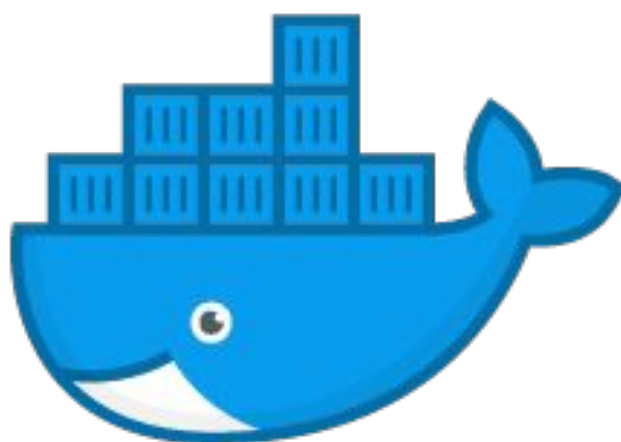


# Laboratório SO



**docker**  
**docs 22**

Uma introdução à  
containers com docker

# Sobre o material


Esse material foi elaborado para disciplina TT304 - Sistemas Operacionais.

Pelos monitores **Arthur G. S. Conti** e **Myrelle S. Lopes**, supervisionados pelo **Prof. Dr. André Leon S. Gradwohl**.


Esse material, tem como objetivo introduzir os alunos da disciplina a utilização de containers através do docker, dando exemplos práticos e de utilização, tanto para desenvolvimento local, quanto utilizando a **AWS**.

Vale ressaltar, que esse material **assume que seu ambiente na AWS já está configurado** conforme o material de configuração disponibilizado.

Esperamos que seja de grande ajuda e utilidade para vocês alunos e qualquer sugestão para a melhoria do material é bem-vinda!



# Índice

- **Introdução**
  - **Imagens**
  - **Containers**
  - **Dockerfile**
    - O que é ?
    - Como é um Dockerfile?
  - **Docker Compose**
    - O que é ?
    - Como é um docker-compose?
    - Exemplo docker-compose
    - Docker compose CLI
- 

# Introdução

O **Docker** é uma ferramenta que utiliza **virtualização** de nível de sistema operacional para empacotar **softwares** em **containers**

Um **container** é como se fosse um **software** que foi **empacotado** junto com todas suas dependências para rodar, dessa forma, **conseguimos criar uma área isolada** no SO para rodar essa aplicação, que **executa como** um **processo isolado compartilhando** o **kernel e recursos**.

Dessa forma, os **containers utilizam** muito menos recursos que VMs, por exemplo.

# Imagens

Imagens são **utilizadas** para **instanciar containers**, podemos pegar imagens já prontas e oficiais do Docker no site do **DockerHub** ou criar as nossas próprias utilizando Dockerfile, que será abordado mais para frente!

Para baixar uma imagem do docker hub utilizamos o comando:

```
docker pull nome_da_imagem [opções/flags]
```

Opcionalmente, também podemos passar a tag da imagem indicada no docker hub se quisermos uma imagem mais específica o comando ficaria dessa forma

```
docker pull nome_da_imagem:tag [opções/flags]
```

# Imagens

Algumas flags que esse comando pode ter:

flag	Significado	Função
-q	quiet	desabilitar o modo verbose, ou seja a saída do comando no console

# Imagens

Um exemplo de utilização, vamos baixar a imagem de um container **node** para nossa máquina!

## 1. Primeiro buscamos o nome da imagem no DockerHub

The screenshot shows the DockerHub search results for the query 'node'. The search bar at the top contains 'node'. The results are displayed in a list format. The first result is 'node', which is marked as an 'Official Image' with a green checkmark. A red arrow points to the 'node' icon. The second result is 'mongo-express', also marked as an 'Official Image'. The third result is 'circleci/node', marked as a 'Verified Publisher'. The fourth result is 'bitnami/node', also marked as a 'Verified Publisher'. The left sidebar shows filters for 'Images' and 'Categories'.

dockerhub  Explore Pricing Sign In [Sign Up](#)

[Docker](#) [Containers](#) [Plugins](#)

Filters 1 - 25 of 104,210 results for **node**. [Clear search](#) Best Match

☐ Verified Publisher   
 ☐ Official Images   
 Official Images Published By Docker

Categories   
 ☐ Analytics   
 ☐ Application Frameworks   
 ☐ Application Infrastructure   
 ☐ Application Services   
 ☐ Base Images   
 ☐ Databases   
 ☐ DevOps Tools   
 ☐ Featured Images   
 ☐ Messaging Services   
 ☐ Monitoring   
 ☐ Operating Systems   
 ☐ Programming Languages   
 ☐ Security   
 ☐ Storage

Operating Systems

**node** **Official Image**   
 Updated 6 days ago   
 1B+ Downloads 10K+ Stars   
 Node.js is a JavaScript-based platform for server-side and networking applications.   
 Container Linux PowerPC 64 LE x86-64 ARM 64 386 IBM Z ARM Application Infrastructure

**mongo-express** **Official Image**   
 Updated 3 months ago   
 100M+ Downloads 1.1K Stars   
 Web-based MongoDB admin interface, written with Node.js and express   
 Container Linux ARM 64 x86-64 DevOps Tools Application Frameworks

**circleci/node** **Verified Publisher**   
 By CircleCI • Updated a month ago   
 100M+ Downloads 121 Stars   
 Node.js is a JavaScript-based platform for server-side and networking applications.   
 Container Linux x86-64

**bitnami/node** **Verified Publisher**   
 10M+ Downloads 57 Stars

# Imagens

Um exemplo de utilização, vamos baixar a imagem de um container **node** para nossa máquina!

