

FIAP GRADUAÇÃO

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

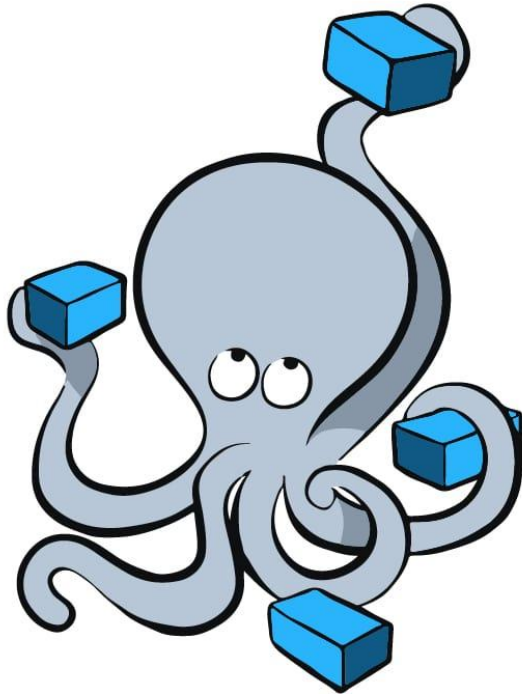
profthiago.rocha@fiap.com.br

PROF. Thiago Moraes

proftiago.moraes@fiap.com.br

PROF. Rafael Pereira

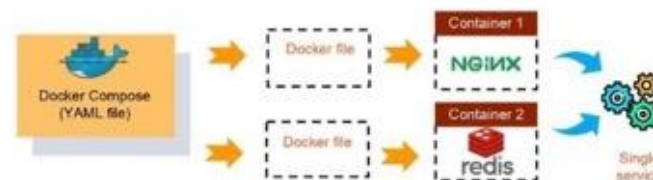
profrafael.pereira@fiap.com.br



docker 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

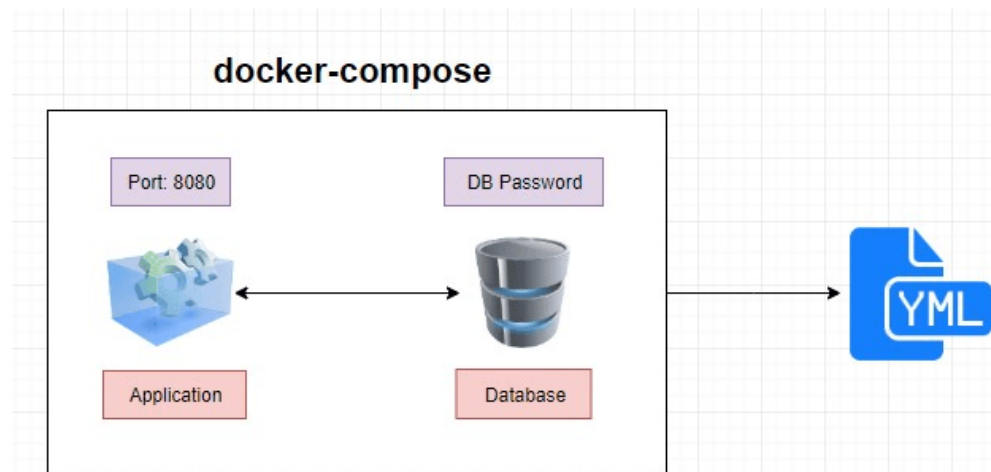
- Esse arquivo é composto de várias seções, cada uma delas correspondendo a um serviço, que pode ser um container ou um conjunto de containers que trabalham juntos para oferecer um serviço completo

