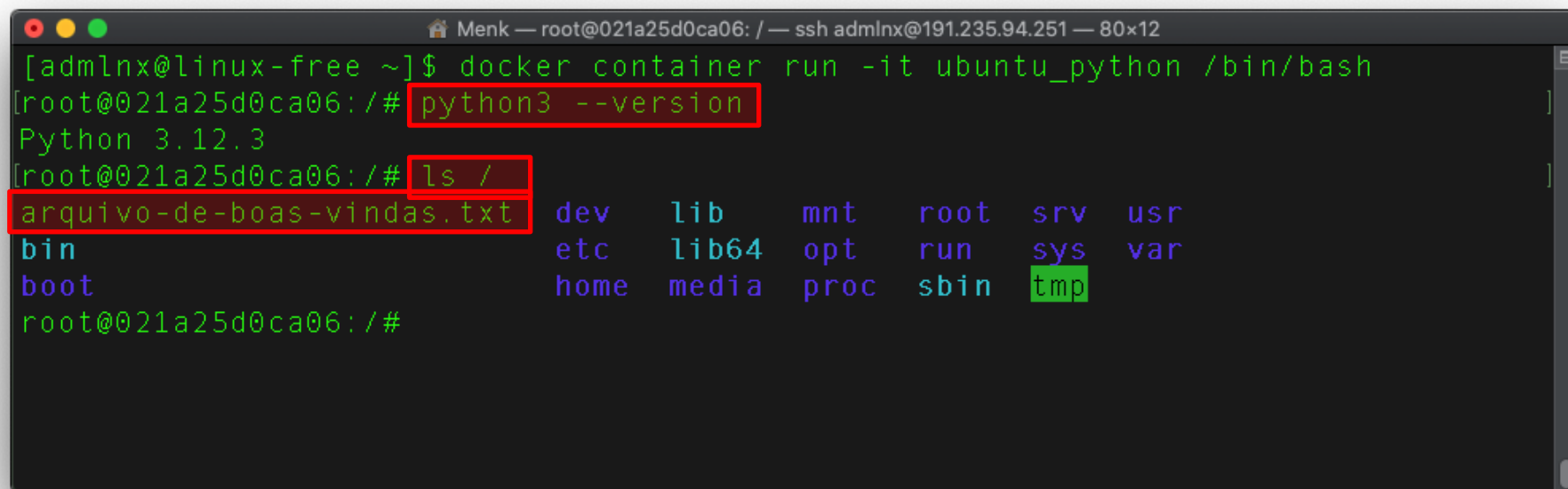
O comando BUILD consegue reutilizar diversas camadas e isso torna o processo muito mais rápido

```
Menk — root@4678c2fee316: / — -bash — 82x19
[iMac:~ Menk$ docker build -t ubuntu_python .
[+] Building 2.8s (8/8) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 145B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 2.1s
=> [1/4] FROM docker.io/library/ubuntu@sha256:4b1d0c4a2d2aaf63b37111f34eb9 0.0s
=> CACHED [2/4] RUN apt-get -y update 0.0s
=> CACHED [3/4] RUN apt-get -y install python3 0.0s
=> [4/4] RUN touch arquivo-de-boas-vindas.txt 0.2s
=> exporting to image 0.4s
=> => exporting layers 0.4s
=> => writing image sha256:753d7b3e2600adb42da2257b20e548d19e798894d606ed9 0.0s
=> => naming to docker.io/library/ubuntu_python 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
iMac:~ Menk$
```