## 1. Primeiro buscamos o nome da imagem no DockerHub

The screenshot shows the DockerHub search results for the term 'node'. The search bar at the top contains 'node'. Below the search bar, there are tabs for 'Docker', 'Containers', and 'Plugins'. The search results are displayed in a list format. The first result is 'node', which is marked as an 'Official Image' with a green checkmark. A red arrow points to this result. The second result is 'mongo-express', also marked as an 'Official Image'. The third result is 'circleci/node', marked as a 'Verified Publisher'. The fourth result is 'bitnami/node', also marked as a 'Verified Publisher'. The search results are filtered by 'Best Match'.

dockerhub node Explore Pricing Sign In Sign Up

Docker Containers Plugins

Filters 1 - 25 of 104,210 results for **node**. [Clear search](#) Best Match

Images

☐ Verified Publisher ☐ Official Images Official Images Published By Docker

Categories

☐ Analytics ☐ Application Frameworks ☐ Application Infrastructure ☐ Application Services ☐ Base Images ☐ Databases ☐ DevOps Tools ☐ Featured Images ☐ Messaging Services ☐ Monitoring ☐ Operating Systems ☐ Programming Languages ☐ Security ☐ Storage

Operating Systems

**node** Official Image Updated 6 days ago 1B+ Downloads 10K+ Stars

Node.js is a JavaScript-based platform for server-side and networking applications.

Container Linux PowerPC 64 LE x86-64 ARM 64 386 IBM Z ARM Application Infrastructure

**mongo-express** Official Image Updated 3 months ago 100M+ Downloads 1.1K Stars

Web-based MongoDB admin interface, written with Node.js and express

Container Linux ARM 64 x86-64 DevOps Tools Application Frameworks

**circleci/node** Verified Publisher By CircleCI • Updated a month ago 100M+ Downloads 121 Stars

Node.js is a JavaScript-based platform for server-side and networking applications.

Container Linux x86-64

**bitnami/node** Verified Publisher 10M+ Downloads 57 Stars

**Note** que existe uma insígnia verde escrita "**oficial image**", isso significa que essa imagem é uma imagem curada e mantida por eles, elas provêm a base para a utilização sem problemas, além de estarem sempre atualizadas servem de ponto de partida para a maioria dos usuários.



# Imagens

2. Após encontrarmos a imagem, podemos baixar ela utilizando o comando abaixo:

```
docker pull node
```

Aqui não utilizaremos nenhuma flag

Caso quisesse uma versão específica do node, você consegue ver as tags acessando o menu de tags da imagem



# Imagens



node

Official Image ☆

Node.js is a JavaScript-based platform for server-side and networking applications.

↓ 1B+

Container Linux PowerPC 64 LE x86-64 ARM 64 386 IBM Z ARM Application Infrastructure  
Official Image

Description

Reviews

Tags

Copy and paste to pull this image

```
docker pull node
```

[View Available Tags](#)

## Quick reference

- **Maintained by:** The Node.js Docker Team
- **Where to get help:** the Docker Community Forums, the Docker Community Slack, or Stack Overflow

## Supported tags and respective Dockerfile links

- 17-alpine3.14 , 17.3-alpine3.14 , 17.3.1-alpine3.14 , alpine3.14 , current-alpine3.14
- 17-alpine , 17-alpine3.15 , 17.3-alpine , 17.3-alpine3.15 , 17.3.1-alpine , 17.3.1-alpine3.15 , alpine , alpine3.15 , current-alpine , current-alpine3.15
- 17 , 17-bullseye , 17.3 , 17.3-bullseye , 17.3.1 , 17.3.1-bullseye , bullseye , current , current-bullseye , latest

# Imagens

ContainerLinuxPowerPC 64 LEx86-64ARM 64386IBM ZARMApplication Infrastructure

Official Image

Copy and paste to pull this image

`docker pull node`

[View Available Tags](#)

DescriptionReviewsTags

Sort byNewestFilter Tags

TAG

**fermium-buster-slim** Log4Shell CVE not detected

Last pushed 6 days ago by [doijanky](#)

DIGEST

1cc39235887e

f394d87fddd7

14ea9d3661b8

+2 more...

OS/ARCH

linux/amd64

linux/arm/v7

linux/arm64/v8

COMPRESSED SIZE

62.4 MB

56.46 MB

61.34 MB

`docker pull node:fermium-buster-slim`

TAG

**fermium-buster** Log4Shell CVE not detected

Last pushed 6 days ago by [doijanky](#)

DIGEST

542f572bc5ed

aea2d9296c99

3dea9434aa8e

+2 more...

OS/ARCH

linux/amd64

linux/arm/v7

linux/arm64/v8

COMPRESSED SIZE

333.7 MB

299.34 MB

324.55 MB

`docker pull node:fermium-buster`

TAG

Você pode observar que o próprio hub te apresenta o comando para baixar a imagem

# Imagens

3. Após a imagem ter sido baixada, podemos listar as imagens presente no nosso sistema utilizando o comando:

```
docker image ls
```

# ou

```
docker images
```

Caso você não queria fazer o processo de baixar uma imagem para só depois rodar o container, você pode fazer o download da imagem direto na instanciação do container, como abordaremos no nosso próximo tópico

# Containers

O Docker por **padrão**, sempre **busca a imagem** para subir o container **primeiramente de forma local, caso não encontre** nenhuma imagem localmente, ele então **passa a buscar online** no DockerHub, baixa ela, e então instancia o container.

Para instanciar um container, utilizamos o comando:

*`docker run [opções/flags] nome_da_imagem`*

*O comando docker run, na realidade faz a função de outros dois comandos simultaneamente, o docker create e o docker start, porém podemos utilizar o run sem problemas :)*

# Containers

Esse comando possui diversas flags porém vamos ver as principais aqui:

Flag	Significado	Parâmetros	Função
--rm	Remover	Nenhum	Automaticamente remove o container quando ele terminar/parar
-d	Desacoplar (detach)	Nenhum	Roda o container em background e retorna o id
--name	Nome do container	nome que deseja para o container (o docker gera um nome aleatório se não usar)	Permite nomear o container que será criado
-v	Volumes	pastHost:pastContainer (./data:/usr/share/www)	Vincula o ponto de montagem dos volumes, a pasta local com a pasta interna do container
-p	Publicar portas	portHost:portContainer (7777:3306)	Torna pública uma porta para o host, deixando o container acessível, mapeia portas

# Containers

Esse comando possui diversas flags porém vamos ver as principais aqui:

