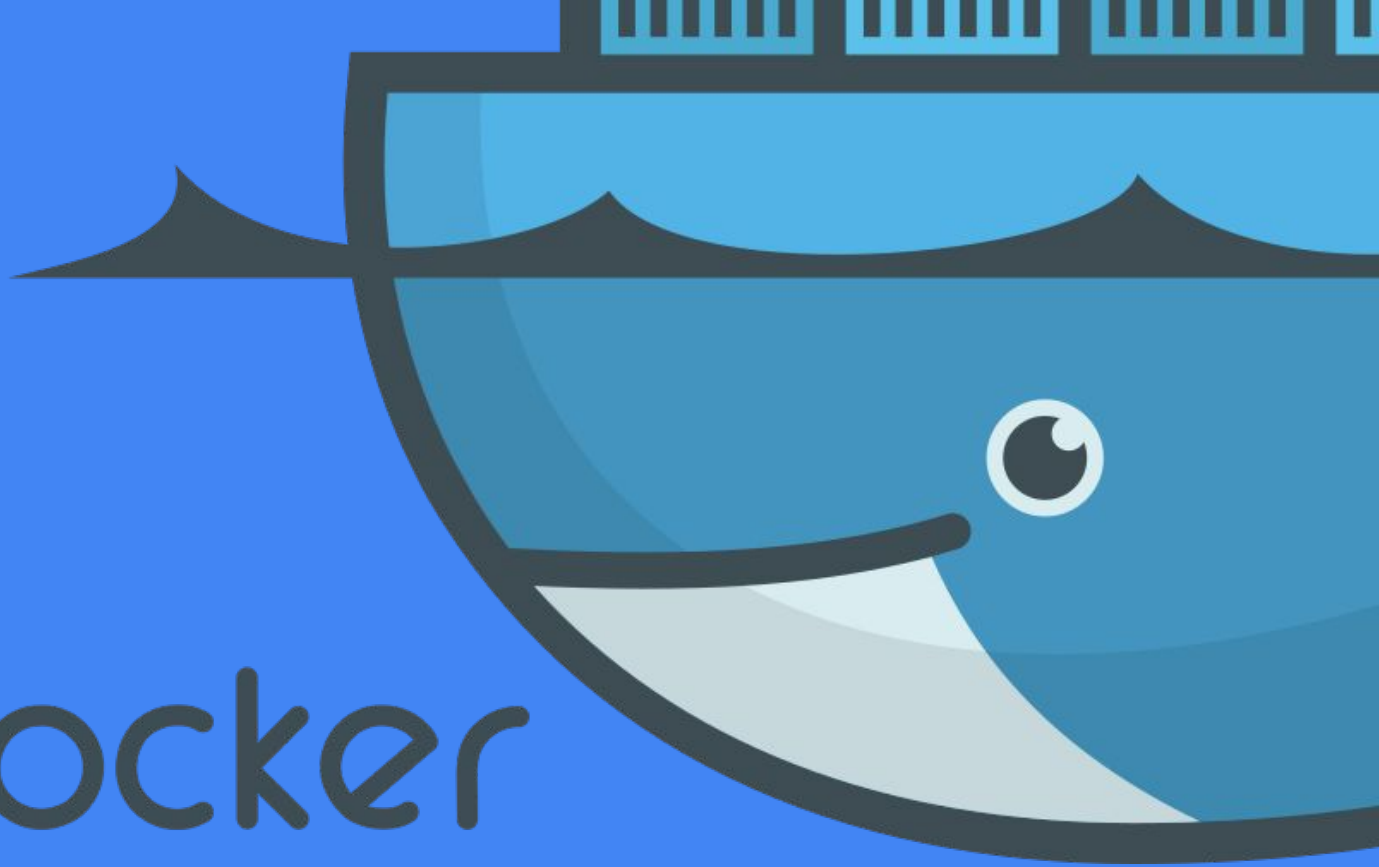


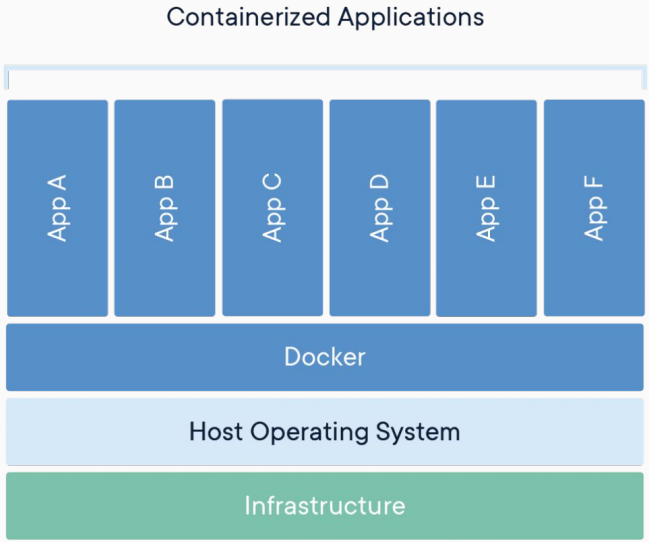
Módulo 1 - Trabalhando com containers

Por Fabio Szostak (2020)

O que é um container?

Resposta: **É uma unidade padronizada de software**

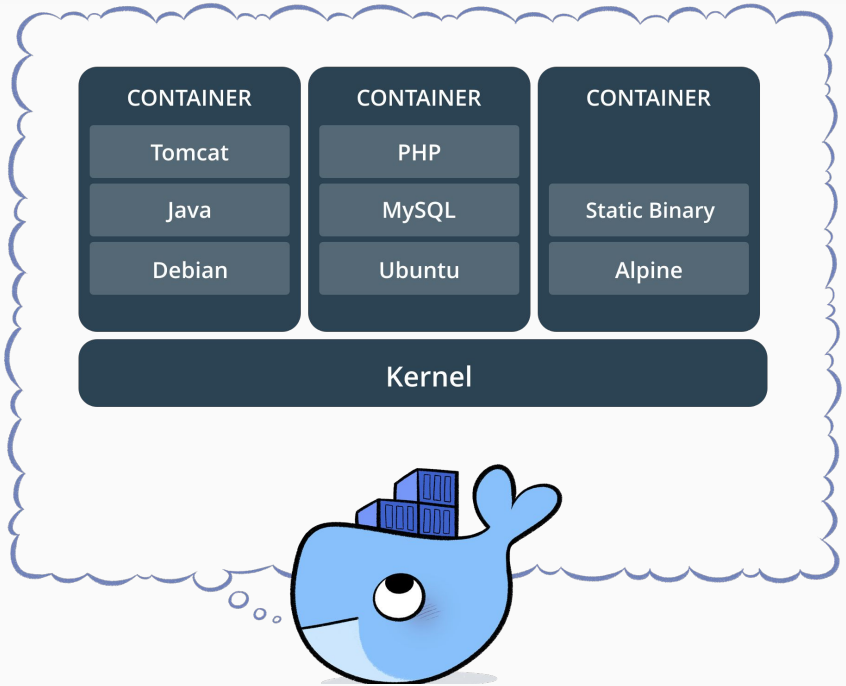
São utilizados para gerar um pacote de software em unidades padronizadas para desenvolvimento e implantação



Containers

A tecnologia do Docker é única porque se concentra nos requisitos de desenvolvedores e operadores de sistemas para separar dependências de aplicativos da infraestrutura.

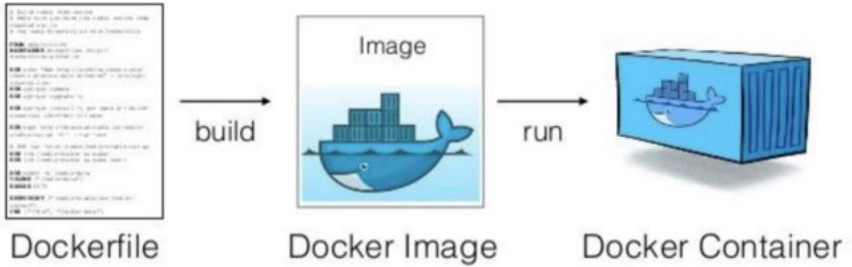
Um contêiner é uma instância de tempo de execução de uma imagem docker.