Executando um novo Container

docker container run -it ubuntu_python /bin/bash

A terminal window titled 'Menk — root@021a25d0ca06: / — ssh adminx@191.235.94.251 — 80x12'. The prompt is [admlnx@linux-free ~]\$ and the command docker container run -it ubuntu_python /bin/bash is entered. The prompt changes to [root@021a25d0ca06:/#]. The command python3 --version is entered and returns Python 3.12.3. The command ls / is entered and returns a directory listing. The listing shows files and directories: arquivo-de-boas-vindas.txt, bin, boot, dev, etc, home, lib, lib64, media, mnt, opt, proc, root, run, sbin, srv, sys, tmp, usr, and var. The prompt returns to [root@021a25d0ca06:/#].

```
[admlnx@linux-free ~]$ docker container run -it ubuntu_python /bin/bash
[root@021a25d0ca06:/#] python3 --version
Python 3.12.3
[root@021a25d0ca06:/#] ls /
arquivo-de-boas-vindas.txt  dev    lib    mnt    root   srv    usr
bin                          etc    lib64  opt    run    sys    var
boot                        home   media  proc   sbin   tmp
root@021a25d0ca06:/#
```

- Exemplos de outros comandos no Dockerfile

ADD / COPY

Faz a cópia de um arquivo, diretório ou até mesmo fazer o download de uma URL de nossa máquina host e inserir dentro da imagem (ADD)

```
FROM ubuntu:18.04
```

```
RUN apt-get update -y
```

```
RUN apt-get install npm -y
```

```
ADD Dockerfile /root/arquivo-host-transferido.txt
```

➔ Direção: Host -> Container

docker build -t teste .

docker container run --name testeadd -it teste /bin/bash

Execute o comando `npm -v` para verificar a instalação, e após, `exit`, para sair do Container

DOCKER FILE

- Exemplos de outros comandos no Dockerfile

EXPOSE

Expõe uma porta específica com um protocolo especificado dentro de um Docker Container

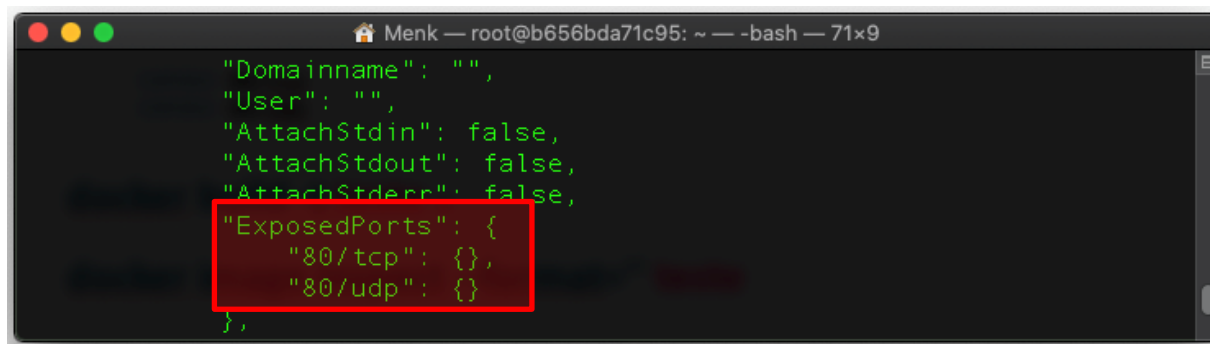
```
FROM ubuntu
```

```
EXPOSE 80/tcp
```

```
EXPOSE 80/udp
```

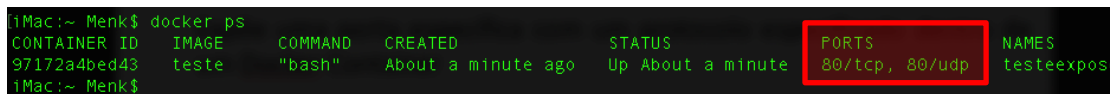
docker build -t teste .

docker image inspect teste



```
Menk — root@b656bda71c95: ~ — -bash — 71x9
{
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "ExposedPorts": {
    "80/tcp": {},
    "80/udp": {}
  },
}
```

docker container run --name testeexpose -it teste bash



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
97172a4bed43	teste	"bash"	About a minute ago	Up About a minute	80/tcp, 80/udp	testeexpose

Em outro Terminal

- Exemplos de outros comandos no Dockerfile

EXPOSE

É importante entender que a instrução EXPOSE atua apenas como uma plataforma de informações (como Documentação) entre o criador da imagem Docker e o indivíduo que executa o Container. Esse comando não faz a publicação da porta.

