

FIAP GRADUAÇÃO

FIAP

MBA⁺
2015

TECNOLOGIA EM DESENVOLVIMENTO DE SISTEMAS

DevOps Tools & Cloud Computing

Docker File e Docker Compose

PROF. João Menk

profjoao.menk@fiap.com.br

PROF. Sálvio Padlipskas

salvio@fiap.com.br

PROF. Antonio Figueiredo

profantonio.figueiredo@fiap.com.br

PROF. Marcus Leite

profmarcus.leite@fiap.com.br

PROF. Thiago Rocha

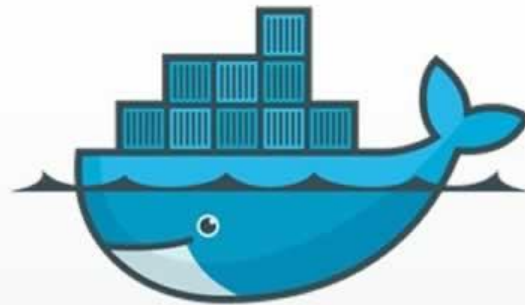
profthiago.rocha@fiap.com.br

PROF. Thiago Moraes

proftiago.moraes@fiap.com.br

PROF. Rafael Pereira

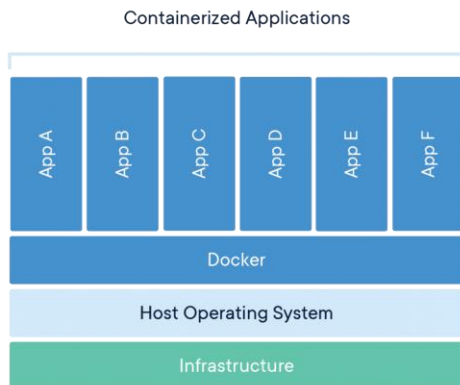
profrafael.pereira@fiap.com.br



docker

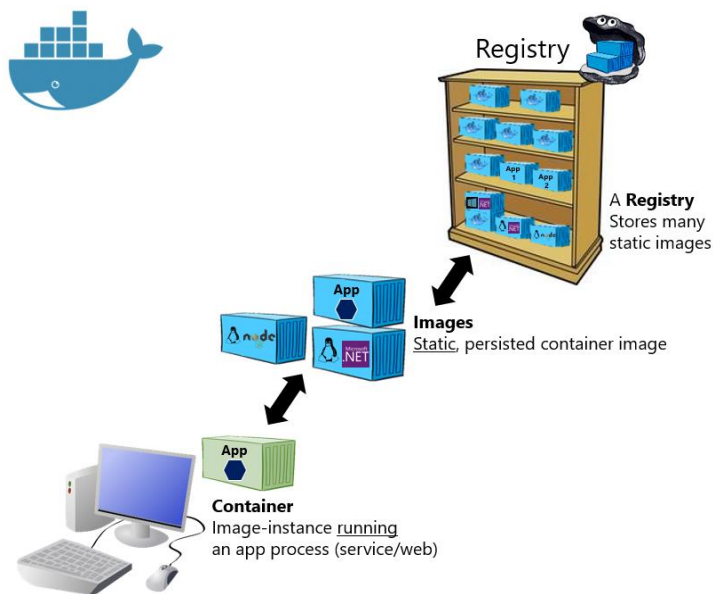
DOCKERFILE

1 Container

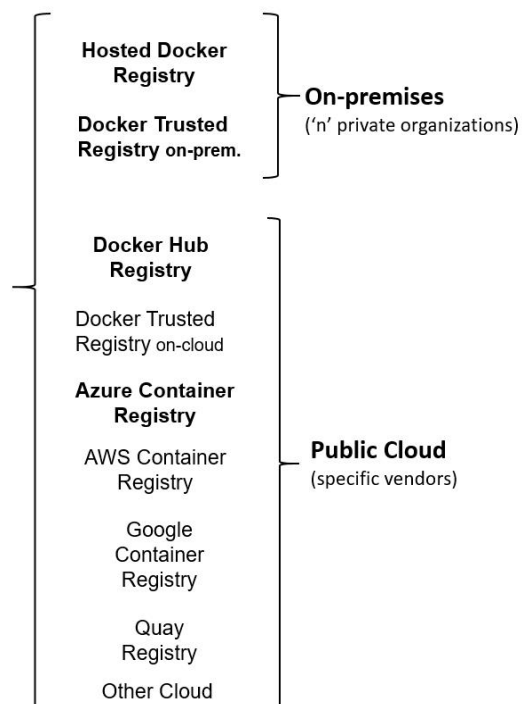


2 Imagem

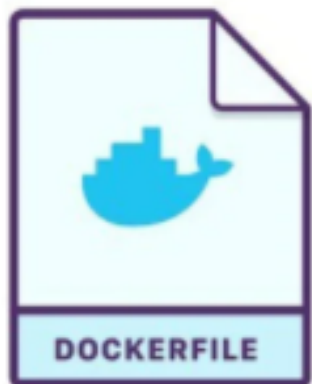
Basic taxonomy in Docker



3 Docker Registry



- A interface de linha de comando (CLI) é uma forma manual de realizar a administração. Visto em nossos exemplos não é complexo fazer o pull, run, ps e stop de uma Imagem
- 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 um Container**, 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

DOCKER FILE

Crie um arquivo com o fonte 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 dele 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
dica*

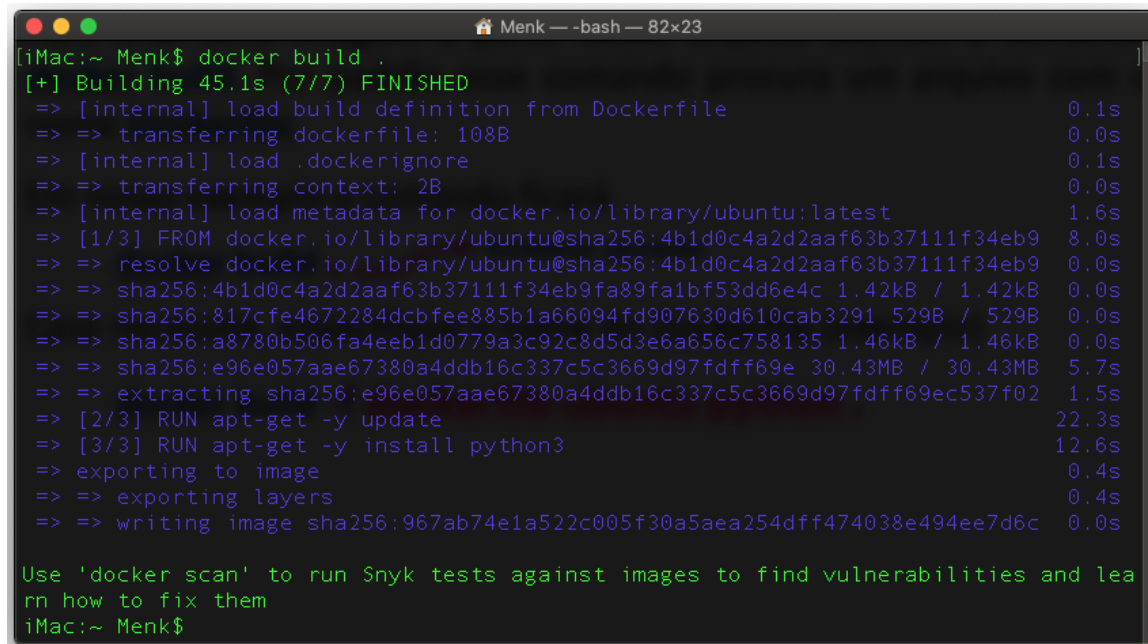
Caso queira criar uma imagem do zero, sem a preocupação de utilizar imagem alguma, utilize a imagem **scratch**

FROM scratch

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@sha256:4b1d0c4a2d2aaf63b3711f34eb9      8.0s
=> => resolve docker.io/library/ubuntu@sha256:4b1d0c4a2d2aaf63b3711f34eb9      0.0s
=> => sha256:4b1d0c4a2d2aaf63b3711f34eb9fa89fa1bf53dd6e4c 1.42kB / 1.42kB      0.0s
=> => sha256:817cfe4672284dcbfee885b1a66094fd907630d610cab3291 529B / 529B    0.0s
=> => sha256:a8780b506fa4eeb1d0779a3c92c8d5d3e8a656c758135 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$
```

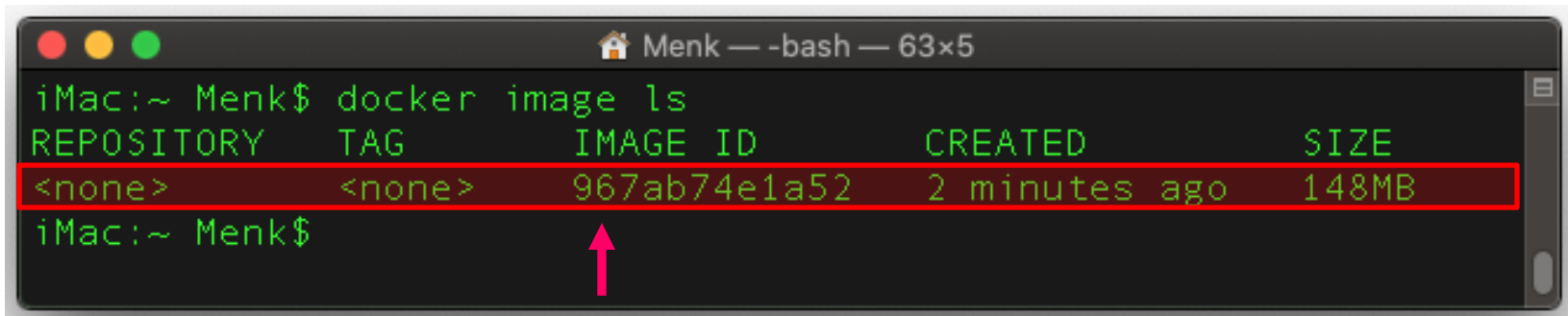
Caso necessite, especifique o nome do Dockerfile com a opção **-f** (exemplo)

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



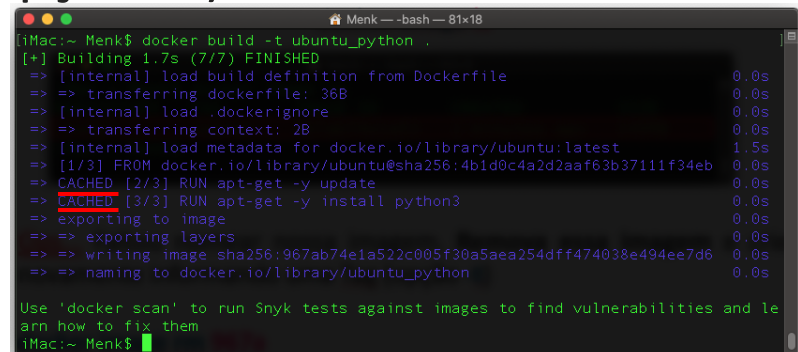
```
iMac:~ Menk$ docker image ls
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
<none>        <none>    967ab74e1a52  2 minutes ago  148MB
iMac:~ Menk$
```

A red box highlights the row for the image with ID 967ab74e1a52. A red arrow points to the ID.

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

docker image rm 967a

docker build -t ubuntu_python .



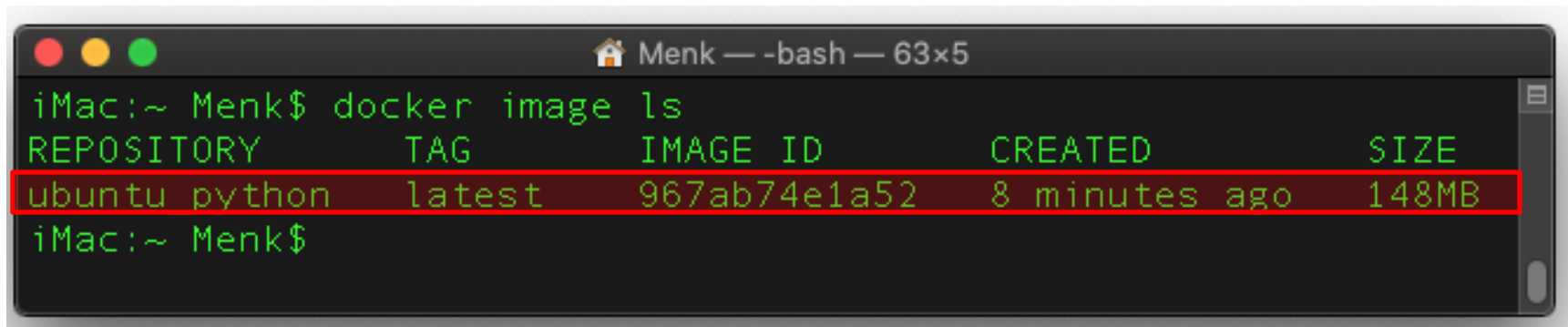
```
iMac:~ Menk$ docker build -t ubuntu_python .
[+] Building 1.7s (7/7) FINISHED
=> [internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 36B                       0.0s
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 2B                             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 sha256:967ab74e1a522c005f30a5ea254dff474038e494ee7d6 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`



```
Menk — -bash — 63x5
iMac:~ Menk$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu python  latest       967ab74e1a52   8 minutes ago  148MB
iMac:~ Menk$
```

