

FIAP GRADUAÇÃO

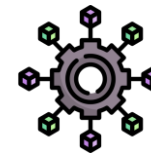
# TECNOLOGIA EM DESENVOLVIMENTO DE SISTEMAS

DevOps Tools & Cloud Computing

**Docker Compose Microserviços - Aprofundamento**

PROF. João Menk	<a href="mailto:profjoao.menk@fiap.com.br">profjoao.menk@fiap.com.br</a>
PROF. Sálvio Padlipskas	<a href="mailto:salvio@fiap.com.br">salvio@fiap.com.br</a>
PROF. Antonio Figueiredo	<a href="mailto:profantonio.figueiredo@fiap.com.br">profantonio.figueiredo@fiap.com.br</a>
PROF. Marcus Leite	<a href="mailto:profmarcus.leite@fiap.com.br">profmarcus.leite@fiap.com.br</a>
PROF. Thiago Rocha	<a href="mailto:profthiago.rocha@fiap.com.br">profthiago.rocha@fiap.com.br</a>
PROF. Thiago Moraes	<a href="mailto:proftiago.moraes@fiap.com.br">proftiago.moraes@fiap.com.br</a>

# Microserviço



FIAP

Um micro serviço é uma abordagem de arquitetura de software que se concentra em dividir a aplicação em componentes menores, independentes e especializados, conhecidos como Serviços

Cada Serviço é projetado para executar uma tarefa específica dentro da aplicação

Benefícios:

- ✓ Capacidade de dimensionar micros serviços individuais
- ✓ Uma base de código enxuta e fácil de testar, permitindo ainda a utilização de diferentes tecnologias (linguagens e frameworks diferentes)
- ✓ Maior flexibilidade e aproveitamento dos recursos disponíveis

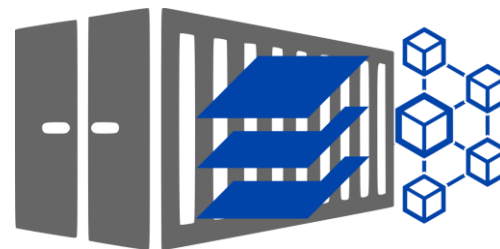


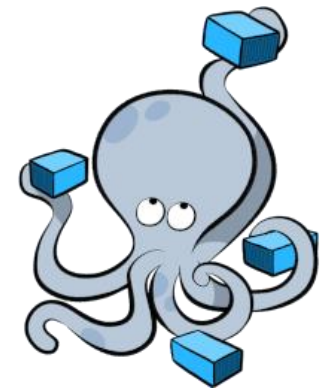
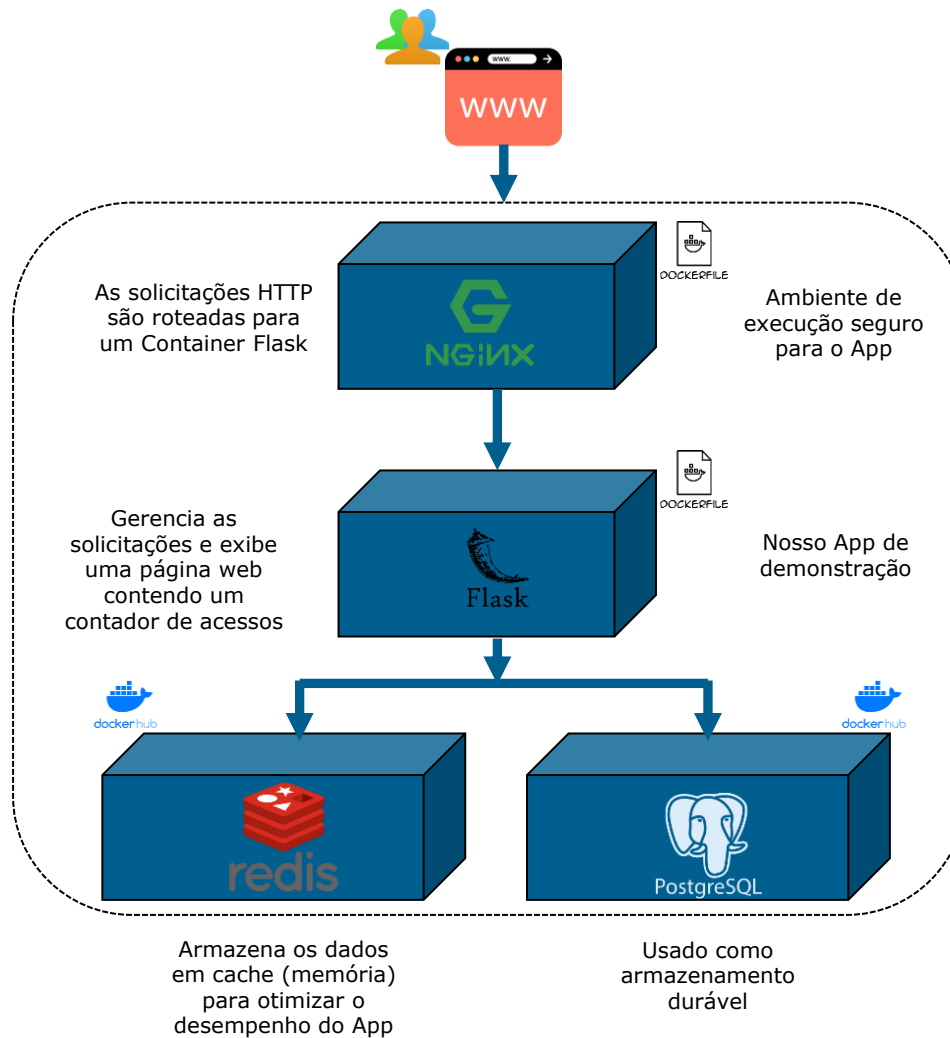
# Microservices

A Dimdim busca explorar as vantagens da arquitetura de micro serviços com o gerenciamento de Containers pelo Docker Compose

Nesta atividade, iremos criar um aplicativo simples de contador de acessos a uma página Web, demonstrando uma arquitetura de Micro serviços, utilizando quatro Containers distintos: Redis, Postgres, App e Nginx