- ✓ Podemos usar o protocolo TCP ou UDP para expor a porta (Padrão TCP)
- ✓ Não mapeia portas na máquina Host
- ✓ Pode ser substituído usando o sinalizador de publicação (**-p**) ao iniciar um Container (docker run)

DOCKER FILE

FIAP



- Exemplos de outros comandos, via **DOCKER RUN**

EXPOSE

Exemplo com sinalizador de publicação

docker run -d --name nginx-server -p 80:80 nginx

↪ Direção: Host -> Container

```
Menk — root@52aca4c95fd0: / — -bash — 80x14
iMac:~ Menk$ docker run -d --name nginx-server -p 80:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
a603fa5e3b41: Pull complete
c39e1cda007e: Pull complete
90cfefba34d7: Pull complete
a38226fb7aba: Pull complete
62583498bae6: Pull complete
9802a2cfdb8d: Pull complete
Digest: sha256:e209ac2f37c70c1e0e9873a5f7231e91dcd83fdf1178d8ed36c2ec09974210ba
Status: Downloaded newer image for nginx:latest
23021825f5d60a42b1285d04e3d2500c3e3f101b129e8f950b5d29f047d1d143
iMac:~ Menk$
```

Acesse seu Web Browser em -> **localhost:80**

docker container stop nginx-server

docker container rm nginx-server

- Exemplos de outros comandos no Dockerfile

WORKDIR

Define o ambiente de trabalho no Container, onde as instruções CMD, RUN, ENTRYPOINT, ADD etc executarão suas tarefas, além de definir o diretório padrão que será aberto ao executarmos o Container no modo Interativo (-it)

Crie o arquivo **no seu Host**, no Terminal execute:

```
echo { "nome": "Robert Plant", "banda": "Led Zeppelin" } > arquivo-host.json
```

```
FROM ubuntu:18.04
```

```
WORKDIR /app-java
```

```
ADD arquivo-host.json arquivo-host-transferido.json
```

```
docker build -t teste .
```

```
docker container run --name testeworkdir -it teste bash
```

- Exemplos de outros comandos no Dockerfile

CMD

- ✓ Usado para definir um comando padrão que é executado assim que você roda o Container, **não no Build da Imagem**
- ✓ No caso de vários comandos CMD, apenas o último é executado

```
FROM alpine
```

```
CMD ["echo", "Rodei na execução"]
```

```
CMD = /bin/bash -c
```

docker build -t testecmd .

docker container run --rm testecmd

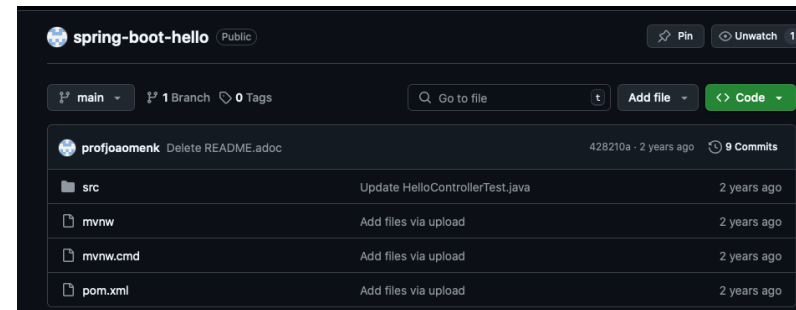


O Docker limpa automaticamente o Container e remove o sistema de arquivos quando for encerrado

Exercício com Dockerfile

Exemplo de App Hello Word Java Spring

- 1) Clonar o Repositório
- 2) Entrar no diretório criado pelo Git
- 3) Criar o Dockerfile para empacotar e rodar o App
- 4) Criar a Imagem do Docker
- 5) Rodar o Container em Segundo Plano com o nome **apphello**
- 6) Limpar o Laboratório