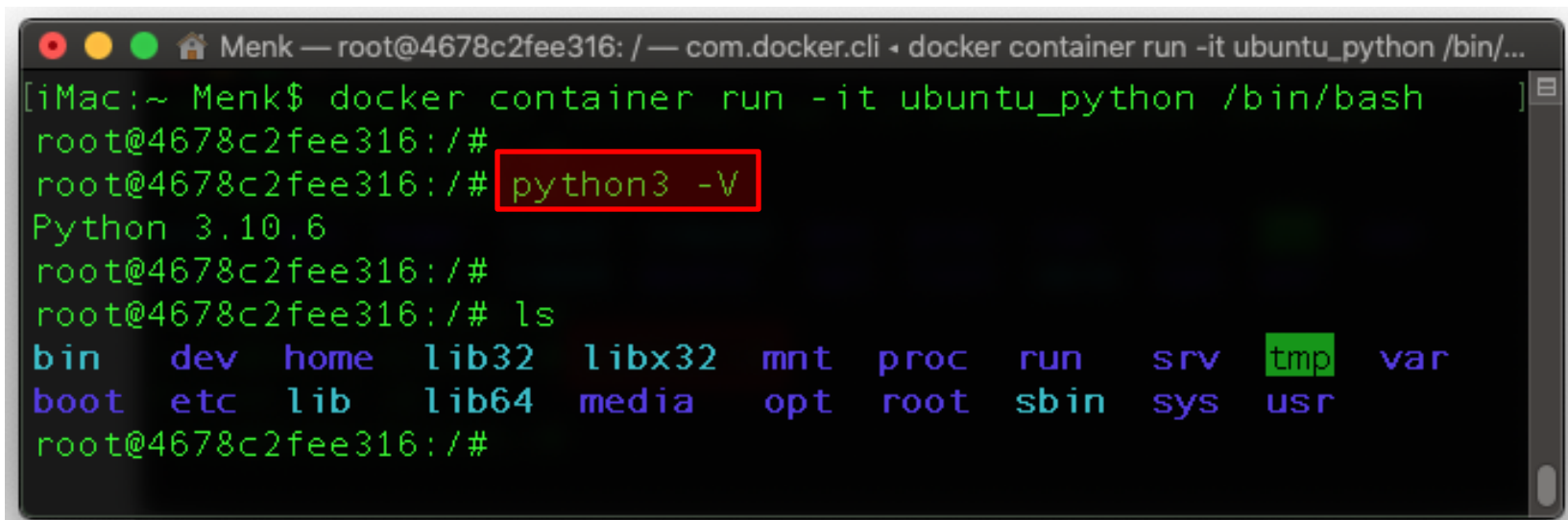
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu python	latest	967ab74e1a52	8 minutes ago	148MB



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

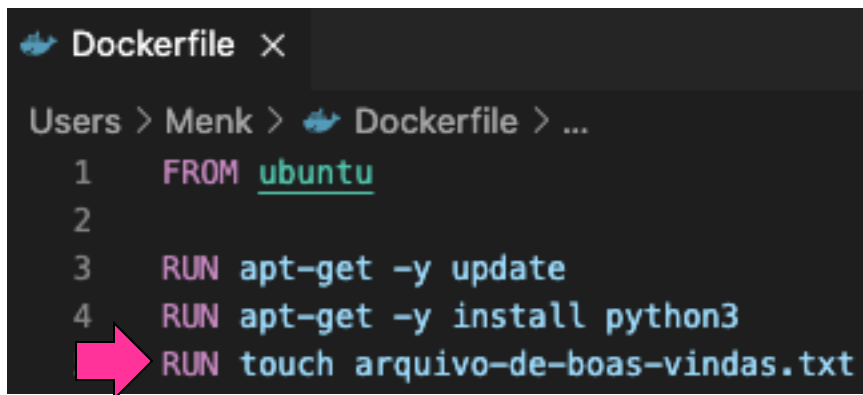
docker container run -it ubuntu_python /bin/bash

A terminal window titled 'Menk' showing the execution of a Docker container. The command 'docker container run -it ubuntu_python /bin/bash' is entered at the prompt. The container starts with a root shell. The user enters 'python3 -V' (highlighted with a red box), which outputs 'Python 3.10.6'. Then, the user enters 'ls', which lists the directory contents: 'bin', 'dev', 'home', 'lib32', 'libx32', 'mnt', 'proc', 'run', 'srv', 'tmp', 'var', 'boot', 'etc', 'lib', 'lib64', 'media', 'opt', 'root', 'sbin', 'sys', and 'usr'. The 'tmp' directory is highlighted with a green box.

```
Menk — root@4678c2fee316: / — com.docker.cli • docker container run -it ubuntu_python /bin/...
[iMac:~ Menk$ docker container run -it ubuntu_python /bin/bash ]
root@4678c2fee316:/#
root@4678c2fee316:/# python3 -V
Python 3.10.6
root@4678c2fee316:/#
root@4678c2fee316:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr
root@4678c2fee316:/#
```

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
   RUN touch arquivo-de-boas-vindas.txt
```

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`

```
Menk — root@52aca4c95fd0: / — com.docker.cli - docker container run -it ubuntu_python /bin/bash — 7...
iMac:~ Menk$ docker container run -it ubuntu_python /bin/bash
root@52aca4c95fd0:/#
root@52aca4c95fd0:/# python3 -V
Python 3.10.6
root@52aca4c95fd0:/#
root@52aca4c95fd0:/# ls /
arquivo-de-boas-vindas.txt  dev  lib  libx32  opt  run  sys  var
bin  etc  lib32  media  proc  sbin  tmp
boot  home  lib64  mnt  root  srv  usr
root@52aca4c95fd0:/#
```


- 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
- 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

<https://docs.docker.com/reference/dockerfile/#understand-how-cmd-and-entrypoint-interact>

- 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

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

DOCKER FILE

- Exemplos de outros comandos no Dockerfile

EXPOSE

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

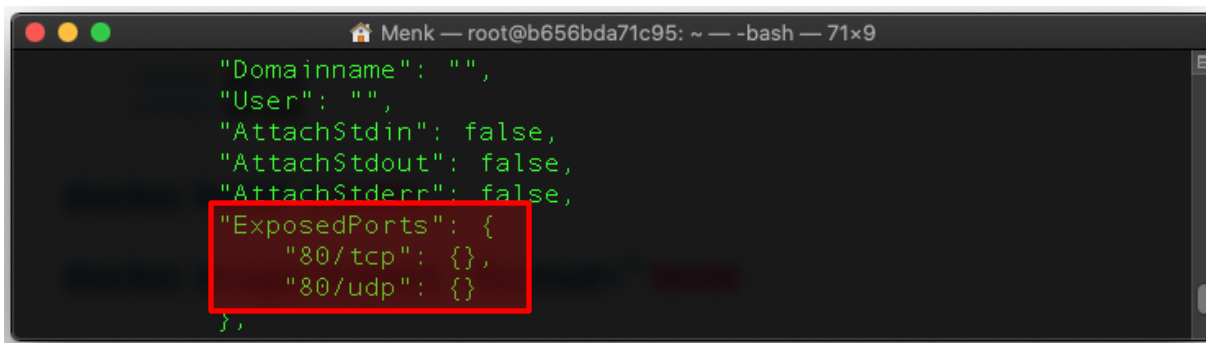
```
FROM ubuntu
```

```
EXPOSE 80/tcp
```

```
EXPOSE 80/udp
```

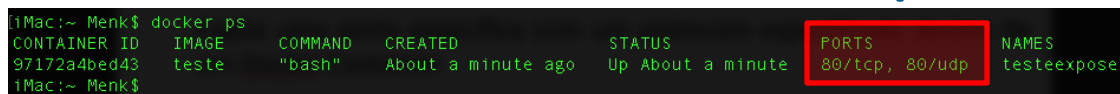
docker build -t teste .

docker image inspect --format=" 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

- 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

CMD

- ✓ Usado para definir um comando padrão que é executado assim que você executa 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"]
```

docker build -t testecmd .

docker container run --rm testecmd



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

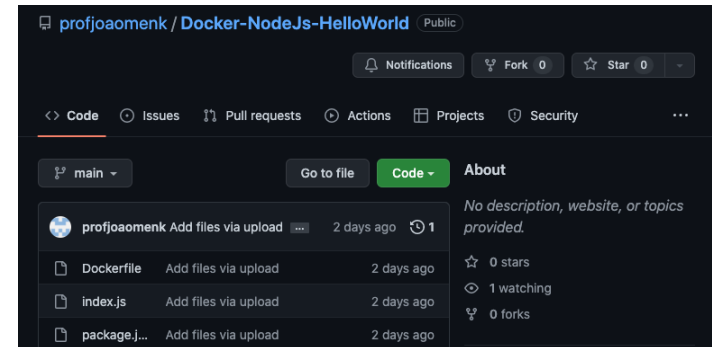
DOCKER FILE

- Exemplos de outros comandos no Dockerfile

CMD

Exemplo de App Hello Word com CMD

- 1) Clonar o Repositório
- 2) Entrar no diretório criado pelo Git
- 3) Criar a Imagem do Docker
- 4) Rodar o Container
- 5) Parar e Iniciar novamente o Container



```
git clone https://github.com/profjoaomenk/Docker-NodeJs-HelloWorld.git
```

```
cd Docker-NodeJs-HelloWorld
```

```
docker build -t nodehelloworld .
```

```
docker container run -d -p 3030:3030 --name apphello nodehelloworld
```

```
docker container stop apphello
```

```
docker container start apphello
```


- 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

Shell form

```
FROM alpine
ENTRYPOINT exec top -b
```

Exec form

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

#fica
dica

Também podemos utilizar ambas as formas no comando CMD e RUN

docker build -t ptoentrada .

Shell form

docker container run --rm ptoentrada

Exec form

docker container run --rm ptoentrada

docker container run --rm ptoentrada -n 3

Envia parâmetros e adiciona, ao invés de executar -b, executa -b -n 3

- 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 ad hoc 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

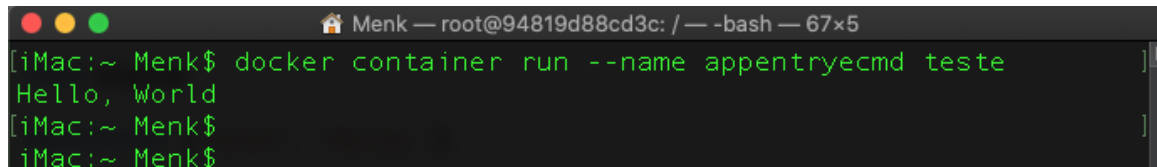
- Exemplos de outros comandos no Dockerfile

ENTRYPOINT e CMD - Juntos

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

docker build -t teste .