Containers

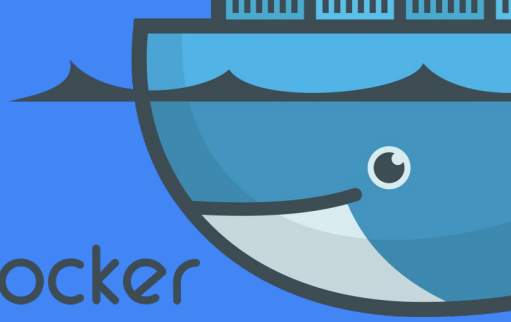
Um contêiner Docker consiste em

1) Uma imagem Docker
2) Um ambiente de execução
3) Um conjunto padrão de instruções



Docker define um padrão para entrega do software.

docker images

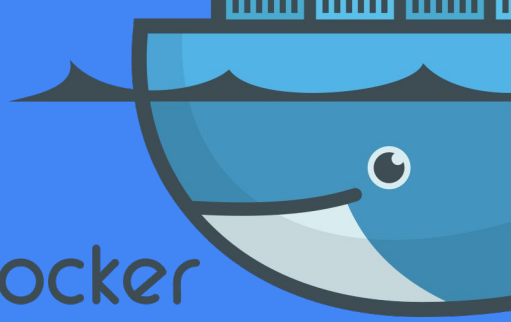


As imagens oficiais dos produtos são encontradas no Docker Hub.

O AWS ECR é onde registramos as imagens de nossas aplicações.

Sempre utilizar imagens recomendadas pela empresa.

docker run



O comando "docker run" irá criar uma instância de um container especificado.

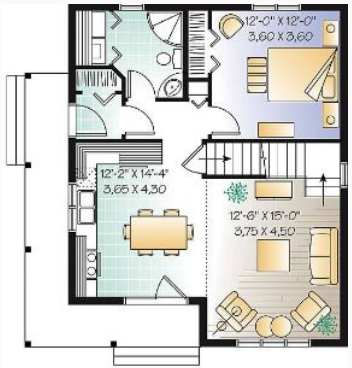
Criando uma instância de container Nginx

```
$ docker run -p 80:80 nginx:1.19
```

Container Instance

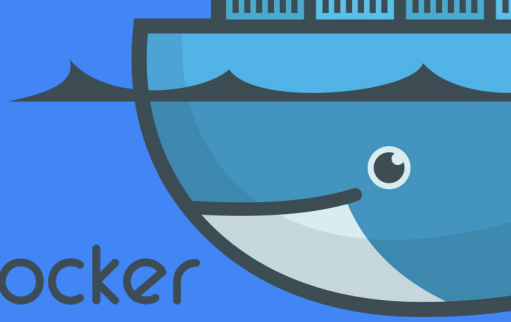
Podemos fazer uma analogia a construção de uma casa, onde o projeto seria a imagem e a casa construída seria a instância do container sendo executada.

Image



Container instance

docker build



Criando e executando a sua própria imagem

Criar arquivo "Dockerfile"

```
FROM nginx:1.19

RUN echo "Hello" > /usr/share/nginx/html/index.html
```

Criar imagem

```
$ docker build -t my-nginx:1 .
```

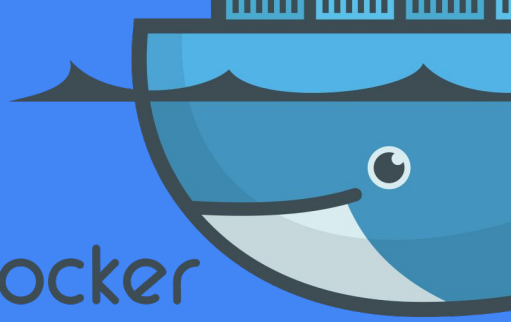
Criar instância

```
$ docker run -p 90:80 my-nginx:1
```

docker ports

O Docker criar uma rede interna para os containers e ela fica isolada, ou seja, os containers não ficam acessíveis a não ser que seja feito um mapeamento.

docker images



Com o comando "docker images" você conseguirá consultar as imagens que foram criadas e também as que foram baixadas em sua máquinas.