01) git clone <https://github.com/profjoaomenk/spring-boot-hello.git>

02) cd spring-boot-hello

Imagem a Utilizar no Dockerfile: maven:3.9.5-eclipse-temurin-17

Expor a porta 8080

Comando CMD:

CMD ["java", "-jar", "target/spring-boot-complete-0.0.1-SNAPSHOT.jar"]

DOCKER FILE

Exercício com Dockerfile

```
git clone https://github.com/profjoaomenk/spring-boot-hello.git
```

```
cd spring-boot-hello
```

```
nano Dockerfile
```

```
FROM maven:3.9.5-eclipse-temurin-17
```

```
WORKDIR /app
```

```
COPY . .
```

```
RUN mvn clean package -DskipTests
```

```
EXPOSE 8080
```

```
CMD ["java", "-jar", "target/spring-boot-complete-0.0.1-SNAPSHOT.jar"]
```

```
docker build -t hello .
```

```
docker container run -d -p 8080:8080 --name apphello hello
```

```
docker container rm -f apphello
```

```
docker image rm hello
```

Não esqueça de voltar ao diretório anterior ao final desse exemplo

- Exemplos de outros comandos no Dockerfile

ENTRYPOINT

É um ponto de entrada para seu Container, o que ele irá fazer ao iniciar. Permite que você configure **um Container que será executado como um executável**

```
FROM alpine  
ENTRYPOINT ["top", "-b"]
```

docker build -t ptoentrada .

docker container run --rm ptoentrada

docker container run --rm ptoentrada -n 3

Envia parâmetros ao ENTRYPOINT. Executa:
-b -n 3

Documentação Linux: -n N -> Exit after N iterations

- Exemplos de outros comandos no Dockerfile

ENTRYPOINT e CMD - Juntos

- ✓ ENTRYPOINT deve ser definido ao usar o Container como um executável
- ✓ O CMD deve ser usado como uma forma de definir argumentos padrão para um comando ENTRYPOINT ou para executar um comando em um Container
- ✓ O CMD será substituído ao executar o Container com argumentos alternativos

```
...  
ENTRYPOINT ["dotnet", "seuapp.dll"]  
CMD ["meuparam"]  
...
```

O resultado na execução é:
dotnet seuapp.dll meuparam

DOCKER FILE

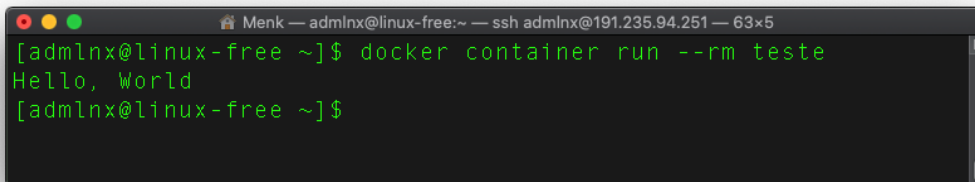
- Exemplos de outros comandos no Dockerfile

ENTRYPOINT e CMD - Juntos

```
FROM alpine  
  
ENTRYPOINT ["echo", "Hello, "]  
  
CMD ["World"]
```

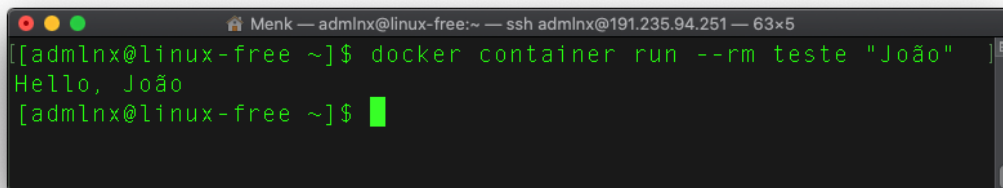
docker build -t teste .