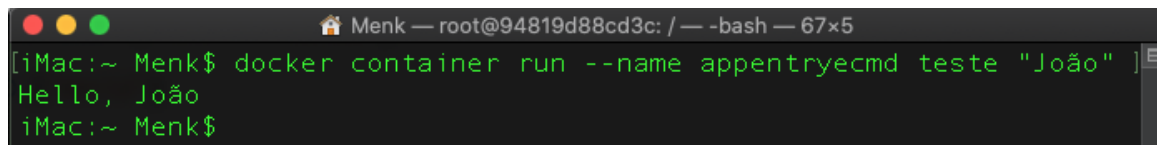
docker container run --name appentrycmd teste



```
Menk — root@94819d88cd3c: / — -bash — 67x5  
[iMac:~ Menk$ docker container run --name appentrycmd teste]  
Hello, World  
[iMac:~ Menk$]  
iMac:~ Menk$
```

docker container rm appentrycmd

docker container run --name appentrycmd teste "João"

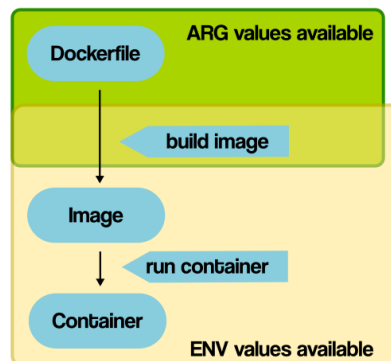


```
Menk — root@94819d88cd3c: / — -bash — 67x5  
[iMac:~ Menk$ docker container run --name appentrycmd teste "João"]  
Hello, João  
iMac:~ Menk$
```

- 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

- Exemplos de outros comandos no Dockerfile

ARG

FROM alpine

ARG nome=João

RUN echo "Olá! Bem-vinde \$nome" > bem-vinde.txt

ENTRYPOINT cat bem-vinde.txt

docker build -t arg-demo .

docker container run --rm arg-demo

Substituindo o valor padrão

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

docker container run --rm arg-demo

```
Menk — -bash — 91x21
[iMac:~ Menk$ docker container run --rm arg-demo
Olá! Bem-vinde João
[iMac:~ Menk$ docker build -t arg-demo --build-arg nome=Maria .
[+] Building 1.3s (6/6) 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/alpine:latest 0.8s
=> CACHED [1/2] FROM docker.io/library/alpine@sha256:8914eb54f968791faf6 0.0s
=> [2/2] RUN echo "Olá! Bem-vinde Maria" > bem-vinde.txt 0.2s
=> exporting to image 0.2s
=> => exporting layers 0.2s
=> => writing image sha256:646ec4da10c4ac12d8e9a5c46295b99a8fa184373124b 0.0s
=> => naming to docker.io/library/arg-demo 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to
fix them
[iMac:~ Menk$ docker container run --rm arg-demo
Olá! Bem-vinde Maria
[iMac:~ Menk$
```

- Exemplos de outros comandos no Dockerfile

ARG

Mais um exemplo de utilização

```
FROM ubuntu
```

```
ARG PYTHON_VERSION
```

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

```
RUN apt-get install python${PYTHON_VERSION} -y
```

```
docker build -t arg-version --build-arg PYTHON_VERSION="2" .
```

```
docker container run --rm -it arg-version bash
```

```
python2 -V
```

```
docker build -t arg-version --build-arg PYTHON_VERSION="3" .
```

```
docker container run --rm -it arg-version bash
```

```
python2 -V
```

```
python3 -V
```

- Exemplos de outros comandos no Dockerfile

ENV

Exemplo de utilização

```
FROM ubuntu
ENV hey="Olá"
ENV dir="/"
# O 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 bash

- 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
```


docker build -t user-demo .

docker container run --rm user-demo


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=CNTR_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

network

-P, --publish-all Publish all exposed ports to random ports.
-p HOST_PORT:CNTR_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:]CNTR_DEST Mount a volume between host and the container file system.
--volumes-from=CNTR_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

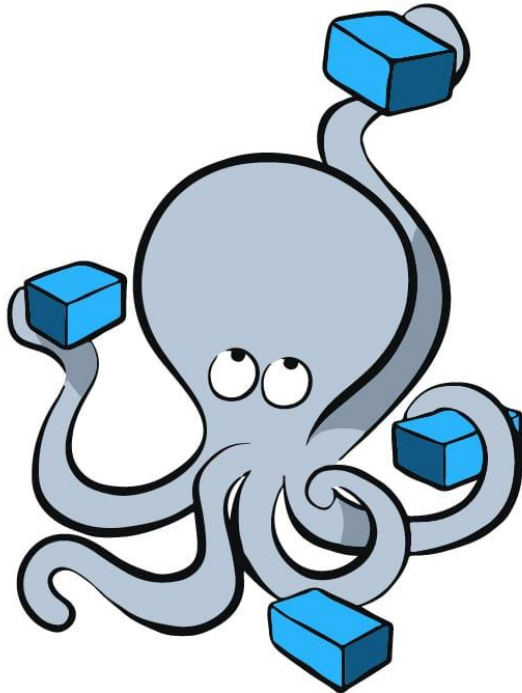
STOPSIGNAL <signal>
 defines signal to use to notify container process to stop

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`

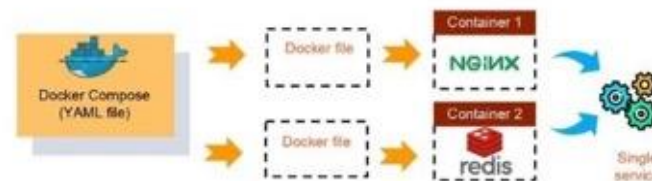




docker Compose

Conhecendo o Docker Compose

- ✓ O Docker Compose é uma ferramenta para definir e gerenciar aplicações docker com múltiplos containers de maneira mais fácil. Com o Docker Compose, você pode criar, configurar e gerenciar vários contêineres Docker como um aplicativo. Neste contexto os containers são chamados de serviços
- ✓ É uma Ferramenta de Coordenação (não orquestração) de Containers
 - ✓ Auxiliar a executar e compor diversos containers com diversos arquivos
 - ✓ Trabalha com múltiplos Containers
- ✓ O Docker Compose já vem instalado por padrão quando instalamos o Docker no Windows ou no Mac, porém no Linux, precisamos realizar sua instalação



Arquitetura do Docker Compose

```
version: "3"
services:
  db:
    container_name: db
    image: mysql
    environment:
      MYSQL_USER: admdimdim
      MYSQL_PASSWORD: admdimdim
      MYSQL_DATABASE: out_stock
      MYSQL_ROOT_PASSWORD: admdimdim
    command: --default-authentication-plugin=mysql_native_password
    ports:
      - "3306:3306"
    networks:
      - outstock_network
    volumes:
      - db_data:/var/lib/mysql
  app:
    container_name: outstock
    build: .
    ports:
      - "5000:5000"
    environment:
      DB_HOST: db
      DB_PORT: 3306
      DB_NAME: out_stock
      DB_USER: admdimdim
      DB_PASSWORD: admdimdim
      AUTH_PLUGIN: mysql_native_password
    depends_on:
      - db
    networks:
      - outstock_network
networks:
  outstock_network:
volumes:
  db_data:
```

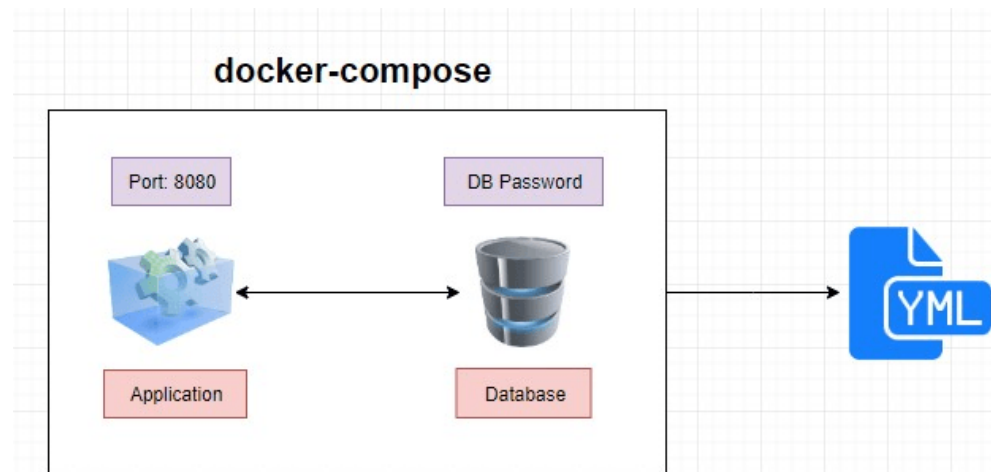
- A arquitetura do Docker Compose é baseada em um arquivo YAML (Yet Another Markup Language), que contém a definição do aplicativo e sua configuração. O arquivo YAML é usado para criar e gerenciar um ou mais containers de aplicativos

- Esse arquivo é composto de várias seções, cada uma delas correspondendo a um serviço, que pode ser um container ou um conjunto de containers que trabalham juntos para oferecer um serviço completo

- Cada serviço é definido por um conjunto de configurações, incluindo a imagem do Container, a porta em que o Container está exposto, as variáveis de ambiente, o volume e a rede a que o Container está conectado

Quando o Docker Compose é executado, ele lê o arquivo YAML e cria e gerencia os containers de aplicativos conforme especificado no arquivo. Ele usa as configurações definidas no arquivo YAML para criar os containers, atribuir os recursos necessários, conectá-los e configurar as variáveis de ambiente

Os Containers criados pelo Docker Compose podem ser executados em um único Host ou em vários Hosts, e o Docker Compose é capaz de gerencia-los de forma centralizada



A DimDim precisa de uma aplicação para gerenciar as solicitações de produtos e que emitam um alarme de baixo estoque. Para isso, você foi designado para desenvolver uma API que permita realizar operações de CRUD em uma tabela chamada OUT_STOCK, com os campos código, nome e data da solicitação

