Containers

O que é e qual a sua importância

Agenda

1. Quem sou eu
2. *Containers*
3. *Images*
4. *Container Registries*
5. Mãos na massa!
6. Considerações finais

Quem sou eu

Diego Ferreira
DevOps Engineer @ Blip.pt

28 anos, engenheiro eletrônico de formação pela UFPE e DevOps Engineer por paixão.

https://bit.ly/diegodevops

ada

Containers

Motivação

Como eu posso garantir que a minha aplicação **pode ser executada em qualquer** computador ou servidor de forma rápida, simples e leve?

Spoiler: criando um ambiente dedicado só pra ela!

ada

Containers

O que são? Onde vivem? De onde vem?
Para que servem?

- São semelhante a máquinas virtuais;
- São criados para um trabalhos específicos, com dependências específicas;
- Os containers “alugam” um espaço no sistema operacional do computador físico e compartilham vários recursos, com um isolamento menor.

Containers

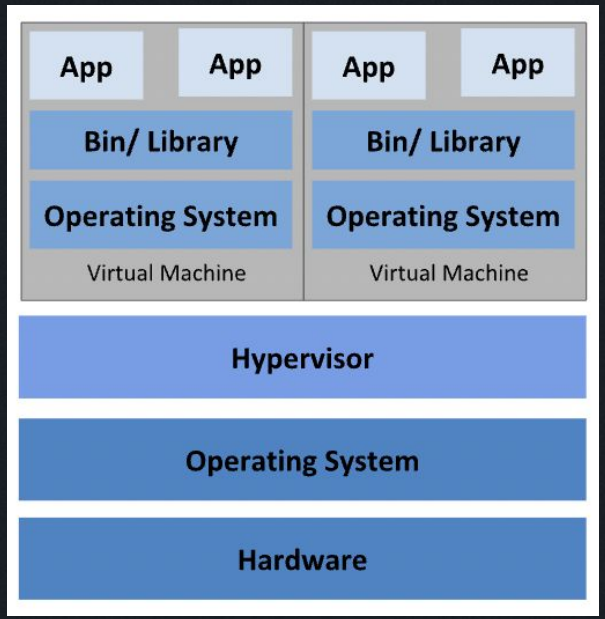
Mas e as máquinas virtuais?

- As máquinas virtuais são virtualizações completas de um computador a nível de hardware, software e até IO;
- As VMs são quase que totalmente isoladas do host principal, isso garante maior isolamento e segurança para elas;

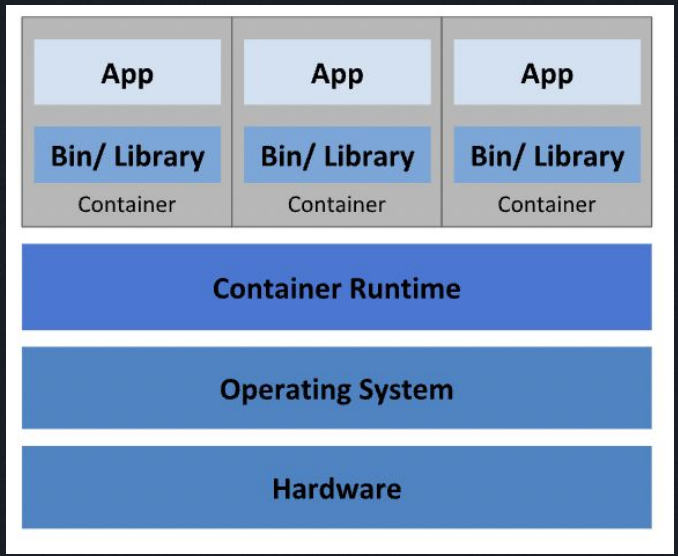
Containers

Máquina Virtual ou Container?