Flag	Significado	Parâmetros	Função
-e	Variáveis de ambiente	VARIAVEL=valor (-e NODE_ENV=test)	Responsável por setar variáveis de ambiente ao container
-t	Terminal	Nenhum	Acopla um pseudo-terminal ao container
-i	Iterativo	Nenhum	O processo não vai ser finalizado até a conclusão

*A flag -i pode ser combinada com a flag t e/ou com a flag -d, ficando -dit*

# Containers

Esse comando possui diversas flags porém vamos ver as principais aqui:

Flag	Significado	Parâmetros	Função
--network	Rede	nome da rede criada	Conecta o container a uma rede específica de container
--link	Vincular	nome do container	Vincula um container a outro ou outros containers, podendo substituir o ip nos projetos pelo nome do container
--restart	Reiniciar	opções: {no,on-failure, on-failure:maxTentativas, unless-stopped, always}	Define a política para reiniciar o container, padrão é no, ou seja, o docker nunca vai reiniciar o container, as outras políticas definem a condição para que o docker reinicie o container baseado em quando ele parar/sair



# Containers

Um exemplo de docker run utilizando uma imagem node que não existe localmente e alguma das flags descritas

```
🐳 DockerDocs22 [main] ⚡ docker run -d --name nodejs -p 3000:3000 node:latest
Unable to find image 'node:latest' locally
latest: Pulling from library/node
0c6b8ff8c37e: Pull complete
412caad352a3: Pull complete
e6d3e61f7a50: Pull complete
461bb1d8c517: Pull complete
808edda3c2e8: Pull complete
2d01df36eabd: Pull complete
b174a8d53337: Pull complete
07c1216b144e: Pull complete
8216eb213fee: Pull complete
Digest: sha256:88becea956ea5ec0262b8aac011a234f95310e5cacc38cc9d2468a836d67ffc9
Status: Downloaded newer image for node:latest
2bde9c3a2d07cfc4f0420a9eed8ce9e1e81f57056a9049000924838e3a47b7f7
```

# Containers

Após instanciarmos um container, ou vários containers, podemos listar os containers ativos usando o comando:

```
docker ps
```

Caso você queira listar todos os containers que você tem criado e informações específicas de cada um, basta usar o comando:

```
docker ps -a
```

```
img5 [main] ⚡ docker ps -a
```

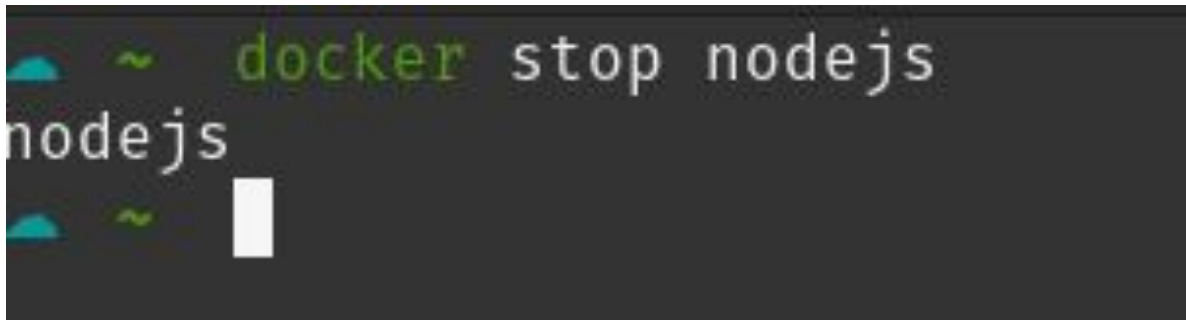
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2bde9c3a2d07	node:latest	"docker-entrypoint.s..."	11 minutes ago	Exited (0) 2 minutes ago		nodejs
1dc575e1ddb4	mysql:latest	"docker-entrypoint.s..."	5 months ago	Up 18 seconds	3306/tcp, 33060/tcp	mysqlBD
e9a711325343	hello-world	"/hello"	5 months ago	Exited (0) 5 months ago		reverent_carv

```
er
```

# Containers

Para pararmos a execução de um container utilizamos o comando

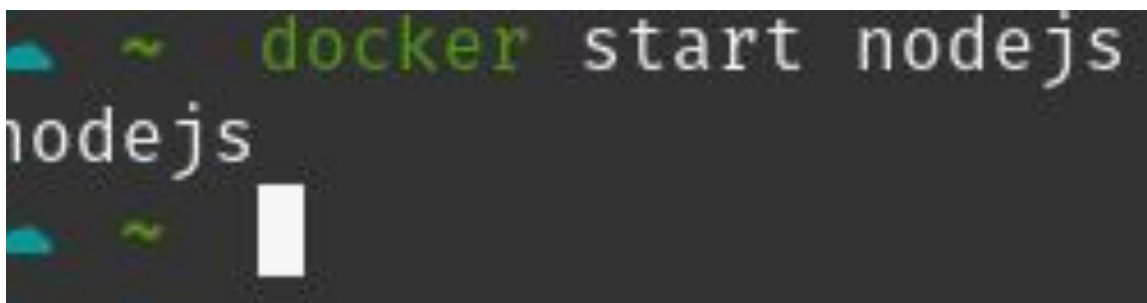
*docker stop nome\_container | id\_container*

A terminal window with a dark background. The prompt is a teal fish icon followed by a tilde '~'. The command 'docker stop nodejs' is entered in green text. Below the command, the output 'nodejs' is shown in white text. A white cursor is positioned at the end of the command line.

```
~ docker stop nodejs
nodejs
```

E após pararmos um container, ou quando iniciamos nosso computador, para subir um iniciar um container criado, utilizamos

*docker start nome\_container | id\_container*

A terminal window with a dark background. The prompt is a teal fish icon followed by a tilde '~'. The command 'docker start nodejs' is entered in green text. Below the command, the output 'nodejs' is shown in white text. A white cursor is positioned at the end of the command line.

```
~ docker start nodejs
nodejs
```

# Containers


Nesse momento, **já sabemos** como **procurar imagens** na internet e onde encontrá-las, também aprendemos a **instanciar, parar e subir** novamente um **container**!