O App em Python Flask será responsável por exibir uma página Web contendo um contador de acessos e recebendo solicitações de um servidor Nginx. Cada vez que a página for carregada, o contador será incrementado e os dados do contador são armazenados em dois bancos de dados diferentes: o Redis, e o Postgres





Responsável por monitorar e gerenciar todos os containers, garantindo que o ambiente esteja sempre funcionando corretamente

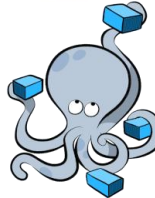
# Pré Reqs

Iremos utilizar as seguintes ferramentas

✓ Docker



✓ Docker Compose



✓ VSC



✓ Terminal

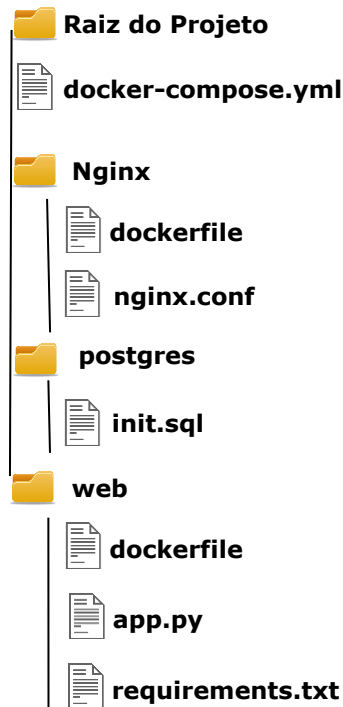


✓ Docker Desktop



## Preparando o ambiente

Para uma criação do zero, precisaríamos seguir vários passos, criar o fonte, os Dockerfiles, arquivos de configuração e o Docker Compose. Assim no final do projeto teremos um estrutura parecida com essa



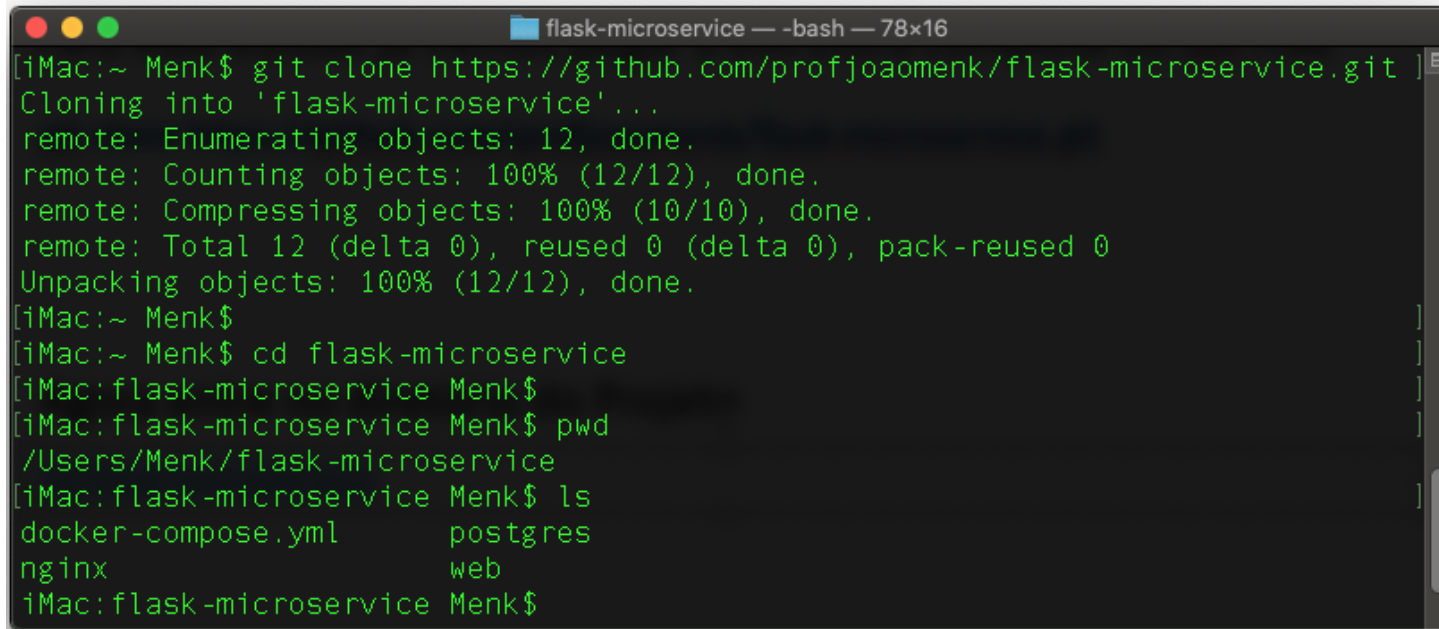
## Preparando o ambiente

Para esse estudo já iremos pegar a estrutura completa do Git Hub

```
git clone https://github.com/profjoaomenk/flask-  
microservice.git
```