Máquina Virtual

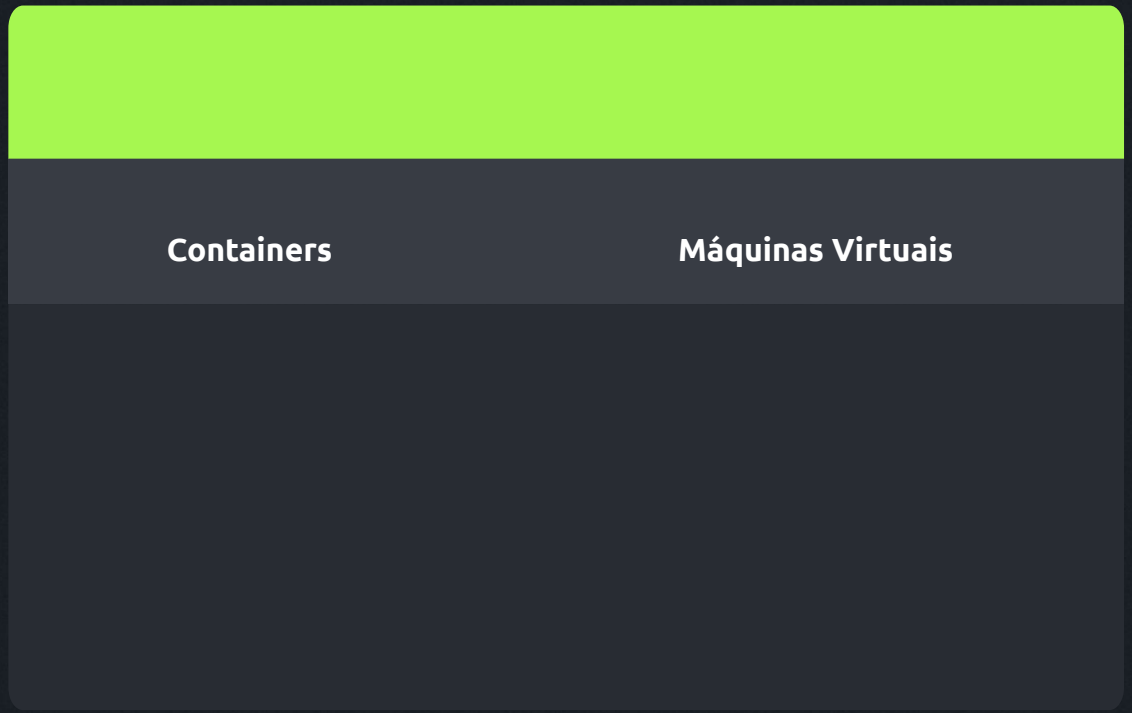


Container



Containers

Comparativo entre Máquinas Virtuais e Containers



Containers

Comparativo entre Máquinas Virtuais e Containers

Containers	Máquinas Virtuais
❌ Menor Isolamento	✅ Maior Isolamento
✅ Menor consumo de recursos	❌ Maior consumo de recursos
✅ Muito fácil de compartilhar ambientes	❌ Mais difícil de compartilhar ambientes
✅ Imagens mais leves, apenas com o essencial para executar o serviço	❌ Imagens pesadas, com coisas que podem não ser necessárias

Containers

Containers ≠ Docker

- Mas containers não se resumem só ao Docker… **tem muito mais por trás**.