Subimos um container node, para demonstrar como funciona, porém, esse container node, **ainda não terá tanta utilidade**, voltaremos a utilizar ele mais para frente quando falarmos de Dockerfile, porém, vamos agora, fazer a **instanciação completa** de um **container** com uma coisa que para nós programadores é de suma importância, um **SGBD**, ao invés de baixarmos diversos gerenciadores, **podemos ter várias imagens** configuradas e só **subirmos quando e qual for necessária**!

# Containers

Vamos trabalhar com o **MySQL** mas também funciona com o Postgres, basta ver quais flags e opções são necessárias para essa imagem, você pode consultar isso [nesse link](#)

Como vimos, devemos sempre começar pela imagem, nesse exemplo, utilizaremos a imagem oficial do DockerHub [disponível neste link](#)



# Containers

Tendo escolhido nossa imagem, podemos rodar o seguinte comando para instanciar nosso container, tendo em mente, que optamos pelo download da imagem na nuvem

```
docker run -d --name exemploMySQL -p 3306:3306  
-e MYSQL_ROOT_PASSWORD=root mysql:latest
```

*#vamos também criar um container para podermos  
remover futuramente*

```
docker run -d --name containerRemover -p 3306 -e  
MYSQL_ROOT_PASSWORD=root mysql:latest
```

Observação: A variável de ambiente **MYSQL\_ROOT\_PASSWORD** é **obrigatória** para definirmos a senha padrão de acesso ao bd, e essa variável está na documentação oficial da imagem, o usuário por padrão do mysql é o **root**

# Containers

Ao rodarmos esse comando e logo na sequência:

*docker ps*

Obtemos o seguinte retorno no terminal:

```
~ docker run -d --name exemploMySQL -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root
mysql:latest
a85aeea2d0e9befd102c50f6c83e255b2ac911de9ac0a900062a3610350d93a7
~ docker run -d --name containerRemover -p 3306 -e MYSQL_ROOT_PASSWORD=root
mysql:latest
e8423d9b818b48cc45fad22437c662c92406b1f5284e84d3cd879dfb84e623ed
~ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
e8423d9b818b   mysql:latest   "docker-entrypoint.s..." 11 seconds ago Up 9 seconds  33060/tcp, 0.0.0.0:49155->3306/tcp, :::49154->3306/tcp containerRemover
a85aeea2d0e9   mysql:latest   "docker-entrypoint.s..." 15 seconds ago Up 14 seconds  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp exemploMySQL
```

# Containers

Podemos visualizar também os container ativos e inativos por uma interface, como, por exemplo do [ctop](#), em verde todos os containers rodando, e em vermelho, o que estão parados:

ctop - 16:46:38 -03 8 containers									
NAME	CID	CPU	MEM	NET RX/...	IO R /W	PIDS	UPTI ME		
▶ containerRemover	e8423d9b818b	0%	368M	/6K ...	0B ...	37	1m3...		
▶ exemploMySQL	a85aeea2d0e9	0%	357M	/7K ...	0B ...	37	1m3...		
● backend_bd_1	cf1351c19e8e	-	-	-	-	-	2h1...		
● mysqlBD	1dc575e1ddb4	-	-	-	-	-	4h1...		
● nodejs	2bde9c3a2d07	-	-	-	-	-	0s		
● postgres	1e8c300eb9e0	-	-	-	-	-	1h1...		



# Containers

Podemos ver os logs desse container rodando utilizando o comando:

*docker logs exemploMySQL*

e temos de retorno do comando todos os logs que o container emitiu:

```
~ docker logs exemploMySQL
2022-06-18 19:44:56+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Serve
r 8.0.26-1debian10 started.
2022-06-18 19:44:56+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysq
l'
2022-06-18 19:44:56+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Serve
r 8.0.26-1debian10 started.
2022-06-18 19:44:56+00:00 [Note] [Entrypoint]: Initializing database files
2022-06-18T19:44:56.789551Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (my
sqld 8.0.26) initializing of server in progress as process 42
2022-06-18T19:44:56.804105Z 1 [System] [MY-013576] [InnoDB] InnoDB initializatio
n has started.
2022-06-18T19:44:57.582432Z 1 [System] [MY-013577] [InnoDB] InnoDB initializatio
n has ended.
2022-06-18T19:44:59.136674Z 0 [Warning] [MY-013746] [Server] A deprecated TLS ve
rsion TLSv1 is enabled for channel mysql_main
2022-06-18T19:44:59.137518Z 0 [Warning] [MY-013746] [Server] A deprecated TLS ve
rsion TLSv1.1 is enabled for channel mysql_main
2022-06-18T19:44:59.273278Z 6 [Warning] [MY-010453] [Server] root@localhost is c
reated with an empty password ! Please consider switching off the --initialize-i
nsecure option.
2022-06-18 19:45:03+00:00 [Note] [Entrypoint]: Database files initialized
```

# Containers

Na sequência, precisamos executar alguns comandos, como por exemplo rodar um script para criar nosso banco de dados e nossa tabela, existem diversas formas para se fazer isso, nesse exemplo, vamos passar um script sql para nosso banco de dados, esses script foi previamente criado e não entraremos em detalhe de como criá-lo pois não é o nosso foco, após isso entraremos no banco para vermos se tudo foi executado corretamente, para isso, precisamos usar o comando **docker exec**, que vamos a seguir:

*docker exec [FLAGS] container [COMANDO]*



# Containers

O docker exec aceita qualquer comando Linux ou referente a imagem como o mysql

Flag	Significado	Parâmetros	Função
-i	Iterativo	Nenhum	Mantém a interação no terminal enquanto o comando do container está em execução
-t	TTY (terminal)	Nenhum	Aloca um terminal ao container

# Containers

Nesse nosso exemplo, vamos passar o comando de conexão ao banco mysql com o usuário e a senha e logo após com o operador < passamos o arquivo para ser executado pelo mysql, dessa forma nosso banco será criado.

```
docker exec -i exemploMySQL mysql -uroot -proot < exemplo.sql
```

Para vermos se o banco foi criado, temos diversas formas de fazer, mas iremos via terminal, para isso precisaremos acessar o mysql via linha de comando e passar os comandos para visualização de banco de dados e tabelas!