docker container run --rm teste



```
Menk — admlnx@linux-free:~ — ssh admlnx@191.235.94.251 — 63x5  
[admlnx@linux-free ~]$ docker container run --rm teste  
Hello, World  
[admlnx@linux-free ~]$
```

docker container run --rm teste "João"

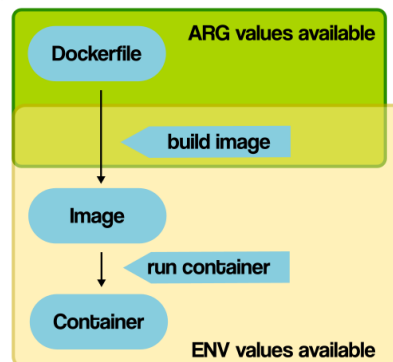


```
Menk — admlnx@linux-free:~ — ssh admlnx@191.235.94.251 — 63x5  
[admlnx@linux-free ~]$ docker container run --rm teste "João"  
Hello, João  
[admlnx@linux-free ~]$
```

- Exemplos de outros comandos no Dockerfile

ARG e ENV

- ✓ A instrução **ARG** define uma variável que os usuários podem passar em tempo de compilação para o construtor da Imagem (comando **docker build**) usando o sinalizador: **--build-arg <varname>=<value>**
- ✓ **ARG** é a única instrução que pode preceder FROM no Dockerfile
- ✓ As variáveis de ambiente definidas usando a instrução **ENV** sempre substituem uma instrução **ARG** com o mesmo nome
- ✓ Ao contrário do **ARG**, as variáveis **ENV** também são acessíveis ao executar os Containers
- ✓ Os valores **ENV** também podem ser substituídos ao iniciar um Container (**-e** ou **--env-file**)



ARG nome # O ARG espera um valor
ARG nome=João # O ARG recebe um valor padrão
ENV estado=PB # O ENV recebe um valor padrão
ENV nome2=\$nome # O ENV recebe um valor padrão de um ARG
ENV nome2=\${nome} # O ENV recebe um valor padrão de um ARG

#fica
dica

NÃO PASSE VALORES DE SEGREDOS

Variáveis em tempo de compilação são visíveis através do comando **docker history**

DOCKER FILE

FIAP



- Exemplos de outros comandos no Dockerfile

ARG

```
FROM alpine
```

```
ARG nome=João
```

```
RUN echo "Olá! Bem-vindo(a) $nome" > bem-vindo.txt
```

```
ENTRYPOINT cat bem-vindo.txt  
#ENTRYPOINT ["cat", "bem-vindo.txt"]
```



docker build -t arg-demo .

docker container run --rm arg-demo

```
Menk — adminx@linux-free:~ — ssh adminx@191.235.94.251 — 75x17  
[adminx@linux-free ~]$ docker container run --rm arg-demo  
Olá! Bem-vindo(a) João  
[adminx@linux-free ~]$
```

Substituindo o valor padrão

docker build -t arg-demo --build-arg nome=Maria .

docker container run --rm arg-demo

```
Menk — adminx@linux-free:~ — ssh adminx@191.235.94.251 — 75x17  
[adminx@linux-free ~]$ docker container run --rm arg-demo  
Olá! Bem-vindo(a) Maria  
[adminx@linux-free ~]$
```

- Exemplos de outros comandos no Dockerfile

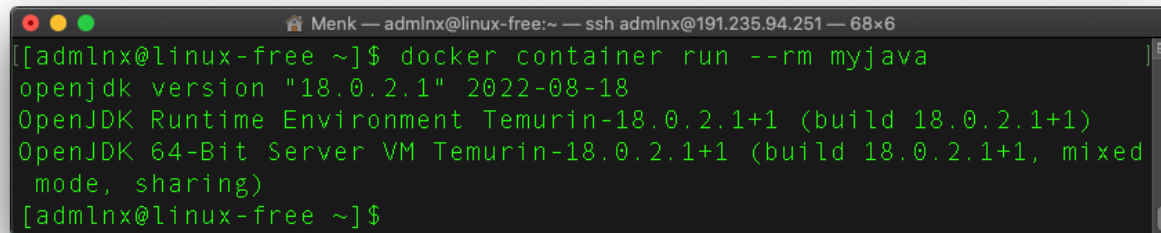
ARG

Mais um exemplo de utilização

```
ARG JAVA_VERSION=17
FROM eclipse-temurin:${JAVA_VERSION}-jdk
CMD ["java", "-version"]
```