https://www.mundodocker.com.br/o-que-e-docker/

Containers ≠ Docker

Container Engine

Docker, CRI-O, RKT e LXD

Container Daemon

containerd

Container Runtime

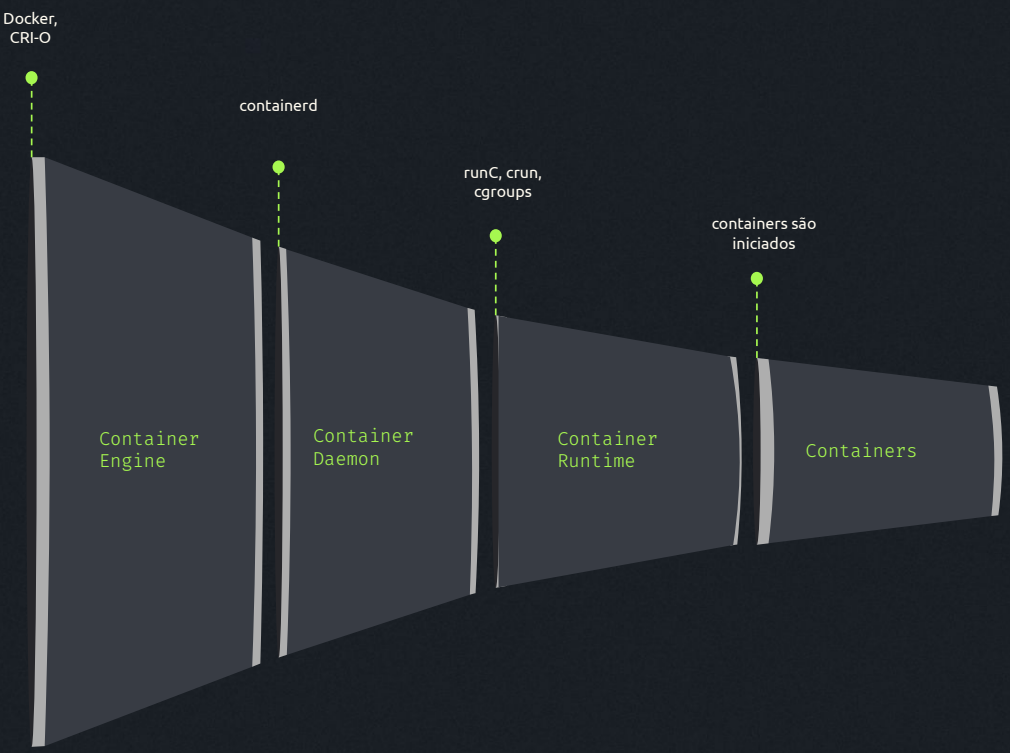
runC, cgroups, crun

Mas como isso tudo funciona?
São muitas opções e variações…

Containers

Containers ≠ Docker

```
docker container run
MEU_CONTAINER
```



Containers

Open Container Initiative (OCI), como proteger os containers

- *“The Open Container Initiative is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.” - The Linux Foundation*

- O OCI serve como um contrato que determina o modo com que os containers runtime e as imagens funcionam;
- Garante a compatibilidade e troca entre as várias implementações.

Empresas que apoiam o OCI

Fonte: https://opencontainers.org/

ada

Images

Imagens

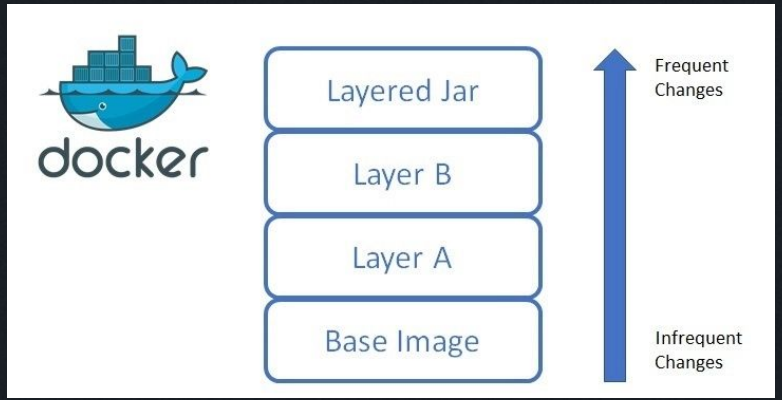
O que são as imagens

- As imagens podem servem como a base dos containers;
- Imagens funcionam como as ISOs para instalação de sistemas operacionais, servem de template para criar containers;
- Os containers são criados a partir de imagens e é possível personalizá-los a partir de uma base sólida e confiável.

Imagens

Layers e qual a sua importância

- As imagens são divididas em *layers* ou camadas e servem para facilitar o build de novas imagens;
- Cada *layer* é criado com base na anterior, através de um *diff;*
- Desse modo, quando a imagem é recriada, **apenas o que mudou efetivamente é atualizado, acelerando o processo de criação.**