# Containers

Segue o fio:

*# Instanciando o terminal e acessando o mysql  
docker exec -it exemploMySQL mysql -uroot -proot*

*# Dentro do terminal do mysql digitamos para  
vermos se o banco está criado  
show databases;*

*# Após isso executamos para ver se nossa tabela  
está criada corretamente dentro do banco  
use labso; show tables;*

*# Após isso podemos usar o select e confirmar se  
os dados foram inseridos corretamente  
select \* from pessoa;*

*# E finalmente para sairmos do terminal basta  
digitar o comando  
exit*

# Containers

E agora para removermos um container precisamos primeiramente pará-lo e em seguida utilizamos o seguinte comando:

*docker stop containerRemover*

*docker container rm containerRemover*





# Dockerfile

# O que é?

Dockerfile nada mais é do que um arquivo onde conseguimos especificar todos os comandos que poderíamos passar pela linha de comando para criarmos nossa própria imagem.

Após escrevermos todas as instruções, basta rodar o comando:

```
docker build -f [file] -t [tagname] [PATH]
```





# O que é?

Flag	Significado	Parâmetros	Função
-f	file	Nome do Dockerfile	Especificar qual o arquivo vai servir de base para a construção da imagem, por padrão, o docker build procura no contexto o arquivo chamado Dockerfile, mas através desse comando pode-se especificar outro caminho para outra Dockerfile
-t	tagname	nome da imagem (name:tag)	Especificar qual será o nome customizado da imagem, a tag que vem após o nome é opcional

*Temos também o **PATH**, o **PATH especifica onde encontrar os arquivos para o contexto de criação da imagem**, o path pode ser um caminho para pasta, uma url ou até mesmo um repositório do git, aqui, nos limitaremos a trabalhar só com o caminho para pastas locais*

*Caso você não vá utilizar a flag -f, necessariamente seu arquivo dockerfile **DEVE** se chamar "Dockerfile"*

# Como é um Dockerfile?

O Dockerfile possui diversos comandos para dar suporte a criação de uma imagem customizada, veremos alguns exemplos e detalharemos alguns dos comandos principais, vale ressaltar que obrigatoriamente, um arquivo dockerfile deve ser nomeado como Dockerfile!



# Como é um Dockerfile?

Instruções	Função
FROM	Ponto de partida para a criação de uma imagem, se quiser algo baseado em Mysql, pode especificar, se quiser fazer uma imagem do zero, basta informar SCRATCH
RUN	Pode ser executada uma ou mais vezes, definimos quais os comandos serão executados nas etapas de criação de uma imagem.
CMD	Definimos quais comandos serão executados na etapa de criação do container, caso o container não tenha nenhum comando
ENTRYPOINT	A mesma coisa que o CMD, porém seus parâmetros não são sobrescritos
ADD	Fazer cópia de arquivos, diretório ou até mesmo baixar arquivos, da máquina host ou internet para a imagem
COPY	Permite apenas a passagem de arquivos ou diretórios, diferente do ADD que permite downloads
EXPOSE	Serve para documentar qual a porta será exposta, mas efetivamente <b>não</b> publica a porta
VOLUME	Cria uma pasta que será compartilhada entre o container e o host
WORKDIR	Tem o propósito de definir onde as instruções acima executarão suas tarefas, além de definir o diretório padrão que será aberto ao executarmos o container

# Como é um Dockerfile?

Agora, vamos a um exemplo prático de como utilizar essas instruções!

Vamos criar uma aplicação básica em nodejs, só para acessarmos algumas rotas e vocês poderão ver diversas partes do Dockerfile em ação!

o código em JS é um exemplo simples e estará disponibilizado, pois, todo o desenvolvimento dele não faz parte dessa matéria



# Como é um Dockerfile?

*#Especificando qual imagem vamos utilizar de base*  
**FROM** node

*#Indicamos a pasta que será criada dentro do container, para ser o ponto inicial dos arquivos*  
**WORKDIR** /server-example

*#Copiando o arquivo package.json da pasta atual para a pasta server-example do container*  
**COPY** package.json .

*# Copiando o server.js da pasta atual para a pasta server-example do container*  
**ADD** server.js .

*# Documentando que o container terá a porta 3333 exposta*  
**EXPOSE** 3333

*# rodando npm install durante a criação da imagem, para quando baixarmos ela já vir com o node\_modules pronto*  
**RUN** npm install --production

*# Rodando npm start após a criação do container para o container já subir com o servidor inicializado*  
**CMD** ["npm","start"]


# Como é um Dockerfile?

Após montarmos o Dockerfile, precisamos criar essa image, para fazer isso, basta rodar o docker build

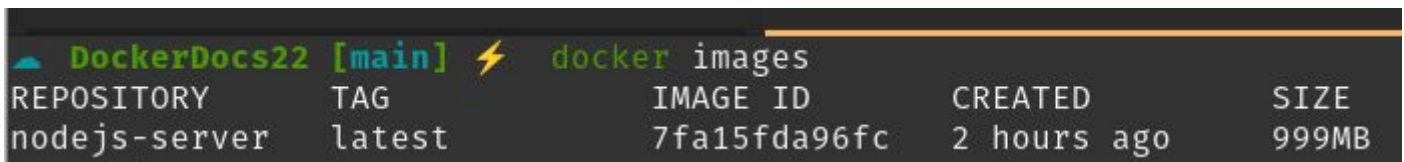
*docker build -t nodejs-server .*

*Obs: Para o usarmos o '.', precisamos estar na pasta em que existe o Dockerfile, caso contrário é necessário passar o caminho para esse dockerfile*

Dessa forma se rodarmos o docker images vemos nossa imagem criada e pronta para ser instanciada com o comando docker run, como demonstrado a seguir!



# Como é um Dockerfile?



A terminal window showing the command `docker images` and its output. The output is a table with columns: REPOSITORY, TAG, IMAGE ID, CREATED, and SIZE. The first row shows `nodejs-server` with tag `latest`, image ID `7fa15fda96fc`, created 2 hours ago, and size 999MB.