- 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

Arquitetura do Docker Compose

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

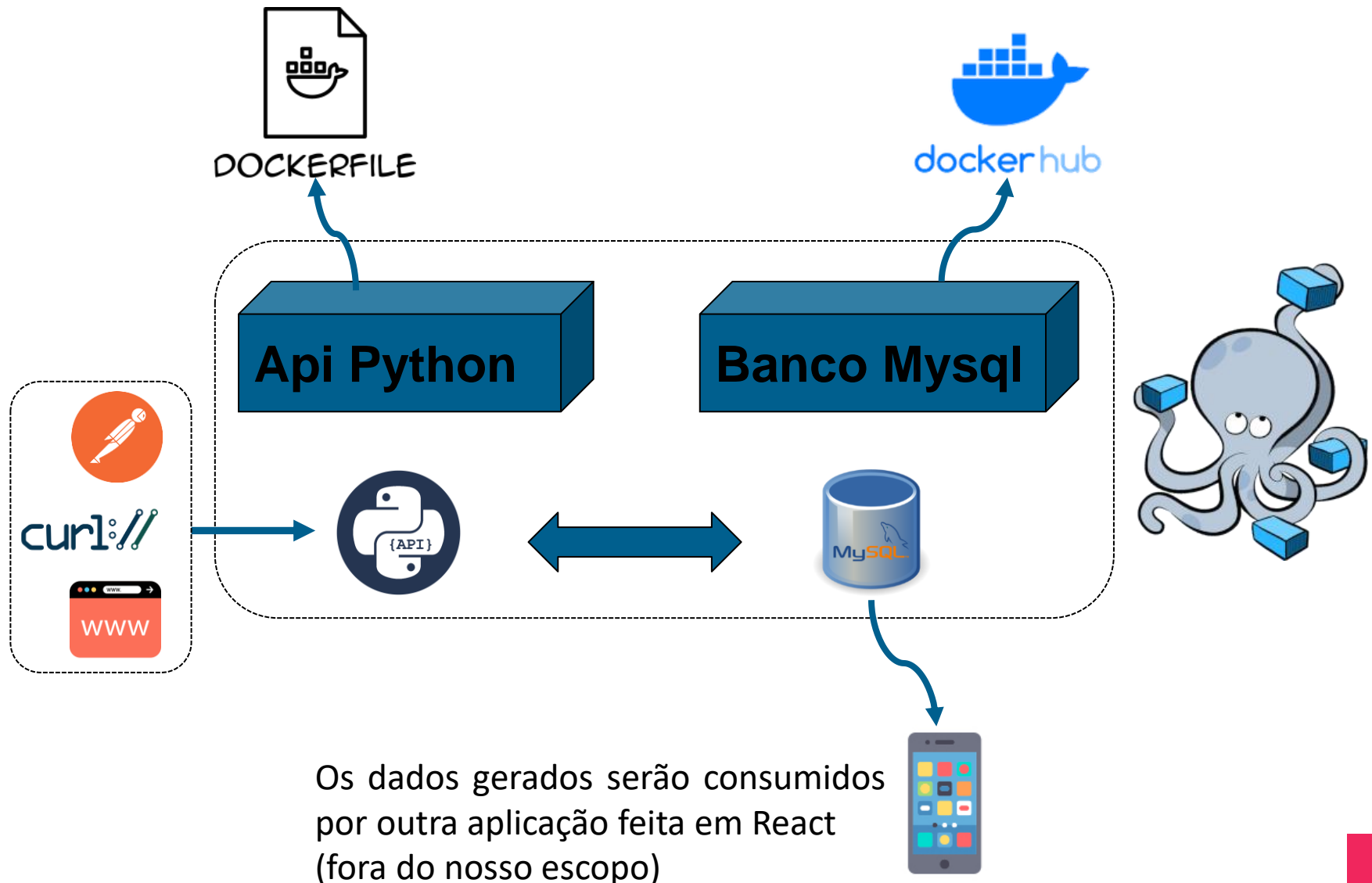
Para o desenvolvimento da aplicação, você optou por utilizar Python como o App e MySQL como o Banco de Dados. Além disso, para facilitar a implantação e o gerenciamento dos componentes da aplicação, você decidiu usar o Docker Compose para criar os Containers e o Serviço



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

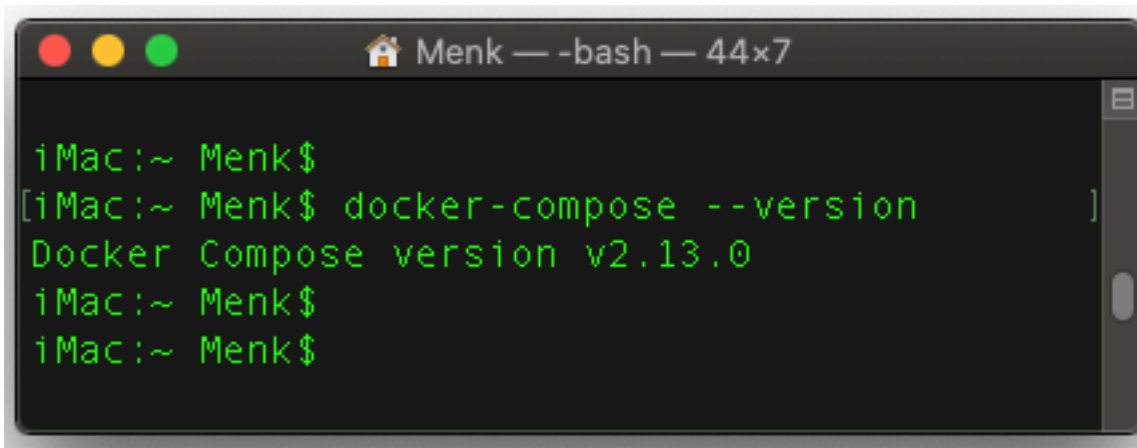
O Desenvolvimento do código da API, Dockerfile e o docker-compose.yml já foram feitos para esse exercício assistido

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 text is as follows:

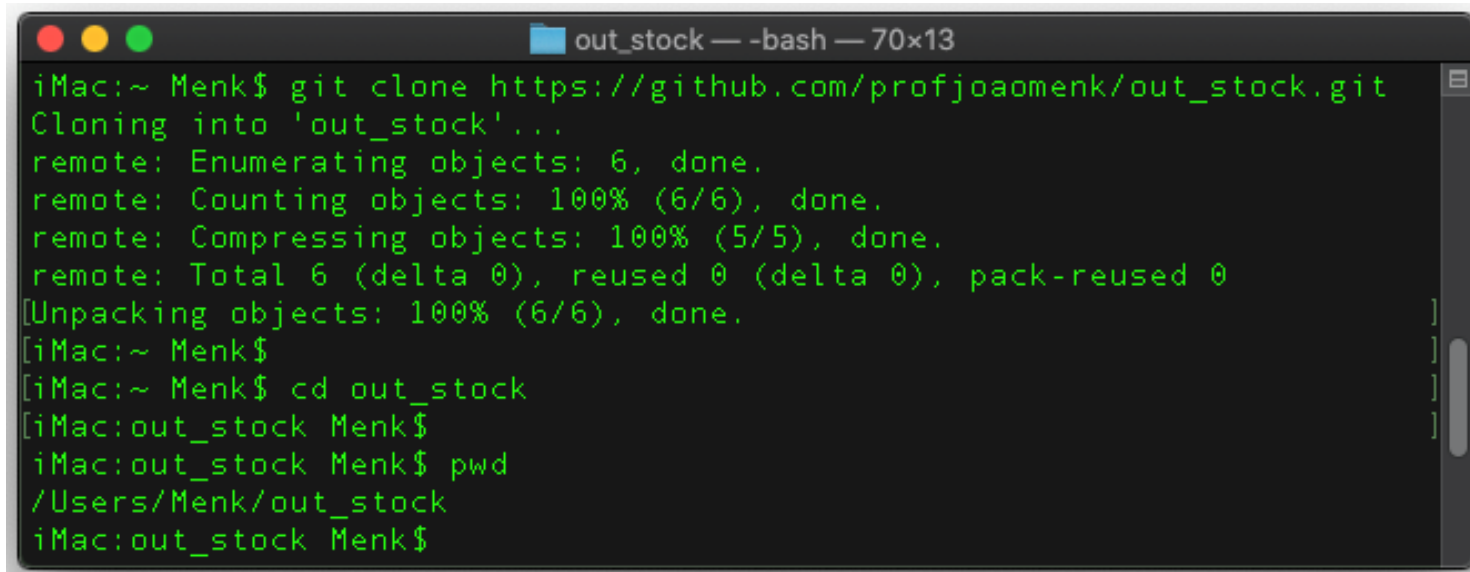
```
iMac:~ Menk$  
[iMac:~ Menk$ docker-compose --version ]  
Docker Compose version v2.13.0  
iMac:~ Menk$  
iMac:~ Menk$
```

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
```

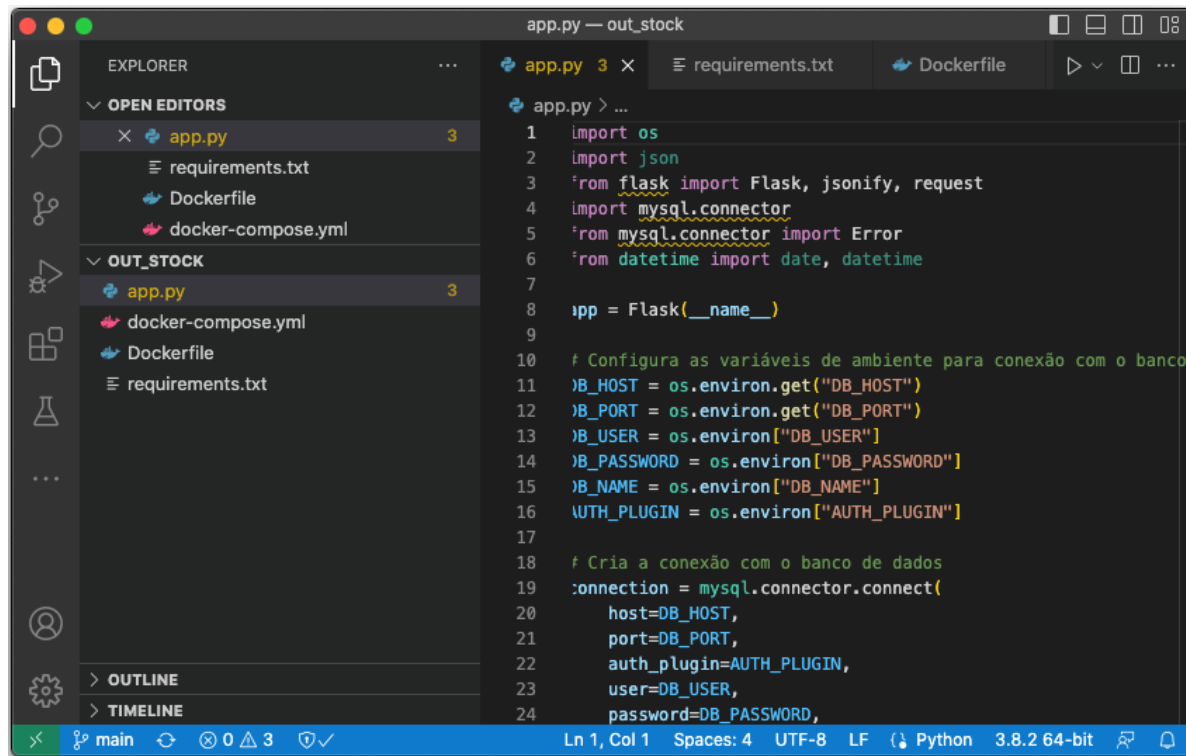