Cuidado com o consumo de espaço em disco

Listar imagens criadas

```
$ docker images
```

Remover imagem

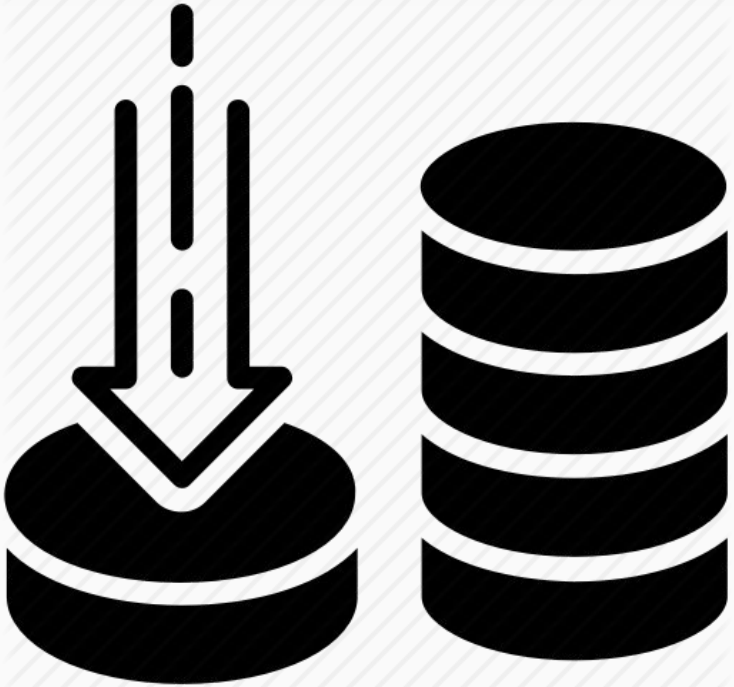
```
$ docker rmi my-nginx:1
```

Dica: Removendo somente as imagens que não estão sendo utilizadas.

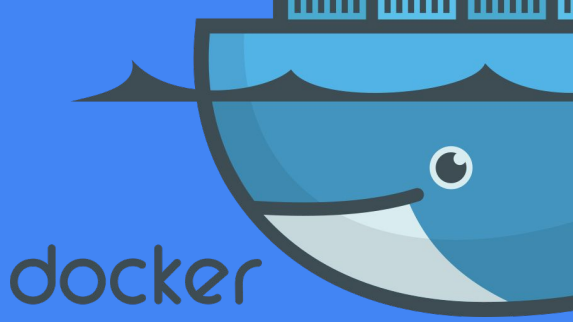
```
$ docker rmi $(docker images --filter
"dangling=true" -q --no-trunc) > /dev/null 2>&1
```

docker volume

Os volumes são áreas de armazenamento de arquivos. Eles podem ser internos ao Docker ou um compartilhamento com o armazenamento da máquina hospedeira.



docker volume



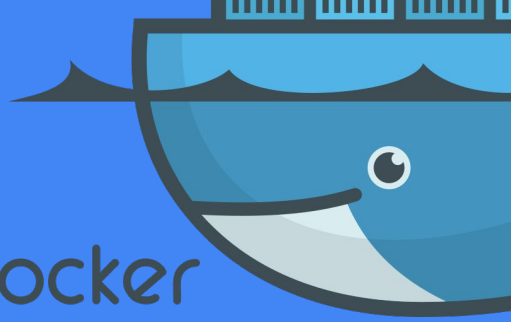
Conceito

É uma área para leitura/gravação de arquivos.

Você escutará o termo "montar o volume".

Isto significa que aquele volume será disponibilizado em uma determinada área do sistema de arquivos ("filesystem") da instância do container.

docker volume



Volume Compartilhado

Os arquivos serão compartilhados entre sua máquina e o container.