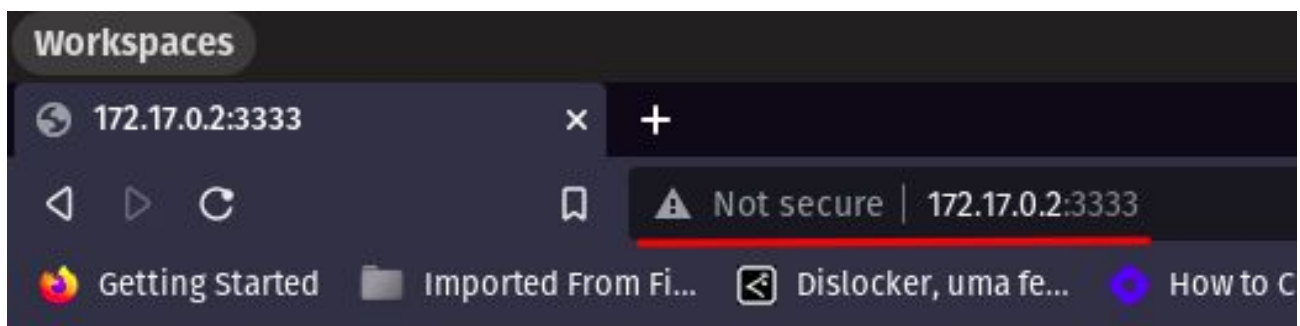
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nodejs-server	latest	7fa15fda96fc	2 hours ago	999MB

```
docker run --rm -d --name server-labso  
nodejs-server
```

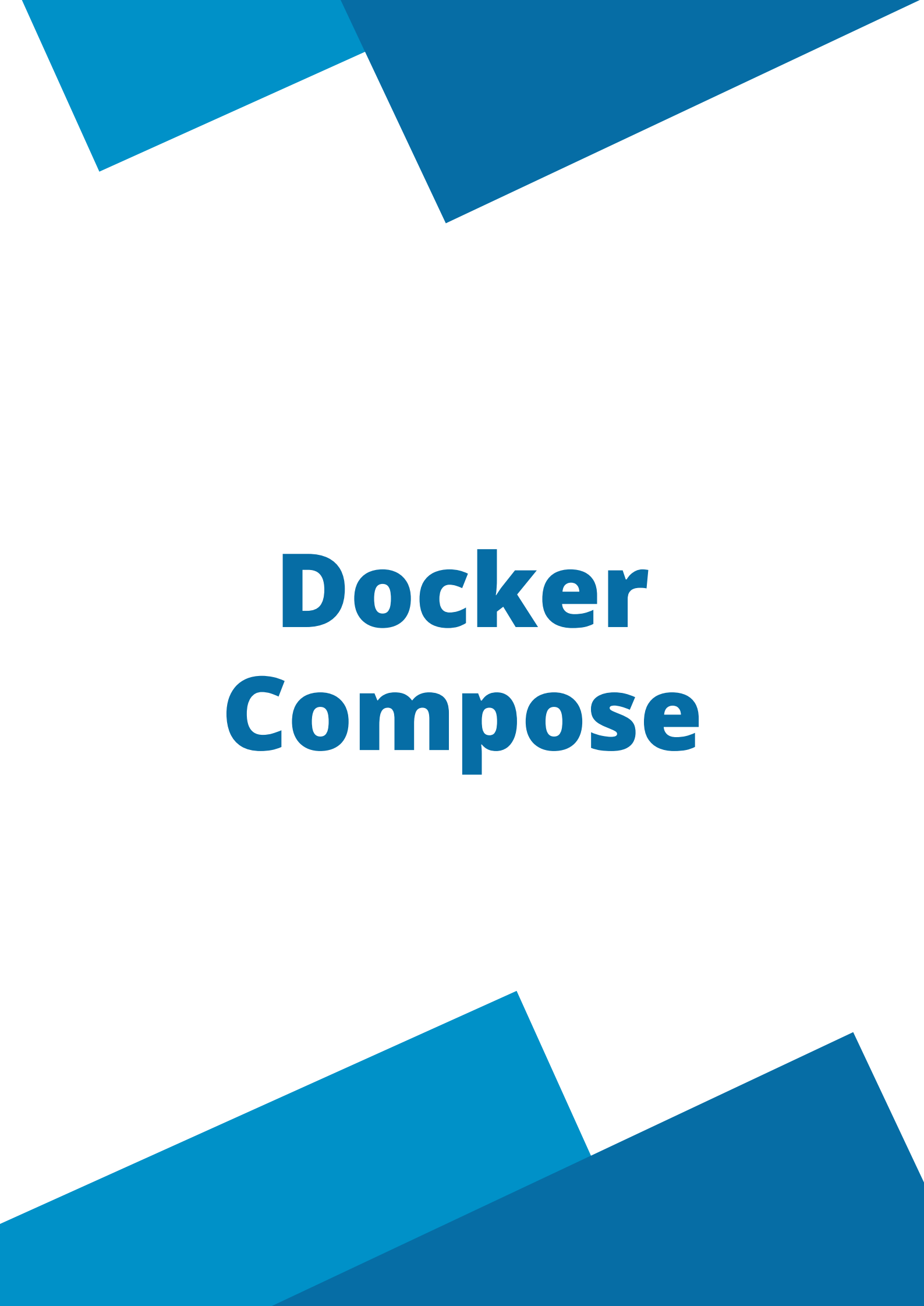
# Se estiver rodando o container na sua máquina, precisamos pegar o ip para poder acessar ela, usamos o seguinte comando para saber qual o ip, se for na aws basta passar o seu ipv4 público com a porta

```
docker inspect server-labso | grep IPAddress
```

e com isso podemos acessar a api pelo navegador ou fazer requisições



Olá bem vindo ao servidor express



# **Docker Compose**



# O que é?

O Docker compose é uma ferramenta para definir e rodar múltiplos containers de aplicação, para o docker compose, utilizamos um arquivo do tipo YAML para configurar as aplicações, então com um único comando, conseguimos criar e iniciar todos os serviços configurados no arquivo.