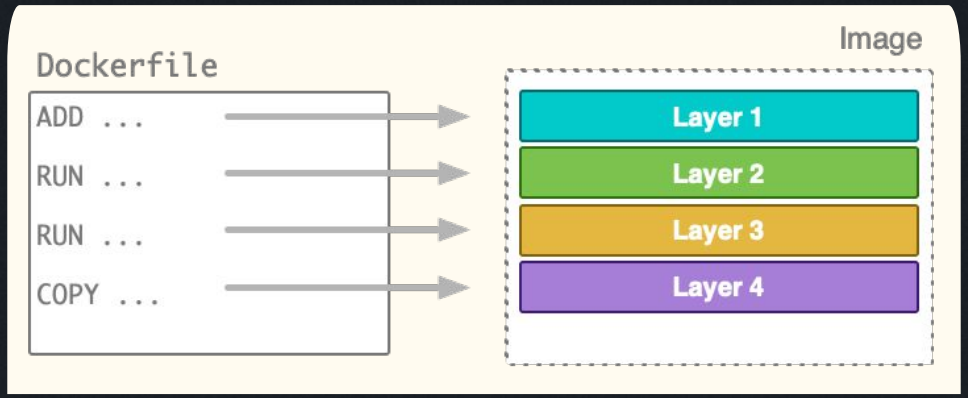


https://www.baeldung.com/docker-layers-spring-boot

Imagens

Dockerfile e a sua importância

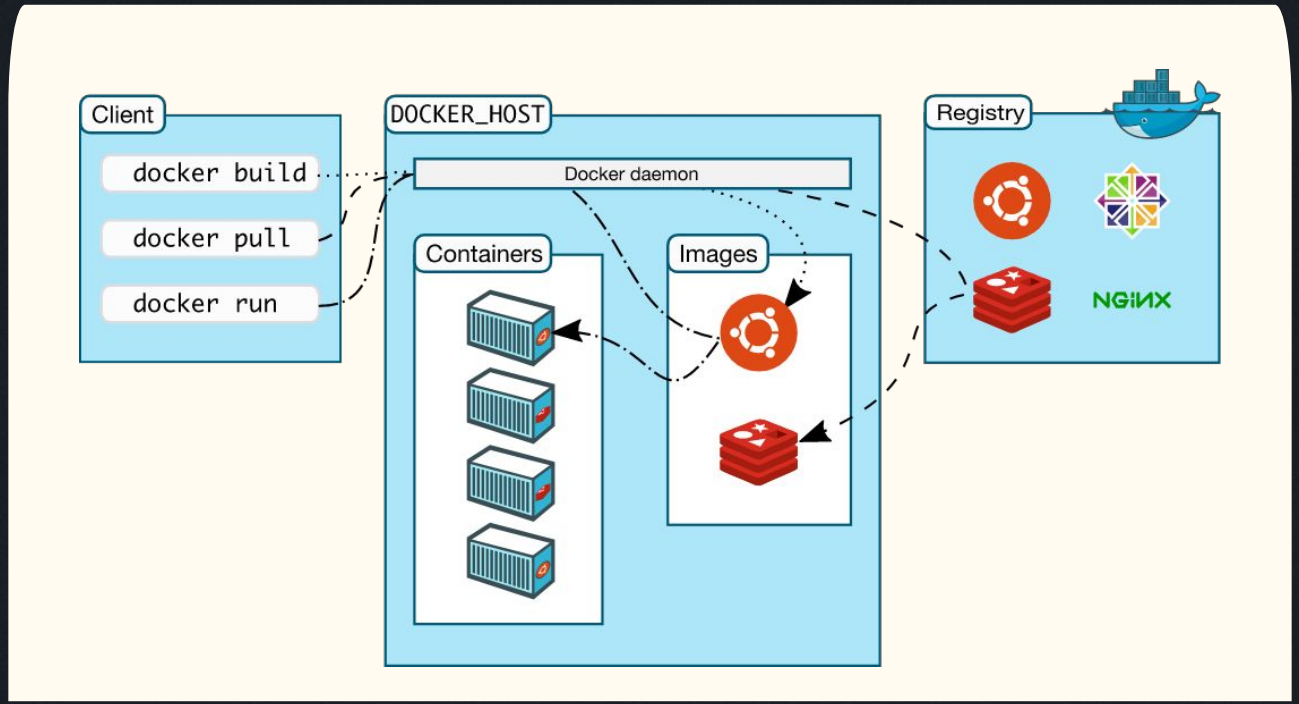
- O Dockerfile é um arquivo declarativo que serve para criar uma imagem docker “em etapas” a partir de comandos pré definidos
- É possível executar comandos de:
  - Copiar arquivos;
  - Instalar dependências;
  - Executar comandos;
  - Expor portas.
- Ou seja, **tudo que é necessário para preparar um ambiente para a execução de uma aplicação!**



Container Registries

Container Registries

Imagens, registry e containers: como interagem?



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

Container Registries

Armazenando e compartilhando a sua Imagem

- Os container registries servem para compartilhar e armazenar imagens;
- Eles podem ser de dois tipos:
  - **Públicos**: Docker Hub;
  - **Privados**: Amazon ECR, Microsoft Azure Containers, etc.