Agora entre no diretório do Projeto

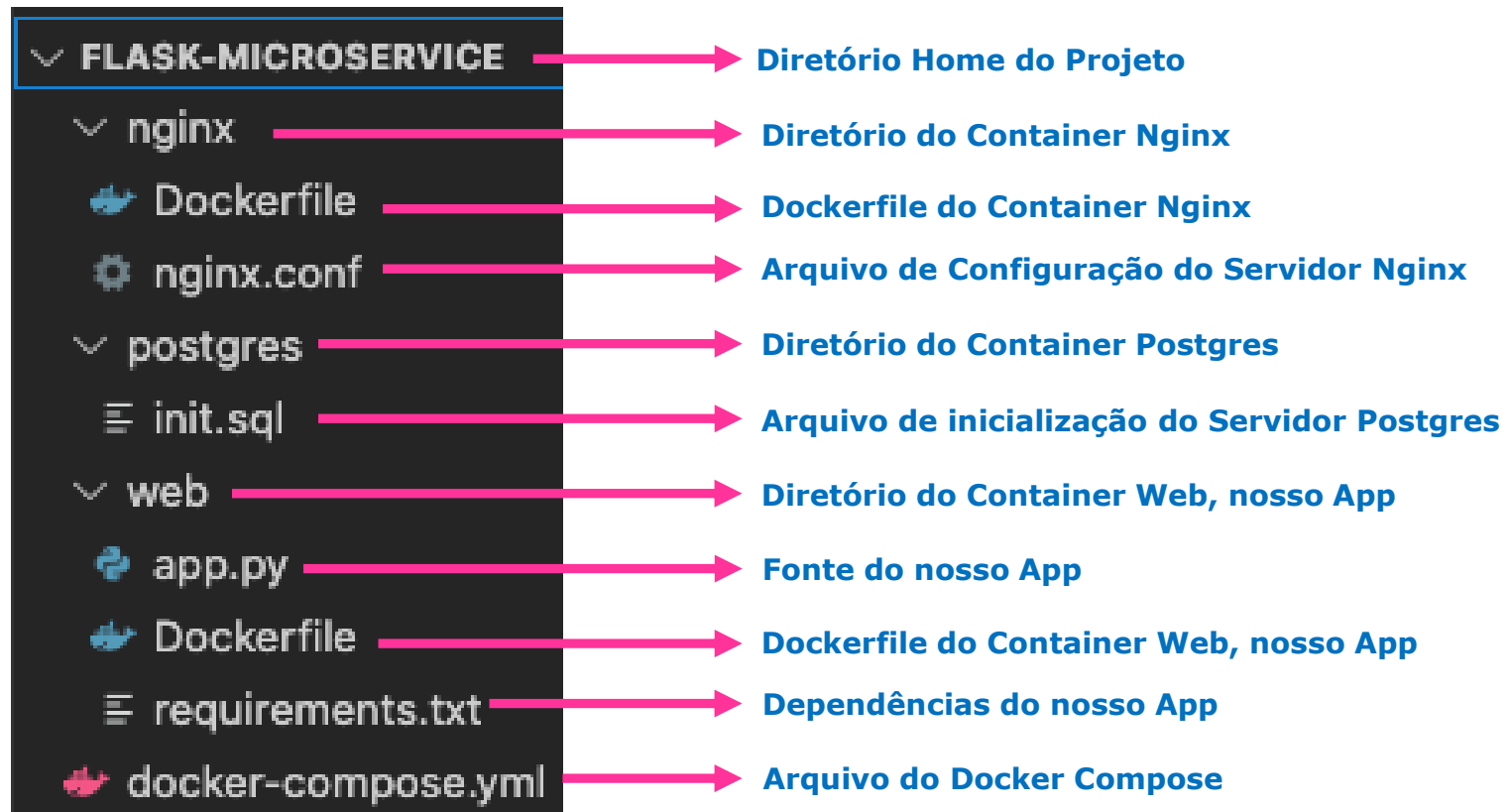
```
cd flask-microservice
```



```
flask-microservice — -bash — 78x16  
[iMac:~ Menk$ git clone https://github.com/profjoaomenk/flask-microservice.git ]  
Cloning into 'flask-microservice'...  
remote: Enumerating objects: 12, done.  
remote: Counting objects: 100% (12/12), done.  
remote: Compressing objects: 100% (10/10), done.  
remote: Total 12 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (12/12), done.  
[iMac:~ Menk$ ]  
[iMac:~ Menk$ cd flask-microservice ]  
[iMac:flask-microservice Menk$ ]  
[iMac:flask-microservice Menk$ pwd ]  
/Users/Menk/flask-microservice ]  
[iMac:flask-microservice Menk$ ls ]  
docker-compose.yml      postgres  
nginx                   web  
[iMac:flask-microservice Menk$ ]
```



Agora que temos o projeto vamos realizar uma análise e compreender o que foi feito. Inicie o VSC e abra a pasta do Projeto



Não criamos o diretório do **Container Redis** pois não temos um Dockerfile, não utilizamos persistência em disco nem variáveis de ambiente ou configurações adicionais. A imagem vem do Docker Hub sem nada a adicionar

Analise juntamente com o Professor os arquivos que compõem esse Projeto através dos diretórios, entendendo todo o Contexto da Arquitetura

Dockerfile > ...  
FROM python:3.9-slim

Dockerfile > ...  
FROM python:3.9-slim

Dockerfile > ...  
FROM nginx:alpine

nginx.conf

```
server {  
    listen 80;  
    proxy_pass_header Server;  
  
    location / {  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
  
        # Aponte para o nome do Serviço da Web  
        proxy_pass http://flaskapp:8000/;  
    }  
}
```

init.sql

```
CREATE TABLE visitors (  
    site_id integer,  
    site_name text,  
    visitor_count integer  
);  
  
ALTER TABLE visitors OWNER TO postgres;  
  
GRANT ALL PRIVILEGES ON DATABASE admdimdim TO postgres;  
  
INSERT INTO visitors (site_id, site_name, visitor_count) values(1, 'fiap.example.com.br', 0);
```

app.py

```
# Coenxão com o Redis - Procure pela palavra "cache" no fonte para verificar as operações  
cache = redis.StrictRedis(host='redis', port=6379)
```

```
# Realiza a conexão com o Postgres  
conn = psycopg2.connect("host='postgres' dbname='admdimdim' user='postgres' password='admdimdim'")
```

```
# Rota para a página principal  
@app.route('/')
```

```
# Rota para zerar o contador de visitas  
@app.route('/resetcounter')
```

