
# Conceitos

Documentação Docker:

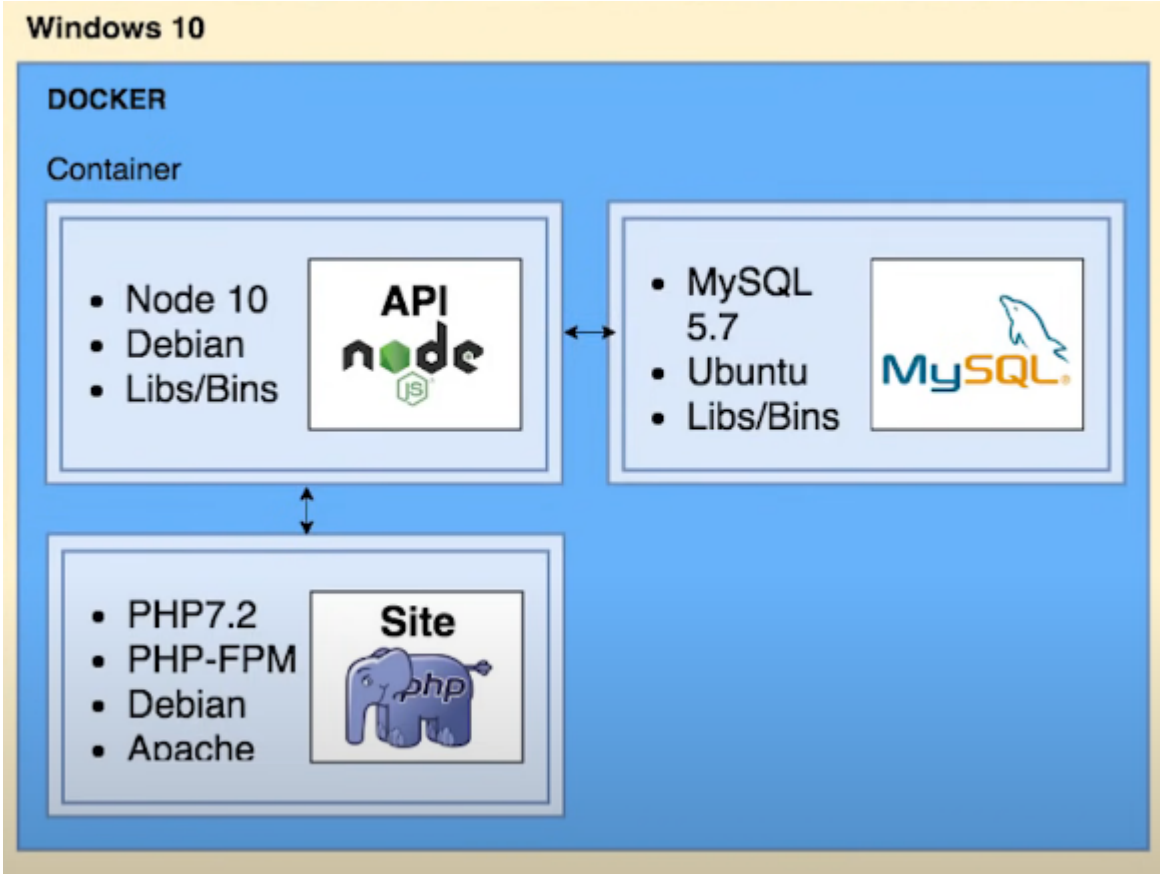
Reference documentation

This section includes the reference documentation for the Docker platform's various APIs, CLIs, and file formats.

https://docs.docker.com/reference/

## O que é Docker?

Junta toda aplicação  do repositório em uma imagem completa, incluindo as dependências para executar a aplicação. Essa imagem é em forma de containers e o processo se chama Conteinerização.