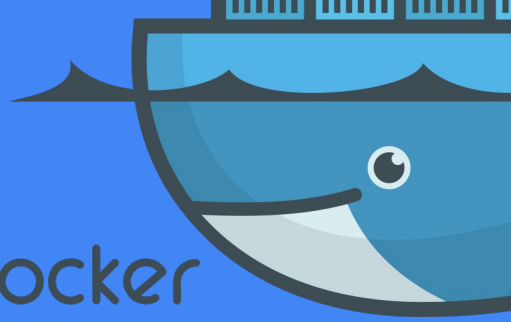
Criar instância montando um volume local

```
$ docker run -p 90:80 -v
$PWD/src:/usr/share/nginx/html my-nginx:1
```

No caso acima, irá disponibilizar ./src como um volume a ser compartilhado com container no diretório /usr/share/nginx/html

Tudo que você alterar em sua máquina local, irá refletir dentro do container.

docker volume



Volume Interno

Volumes internos podem ser criados, podemos utilizar o disco local ou até mesmo uma parte da memória RAM para acelerar algum processo. Eles podem ser compartilhados entre containers.

Lembrando que se os volumes removidos os dados serão perdidos.

Criar um volume para armazenar dados do mysql

```
$ docker volume create mysql-data
```

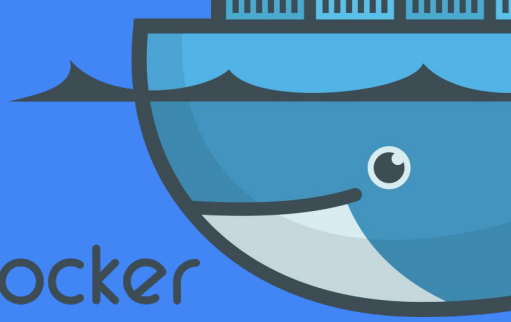
Executar container

```
$ docker run -p 3306:3306 -v
mysql-data:/usr/lib/mysql mysql:5.7
```

Executar container nginx montando /app com 1Mb de espaço

```
$ docker run --mount
type=tmpfs,destination=/app,tmpfs-size=1024000,tmpfs-mode=1775
nginx:1.19
```

Dockerfile RUN



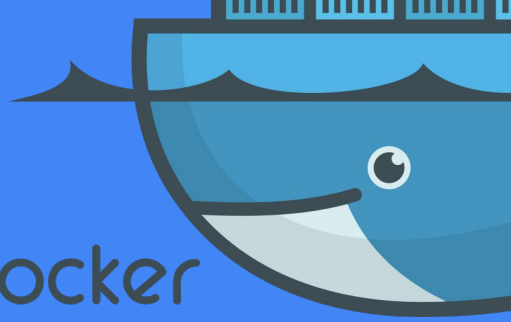
RUN

Executa comandos do sistema operacional da imagem base.

```
FROM nginx:1.19

RUN echo "<br/> Data no momento do build <br/>" >> /usr/share/nginx/html/index.html
RUN date >> /usr/share/nginx/html/index.html
```

O texto e a data corrente serão adicionados ao arquivo index.html

Dockerfile ARG / ENV



ARG

Argumento para ser utilizado durante o processo de build.

```
FROM nginx:1.19

ARG  environ=development

COPY ./default.${environ}.conf /etc/nginx/conf.d/
```

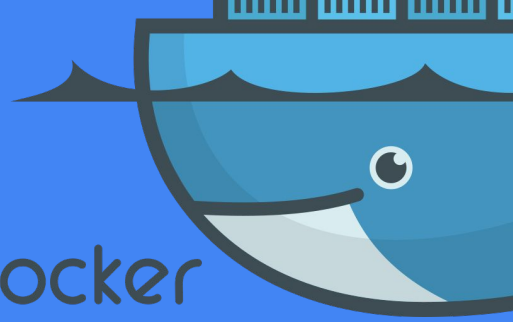
Caso o argumento não seja especificado para build, o valor "development" será atribuido como default.