```
out_stock — -bash — 70x13
iMac:~ Menk$ git clone https://github.com/profjoaomenk/out_stock.git
Cloning into 'out_stock'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0),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

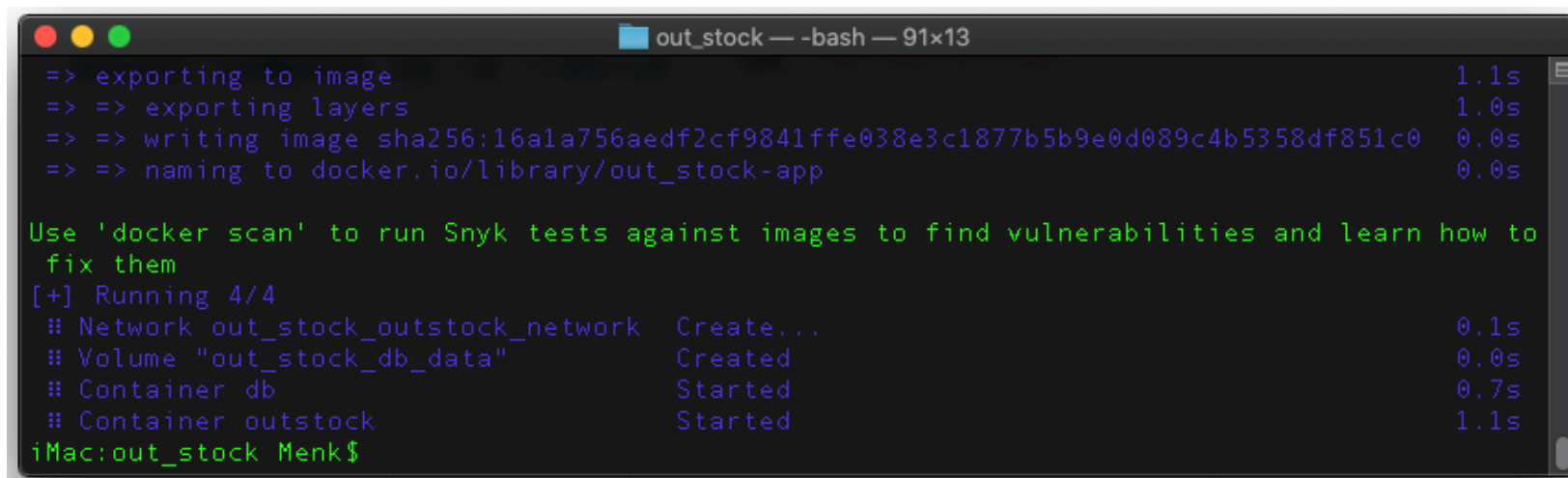
Agora, como descrito no passo 3, iremos subir o Docker Compose