Dockerfile

```
USER flask  
ENTRYPOINT ["/usr/local/bin/gunicorn", "--bind", ":8000", "app:app", "--reload", "--workers", "16"]
```

docker-compose.yml

```
volumes:  
  - ./web:/home/flask/app/web
```

```
# Realiza o Link com os Containers redis e postgres, abilitando a comunicação entre eles  
links:  
  - redis  
  - postgres
```

```
# Define limites para o Container  
resources:  
  limits:  
    cpus: '1' # Limite máximo de recursos  
    memory: 2048M  
  reservations:  
    cpus: '0.50' # Definida como a quantidade mínima para a execução  
    memory: 1048M
```

redis:

```
image: redis:alpine
```

postgres:

```
restart: always  
image: postgres:alpine
```

Agora vamos realizar a execução do Serviço com o comando abaixo

**docker-compose up -d --build**

```
flask-microservice — -bash — 81x20
=> [flask-microservice-nginx 2/2] COPY nginx.conf /etc/nginx/nginx.conf 2.2s
=> [flask-microservice-flaskapp 9/10] RUN pip install --no-cache-dir -r 7.3s
=> [flask-microservice-flaskapp] exporting to image 0.4s
=> => exporting layers 0.4s
=> => writing image sha256:03472f1e4113109f46375093ee84ef8186ddd0bd77904a 0.0s
=> => naming to docker.io/library/flask-microservice-nginx 0.0s
=> => writing image sha256:d0d6f9c656d60fbcc5e2853f26552962b4bbbc85e1fcf4 0.0s
=> => naming to docker.io/library/flask-microservice-flaskapp 0.0s
=> [flask-microservice-flaskapp 10/10] RUN chown -R flask:flaskgroup /hom 0.5s































Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
[+] Running 6/6
  :: Network flask-microservice_default Created 0.1s
  :: Container flask-microservice-redis-2 Started 1.3s
  :: Container flask-microservice-postgres-1 Started 1.4s
  :: Container flask-microservice-redis-1 Started 1.2s
  :: Container flask-microservice-flaskapp-1 Started 2.6s
  :: Container flask-microservice-nginx-1 Started 3.8s
iMac:flask-microservice Menk$
```

## Verifique no Docker Desktop o Serviço

### Containers [Give feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

☐ Only show running containers

<input type="checkbox"/>		NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	✓ 	flask-microservice	-	Running (5/5)			  
<input type="checkbox"/>		redis-2 828cdd770668 	<a href="#">redis:alpine</a>	Running		4 minutes ago	  
<input type="checkbox"/>		redis-1 95a5d86516bc 	<a href="#">redis:alpine</a>	Running		4 minutes ago	  
<input type="checkbox"/>		postgres-1 6c2c2722e9b4 	<a href="#">postgres:alpine</a>	Running		3 minutes ago	  
<input type="checkbox"/>		flaskapp-1 dba6e1819806 	<a href="#">flask-microserv</a>	Running		4 minutes ago	  
<input type="checkbox"/>		nginx-1 a8ce623b76a1 	<a href="#">flask-microserv</a>	Running	<a href="#">80:80</a> 	4 minutes ago	  

# Docker Compose Microserviços - Exercício

Verifique agora pelo Terminal com os seguintes comandos

## docker-compose ps

```
iMac:flask-microservice Menk$ docker-compose ps
```

NAME	COMMAND	SERVICE	STATUS	PORTS
flask-microservice-flaskapp-1	"/usr/local/bin/guni..."	flaskapp	running	8000/tcp
flask-microservice-nginx-1	"/docker-entrypoint..."	nginx	running	0.0.0.0:80->80/tcp
flask-microservice-postgres-1	"docker-entrypoint.s..."	postgres	running	5432/tcp
flask-microservice-redis-1	"docker-entrypoint.s..."	redis	running	6379/tcp
flask-microservice-redis-2	"docker-entrypoint.s..."	redis	running	6379/tcp

```
iMac:flask-microservice Menk$
```

## docker-compose logs

```
iMac:flask-microservice Menk$ docker-compose logs
```

```
omatic recovery in progress
flask-microservice-postgres-1 | 2023-03-18 19:36:21.021 UTC [25] LOG:  redo starts at 0/1501598
flask-microservice-postgres-1 | 2023-03-18 19:36:21.432 UTC [25] LOG:  invalid record length at 0/1923F10: wanted 24,
got 0
flask-microservice-postgres-1 | 2023-03-18 19:36:21.432 UTC [25] LOG:  redo done at 0/1923EE8 system usage: CPU: user:
0.01 s, system: 0.04 s, elapsed: 0.41 s
flask-microservice-postgres-1 | 2023-03-18 19:36:21.436 UTC [23] LOG:  checkpoint starting: end-of-recovery immediate
wait
flask-microservice-postgres-1 | 2023-03-18 19:36:22.113 UTC [23] LOG:  checkpoint complete: wrote 920 buffers (5.6%);
0 WAL file(s) added, 0 removed, 0 recycled; write=0.668 s, sync=0.002 s, total=0.679 s; sync files=252, longest=0.001 s
, average=0.001 s; distance=4234 kB, estimate=4234 kB
flask-microservice-postgres-1 | 2023-03-18 19:36:22.125 UTC [1] LOG:  database system is ready to accept connections
iMac:flask-microservice Menk$
```