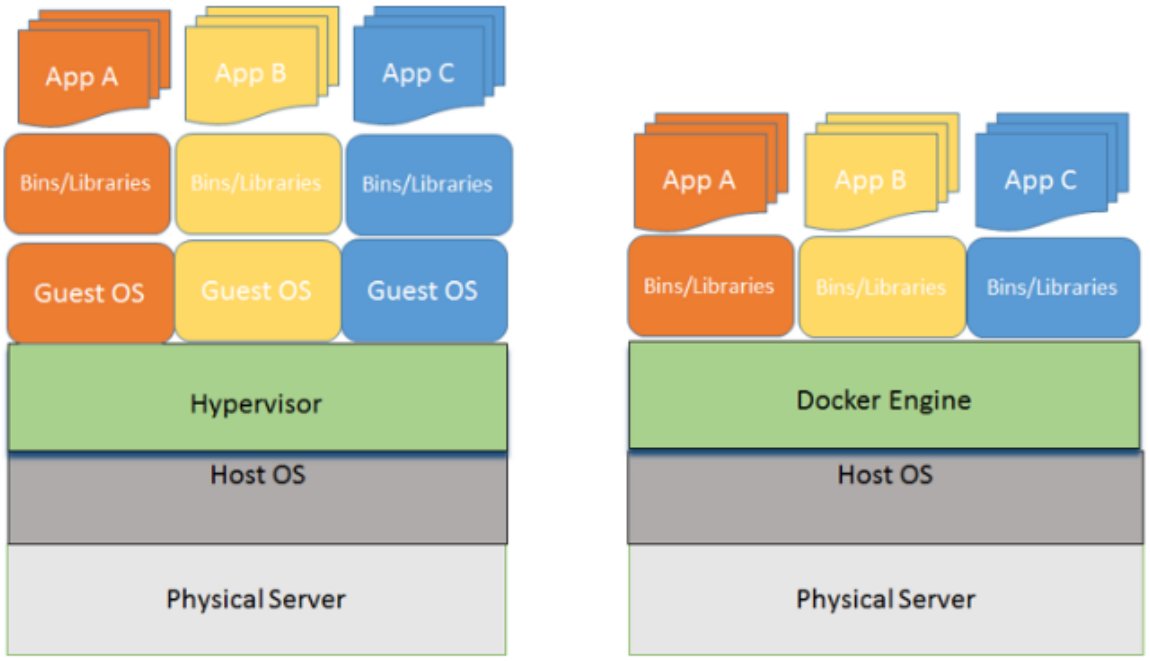

Cada aplicação está em um container diferente. Este docker é um template/modelo ou imagem.

# Como funciona?

O Docker utiliza o Kernel(Núcleo) do Linux e seus recursos. O objetivo é criar a independência para executar múltiplas aplicações e processos de forma separada;

# Qual a Diferença entre máquina virtual(VM) e docker?

Docker é um ambiente descartável, escalável e segue o mesmo padrão de configuração, pois não necessita de sistema operacional gráfico para funcionar. Conforme imagem abaixo, na esquerda contem um sistema de operação para cada aplicação, enquanto no Docker Engine todos compartilham a camada OS.




# Quais as vantagens da tecnologia?

Agilidade→ Os containers podem ser facilmente transportados entre diversos ambientes.

Maior disponibilidade do sistema→ Mais espaço livre pois há o compartilhamento do sistema operacional e  de outros componentes.

Reversibilidade dos containers→ Caso haja algum erro, basta retornar o container para sua versão prévia.

Abordagem em microsserviços→  Existe a possibilidade de interromper apenas uma parte do software em execução, fazendo-o continuar normalmente com exceção deste modulo. Aliado a isso existe a arquitetura SOA que compartilha recursos entre diversas aplicações.

Leve → Os contêineres compartilham o kernel do OS, eliminando a necessidade de uma instância completa do OS por aplicativo.

# Caso de usi para contêineres

Microsserviços → eles são pequenos e leves, o que os tonar uma boa combinação para arquitetura de microsserviço.

DevOps→ Usado no DevOps como forma de construir, enviar e executar software.

Híbrido, multicloud →

# Qual a diferença entre imagem e container?

Um paralelo com o conceito de OOP, a imagem é classe e o container o objeto. Todo container é iniciado a partir de uma imagem. Um container só pode ser iniciado a partir de uma única imagem, caso queira algo diferente, é necessário customizar a imagem.