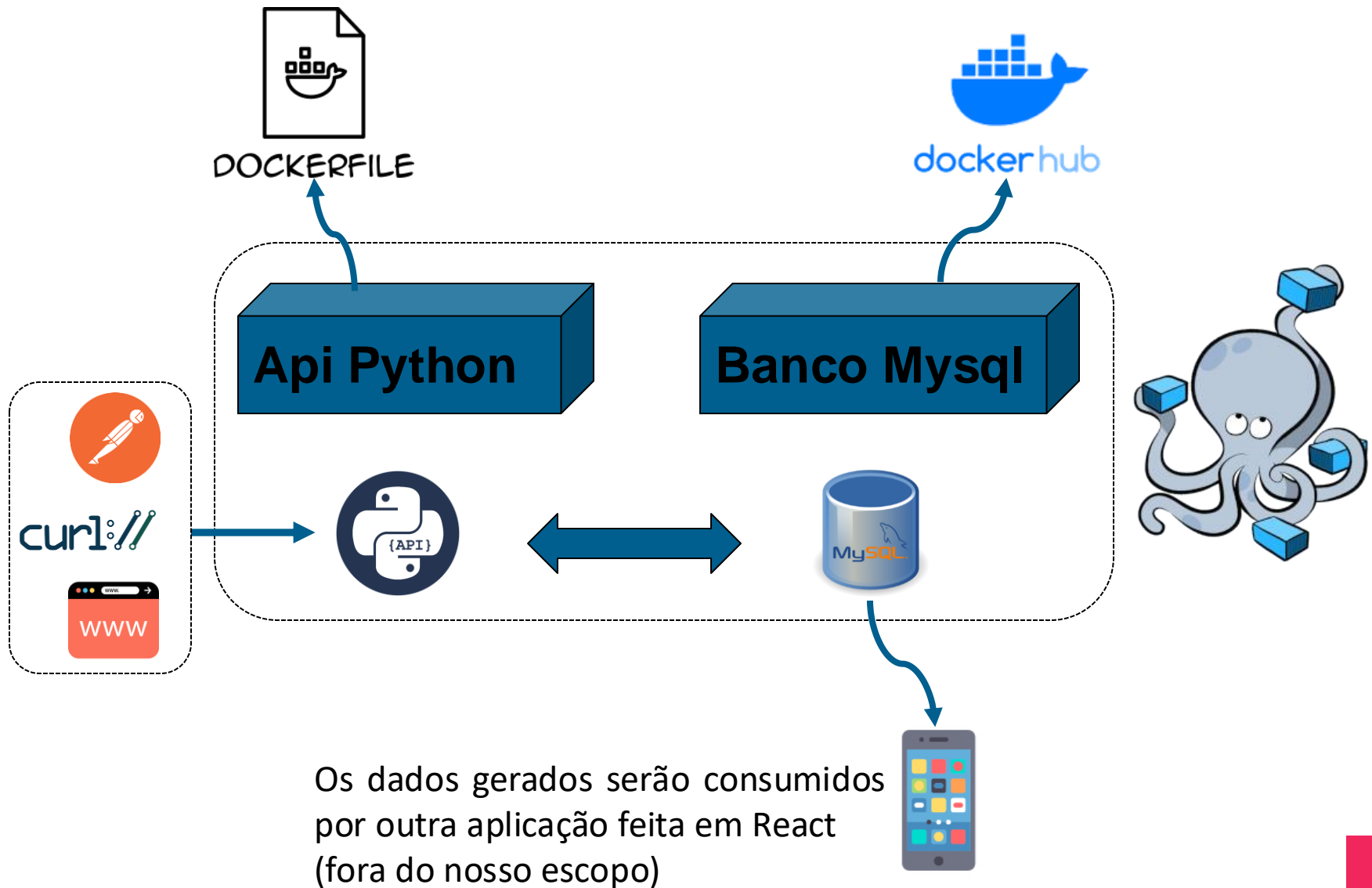
Para o desenvolvimento da aplicação, você optou por utilizar Python como o App e MySQL como o Banco de Dados. Além disso, para facilitar a implantação e o gerenciamento dos componentes da aplicação, você decidiu usar o Docker Compose para criar os Containers e o Serviço



Para começar, precisamos criar um Docker Compose que gerencie dois containers: uma API em Python e um Banco de Dados MySQL. A aplicação em Python é responsável por gerenciar os produtos que irão emitir um alarme de baixo estoque, enquanto o banco de dados armazena as informações dessas transações

Nos próximos passos, vamos ver como a DimDim pode criar um Docker Compose para gerenciar esses dois Containers





Seu objetivo é criar um ambiente de desenvolvimento local que inclua dois containers: um container para a API e outro container para o Banco de Dados. Esses containers devem se comunicar na mesma rede e os dados do Banco devem ser persistidos

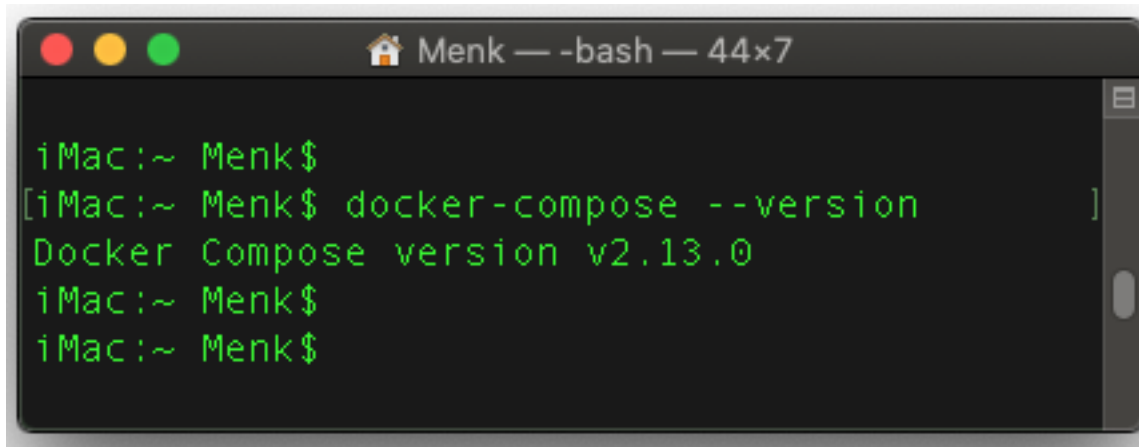
O Desenvolvimento do código da API, Dockerfile e o docker-compose.yml já foram feitos para esse exercício assistido

1. Clonar o Repositório do Git Hub
2. Revisar os códigos fontes (app.py, requirements etc), o arquivo Dockerfile e o arquivo docker-compose.yml

3. Subir o Docker Compose
4. Verificar o ambiente do Banco (Banco, Tabela etc)
5. Verificar as informações sobre Logs, Rede, Volume
6. Realizar os testes do Serviço

Primeiramente vamos verificar se o Docker Compose está instalado e qual sua versão através do comando abaixo

docker-compose --version

A screenshot of a macOS terminal window. The title bar shows a home icon, the name 'Menk', and the shell '-bash' with a window size of '44x7'. The terminal has a dark background with green text. It shows the prompt 'iMac:~ Menk\$' followed by the command 'docker-compose --version' and its output 'Docker Compose version v2.13.0'. The prompt appears again twice more.

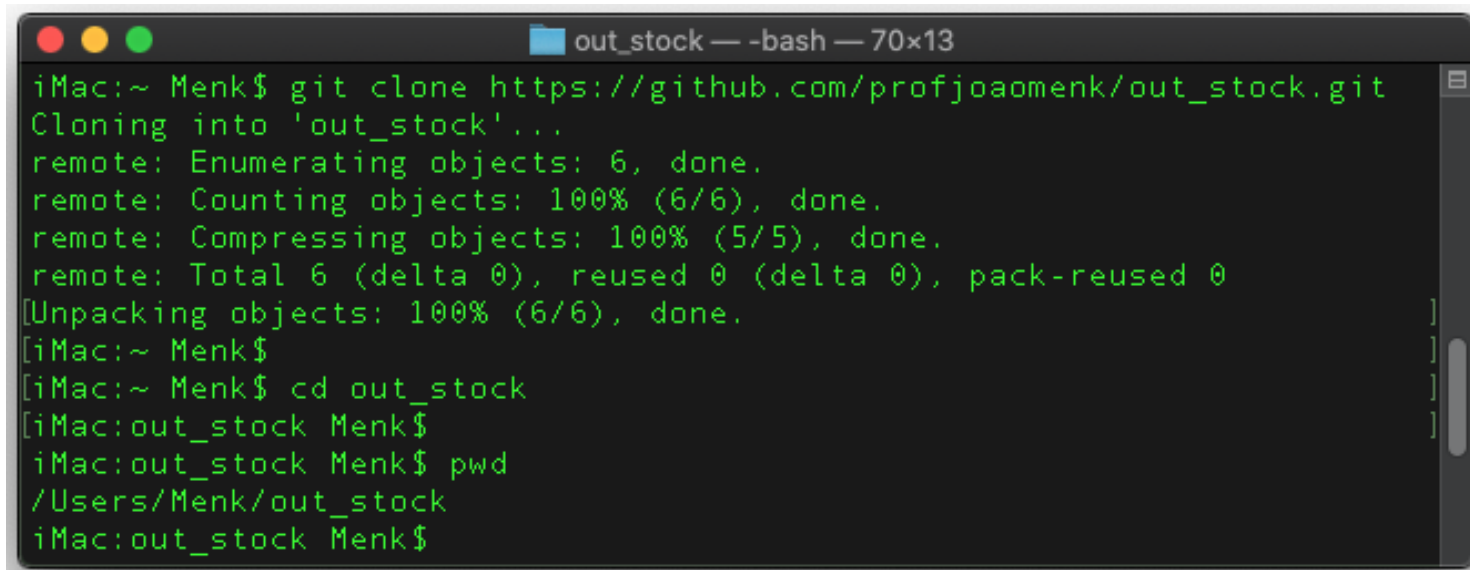
```
iMac:~ Menk$  
[iMac:~ Menk$ docker-compose --version ]  
Docker Compose version v2.13.0  
iMac:~ Menk$  
iMac:~ Menk$
```

Vamos começar a tarefa clonando o fonte do Git Hub

```
git clone https://github.com/profjoaomenk/out_stock.git
```

Agora entre no diretório do Projeto

```
cd out_stock
```

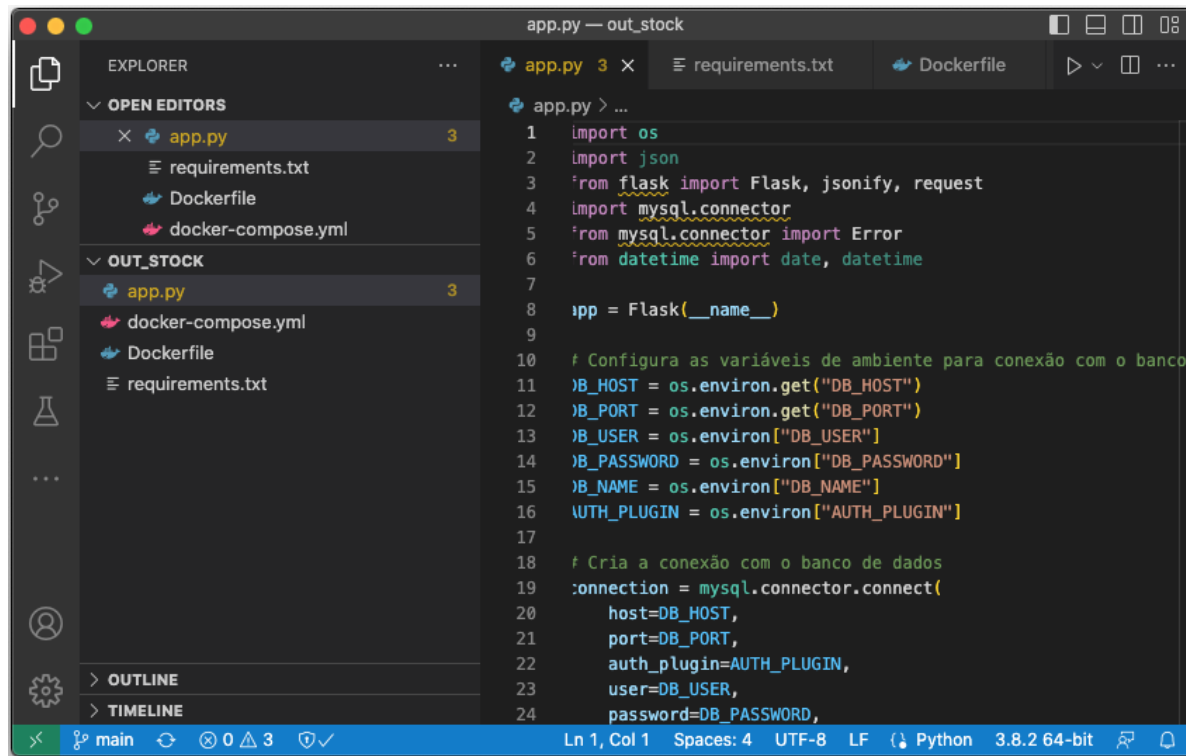


```
out_stock — -bash — 70x13
iMac:~ Menk$ git clone https://github.com/profjoaomenk/out_stock.git
Cloning into 'out_stock'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
[Unpacking objects: 100% (6/6), done.
iMac:~ Menk$
iMac:~ Menk$ cd out_stock
iMac:out_stock Menk$
iMac:out_stock Menk$ pwd
/Users/Menk/out_stock
iMac:out_stock Menk$
```