- 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*

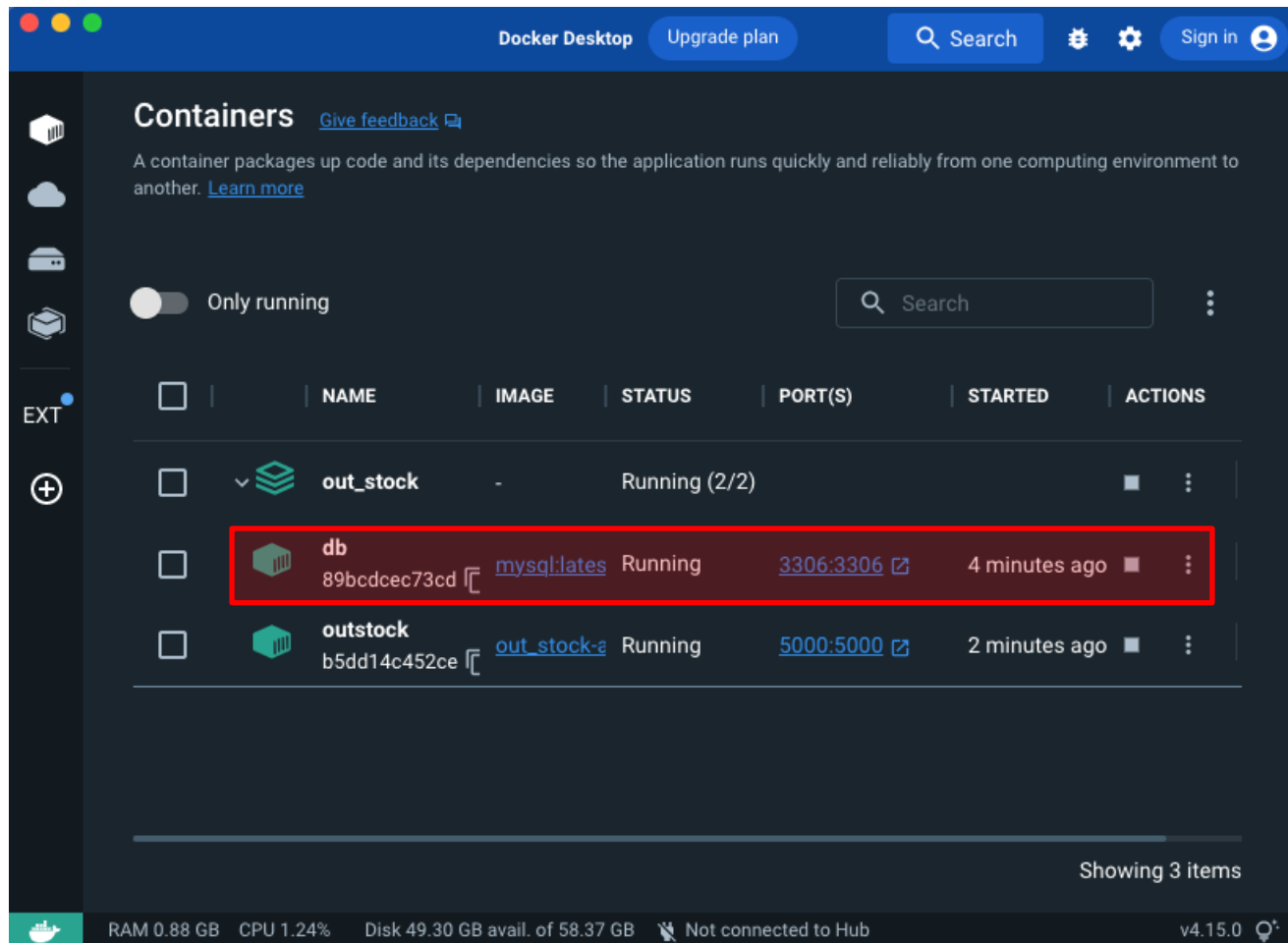
A opção **--build** refaz as imagens



```
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**



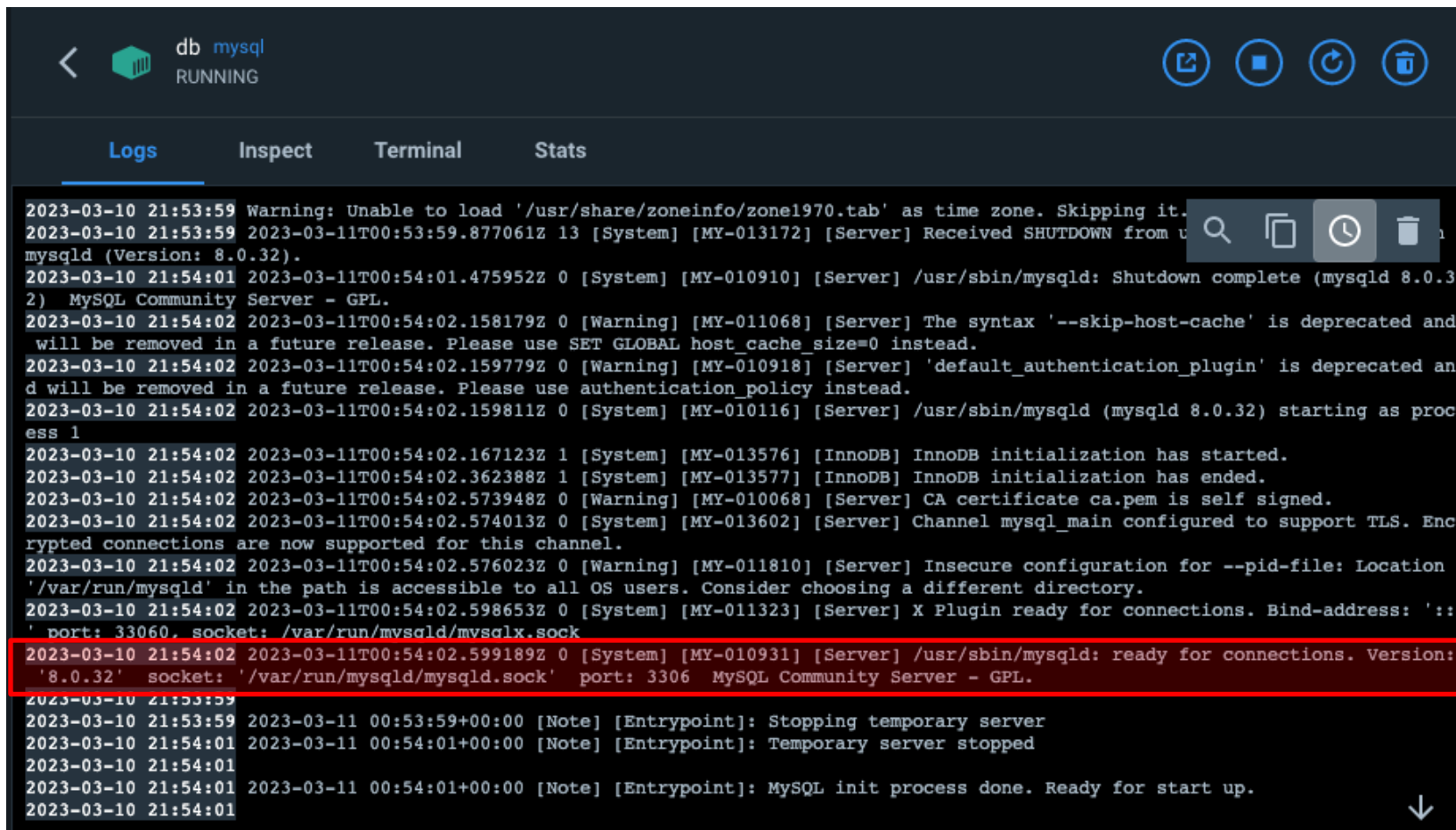
The screenshot shows the Docker Desktop interface. The 'Containers' tab is active, displaying a list of containers. A red box highlights the 'db' container, which is running. The interface includes a sidebar with navigation icons, a top bar with 'Docker Desktop' and 'Upgrade plan' buttons, and a bottom status bar showing system resources.

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	out_stock	-	Running (2/2)			
<input type="checkbox"/>	db 89bcdcec73cd	mysql:lates	Running	3306:3306	4 minutes ago	
<input type="checkbox"/>	outstock b5dd14c452ce	out_stock-a	Running	5000:5000	2 minutes ago	

Showing 3 items

RAM 0.88 GB CPU 1.24% Disk 49.30 GB avail. of 58.37 GB Not connected to Hub v4.15.0

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
2023-03-10 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
2023-03-10 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"/>	<div></div>	out_stock -	Running (1/2)					
<input type="checkbox"/>	<div></div>	db 89bcdcec73cd 	mysql:latest	Running	3306:3306 	10 minutes ago 		
<input type="checkbox"/>	<div></div>	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 square, a refresh arrow, and a trash can. 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 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." The prompt "mysql>" is at the bottom. On the right side of the terminal window, there are three icons: a magnifying glass, a document, and a trash can.