Imagens proprietárias podem e é fortemente recomendado que sejam armazenadas em **registries privados!**

Container Registries

Como usar os Registries a seu favor

- **Imagens de aplicações prontas:**
  - NGINX
  - Postgres
  - MySQL
  - Redis
- **Imagens base:**
  - OpenJDK
  - Python
  - Node
- Um uso muito comum é construir a imagem da sua aplicação a partir de imagens base (com um Dockerfile) e, então, fazer upload para um registry privado.

Mãos na massa!

Mãos na massa

99%

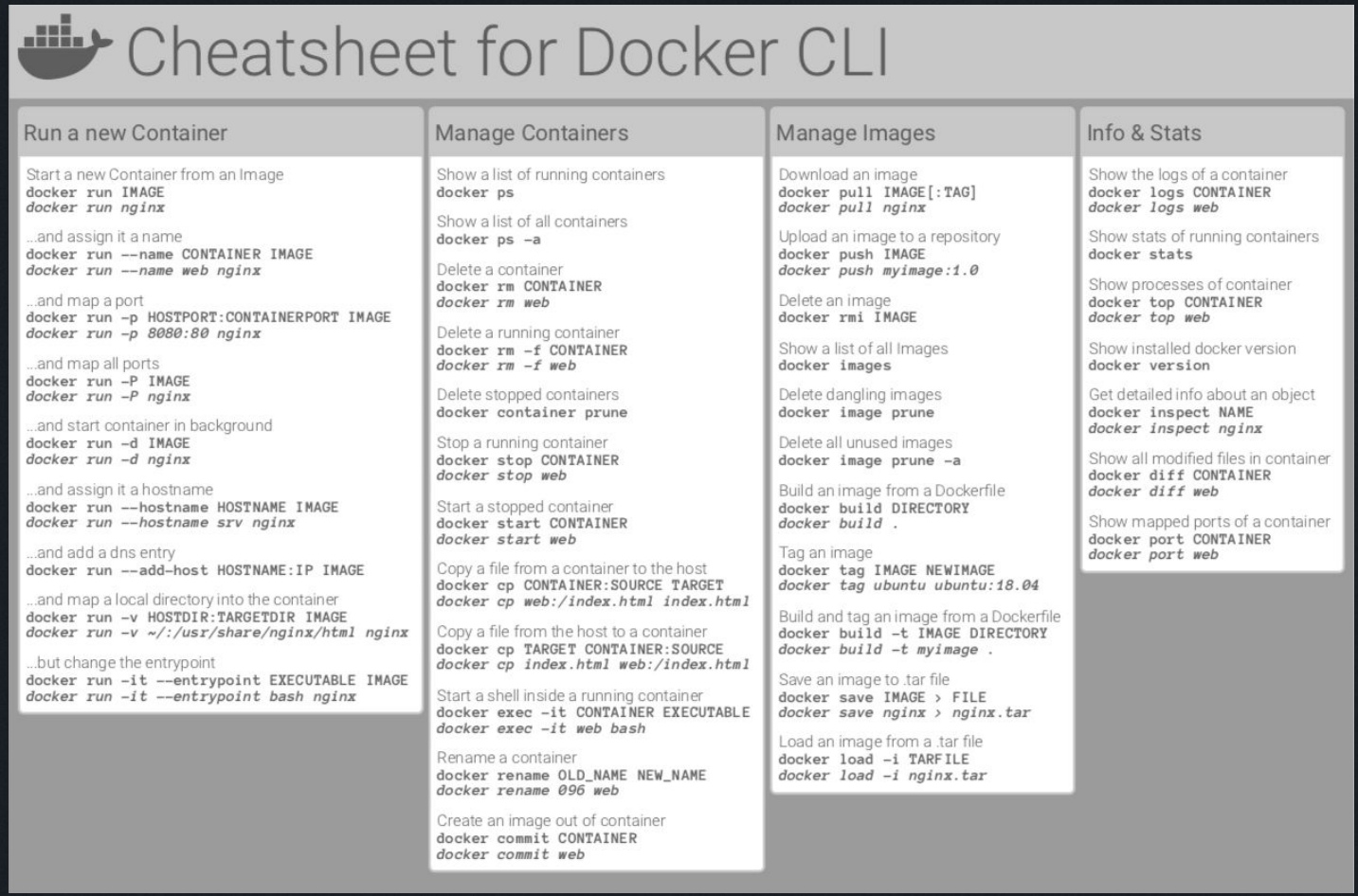
código: https://github.com/desferreira/meetup-containers