Inicie o Visual Studio Code e abra a pasta referente ao projeto

Vamos fazer o passo 2 agora:

Revisar os códigos fontes (app.py, requirements etc), o arquivo Dockerfile e o arquivo docker-compose.yml (descrição no próximo Slide)



The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure with folders 'OUT_STOCK' and 'OUTLINE', and files 'app.py', 'requirements.txt', 'Dockerfile', and 'docker-compose.yml'.
- OPEN EDITORS:** Lists the open files: 'app.py' (3 lines), 'requirements.txt', 'Dockerfile', and 'docker-compose.yml'.
- app.py:** The main editor shows the following code:

```
1 import os
2 import json
3 from flask import Flask, jsonify, request
4 import mysql.connector
5 from mysql.connector import Error
6 from datetime import date, datetime
7
8 app = Flask(__name__)
9
10 # Configura as variáveis de ambiente para conexão com o banco
11 DB_HOST = os.environ.get("DB_HOST")
12 DB_PORT = os.environ.get("DB_PORT")
13 DB_USER = os.environ["DB_USER"]
14 DB_PASSWORD = os.environ["DB_PASSWORD"]
15 DB_NAME = os.environ["DB_NAME"]
16 AUTH_PLUGIN = os.environ["AUTH_PLUGIN"]
17
18 # Cria a conexão com o banco de dados
19 connection = mysql.connector.connect(
20     host=DB_HOST,
21     port=DB_PORT,
22     auth_plugin=AUTH_PLUGIN,
23     user=DB_USER,
24     password=DB_PASSWORD,
```

```
version: "3"
services:
  db:
    container_name: db
    image: mysql
    environment:
      MYSQL_USER: admdimdim
      MYSQL_PASSWORD: admdimdim
      MYSQL_DATABASE: out_stock
      MYSQL_ROOT_PASSWORD: admdimdim
    command: --default-authentication-plugin=mysql_native_password
    ports:
      - "3306:3306"
    networks:
      - outstock_network
    volumes:
      - db_data:/var/lib/mysql
  app:
    container_name: outstock
    build: .
    ports:
      - "5000:5000"
    environment:
      DB_HOST: db
      DB_PORT: 3306
      DB_NAME: out_stock
      DB_USER: admdimdim
      DB_PASSWORD: admdimdim
      AUTH_PLUGIN: mysql_native_password
    depends_on:
      - db
    networks:
      - outstock_network
  networks:
    outstock_network:
  volumes:
    db_data:
```

Bloco de configuração denominado **services**: O Docker Compose trata todos os Containers que desejamos executar como serviços

Opção **image** e **build**: Essa opção deve ser utilizada para cada serviço que declararmos dentro do arquivo, pois é com o valor desta opção que o Docker entenderá qual imagem de Container deve ser utilizada para a construção. A opção **build** é onde informaremos o contexto e o arquivo (Dockerfile) que possui as instruções para realizar o build de uma imagem personalizada a ser utilizada

A opção **ports** faz referência às portas que serão utilizadas para acessar os serviços providos dentro do Container, onde é utilizada para informar a(s) porta(s) do sistema hospedeiro que receberá as requisições e para qual porta deve encaminhar estas requisições para dentro do Container. Existe a possibilidade de utilizar a opção **expose** ao invés de **ports**, aonde as requisições são tratadas apenas nas redes a qual este Container faz parte (serviços se comuniquem entre si)

Podemos informar com a opção **depends_on** que, para que um serviço seja iniciado, ele depende que outro seja iniciado primeiro, criando uma dependência

Com a opção **environment** conseguimos definir variáveis de ambientes para utilizar dentro de nossos Containers

Na opção **networks** é onde podemos definir as redes que deverão ser criadas para que os serviços dela façam parte. Para que uma rede seja criada é necessário **realizar sua declaração**, onde passamos o nome da rede a ser criada, podemos passar o tipo da rede e driver e até mesmo a Subnet. Feita a declaração é necessário **referenciar a rede na configuração do serviço**, para que este faça uso da rede criada

Assim como na opção **networks**, aqui em **volumes** nós devemos fazer a declaração dos volumes que desejamos criar e referenciar depois dentro de cada serviço que irá utilizá-lo

Podemos nomear cada Container

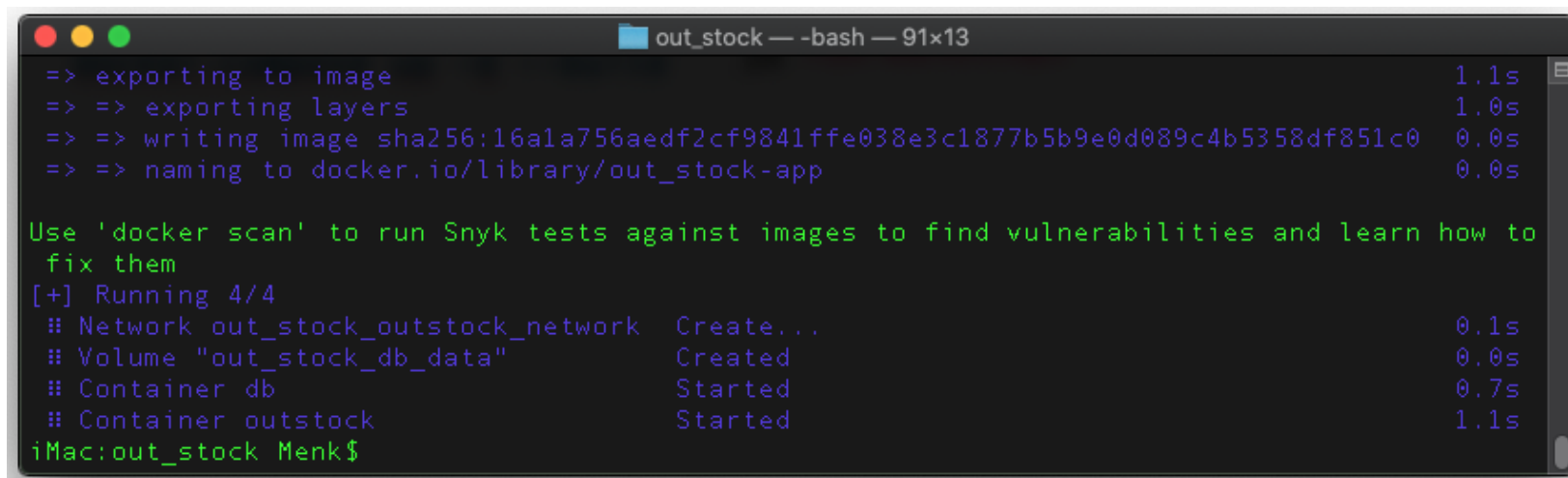
Agora, como descrito no passo 3, iremos subir o Docker Compose

- 1) Abra um Terminal no VSC ou utilize um desacoplado (CMD/Terminal)
- 2) Certifique-se de estar no diretório Home do Projeto
- 3) Com o comando abaixo nos iremos subir o Serviço em Segundo Plano