```
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

```
out_stack — -bash — 118x10
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

```
out_stack — -bash — 113x14
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: 738-868-848
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
```

```
out_stack — -bash — 113x14
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-010911] [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
: '' 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
```

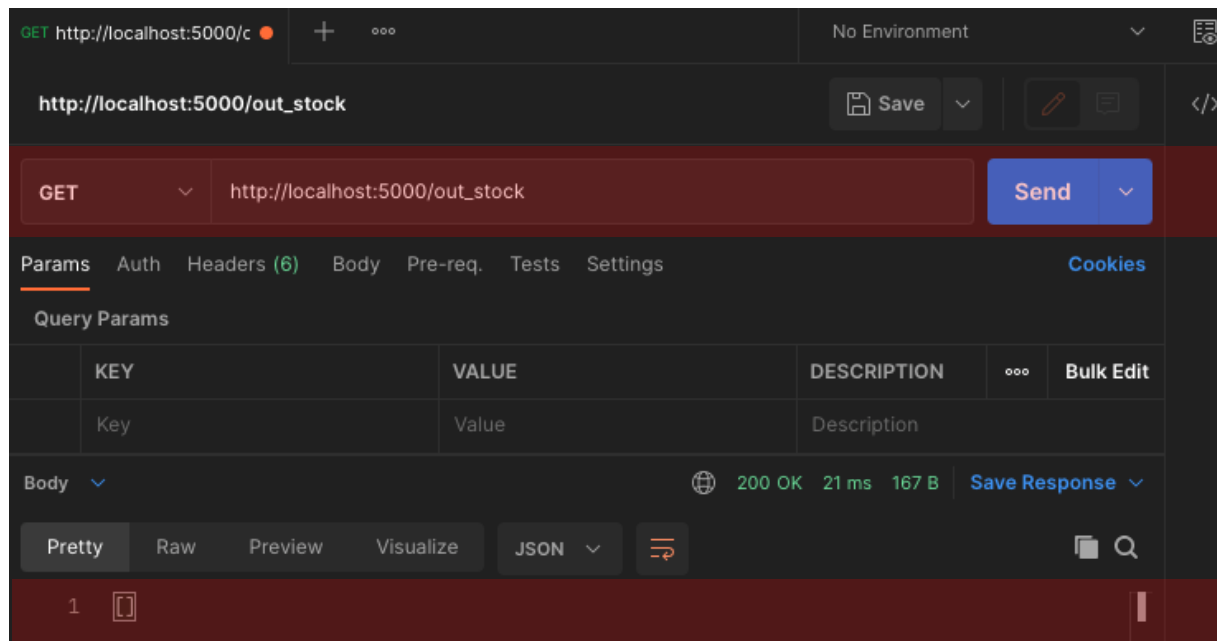
Delete:

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

Pelo Windows iremos realizar os Testes via Postman

Para buscar os registros da tabela

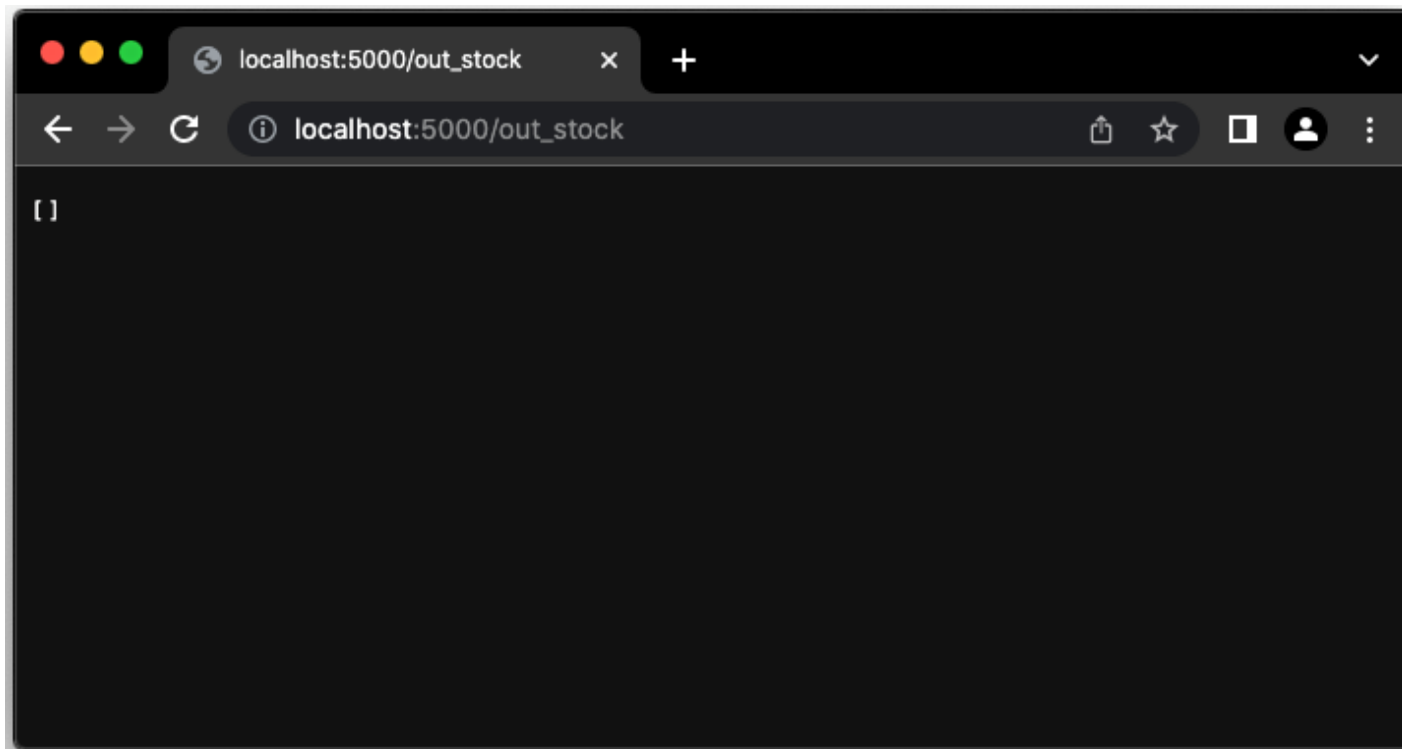
1. Abra o Postman e crie uma nova requisição
2. Selecione o método **HTTP GET** e informe a URL: **http://localhost:5000/out_stock**
3. Clique em "Send" para enviar a requisição
4. O resultado será exibido na aba "Body" da resposta, contendo a representação JSON dos registros da tabela "out_stock"