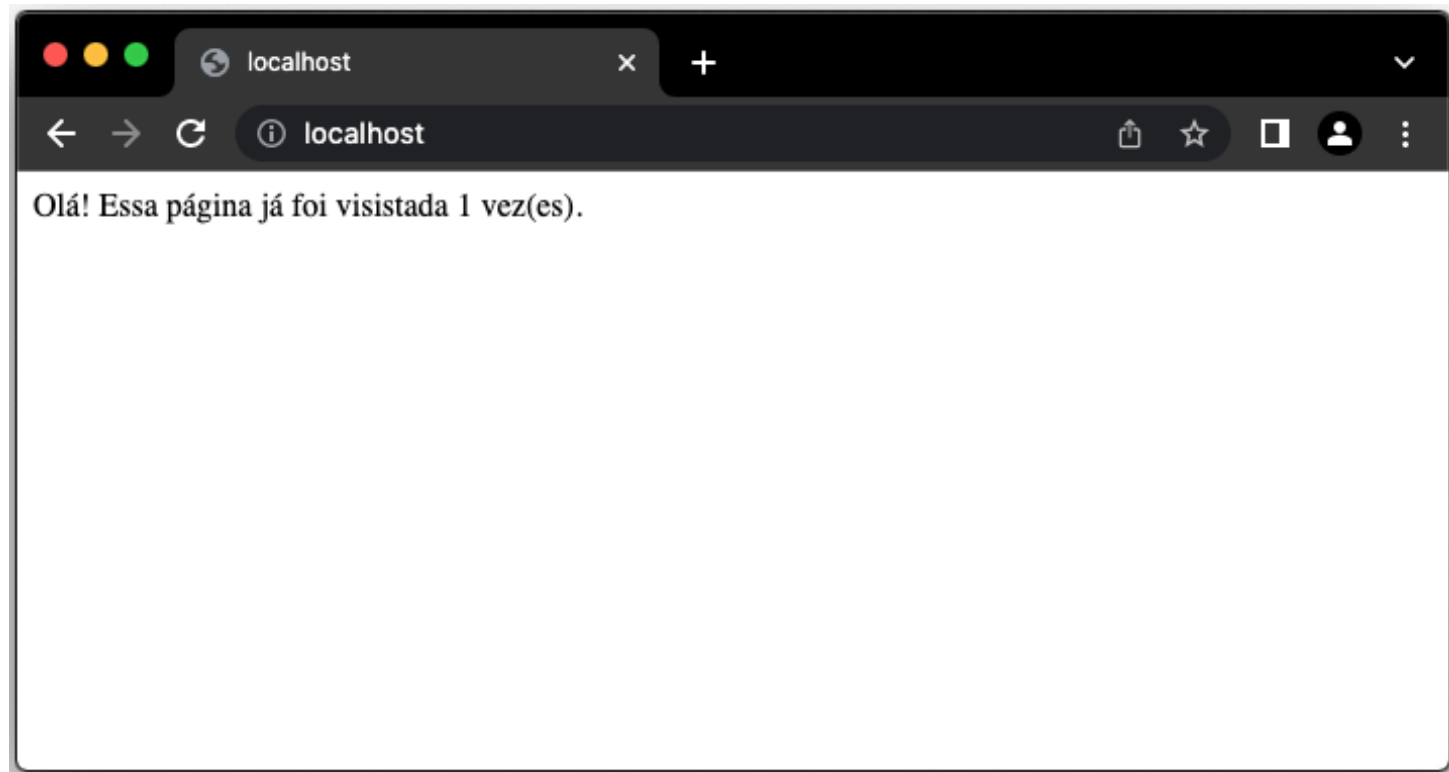
## docker network ls

```
iMac:flask-microservice Menk$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
f2c1baa5c250	bridge	bridge	local
86af5c48f5ee	flask-microservice_web_network	bridge	local
b5f4a5aeb449	host	host	local
7a8860c2561f	none	null	local

```
iMac:flask-microservice Menk$
```

Vamos realizar um Teste do Serviço agora. Abre seu Web Browser e acesse a página: <http://localhost>



**Realize alguns Reloads na Página** e verifique se o contador é atualizado

## Verifique os Dados nos Bancos

Entrar no Terminal do **Postgres**

**docker exec -it flask-microservice-postgres-1 psql -U postgres**

Comando	Descrição
\l	Visualizar os Bancos
\conninfo	Informações sobre a conexão atual
\c	Mudança de Banco
\d	Listar as Tabelas do Banco atual

Altere o Banco de Dados com o comando abaixo

**\c admdimdim**