**A imagem é a planta, o container é a casa. container é o resultado da imagem.**

**Nome da imagem**

> Composto por 2 partes, a primeira chamada “repositório”, a segundo é chamada de “tag”. No caso da imagem “Ubuntu:14:04”, ubuntu é o repositório e 14.04 a tag. Cada conjunto de repositório:tag é uma imagem diferente.

Imagens são imutáveis.

Container → um ambiente isolado que roda uma aplicação.

Virtual machine → uma abstração de uma máquina com hardware físico

Hypervisor→ Software que cria e gerencia maquinas virtuais. Existem os seguintes:

> VirtualBox
>
> VMware
>
> Hyper-v ( apenas windows)

Um container tem as camadas somente leitura e uma adicional para leitura e escrita.

Problemas da máquina virtual:

> Cada VM precisa de um sistema completo  do sistema operacional, é lento de iniciar e usa muito recurso do computador(memoria, Cpu).

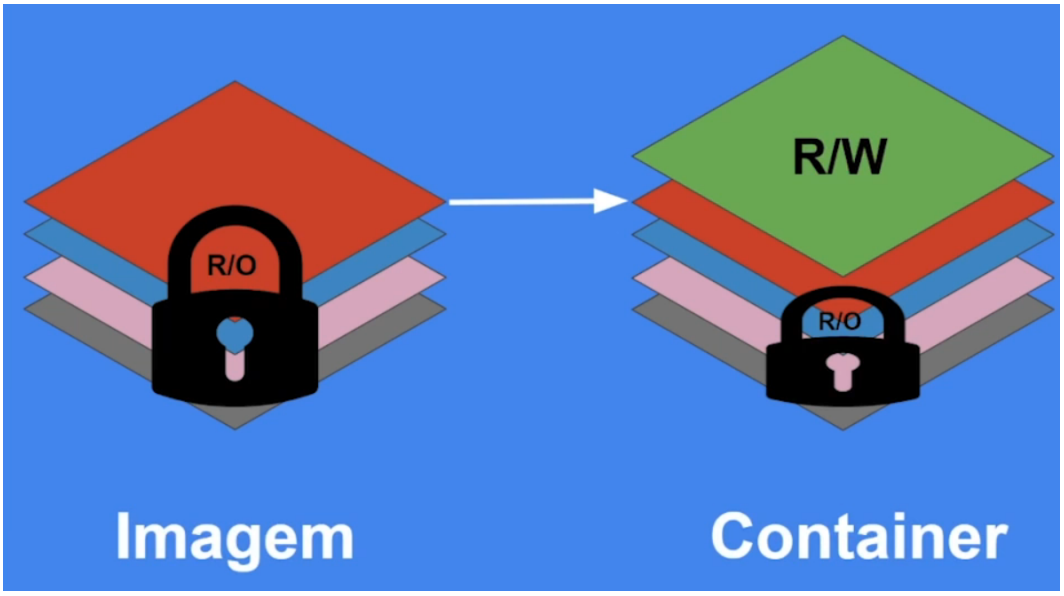
Containers

Permite rodar múltiplos aplicativos isolados, são mais leves, utiliza o sistema operacional do usuário, inicia rapidamente, precisa de menor recursos físicos do computador.

O containers são como caixas e contêm tudo o que é necessário para executar um aplicativo específico. Eles são padronizados e isolados e não há interferência de outros containers.