docker build -t myjava --build-arg JAVA_VERSION=18 .

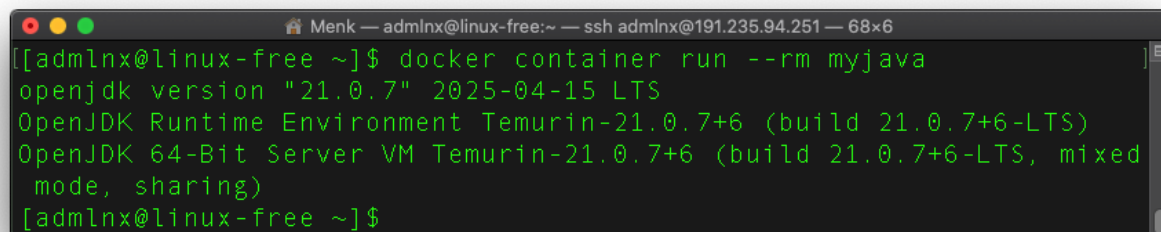
docker container run --rm myjava



```
Menk — admlnx@linux-free:~ — ssh admlnx@191.235.94.251 — 68x6
[admlnx@linux-free ~]$ docker container run --rm myjava
openjdk version "18.0.2.1" 2022-08-18
OpenJDK Runtime Environment Temurin-18.0.2.1+1 (build 18.0.2.1+1)
OpenJDK 64-Bit Server VM Temurin-18.0.2.1+1 (build 18.0.2.1+1, mixed
mode, sharing)
[admlnx@linux-free ~]$
```

docker build -t myjava --build-arg JAVA_VERSION=21 .

docker container run --rm myjava



```
Menk — admlnx@linux-free:~ — ssh admlnx@191.235.94.251 — 68x6
[admlnx@linux-free ~]$ docker container run --rm myjava
openjdk version "21.0.7" 2025-04-15 LTS
OpenJDK Runtime Environment Temurin-21.0.7+6 (build 21.0.7+6-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.7+6 (build 21.0.7+6-LTS, mixed
mode, sharing)
[admlnx@linux-free ~]$
```

- Exemplos de outros comandos no Dockerfile

ENV

Exemplo de utilização

```
FROM alpine
ENV hey="Olá"
ENV dir="/"
# ENV também pode ser utilizado na construção
ADD ./Dockerfile ${dir}
CMD echo $hey
```

docker build -t env-demo .

docker container run --rm env-demo

docker container run --rm -e hey="Salve" env-demo

docker container run --rm -it env-demo sh

Verifique o arquivo copiado

- Exemplos de outros comandos no Dockerfile

USER

Altera o usuário que irá executar os comandos

```
FROM node:alpine
```

```
RUN adduser -h /home/menk -s /bin/bash -D menk
```

→ Criação de um novo usuário

```
USER menk
```

→ Utilização de um usuário já existente

```
RUN whoami
```

```
RUN touch /home/menk/teste.txt
```

```
USER node
```

→ Utilização de um usuário já existente

```
RUN whoami
```

```
RUN touch /home/node/teste.txt
```

#Executar mais de um comando com a instrução CMD

```
CMD ls -l /home/menk; ls -l /home/node
```

```
#CMD ["sh", "-c", "ls -l /home/menk ; ls -l /home/node"]
```

docker build -t user-demo .