Para buscar os registros da tabela

O Browser da Internet também serve para realizar essa operação

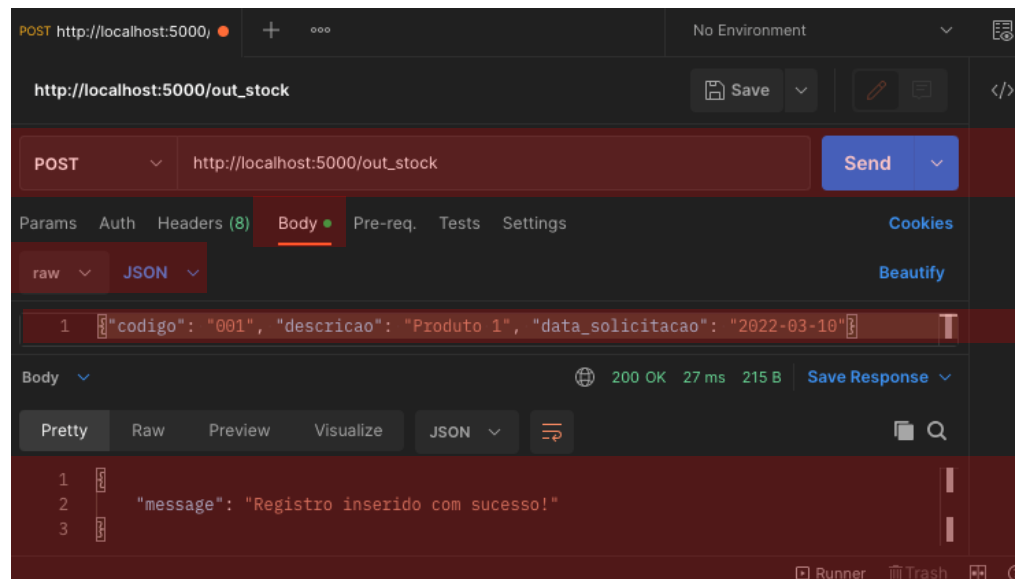
http://localhost:5000/out_stock



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

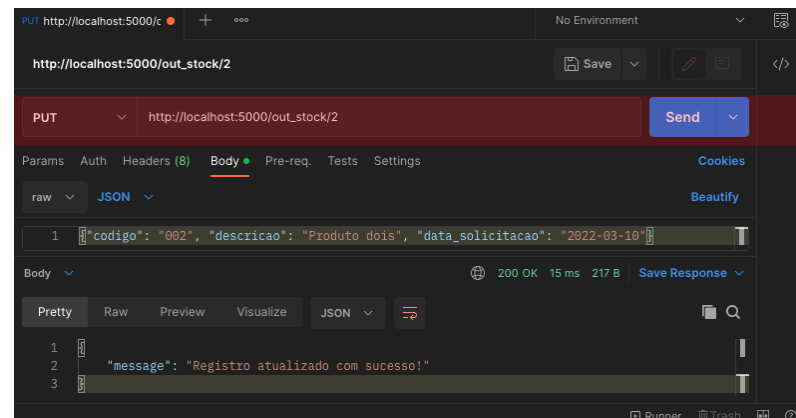
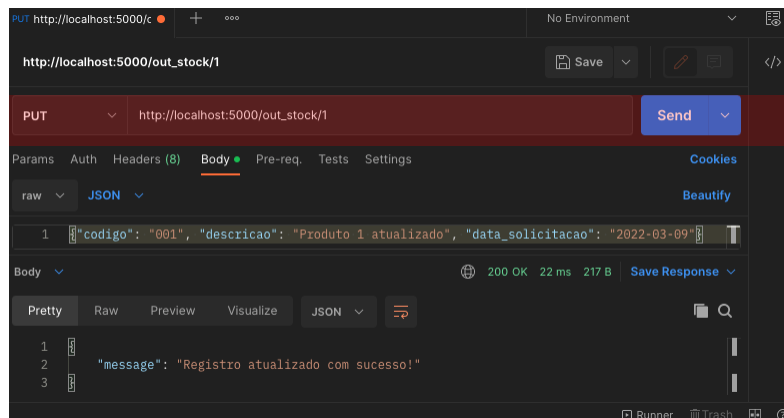
1. Selecione o método **HTTP PUT** e informe a URL com o **id do registro** a ser atualizado (**PK**): **http://localhost:5000/out_stock/1** (no exemplo, o id é "1")
2. Selecione a aba "**Body**", escolha a opção "**raw**" e defina o formato para "**JSON**"
3. No campo de edição, informe os dados atualizados 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

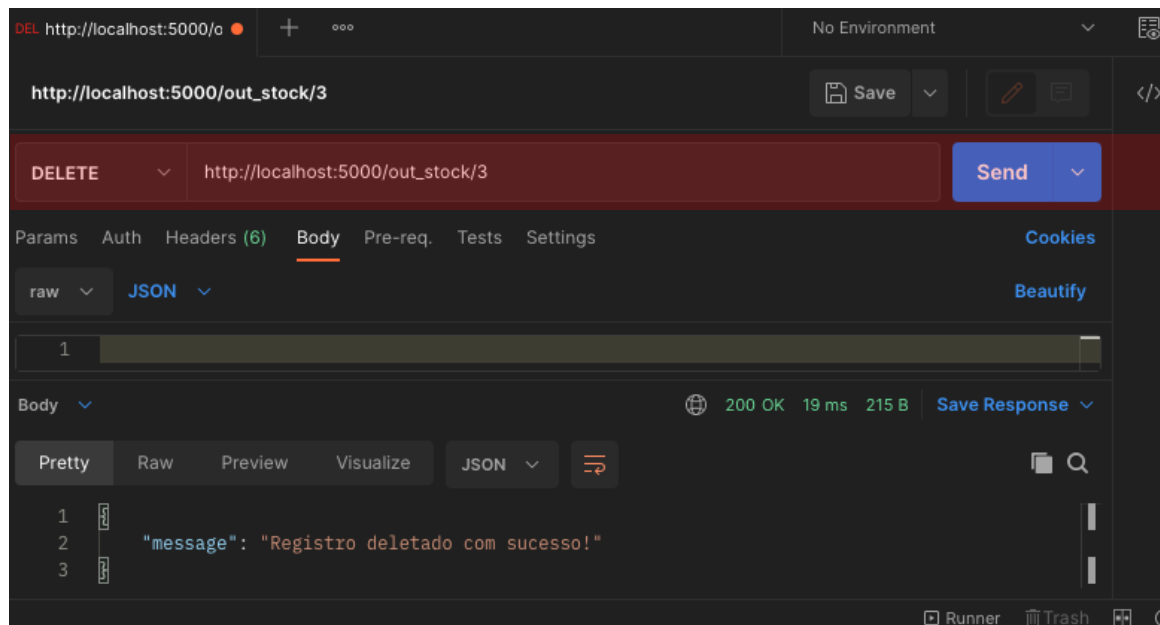
Linha: {"codigo": "002", "descricao": "Produto dois", "data_solicitacao": "2022-03-10"}



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)

```
expose:  
  - "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
```



`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).