```
[postgres=# \c admdimdim  
You are now connected to database "admdimdim" as user "postgres".
```

Realize um **SELECT** na tabela para verificar o valor do contador

## Verifique os Dados nos Bancos

Entrar no Terminal do **Redis**

**docker exec -it flask-microservice-redis-1 redis-cli**

\* Pode estar na réplica 2

Comando	Descrição
INFO keyspace	Informações sobre o Banco e as Chaves
select <b>n</b>	Altera para o Banco <b>n</b>
KEYS *	Mostra as Chaves que o banco possui
get <b>&lt;key&gt;</b>	Mostra os valores da Chave <b>&lt;key&gt;</b>

Altere o Banco de Dados com o comando abaixo

**select 0**

Mostre as Chaves desse Banco com o comando abaixo

**KEYS \***

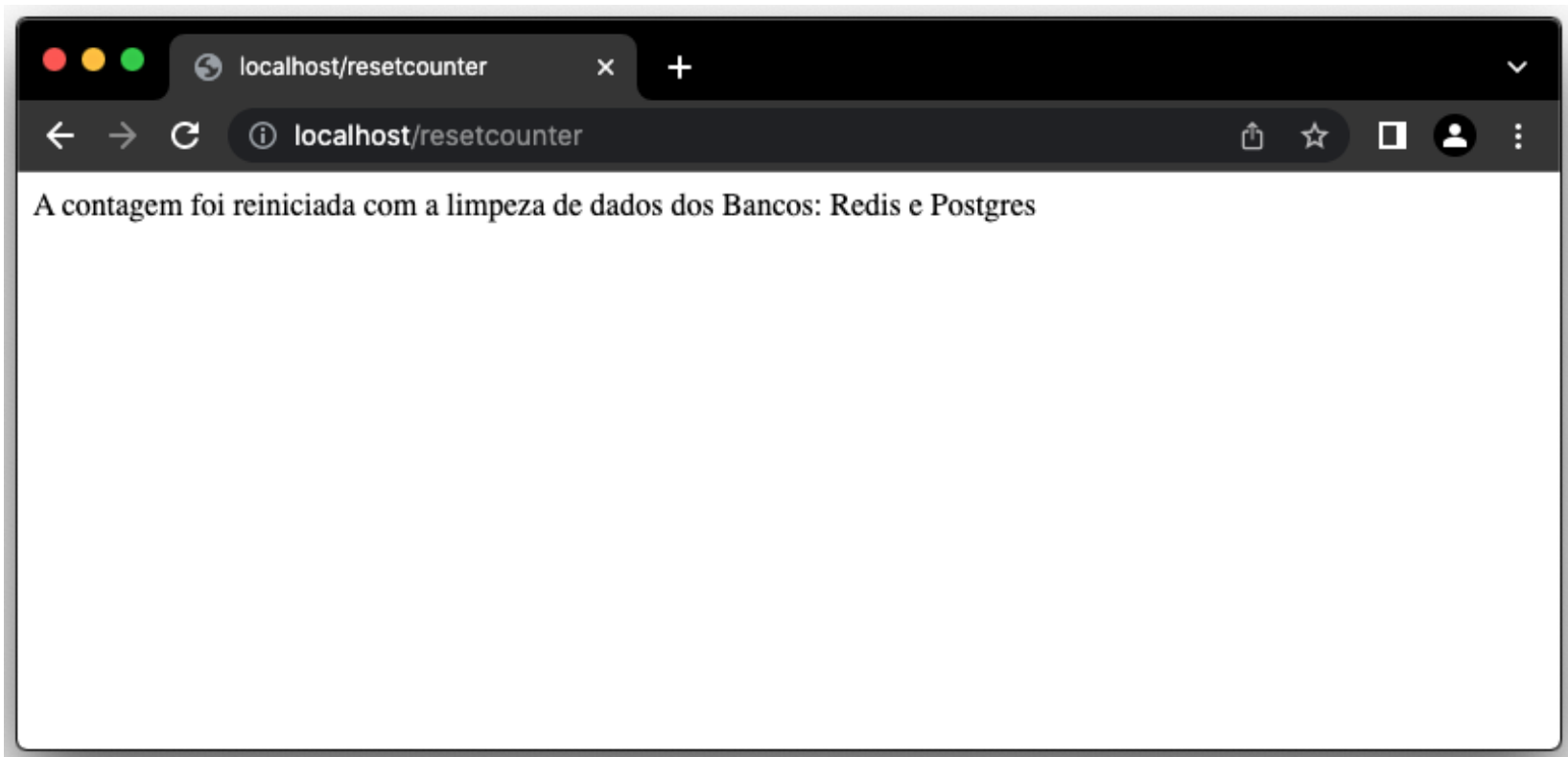
Recupere os valores da Chave com o comando abaixo

**get visitor\_count**



## Docker Compose Microserviços - Exercício

Realize a limpeza do Contador acessando a página:  
<http://localhost/resetcounter>



Verifique novamente os Dados nos Bancos e verifique se o contador foi finalizado

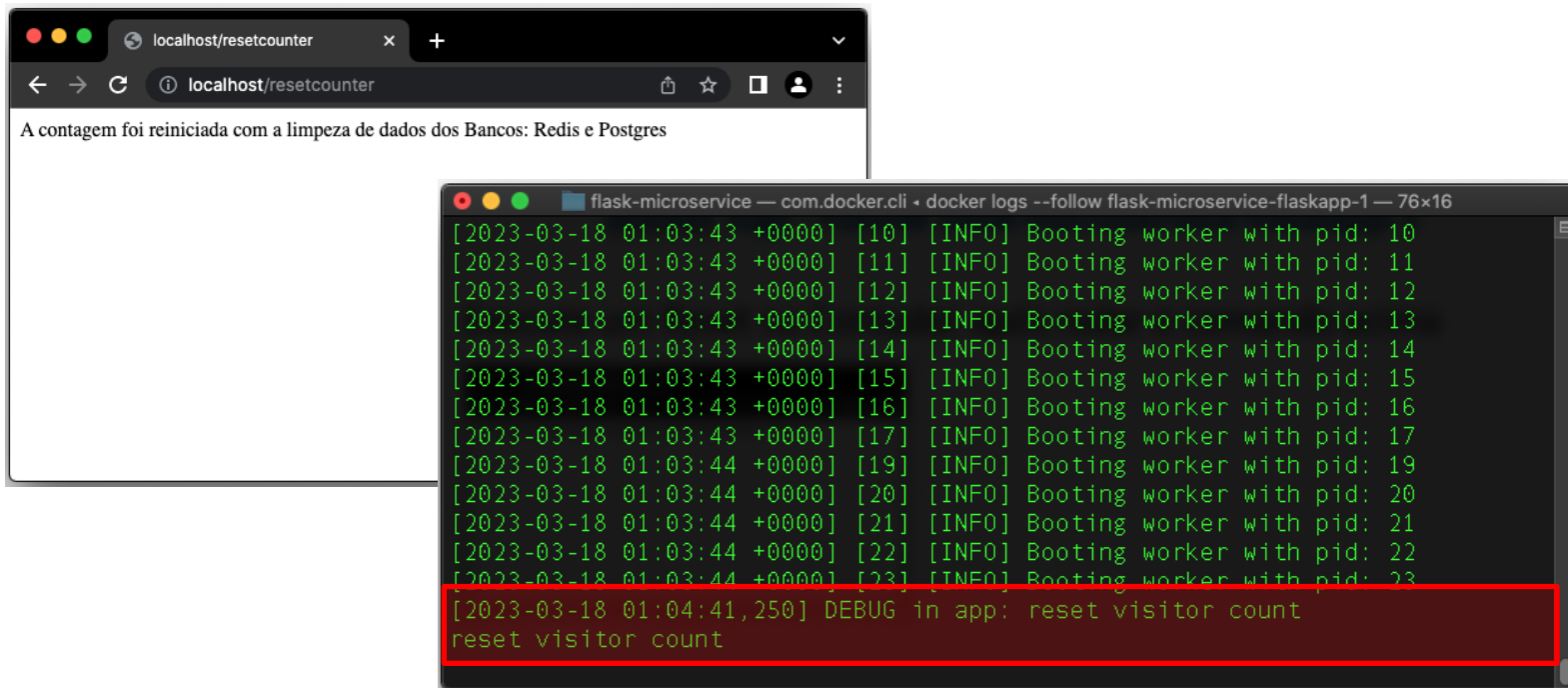
```
admdimdim=# select * from visitors;
 site_id |      site_name      | visitor_count
-----+-----+-----
        1 | fiap.example.com.br |            0
(1 row)
```

```
127.0.0.1:6379> select 0
OK
127.0.0.1:6379> KEYS *
(empty array)
127.0.0.1:6379> get visitor_count
(nil)
127.0.0.1:6379>
```

Você pode acompanhar os logs dos Containers via Terminal com os comandos abaixo (primeiro exemplo):

**docker logs --follow flask-microservice-flaskapp-1**

Execute um Reset no Contador após rodar o comando acima