# Como é um docker-compose?

Assim como no dockerfile precisamos que esse arquivos seja nomeado como Dockerfile, no compose, precisamos nomear o arquivo como **docker-compose.yml** ou **docker-compose.yaml**

Um docker compose possui uma estrutura básica e vamos explicar ela a seguir!

```
version: '3.4'
```

```
services:
```

```
  service_name:
```

```
    image: image_name
```

```
    ports: 0000:0000
```

```
    volumes:
```

```
      - ./pathFolder:/containerPath
```

```
    environment:
```

```
      - KEY=VALUE
```

```
    depends_on:
```

```
      - service
```

```
volumes:
```

```
  volume:
```

```
networks:
```

```
  mydata:
```

# Como é um docker-compose?

Dessas partes vamos detalhar elas:

- **version** especifica qual a versão do docker-compose vamos utilizar
- **services** é a parte onde definimos todos nossos serviços, ou nossos container que serão instanciados, dentro de service, definimos um nome para o service e podemos passar outras informações como
  - **image**: imagem base para o container
  - **ports**: porta que será exposta para nosso container seguindo padrão host:container
  - **volumes**: volume que será criado para o container, assim como a flag -v do docker run
  - **enviroment**: variáveis de ambiente para o container
  - **depends\_on**: basicamente nos informa que antes daquele container subir, o container que esta no depends on precisa já estar rodando, ou seja, através dele conseguimos controlar a ordem em que os containers sobem.
- **volumes** é onde definimos os volumes que serão utilizados, aqui, passa o nome do volume pois podemos definir a pasta diretamente no service
- **networks** é onde usamos para criar redes internas entre os containers e depois passamos isso dentro do service em network

# Como é um docker-compose?

Vale ressaltar, que **existem partes** que **não são obrigatórias**, porém **obrigatoriamente** precisamos **definir os services** e dependendo de como vamos trabalhar com o docker compose, é sempre bom **definir a versão** do docker compose, por exemplo, por padrão, o próprio docker-compose cria uma network para aqueles serviços que estão, não sendo necessário definir uma network, a menos que haja necessidade.

Agora que já falamos dessa parte, vamos a um **exemplo prático**, utilizando tudo que já foi mencionado anteriormente, nesse exemplo, vamos instanciar um banco de dados, uma api e também uma interface web, cada um em um container e todos se comunicando!

A seguir, teremos o arquivo docker-compose.yaml detalhado e comentado, porém ele também pode ser encontrado [nesse link](#)

**Importante:** teremos que construir as imagens e **todos os endereços de ip** do arquivo **docker-compose** deve ser **substituído pelo seu da AWS**

# Exemplo

# docker-compose

*version: '3.4'*

*#Definição dos serviços*  
*services:*

*#container app, ele terá nossa interface*  
*app:*

*#image sendo utilizada, essa imagem será construída através do Dockerfile*

*image: app-labso*

*#container\_name é onde damos um nome customizado ao container que o docker-compose sobe*

*container\_name: compose-app-labso*

*#ports onde mapeamos a porta host:container*

*ports:*

*- 80:8080*

*#depends\_on é onde falamos qual o container precisa subir antes desse*

*# nesse caso, nosso container app, só subirá depois que o container de bd e o da api subirem*

*depends\_on:*

*api:*

*condition: service\_healthy*

# Exemplo

## docker-compose

*#container api, esse container terá nossa api*

*api:*

*image: api-labso*

*container\_name: compose-api-labso*

*#environment passamos as variáveis de ambiente para essa aplicação, isso pode ser feito através de um .env por exemplo*

*environment:*

*- DATABASE\_URL=mysql://root:root@0.0.0.0:3306/labso*

*ports:*

*- 3333:3333*

*#command sobrescreve o CMD da imagem, nessa caso, assim que o container subir ele ira rodar uma migration e depois iniciará o servidor da api*

*command: bash -c "npx prisma db push && npm run start"*

*healthcheck:*

*test: curl --fail http://0.0.0.0:3333/pessoas || exit 1*

*retries: 3*

*interval: 3s*

*#nesse depends\_on, temos um condition, esse condition nos diz que nosso container de api só subira depois que o container db subir e ele estiver em um estado saudável*

*# como queremos rodar uma migration nesse banco de dados, precisamos que além dele estar instanciado, precisamos que ele já esteja inicializado e podendo receber conexões*

*# então dentro do container de bd a gente vai definir o que para aquele container é estar healthy*

*depends\_on:*

*db:*

*condition: service\_healthy*

# Exemplo

## docker-compose

*db:*

*image: mysql*

*container\_name: compose-bd-labso*

*environment:*

*- MYSQL\_ROOT\_PASSWORD=root*

*- MYSQL\_DATABASE=labso*

*ports:*

*- 3306:3306*

*#healthcheck é onde a gente vai definir um teste para nos dizer se o container está healthy/saudável ou não*

*# nesse caso, como precisamos que o container esteja rodando e aceitando conexões, passamos que o teste desse container é dar um ping nele mesmo com o usuário root*

*# caso o teste passe, ele nos retorna que o container é healthy e o container de api pode subir.*

*healthcheck:*

*test: mysqladmin ping -h 0.0.0.0 -u root*  
*-p\$\$MYSQL\_ROOT\_PASSWORD*

*retries: 10*



# Exemplo docker-compose

Agora que temos nosso arquivo docker-compose precisamos subir ele, porém, como temos imagem customizada, precisaremos criar essas imagens!


Nossa imagem do app terá o seguinte Dockerfile:

```
FROM node:lts-alpine  
RUN npm install -g http-server
```

```
WORKDIR /app
```

```
COPY package*.json ./  
RUN npm install --production  
COPY ..
```

```
EXPOSE 8080  
CMD ["http-server","dist"]
```





# Exemplo docker-compose

Nossa imagem da api terá o seguinte Dockerfile:

```
FROM node:lts-alpine  
WORKDIR /server-api
```

```
COPY package.json .  
ADD *.js ./  
COPY prisma/* ./prisma/  
COPY controller/* ./controller/
```

```
EXPOSE 3333
```

```
RUN npm install --production
```

```
RUN ["npx","prisma","generate"]  
CMD ["http-server","dist"]
```

# Exemplo

## docker-compose

Após criarmos os dois dockerfile, precisamos fazer o build delas, então executaremos os seguintes comandos:

```
cd frontend-vue
docker build -t app-labso .
cd ../backend
docker build -t api-labso .
```