Responsáveis por isolamento lógico→namespace

Docker funciona com sistema de cliente e servidor(Docker engine), através de um REST API. Todos os containers compartilham o kernel do usuario




### o Docker tem uma estrutura de ordem  comando

docker run -it  -d ubuntu

docker → nome do programa

run → comando a ser executado

-it -d → Parâmetros

ubuntu → nome da imagem

## Ciclo de vida do container

O ciclo normal do container é subir o container, executar o comando e morrer.

```
docker run -d alpine
```

```
PS C:\Users\guilh> docker run alpine echo "hello world"
hello world
```

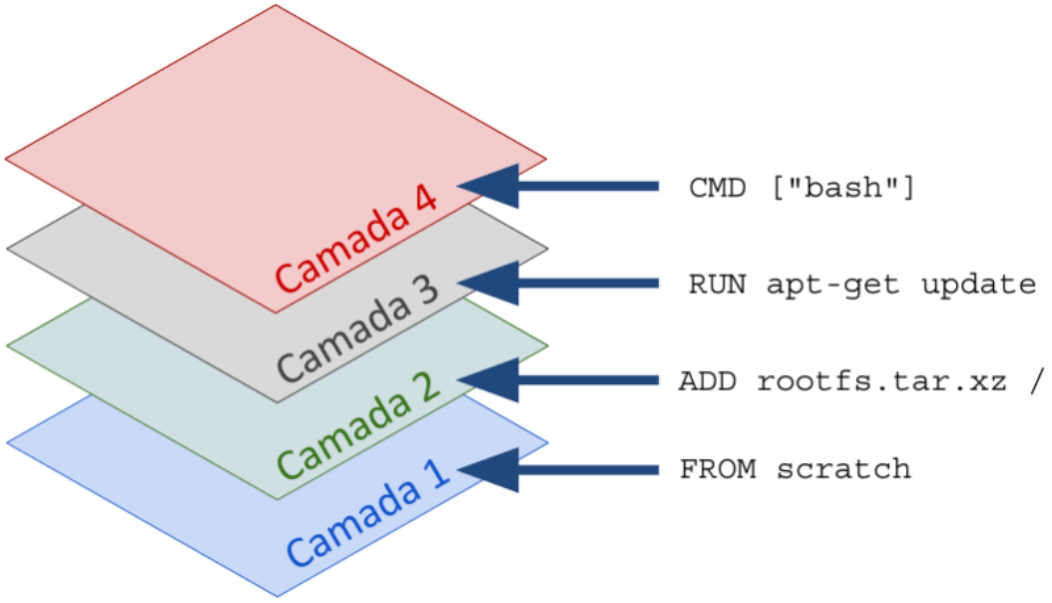
Se eu subo um container e mando executar um comando, e não terminar, ele fica vivo.

Containers zumbis, nasce e morre porque não há processo travando eles.

# Comunicação entre Docker

A solução que precisamos deve:

- Isolar o ambiente de uma aplicação de outras sobre o mesmo hardware (físico ou virtual);
- Alocar apenas o espaço em disco para a aplicação e suas dependências;
- Limitar memória e CPU para o processo em sí;
- Alocar apenas a porta TCP/UDP necessária para comunicação com o serviço na rede;

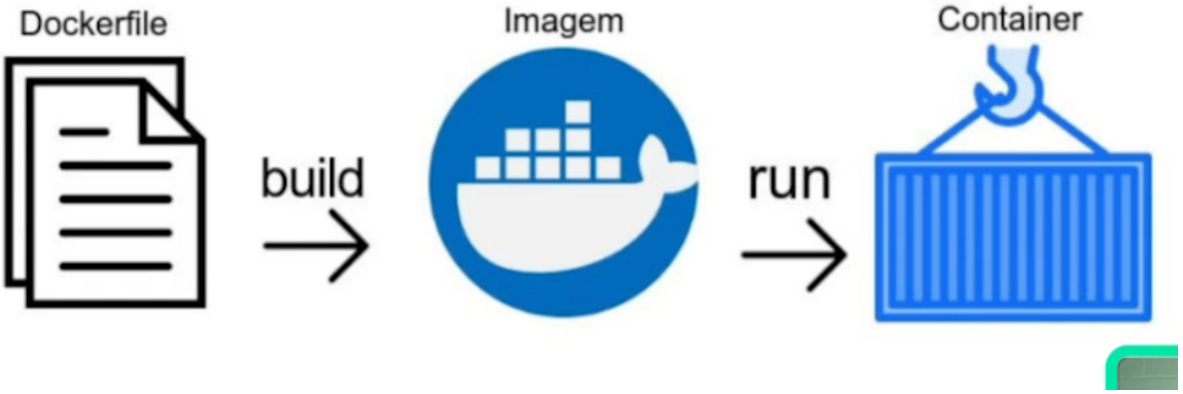



```
ROM scratch
ADD rootfs.tar.xz /
RUN apt-get -y update
CMD ["bash"]
```