ENV

Variáveis de ambiente para o sistema operacional.

```
FROM node:12-buster-slim

ARG  environ=development
ENV  NODE_ENV=${environ}

CMD [ "node", "server.js"]
```

Dockerfile COPY

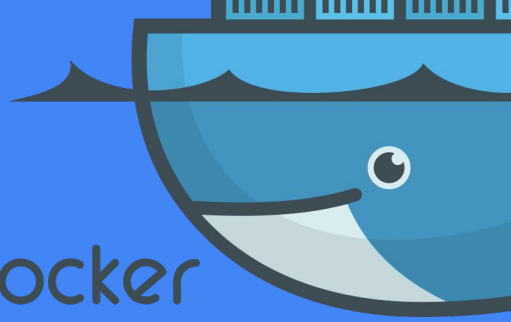


COPY

Copia arquivos do **contexto** corrente para dentro da imagem.

```
FROM nginx:1.19

COPY --chown=1000:1000 ./src/teste.html /usr/share/nginx/html
```

Dockerfile ADD

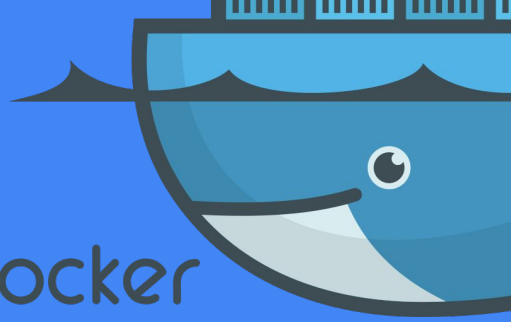


ADD

É utilizado para copiar/importar um arquivo no formato .tar.gz para dentro do container, ele descompacta o contéudo dentro do local especificado.

```
FROM nginx:1.19

ADD ./src/files.tar.gz /usr/share/nginx/html
```

Dockerfile WORKDIR

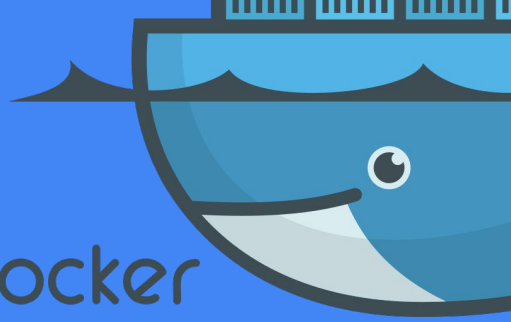


WORKDIR

Especifica o diretório de trabalho. O diretório será criado caso não exista. Após o comando o diretório corrente será o especificado.

```
FROM nginx:1.19

WORDIR /usr/share/nginx/html

COPY ./src/index.html .
ADD ./src/files.tar.gz .

RUN echo "<br/>Data no momento do build <br/>" >> ./index.html
RUN date >> ./index.html
```

Dockerfile CMD

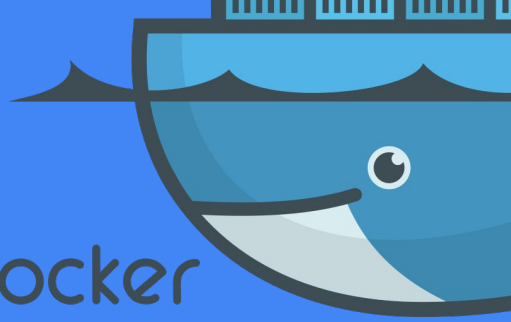


CMD

É o comando que será executado para iniciar o container, caso ele seja interrompido a instância do container será finalizada.

```
FROM nginx:1.19

WORDIR /usr/share/nginx/html

CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile ENTRYPOINT



ENTRYPOINT

Especifica script que será executado durante a inicialização da instância do container, ele será executado antes do CMD.

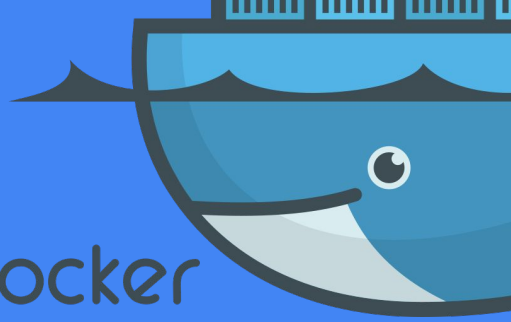
```
FROM nginx:1.19

COPY ./entrypoint.sh
/docker-entrypoint.d/my-entrypoint.sh
RUN chmod 775
/docker-entrypoint.d/my-entrypoint.sh