ada

Considerações finais

Q&A

Mãos na massa



Cheatsheet for Docker CLI

Run a new Container

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

Manage Images

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

Info & Stats

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

ada

Mãos na massa

Inheritance

```
FROM ruby:2.2.2
```

Run commands in strict shell

```
ENV my_var
SHELL ["/bin/bash", "-euo", "pipefail", "-c"]

# With strict mode:
RUN false # fails build like using &&
RUN echo "$myvar" # will throw error due to typ
RUN true | false # will bail out of pipe
```

Using `shell` will turn on strict mode for shell commands.

Metadata

```
LABEL version="1.0"
```

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
```

```
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

Variables

```
ENV APP_HOME /myapp
RUN mkdir $APP_HOME
```

```
ARG APP_HOME=""
RUN mkdir $APP_HOME
```

Onbuild

```
ONBUILD RUN bundle install
# when used with another file
```

Entrypoint

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

Configures a container that will run as an executable.

```
ENTRYPOINT exec top -b
```

This will use shell processing to substitute shell variables, and will ignore any `CMD` or `docker run` command line arguments.

Initialization

```
RUN bundle install
```

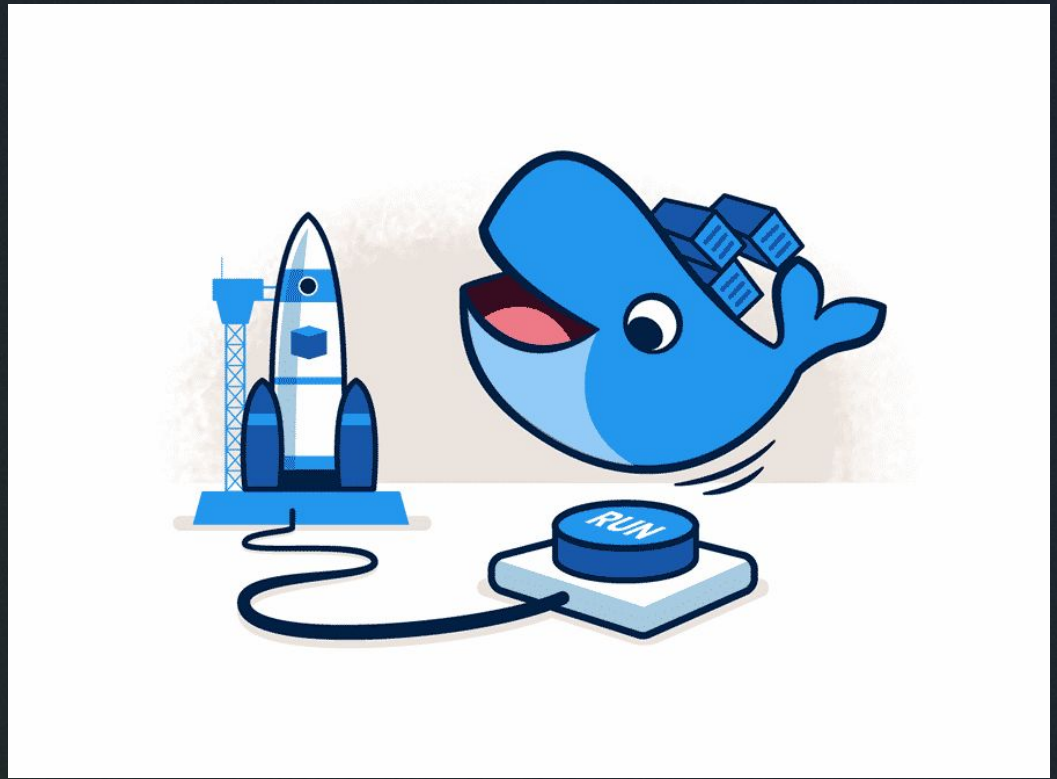
```
WORKDIR /myapp
```

```
VOLUME ["/data"]
# Specification for mount point
```

```
ADD file.xyz /file.xyz
COPY --chown=user:group host_file.xyz /path/con
```

Commands

```
EXPOSE 5900
CMD    ["bundle", "exec", "rails", "server"]
```



Obrigado!