```
localhost/resetcounter x +
localhost/resetcounter
A contagem foi reiniciada com a limpeza de dados dos Bancos: Redis e Postgres

flask-microservice — com.docker.cli • docker logs --follow flask-microservice-flaskapp-1 — 76x16
[2023-03-18 01:03:43 +0000] [10] [INFO] Booting worker with pid: 10
[2023-03-18 01:03:43 +0000] [11] [INFO] Booting worker with pid: 11
[2023-03-18 01:03:43 +0000] [12] [INFO] Booting worker with pid: 12
[2023-03-18 01:03:43 +0000] [13] [INFO] Booting worker with pid: 13
[2023-03-18 01:03:43 +0000] [14] [INFO] Booting worker with pid: 14
[2023-03-18 01:03:43 +0000] [15] [INFO] Booting worker with pid: 15
[2023-03-18 01:03:43 +0000] [16] [INFO] Booting worker with pid: 16
[2023-03-18 01:03:43 +0000] [17] [INFO] Booting worker with pid: 17
[2023-03-18 01:03:44 +0000] [19] [INFO] Booting worker with pid: 19
[2023-03-18 01:03:44 +0000] [20] [INFO] Booting worker with pid: 20
[2023-03-18 01:03:44 +0000] [21] [INFO] Booting worker with pid: 21
[2023-03-18 01:03:44 +0000] [22] [INFO] Booting worker with pid: 22
[2023-03-18 01:03:44 +0000] [23] [INFO] Booting worker with pid: 23
[2023-03-18 01:04:41,250] DEBUG in app: reset visitor count
reset visitor count
```

Você pode acompanhar os logs dos Containers via Terminal com os comandos abaixo (segundo exemplo):

**docker logs --follow flask-microservice-postgres-**

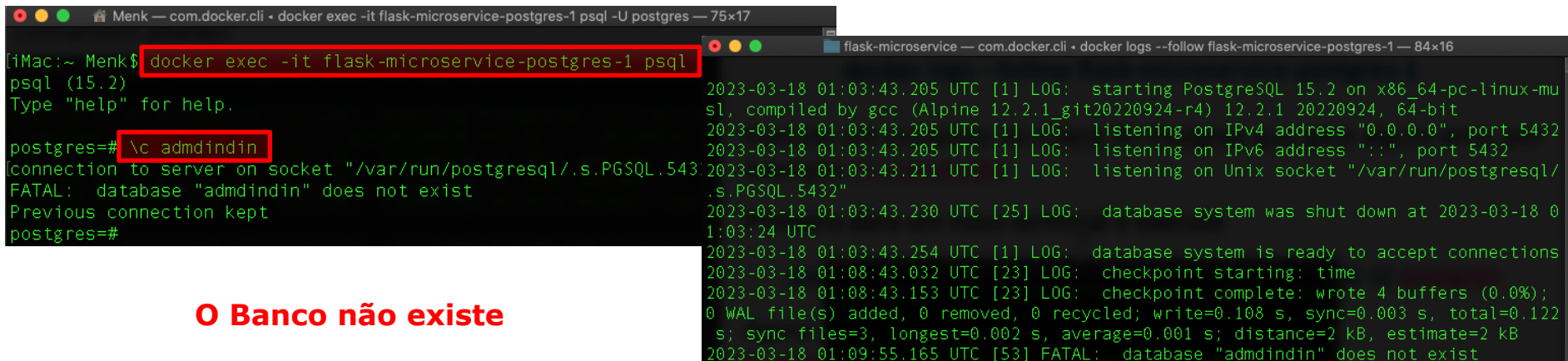
**1**

Agora, abra um novo terminal e execute

**docker exec -it flask-microservice-postgres-1 psql -U postgres**

Agora digite na linha de comando:

**/c admdindin**



```
Menk — com.docker.cli • docker exec -it flask-microservice-postgres-1 psql -U postgres — 75x17
[Mac:~ Menk$ docker exec -it flask-microservice-postgres-1 psql
psql (15.2)
Type "help" for help.

postgres=# \c admdindin
connection to server on socket "/var/run/postgresql/.s.PGSQL.5432":
FATAL:  database "admdindin" does not exist
Previous connection kept
postgres=#
```

```
flask-microservice — com.docker.cli • docker logs --follow flask-microservice-postgres-1 — 84x16
2023-03-18 01:03:43.205 UTC [1] LOG:  starting PostgreSQL 15.2 on x86_64-pc-linux-mu
sl, compiled by gcc (Alpine 12.2.1_git20220924-r4) 12.2.1 20220924, 64-bit
2023-03-18 01:03:43.205 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-03-18 01:03:43.205 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2023-03-18 01:03:43.211 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/
.s.PGSQL.5432"
2023-03-18 01:03:43.230 UTC [25] LOG:  database system was shut down at 2023-03-18 0
1:03:24 UTC
2023-03-18 01:03:43.254 UTC [1] LOG:  database system is ready to accept connections
2023-03-18 01:08:43.032 UTC [23] LOG:  checkpoint starting: time
2023-03-18 01:08:43.153 UTC [23] LOG:  checkpoint complete: wrote 4 buffers (0.0%);
0 WAL file(s) added, 0 removed, 0 recycled; write=0.108 s, sync=0.003 s, total=0.122
s; sync files=3, longest=0.002 s, average=0.001 s; distance=2 kB, estimate=2 kB
2023-03-18 01:09:55.165 UTC [53] FATAL:  database "admdindin" does not exist
```

**O Banco não existe**

## Limitando os recursos do Docker



Além de poder impor os limites de CPU e Memória no Docker Compose, podemos realizar esse procedimento também no comando **docker run**

Exemplos:

```
deploy:
  # Define limites para o Container
  resources:
    limits:           # limite máximo de recursos
      cpus: '2'
      memory: 4096M
    reservations:     # definida como a quantidade mínima para a execução
      cpus: '1'
      memory: 2048M
```

**docker run --name teste -d ubuntu sleep 1d**

```
flask-microservice — bash — 74x20
iMac:flask-microservice Menk$ docker run --name teste -d ubuntu sleep 1d
254811724bf170cb6a096cd6e5eebd81af6318fef985d73f56c1bf5d621a8243
iMac:flask-microservice Menk$
```

**docker inspect teste | findstr /i "mem"**

**docker inspect teste | findstr /i "cpu"**

**docker inspect teste | grep -i mem**

**docker inspect teste | grep -i cpu**