#ENTRYPOINT [ "/docker-entrypoint.sh" ]

CMD ["nginx", "-g", "daemon off;" ]
```

entrypoint.sh

```
#!/usr/bin/env bash

echo "<br/>Data no momento do ENTRYPOINT<br/>" >>
/usr/share/nginx/html/index.html
date >> /usr/share/nginx/html/index.html

exec "$@"
```

invoca o CMD

Dockerfile cache

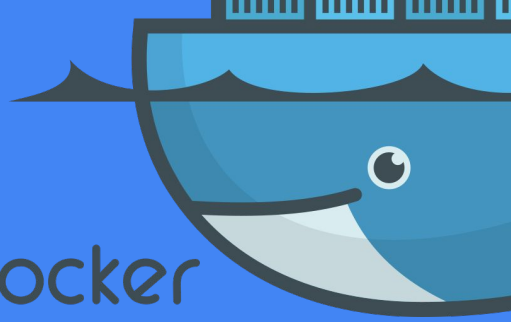


Procure deixar os arquivos mais modificados por último, isso irá reduzir o tempo de build de forma significativa.

somente irá executar o "yarn install" novamente se o arquivo "package.json" for modificado.

```
FROM    node:12-buster-slim
LABEL   maintainer="fabio.szostak@mirumagency.com"
ARG     environ=production
ENV     NODE_ENV=${environ}

WORKDIR /app

COPY    ./src/graphql/package.json /app
COPY    ./src/graphql/yarn.lock /app
RUN     yarn install

COPY    ./src/graphql/. /app/

CMD     [ "node", "./server.js" ]

EXPOSE  3000
```

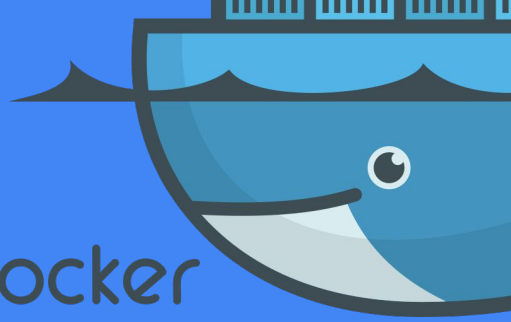
Dockerfile cache



Esta execução do build aproveitou todo cache da última execução, ou seja, nada foi alterado.

```
$ docker build --tag my-graphql:1.0 -f ./Dockerfile .
Sending build context to Docker daemon  12.73MB
Step 1/10 : FROM    node:12-buster-slim
 ---> f18da2f58c3d
Step 2/10 : LABEL   maintainer="fabio.szostak@mirumagency.com"
 ---> Using cache
 ---> 19475fc44e1d
Step 3/10 : ARG     environ=production
 ---> Using cache
 ---> fe0b424d1d38
Step 4/10 : ENV     NODE_ENV=${environ}
 ---> Using cache
 ---> 80a6e84ac386
Step 5/10 : WORKDIR /app
 ---> Using cache
 ---> 2d1848455fec
Step 6/10 : COPY    ./src/graphql/package.json /app
 ---> Using cache
 ---> ad02164e7d41
Step 7/10 : COPY    ./src/graphql/yarn.lock /app
 ---> Using cache
 ---> eb03ca2f4f9c
Step 8/10 : RUN     yarn install --production=true
 ---> Using cache
 ---> f612ad25bd9f
Step 9/10 : COPY    ./src/graphql/. /app/
 ---> Using cache
 ---> f5dcf3b09033
Step 10/10 : CMD     [ "node", "./server.js" ]
 ---> Using cache
 ---> 797a111e8ec1
Successfully built 797a111e8ec1
Successfully tagged my-graphql:1.0
```

Dockerfile cache

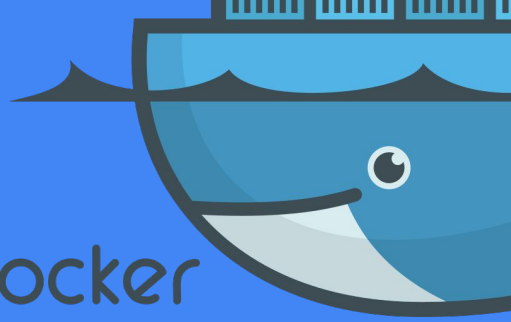


Simulamos que o package.json foi alterado, percebam o comportamento do cache.

O build re-executou todos os comandos a partir de onde houve uma modicação. Dai a importância de ordenar os comandos mais demorados e com pouco modificação e deixá-los no início.