docker-compose up -d --build

*#fica
dica*

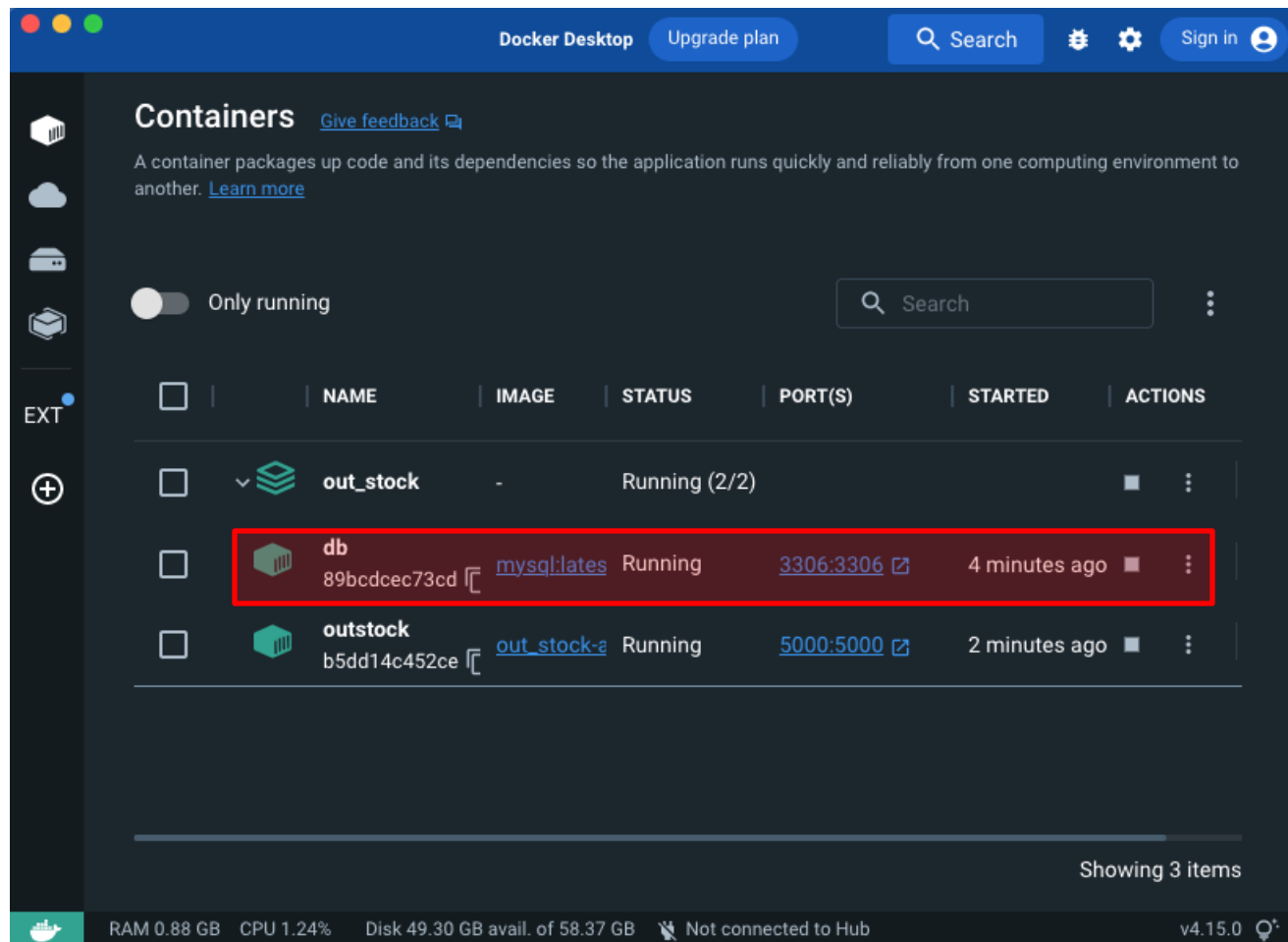
A opção **--build** refaz as imagens



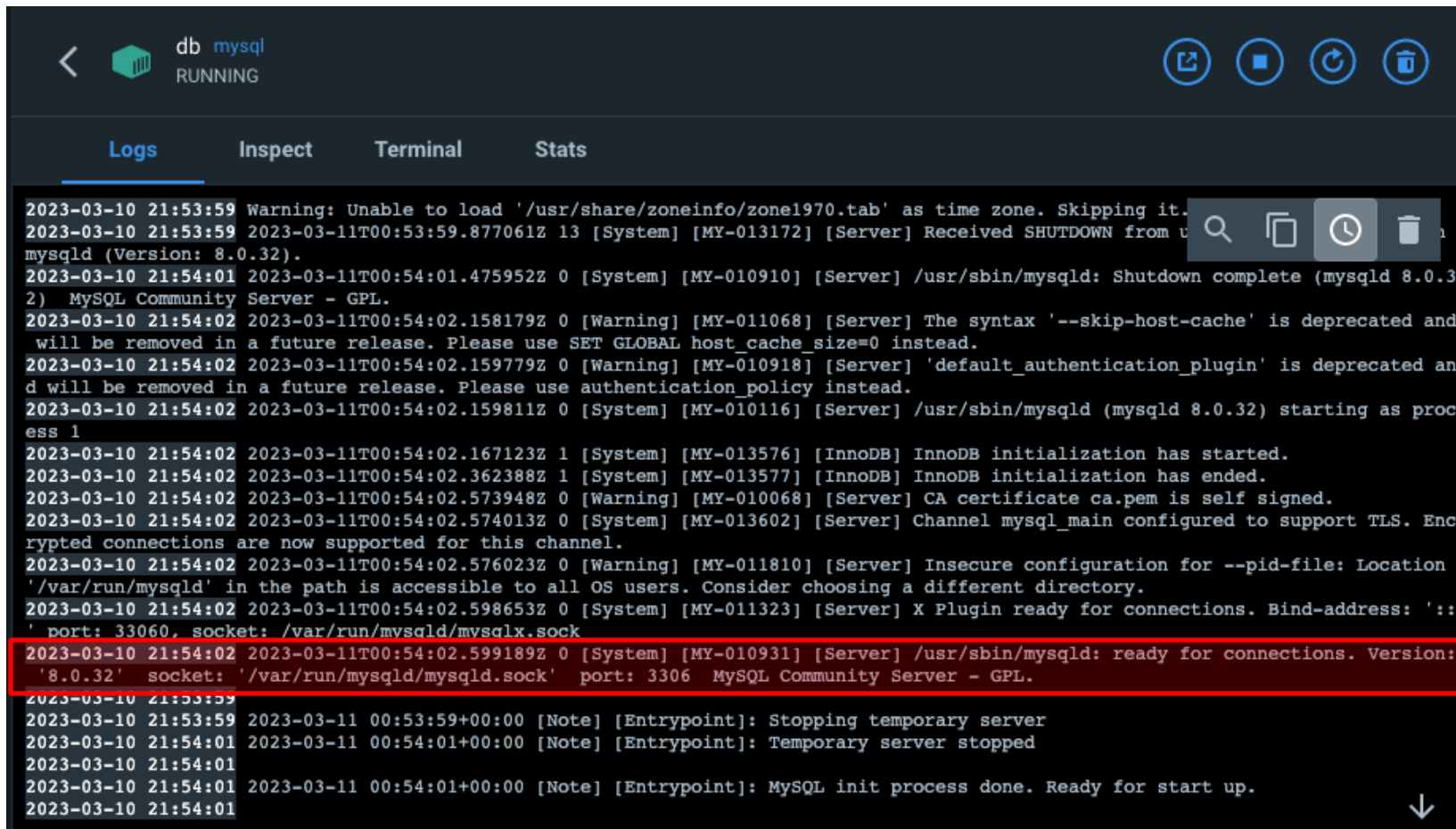
```
out_stock — -bash — 91x13
=> exporting to image 1.1s
=> => exporting layers 1.0s
=> => writing image sha256:16a1a756aedef2cf9841ffe038e3c1877b5b9e0d089c4b5358df851c0 0.0s
=> => naming to docker.io/library/out_stock-app 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to
fix them
[+] Running 4/4
  :: Network out_stock_outstock_network Create... 0.1s
  :: Volume "out_stock_db_data" Created 0.0s
  :: Container db Started 0.7s
  :: Container outstock Started 1.1s
iMac:out_stock Menk$
```


Com o passo 4 vamos verificar o ambiente de Banco no Docker Desktop, expanda nosso Serviço e clique no Container **db**



Verifique o Log do MySQL, certificando que foi iniciado



```
2023-03-10 21:53:59 Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
2023-03-10 21:53:59 2023-03-11T00:53:59.877061Z 13 [System] [MY-013172] [Server] Received SHUTDOWN from u
mysqld (Version: 8.0.32).
2023-03-10 21:54:01 2023-03-11T00:54:01.475952Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 8.0.3
2) MySQL Community Server - GPL.
2023-03-10 21:54:02 2023-03-11T00:54:02.158179Z 0 [Warning] [MY-011068] [Server] The syntax '--skip-host-cache' is deprecated and
will be removed in a future release. Please use SET GLOBAL host_cache_size=0 instead.
2023-03-10 21:54:02 2023-03-11T00:54:02.159779Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plugin' is deprecated an
d will be removed in a future release. Please use authentication_policy instead.
2023-03-10 21:54:02 2023-03-11T00:54:02.159811Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.32) starting as proc
ess 1
2023-03-10 21:54:02 2023-03-11T00:54:02.167123Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2023-03-10 21:54:02 2023-03-11T00:54:02.362388Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2023-03-10 21:54:02 2023-03-11T00:54:02.573948Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2023-03-10 21:54:02 2023-03-11T00:54:02.574013Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Enc
rypted connections are now supported for this channel.
2023-03-10 21:54:02 2023-03-11T00:54:02.576023Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location
'/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
2023-03-10 21:54:02 2023-03-11T00:54:02.598653Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::
' port: 33060, socket: /var/run/mysqld/mysqld.sock
2023-03-10 21:54:02 2023-03-11T00:54:02.599189Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version:
'8.0.32' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
2023-03-10 21:53:59 2023-03-11 00:53:59+00:00 [Note] [Entrypoint]: Stopping temporary server
2023-03-10 21:54:01 2023-03-11 00:54:01+00:00 [Note] [Entrypoint]: Temporary server stopped
2023-03-10 21:54:01 2023-03-11 00:54:01+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up.
2023-03-10 21:54:01
```

Pode acontecer do Container da API não estar em Running, pois o banco ainda não subiu completamente, aguarde a subida completa do Serviço e clique em iniciar no Container do App (outstock)



**DO
NOT
PANIC**

<input type="checkbox"/>		out_stock	-	Running (1/2)			
<input type="checkbox"/>		db 89bcdcec73cd	mysql:latest	Running	3306:3306	10 minutes ago	
<input type="checkbox"/>		outstock b5dd14c452ce	out_stock-app:latest	Exited	5000:5000		



Continuando nossa exploração no Serviço do Banco, clique na aba Terminal e se logue no Banco

```
mysql -u admdimdim -p
```



The screenshot shows the MySQL monitor interface. At the top, there's a header bar with a back arrow, a database icon, and the text "db mysql RUNNING". To the right are four circular icons: a square, a refresh, a trash, and a search. Below the header is a tab bar with four tabs: "Logs", "Inspect", "Terminal" (which is highlighted with a red border), and "Stats". To the right of the tabs is a link that says "Open in external terminal". The main area is a terminal window with a dark background. It shows the command "sh-4.4# mysql -u admdimdim -p" being entered. Below that, it says "Enter password:". Then, it displays the MySQL welcome message: "Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 12 Server version: 8.0.32 MySQL Community Server - GPL". It also shows the copyright notice: "Copyright (c) 2000, 2023, Oracle and/or its affiliates. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners." and the help text: "Type 'help;' or '\h' for help. Type '\c' to clear the current input statement." The prompt "mysql>" is at the bottom.

```
sh-4.4# mysql -u admdimdim -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Verifique algumas informações sobre o Banco, Tabelas etc

```
show databases;
```

```
use out_stock;
```

```
SELECT * FROM out_stock;
```

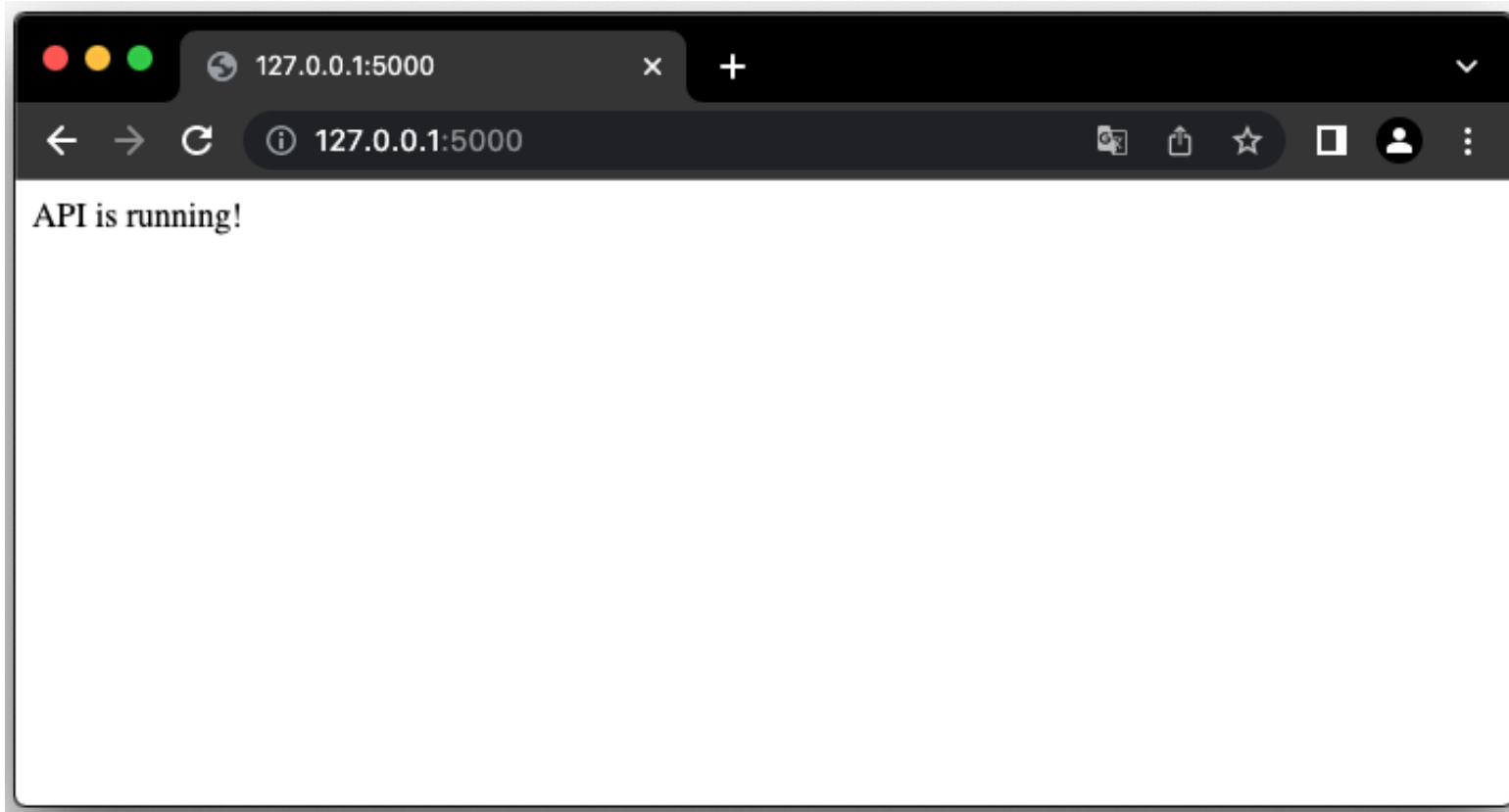
```
DESCRIBE out_stock;
```

```
SHOW TABLES;
```

```
SHOW GRANTS FOR admdimdim;
```

```
exit
```

Teste a execução da API pelo Browser no seguinte endereço



O Passo 5 pede para verificar as informações sobre Logs, Rede, Volume
Vamos realizar esses procedimentos agora pelo Terminal



Similar ao docker ps, mas se limitando aos serviços indicados no docker-compose.yml
docker-compose ps