docker container run --rm user-
demo


- FROM => Inicializa o build de uma imagem a partir de uma imagem base
- RUN => Executa um comando
- WORKDIR => Define o seu diretório corrente
- COPY => Copia arquivos ou diretórios e adiciona ao sistema de arquivos da imagem
- ADD => Copia arquivos ou diretórios ou arquivos remotos e adiciona ao sistema de arquivos da imagem
- LABEL => Adiciona metadados a imagem
- ENV => Define variáveis de ambiente
- VOLUME => Define volumes que devem ser definidos
- ARG => Define um argumento pra ser usado no processo de construção
- EXPOSE => Define que o container precisa expor a porta em questão
- USER => Define o Usuário que vai ser usado
- CMD => Define o comando e/ou os parâmetros padrão
- ENTRYPOINT => Ajuda você a configurar um contêiner que pode ser executado como um executável.




definitions


 **image** a static snapshot of container's configuration.

 **container** an application sandbox. each container is based on an image.


 **layer** image is composed of read-only file system layers. container creates single writable layer.

 **docker registry** remote server for storing Docker images


 **Dockerfile** a configuration file with build instructions for Docker images

 **docker engine** Docker platform installation running on a given host

 **docker client** client application that talks to local or remote Docker daemon

 **docker daemon** service process that listens to Docker client commands over local or remote network

 **docker host** server that runs Docker engine

 **volume** directory shared between host and container

docker run

docker run [OPTIONS] IMAGE[:TAG] [COMMAND]

Run a command in a new container.

metadata	--name=CONTNR_NAME Assign a name to the container.
	-l, --label NAME[=VALUE] Set metadata on the container.
process	-d, --detach Run in the background.
	-i, --interactive Keep STDIN open.
	-t, --tty Allocate a pseudo-TTY.
	--rm Automatically remove the container when process exits.
	-u USER Run as username or UID.
	--privileged Give extended privileges.
	-w DIR Set working directory.
	-e NAME=VALUE Set environment variable.
	--restart=POLICY Restart policy. no on-failure[:RETRIES] always unless-stopped
	-P, --publish-all Publish all exposed ports to random ports.
network	-p HOST_PORT:CONTNR_PORT Expose a port or a range of ports.
	--network=NETWORK_NAME Connect container to a network.
	--dns=DNS_SERVER1[,DNS_SERVER2] Set custom dns servers.
	--add-host=HOSTNAME:IP Add a line to /etc/hosts.
file system	--read-only Mount the container's root file system as read only.
	-v, --volume [HOST_SRC:]CONTNR_DEST Mount a volume between host and the container file system.
	--volumes-from=CONTNR_ID Mount all volumes from another container.

Dockerfile

```
FROM <image_id>
    base image to build this image from
RUN <command> shell form
RUN ["<executable>",
    "<param1>",
    ...,
    "<paramN>"]
    exec form
    executes command to modify container's file system state
MAINTAINER <name>
    provides information about image creator
LABEL <key>=<value>
    adds searchable metadata to image
ARG <name>[=<default value>]
    defines overridable build-time parameter:
    docker build --build-arg <name>=<value> .
ENV <key>=<value>
    defines environment variable that will be visible during
    image build-time and container run-time
ADD <src> <dest>
    copies files from <src> (file, directory or URL) and adds them
    to container file system under <dest> path
COPY <src> <dest> similar to ADD, does not support URLs
VOLUME <dest>
    defines mount point to be shared with host or other containers
EXPOSE <port>
    informs Docker engine that container listens to port at run-time
WORKDIR <dest>
    sets build-time and run-time working directory
USER <user>
    defines run-time user to start container process
STOPSIGNAL <signal>
    defines signal to use to notify container process to stop
ENTRYPOINT shell form or exec form
    defines run-time command prefix that will be added to all
    run commands executed by docker run
CMD shell form or exec form
    defines run-time command to be executed by default when
    docker run command is executed
```


`docker system prune -a -f --volumes`



Copyright © 2025 Prof. João Carlos Menk

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).