```
Sending build context to Docker daemon  12.73MB
Step 1/10 : FROM    node:12-buster-slim
 ---> f18da2f58c3d
Step 2/10 : LABEL   maintainer="fabio.szostak@mirumagency.com"
 ---> Using cache
 ---> 19475fc44e1d
Step 3/10 : ARG     environ=production
 ---> Using cache
 ---> fe0b424d1d38
Step 4/10 : ENV     NODE_ENV=${environ}
 ---> Using cache
 ---> 80a6e84ac386
Step 5/10 : WORKDIR /app
 ---> Using cache
 ---> 2d1848455fec
Step 6/10 : COPY    ./src/graphql/package.json /app
 ---> 23b8c347b6dd
Step 7/10 : COPY    ./src/graphql/yarn.lock /app
 ---> 19f93535a9be
Step 8/10 : RUN     yarn install --production=true
 ---> Running in c9d64f4017f5
yarn install v1.15.2
[1/4] Resolving packages...
[2/4] Fetching packages...
```

Dockerfile Tips



Dica (1) - Quando precisar adicionar comandos ao entrypoint de uma imagem existente, adicione-os no final do arquivo.

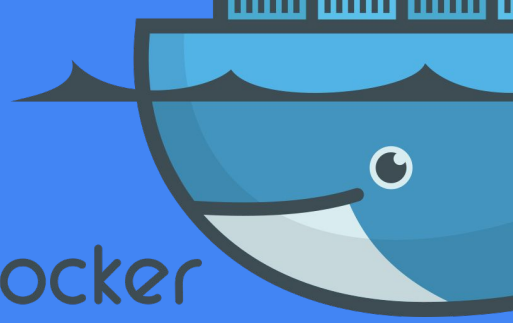
Dica (2) - *faça a instalação de bibliotecas em área temporária e faça a cópia no add-to-entrypoint.sh*

add-to-entrypoint.sh

```
# obtem node_modules gerado na imagem
cp -R /tmp/app/node_modules/ /app
```

```
FROM    node:12-buster-slim
LABEL   maintainer="fabio.szostak@mirumagency.com"
ARG     environ=development
ENV     NODE_ENV=${environ}
ENV     DOCKER_ENTRYPOINT=/usr/local/bin/docker-entrypoint.sh

# (1) add commands to docker-entrypoint.sh
COPY    ./docker/web/scripts/add-to-entrypoint.sh /
RUN     echo -n "$(cat ${DOCKER_ENTRYPOINT} /add-to-entrypoint.sh \
      | grep -v "exec \"\$@\"")\n\nexec \"\$@\"\n" \
      > ${DOCKER_ENTRYPOINT}

# (2) run yarn install and
#     keep into the container for fast startup
RUN     mkdir /tmp/app
COPY    ./src/graphql/package.json /tmp/app/
COPY    ./src/graphql/yarn.lock* /tmp/app/
RUN     cd /tmp/app/ && yarn install --silent --no-optional

WORKDIR /app

CMD     [ "yarn", "start" ]

EXPOSE  3000
```

docker tips



Alguns comandos que podem te ajudar no dia a dia.

Mostrar utilização de disco

```
$ docker system df
```

Fazer uma limpeza do docker

```
$ docker system prune -a
```

Remover todos o volumes

```
$ docker volume prune
```

Remove todas as imagens

```
$ docker image prune
```

Listar todos containers em execução

```
$ docker ps -a
```

Parar um container

```
$ docker stop <containerId>
```

Interromper a execução de container

```
$ docker kill <containerId>
```

Executar um comando dentro container

```
$ docker exec -it <containerId> sh
```

docker learn

Iniciando com Docker

https://docs.docker.com/get-started/

Saber mais sobre o significado dos termos

https://docs.docker.com/glossary/