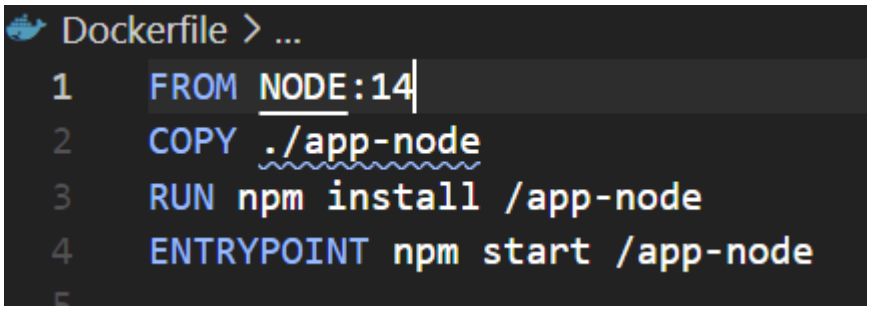
# Arquivo Dockerfile

O Dockerfile é o arquivo de texto que cria imagens do  Docker

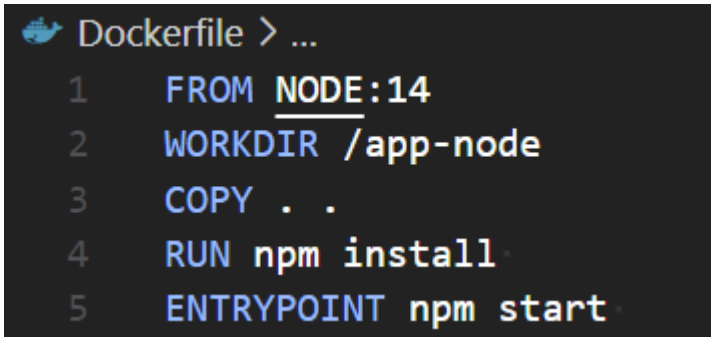
Workdir → define o diretório de trabalho de um contêiner do docker



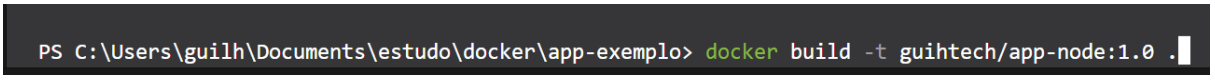

**http://www.patrick.eti.br/?p=artigos&a=docker**




FROM → Imagem escolhida do docker hub+ versão
COPY→ " . " copia todo o conteudo diretório atual para o diretório do container criado
RUN → instala as depêndencias
ENTRYPOINT → ponto de entrada do container starta a aplicação,




WORKDIR→ Diretório padrão para trabalho
COPY . . >abreviação do diretório padrão do trabalho




docker build -t [ etiqueta do container] [versão] [ referência do dir atual]


docker build -t [nome da etiqueta da imagem][ :versão] [ . ]

```
PS C:\Users\guilh\Documents\estudo\docker\app-exemplo> docker run -d -p 8080:3000 g:1.1 .
33dade8f2eb5ab788e42d0a58cb70c10dc949ef896f6cc45cfe1428d5a631871
PS C:\Users\guilh\Documents\estudo\docker\app-exemplo> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                    NAMES
33dade8f2eb5   g:1.1     "/bin/sh -c 'npm sta…"   6 seconds ago   Up 4 seconds   0.0.0.0:8080->3000/tcp   sad_carve
```

docker run -d -p 8080:3000 g:1.1 .
após rodar, usar o comando docker ps para verificar se subiu,

```
Dockerfile > ...
1   FROM node:14
2   WORKDIR /app-exemplo
3   EXPOSE 3000
4   COPY . .
5   RUN npm install
6   ENTRYPOINT npm start
```