```
flask-microservice — root@dc545a806b26: / — bash — 66x9
iMac:flask-microservice Menk$ docker inspect teste | grep -i mem
  "Memory": 0,
  "CpusetMems": "",
  "KernelMemory": 0,
  "KernelMemoryTCP": 0,
  "MemoryReservation": 0,
  "MemorySwap": 0,
  "MemorySwappiness": null,
iMac:flask-microservice Menk$
```

```
flask-microservice — root@dc545a806b26: / — bash — 66x12
iMac:flask-microservice Menk$ docker inspect teste | grep -i cpu
  "CpuShares": 0,
  "NanoCpus": 0,
  "CpuPeriod": 0,
  "CpuQuota": 0,
  "CpuRealtimePeriod": 0,
  "CpuRealtimeRuntime": 0,
  "CpusetCpus": "",
  "CpusetMems": "",
  "CpuCount": 0,
  "CpuPercent": 0,
iMac:flask-microservice Menk$
```

## Limitando os recursos do Docker



Além de poder impor os limites de CPU e Memória no Docker Compose, podemos realizar esse procedimento também no comando **docker run**

Exemplos:

**docker run --name teste-ctrl -m 1G --cpus=1 -d ubuntu sleep 1d**

```
flask-microservice — -bash — 92x19
iMac:flask-microservice Menk$ docker run --name teste-ctrl -m 1G --cpus=1 -d ubuntu sleep 1d
5114b0f06f027082079e0cb68f7f1b46ca82c2e57d57454b21ae08fe3281564f
iMac:flask-microservice Menk$
```

**docker inspect teste-ctrl | findstr /i "mem"**

**docker inspect teste-ctrl | grep -i mem**

```
flask-microservice — root@dc545a806b26: / — -bash — 66x12
iMac:flask-microservice Menk$ docker inspect teste-ctrl | grep -i mem
"Memory": 1073741824,
```

**docker inspect teste-ctrl | findstr /i "cpu"**

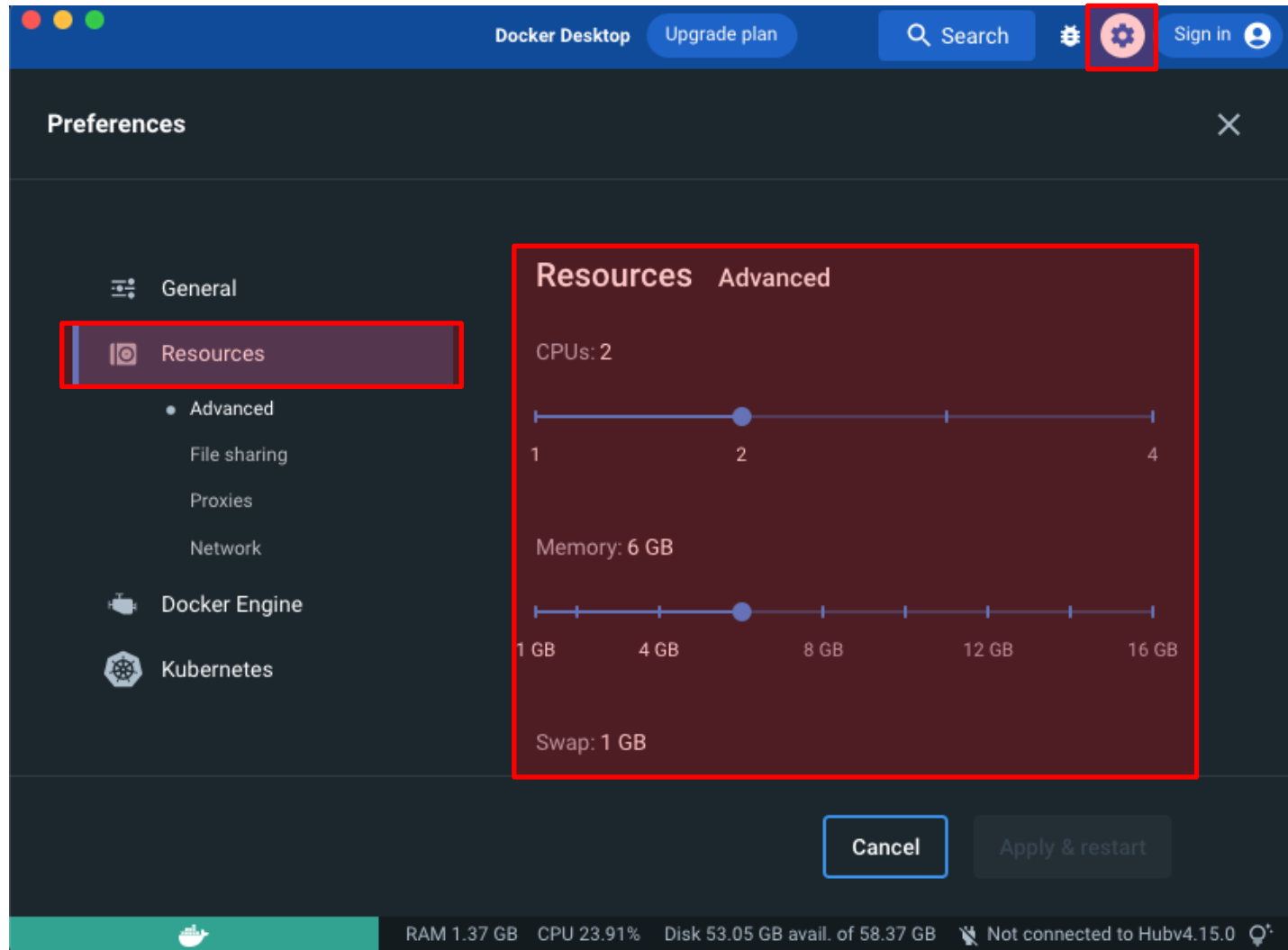
**docker inspect teste-ctrl | grep -i cpu**

```
flask-microservice — root@dc545a806b26: / — -bash — 66x13
iMac:flask-microservice Menk$ docker inspect teste-ctrl | grep -i cpu
"CpuShares": 0,
"NanoCpus": 1000000000,
```

# Limitando os recursos do Docker



## Configurações de Recursos no Docker Desktop



`docker-compose down`  
`docker container stop teste teste-`  
`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).