```
iMac:out_stack Menk$  
iMac:out_stack Menk$ docker-compose ps  
NAME                COMMAND                  SERVICE    STATUS    PORTS  
db                  "docker-entrypoint.s..." db         running   0.0.0.0:3306->3306/tcp, 33060/tcp  
outstock            "python app.py"         app        running   0.0.0.0:5000->5000/tcp  
iMac:out_stack Menk$  
iMac:out_stack Menk$
```

Visualiza os logs dos Containers
docker-compose logs

```
instead.  
outstock * Running on all addresses (0.0.0.0)  
outstock * Running on http://127.0.0.1:5000  
outstock * Running on http://172.18.0.3:5000  
outstock Press CTRL+C to quit  
outstock * Restarting with stat  
outstock * Debugger is active!  
outstock * Debugger PIN: 738-968-848  
outstock * Serving Flask app "app" (lazy loading)  
outstock * Environment: production  
outstock WARNING: This is a development server. Do not use it in a production deployment.  
outstock Use a production WSGI server instead.  
outstock * Debug mode: on  
outstock WARNING: This is a development server. Do not use it in a production deployment. Use a production WSG
```

```
and will be removed in a future release. Please use authentication_policy instead.  
db | 2023-03-11T04:16:16.746444Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.32) starting  
as process 1  
db | 2023-03-11T04:16:16.754248Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.  
db | 2023-03-11T04:16:16.853522Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.  
db | 2023-03-11T04:16:19.235767Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.  
db | 2023-03-11T04:16:19.235875Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support  
TLS. Encrypted connections are now supported for this channel.  
db | 2023-03-11T04:16:19.237433Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: L  
ocat on '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.  
db | 2023-03-11T04:16:19.450852Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections.  
Vers on: '8.0.32' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.  
db | 2023-03-11T04:16:19.450850Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-addr  
ess: '::' port: 33060, socket: /var/run/mysqld/mysqldx.sock
```

Verificar as Redes

docker network ls

```
out_stock — -bash — 63x8
[iMac:out_stock Menk$ docker network ls]
NETWORK ID      NAME                DRIVER  SCOPE
745bb662f17c    bridge              bridge  local
b5f4a5aeb449    host                host    local
7a8860c2561f    none               null    local
51f5f3582d68    out_stock_outstock_network  bridge  local
iMac:out_stock Menk$
```

docker network inspect out_stock_outstock_network

```
out_stock — -bash — 98x16
"Containers": {
  "89bcdcec73cdf6bf7deb0edc73651e39cbab9ac4855a6c0e4042089bf6a0cb70": {
    "Name": "db",
    "EndpointID": "c8615c57aca55ddb286275a0c6e97b6729006751d8b00b9812434d7d42193139",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  },
  "b5dd14c452ce5de03cf00a2c1e1995da41964782a8d196a4b7ec8874c2189b3c": {
    "Name": "outstock",
    "EndpointID": "bbe27b84906e89503f2b985bc2fbe969352826cc333141ec447366048c472b28",
    "MacAddress": "02:42:ac:12:00:03",
    "IPv4Address": "172.18.0.3/16",
    "IPv6Address": ""
  }
}
```


Verificar os Volumes

docker volume ls

```
out_stock — -bash — 44x5
iMac:out_stock Menk$ docker volume ls
DRIVER      VOLUME NAME
local       out_stock_db_data
iMac:out_stock Menk$
```

docker volume inspect out_stock_db_data

```
out_stock — -bash — 75x17
iMac:out_stock Menk$ docker volume inspect out_stock_db_data
[
  {
    "CreatedAt": "2023-03-11T04:16:19Z",
    "Driver": "local",
    "Labels": {
      "com.docker.compose.project": "out_stock",
      "com.docker.compose.version": "2.13.0",
      "com.docker.compose.volume": "db_data"
    },
    "Mountpoint": "/var/lib/docker/volumes/out_stock_db_data/_data",
    "Name": "out_stock_db_data",
    "Options": null,
    "Scope": "local"
  }
]
iMac:out_stock Menk$
```

Don't
forget



Por default, o Docker **Windows** disponibiliza acesso na seguinte localização:

Docker Engine v19:

`\\wsl$\docker-desktop-data\version-pack-data\community\docker\volumes\`

Docker Engine v20:

`\\wsl$\docker-desktop-data\data\docker\volumes`

Por default, o Docker **Mac*** / **Linux** disponibiliza acesso na seguinte localização:

`/var/lib/docker/volumes`

Aprendendo na Prática

Entrar no Terminal do MySQL

docker exec -it db mysql -u admdimdim -p

```
out_stock — com.docker.cli • docker exec -it db mysql -uadmdimdim -p — 78x17
iMac:out_stock Menk$
iMac:out_stock Menk$ docker exec -it db mysql -uadmdimdim -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
[
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

Entrar no Terminal da API

docker exec -it outstock /bin/bash

```
out_stock — com.docker.cli • docker exec -it outstock /bin/bash — 56x9
[iMac:out_stock Menk$ docker exec -it outstock /bin/bash ]
[root@b5dd14c452ce:/app# pwd ]
[/app ]
[root@b5dd14c452ce:/app# ls ]
Dockerfile  docker-compose.yml
app.py      requirements.txt
root@b5dd14c452ce:/app#
```

Agora vamos realizar os testes em nosso Serviço

Quem utiliza Mac / Linux pode realizar os testes pelo **curl** no terminal

Select:

```
curl http://localhost:5000/out_stock
```

Insert:

```
curl -X POST -H "Content-Type: application/json" -d '{"codigo": "001", "descricao": "Produto 1", "data_solicitacao": "2022-03-10"}' http://localhost:5000/out_stock  
curl -X POST -H "Content-Type: application/json" -d '{"codigo": "002", "descricao": "Produto 2", "data_solicitacao": "2022-03-10"}' http://localhost:5000/out_stock  
curl -X POST -H "Content-Type: application/json" -d '{"codigo": "003", "descricao": "Produto 3", "data_solicitacao": "2022-03-10"}' http://localhost:5000/out_stock
```

Update:

```
curl -X PUT -H "Content-Type: application/json" -d '{"codigo": "001", "descricao": "Produto 1 atualizado", "data_solicitacao": "2022-03-09"}' http://localhost:5000/out_stock/1  
curl -X PUT -H "Content-Type: application/json" -d '{"codigo": "001", "descricao": "Produto dois", "data_solicitacao": "2022-03-10"}' http://localhost:5000/out_stock/2
```

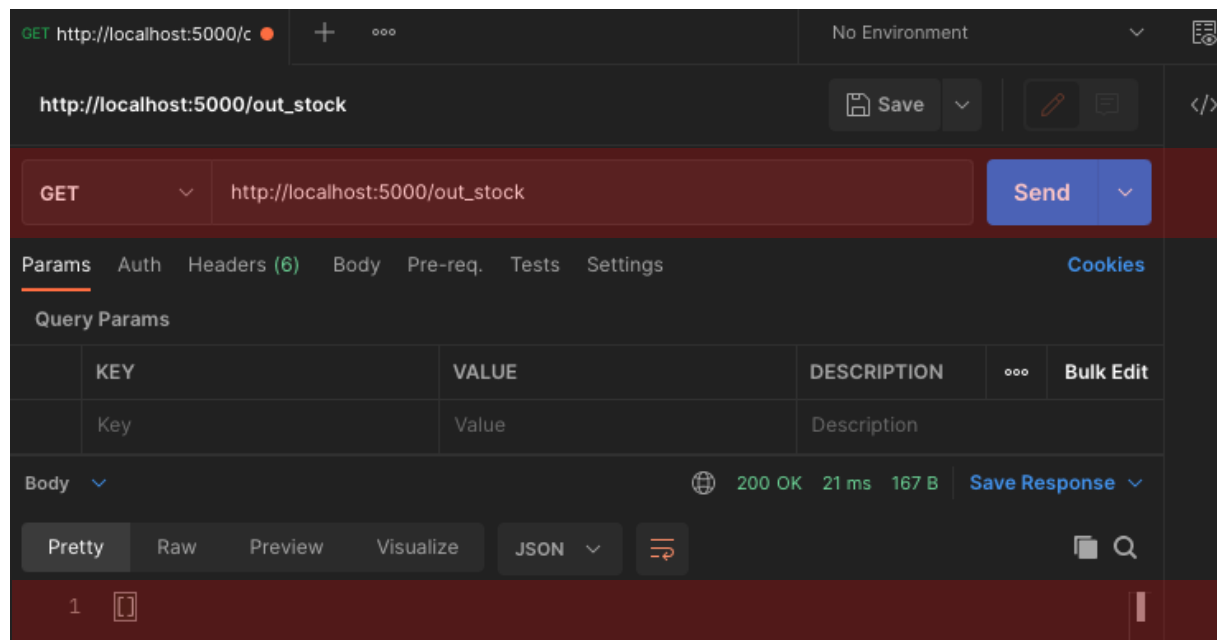
Delete:

```
curl -X DELETE http://localhost:5000/out_stock/003
```

Pelo Windows iremos realizar os Testes via Postman

Para buscar os registros da tabela

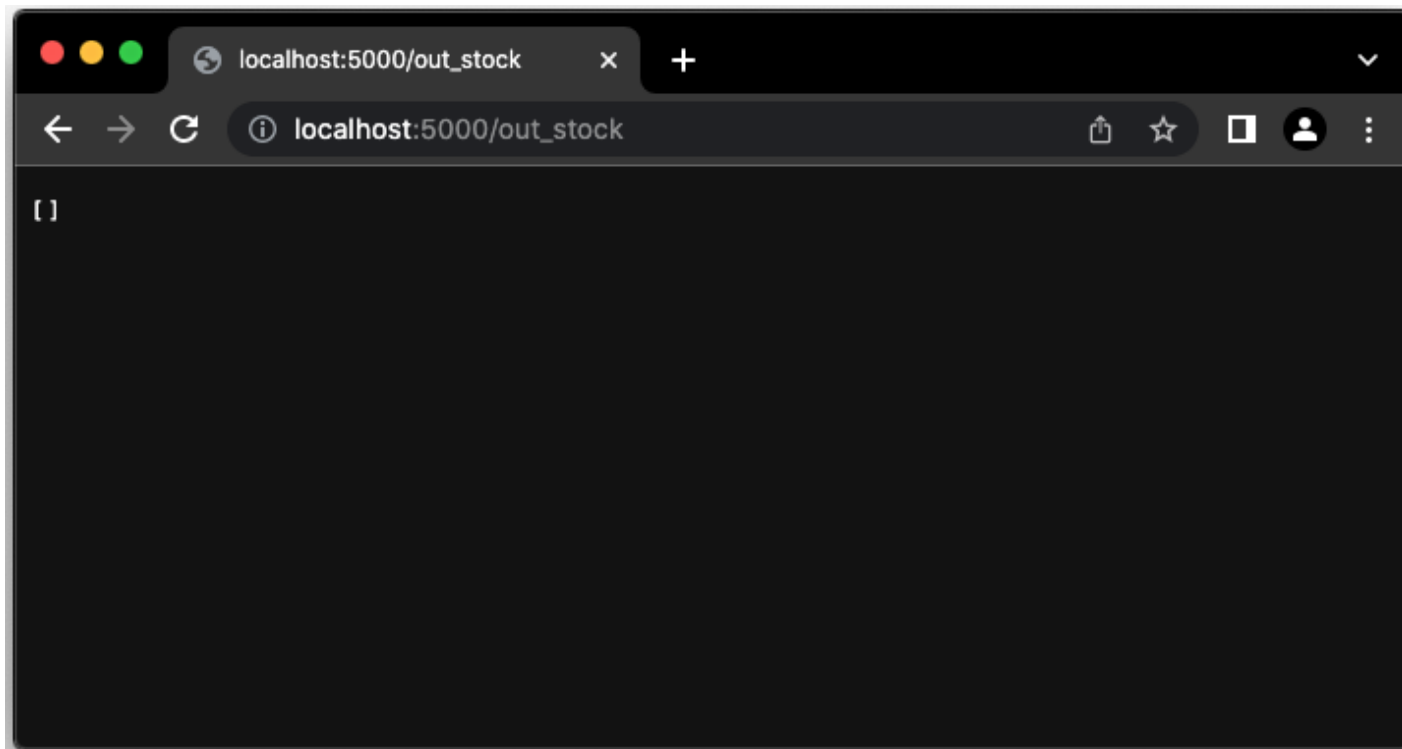
1. Abra o Postman e crie uma nova requisição
2. Selecione o método **HTTP GET** e informe a URL: **http://localhost:5000/out_stock**
3. Clique em "Send" para enviar a requisição
4. O resultado será exibido na aba "Body" da resposta, contendo a representação JSON dos registros da tabela "out_stock"