Após o build feito, se usarmos docker images veremos que as imagens já estão lá!

```
DockerDocs22 [main] ⚡ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
app-labso     latest   4ce52d9beae4   4 days ago    336MB
api-labso     latest   78be11041ccf   4 days ago    1.35GB
```

# Exemplo

## docker-compose

Dessa forma podemos agora então, rodar o docker compose para subir nossos containers!

```
cd ..  
docker-compose up -d
```

Devemos ver a seguinte tela quando executarmos o comando

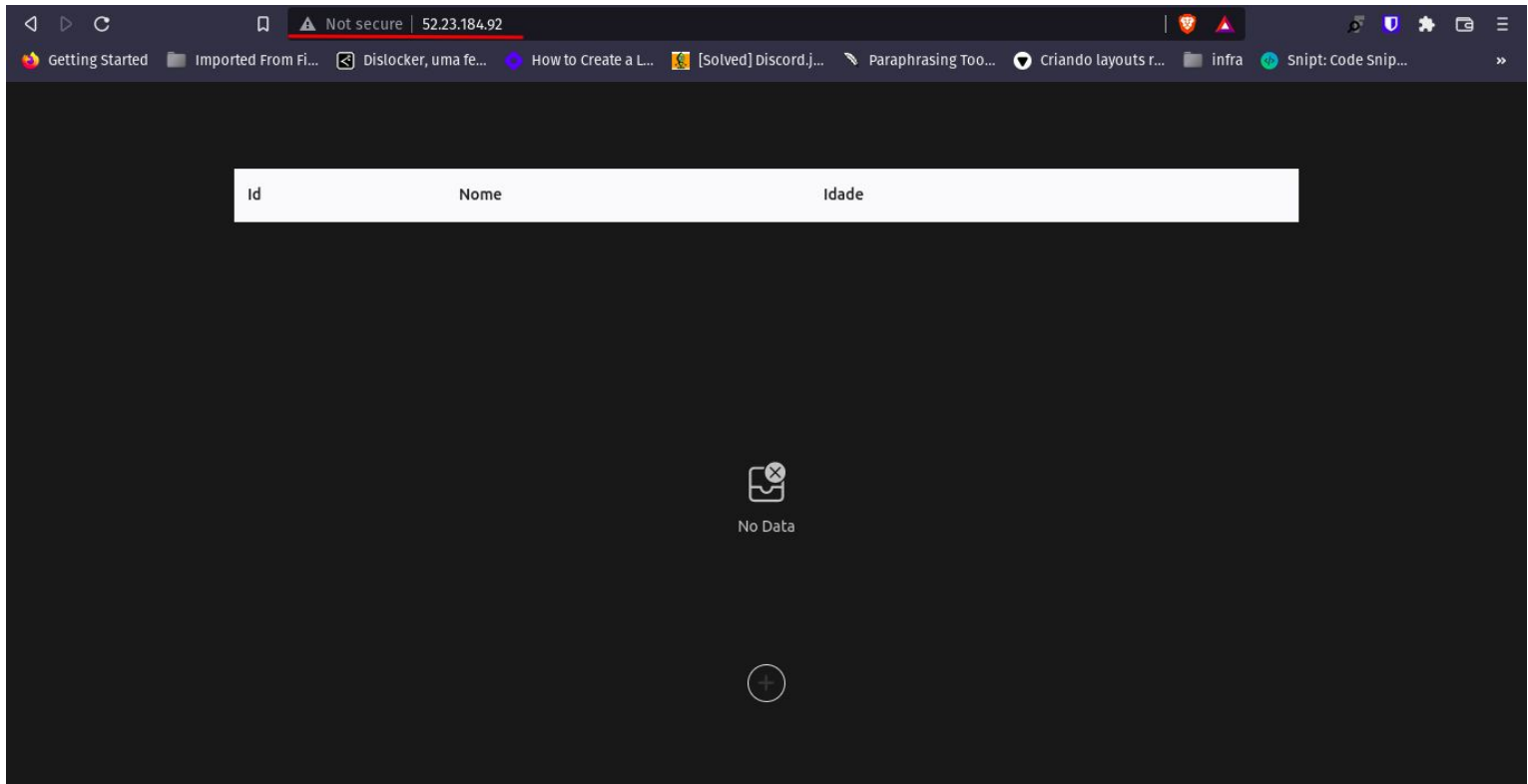
```
[ec2-user@ ██████████ dockerCompose-example]$ docker-compose up -d  
[+] Running 3/4  
:: Network dockercompose-example_default Created  
:: Container compose-bd-labso Waiting  
:: Container compose-api-labso Created  
:: Container compose-app-labso Created
```

Assim que os container estiverem prontos devemos ver a seguinte imagem

```
[ec2-user@ ██████████ dockerCompose-example]$ docker-compose up -d  
[+] Running 4/4  
:: Network dockercompose-example_default Created  
:: Container compose-bd-labso Healthy  
:: Container compose-api-labso Healthy  
:: Container compose-app-labso Started
```

# Exemplo docker-compose

E nesse momento, se digitarmos o ip no nosso navegador, será possível visualizar a aplicação!!!



# Docker-compose CLI

Dentro da CLI do docker-compose temos diversos comandos, vamos mostrar algum deles além do up

Comando	Descrição
up	Cria e inicia os containers dos serviços
down	Para todos os serviços e remove os recursos, networks, containers, etc...
start	Inicia os serviços
stop	Para os containers dos serviços
images	Lista as imagens
rm	Remove/apaga todos os containers parados
top	Mostro todos os processos rodando
kill	Derruba os containers

*No docker-compose up, podemos usar a flag --no-start para somente criar os containers sem iniciá-los*

# Docker-compose CLI

Vale lembrar, que todos esses comandos são feitos baseados no `docker-compose.yml`, então para funcionar, eles precisam ser executados na pasta que contém o arquivo, ou você especificando o caminho para ele.

No caso, por exemplo, do comando *`docker-compose rm`*, ele não irá remover **TODOS** os containers que você criou que estão parados, ele irá remover todos os containers de serviço que o `docker-compose.yml` definiu, ou seja, todos os containers parados referente a esse arquivo serão apagados.