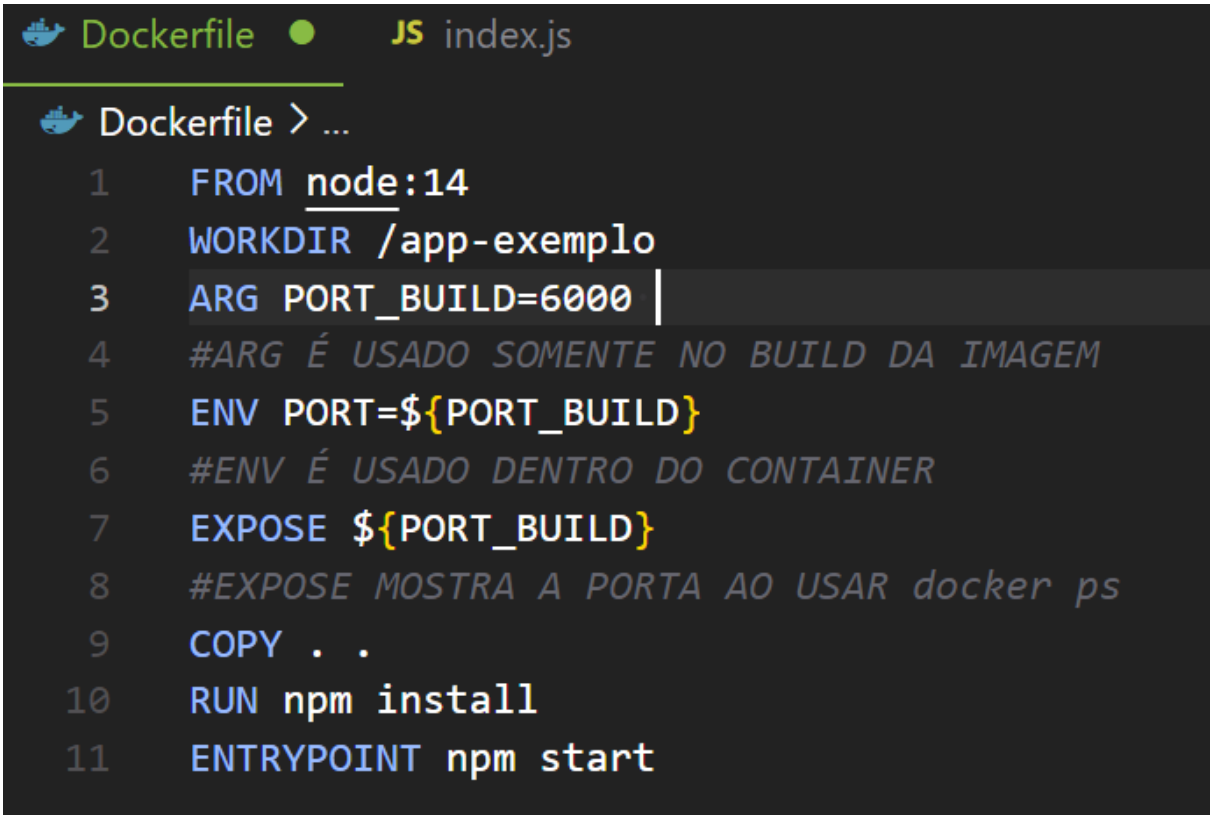
Ao escrever EXPOSE 3000 mostra no terminal a porta da aplicação

```
PS C:\Users\guilh> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS      NAMES
8214eadb8518   g:1.4     "/bin/sh -c 'npm sta…"   33 seconds ago   Up 33 seconds   3000/tcp   mystifying_blackburn
PS C:\Users\guilh>
```

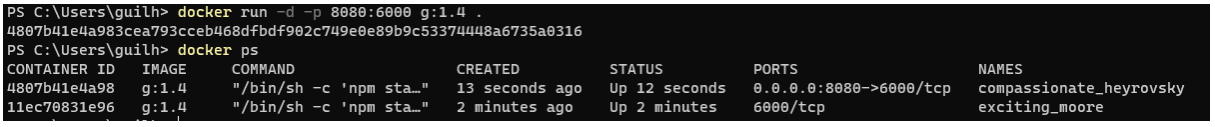
Para definir a imagem no dockerfile.

```
//app.listen("3000", ()=>{
app.listen(process.env.PORT, ()=>{
    console.log("Server is listening on port 3000")
})
```

Alterar no servidor o valor fixo da porta “ 3000” para uma variável de controle.