Para buscar os registros da tabela

O Browser da Internet também serve para realizar essa operação

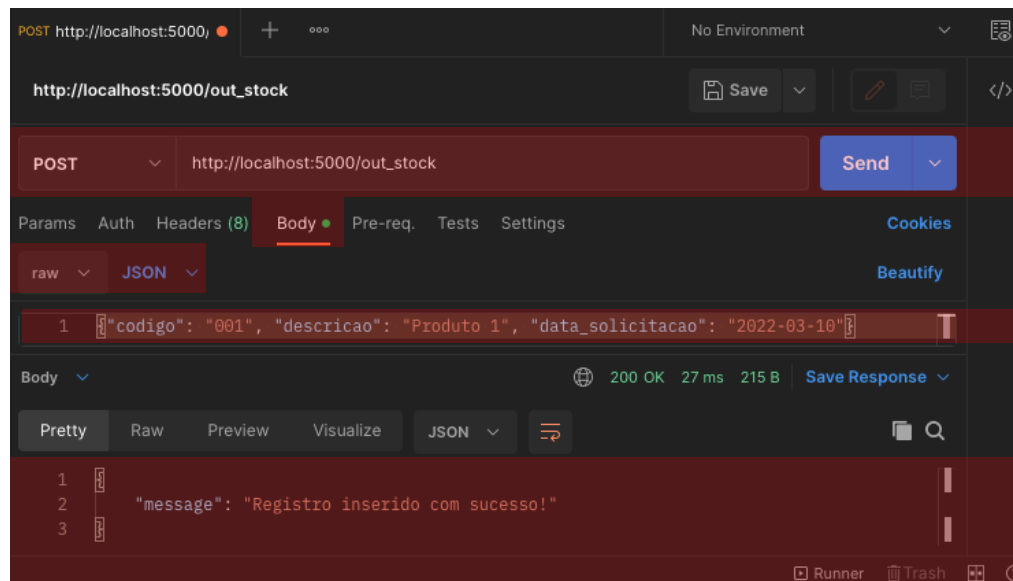
http://localhost:5000/out_stock



Para inserir um novo registro na tabela

1. Selecione o método **HTTP POST** e informe a URL: **http://localhost:5000/out_stock**
2. Selecione a aba "**Body**", escolha a opção "**raw**" e defina o formato para "**JSON**"
3. No campo de edição, informe os dados do novo registro em formato JSON
4. Clique em "Send" para enviar a requisição
5. O resultado será exibido na aba "Body" da resposta, contendo a representação JSON do registro inserido

```
{"codigo": "001", "descricao": "Produto 1", "data_solicitacao": "2022-03-10"}  
{"codigo": "002", "descricao": "Produto 2", "data_solicitacao": "2022-03-10"}  
{"codigo": "003", "descricao": "Produto 3", "data_solicitacao": "2022-03-10"}
```



Para atualizar um registro na tabela

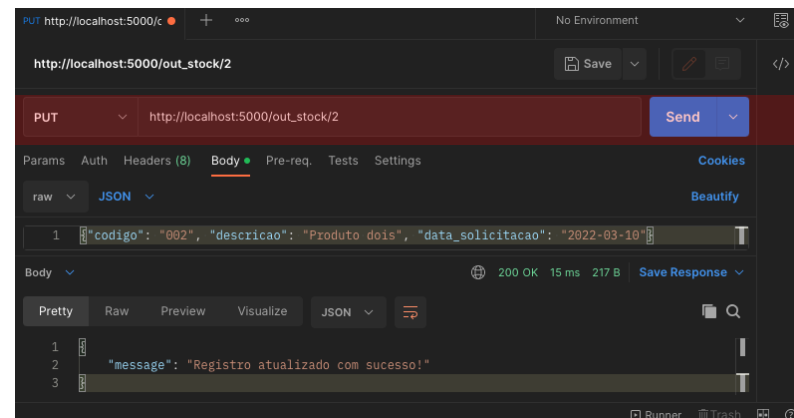
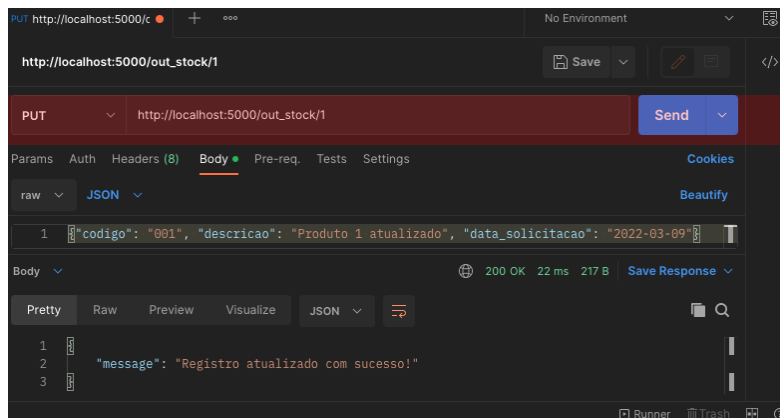
1. Selecione o método **HTTP PUT** e informe a URL com o **id do registro** a ser atualizado (**PK**): **http://localhost:5000/out_stock/1** (no exemplo, o id é "1")
2. Selecione a aba "**Body**", escolha a opção "**raw**" e defina o formato para "**JSON**"
3. No campo de edição, informe os dados atualizados do registro em formato JSON
4. Clique em "Send" para enviar a requisição
5. O resultado será exibido na aba "Body" da resposta, contendo a representação JSON do registro atualizado

URL: http://localhost:5000/out_stock/1

Linha: {"codigo": "001", "descricao": "Produto 1 atualizado", "data_solicitacao": "2022-03-09"}

URL: http://localhost:5000/out_stock/2

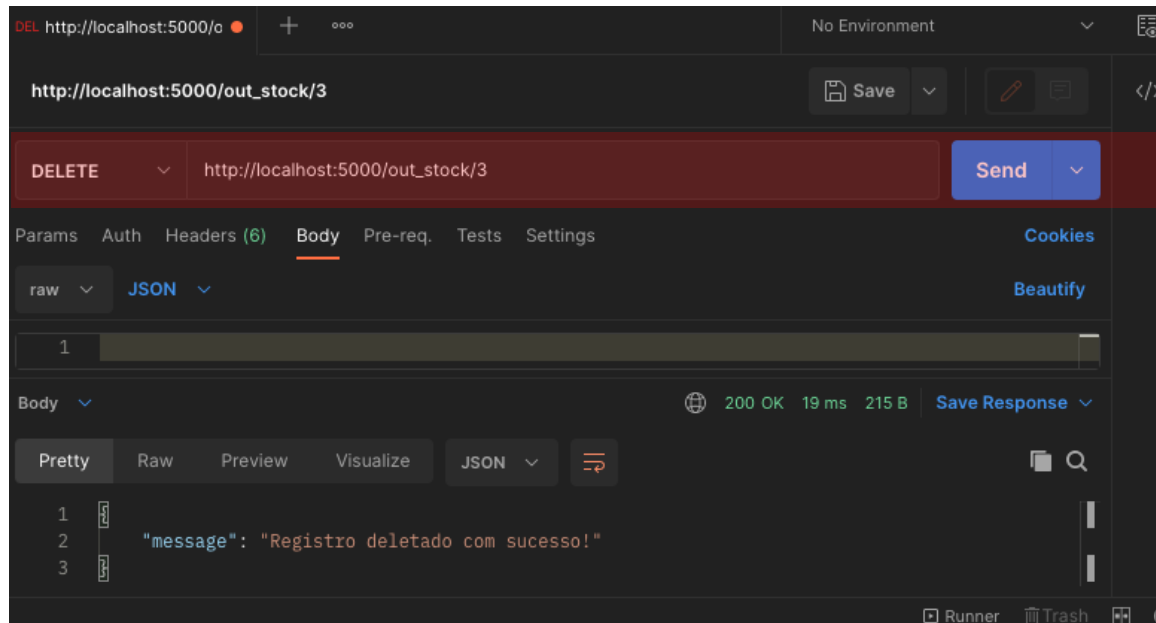
Linha: {"codigo": "002", "descricao": "Produto dois", "data_solicitacao": "2022-03-10"}



Para deletar um registro na tabela

1. Selecione o método **HTTP DELETE** e informe a URL com o **id do registro** a ser deletado (**PK**): **http://localhost:5000/out_stock/3** (no exemplo, o código é "3")
2. Clique em "Send" para enviar a requisição
3. O resultado será exibido na aba "Body" da resposta, contendo uma mensagem indicando se a operação foi realizada com sucesso ou não

http://localhost:5000/out_stock/3



Comandos adicionais importantes

Inicia os Containers:

`docker-compose start`

Reinicia os Containers:

`docker-compose restart`

Paralisa os Containers:

`docker-compose stop`

Para e remove todos os Containers e seus componentes como rede, imagem e volume (Não deleta: Imagens nem Volumes, só elimina o vínculo):

`docker-compose down`

DOCKER COMPOSE CHEAT SHEET

File

structure

```
services:  
  container1:  
    properties: values  
  
  container2:  
    properties: values
```

```
networks:  
  network:
```

```
volumes:  
  volume:
```

Types value

```
key: value
```

array

```
key:  
  - value  
  - value
```

dictionary

```
master:  
  key: value  
  key: value
```

Properties

build

build image from dockerfile
in specified directory

```
container:  
  build: ./path  
  image: image-name
```

image

use specified image

```
image: image-name
```

container_name

define container name to access
it later

```
container_name: name
```

volumes

define container volumes to
persist data

```
volumes:  
  - /path:/path
```

command

override start command for the
container

```
command: execute
```

environment

define env variables for the
container

```
environment:  
  KEY: VALUE  
---  
environment:  
  - KEY=VALUE
```

env_file

define a env file for the
container to set and override
env variables

```
env_file: .env  
---  
env_file:  
  - .env
```

restart

define restart rule
(no, always, on-failure, unless-
stopped)

```
restart:  
  - "9999"
```

networks

define all networks for the
container

```
networks:  
  - network-name
```

ports

define ports to expose to other
containers and host

```
ports:  
  - "9999:9999"
```

expose

define ports to expose only to
other containers

```
expose:  
  - "9999"
```

network_mode

define network driver
(bridge, host, none, etc.)

```
network_mode: host
```

depends_on

define build, start and stop
order of container

```
depends_on:  
  - container-name
```

Other

idle container

send container to idle state
> container will not stop

```
command: tail -f /dev/null
```

named volumes

create volumes that can be used in
the volumes property

```
services:  
  container:  
    image: image-name  
    volumes:  
      - data-  
volume:/path/to/dir
```

```
volumes:  
  data-volume:
```

networks

create networks that can be used
in the networks property

```
networks:  
  frontend:  
    driver: bridge
```



LIMPAR O LABORATÓRIO DO DOCKER COMPOSE

`docker-compose down`

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



Copyright © 2023 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).