ENV carrega as variáveis que serão utilizadas no container




O container com id “ 11ec...” mostra que a porta configurada é a 6000. Já o container “ 4807...” mostra que está no localhost:8080

# Subindo no dockerhub

```
docker login
```

```
PS C:\Users\guilh> docker push g:1.4
The push refers to repository [docker.io/library/g]
af5ad93291df: Preparing
2e8bbe88c965: Preparing
7fff75ecdbe5: Preparing
0d5f5a015e5d: Preparing
3c777d951de2: Preparing
f8a91dd5fc84: Waiting
cb81227abde5: Waiting
e01a454893a9: Waiting
c45660adde37: Waiting
fe0fb3ab4a0f: Waiting
f1186e5061f2: Waiting
b2dba7477754: Waiting
denied: requested access to the resource is denied
PS C:\Users\guilh>
```

docker push [imagem] → foi negado pois não segue o padrão do docker hub

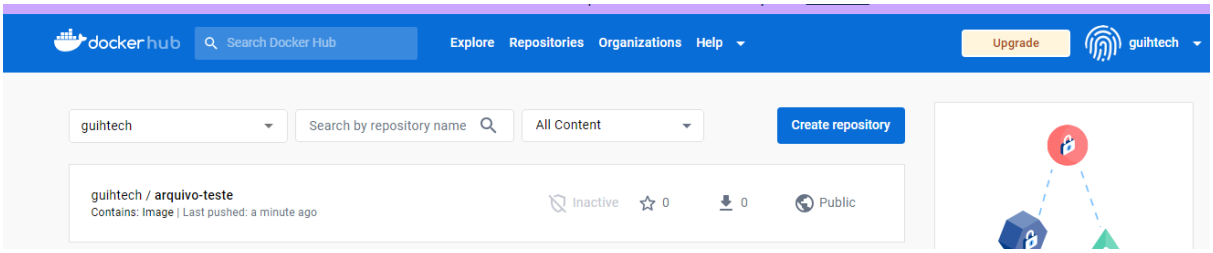
```
PS C:\Users\guilh> docker tag g:1.4 guihtech/arquivo-teste:1.0
PS C:\Users\guilh> docker images
REPOSITORY                  TAG         IMAGE ID       CREATED             SIZE
guihtech/arquivo-teste      1.0         c7655fced101   46 minutes ago      913MB
g                           1.4         c7655fced101   46 minutes ago      913MB
<none>                      <none>      a4dd9067c50f   About an hour ago   913MB
g                           1.1         0c37ff8a42b1   2 hours ago         913MB
```

```
docker tag [imagem] [ nome imagem padrão dockerhub]
```

```
PS C:\Users\guilh> docker push guihtech/arquivo-teste:1.0
The push refers to repository [docker.io/guihtech/arquivo-teste]
af5ad93291df: Pushed
2e8bbe88c965: Pushed
7fff75ecdbe5: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Pushed
f8a91dd5fc84: Pushed
cb81227abde5: Pushed
e01a454893a9: Pushed
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushed
1.0: digest: sha256:b3697a78a666be88974ca16a0e07a9a40206ee073bae3d5e368d68cf0cd32863 size: 2840
```

docker push guihtech/arquivo-teste:1.0

```
docker push guihtech/arquivo-teste:1.0
```




Subiu com sucesso conforme o padrão.

# Docker compose

O compose é uma ferramenta para definir e executar aplicativos docker de **vários containers**

docker-compose