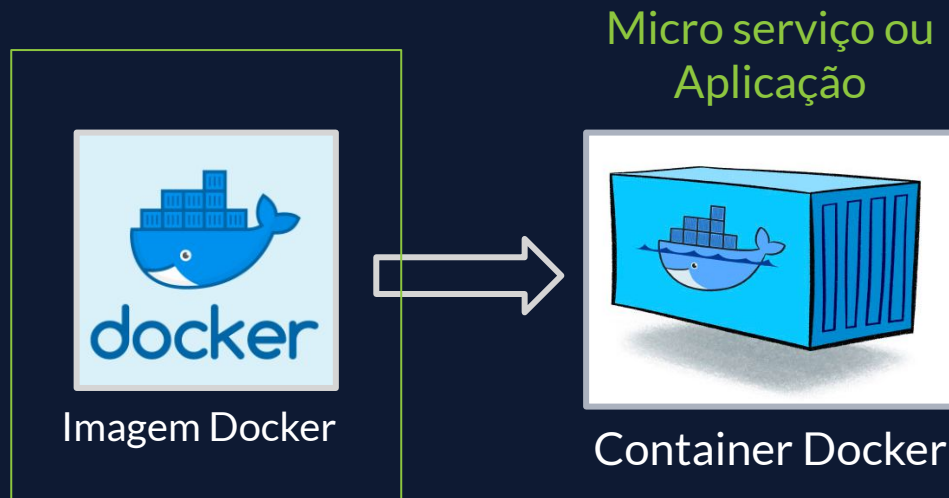




**Como construir imagens de forma efetiva!**

# O que é uma imagem Docker?

Imagens Docker são compostas por sistemas de arquivos de camadas que ficam uma sobre as outras. Ela é a nossa base para construção de uma aplicação, uma imagem pode ser baseada em diversos sistemas operacionais, como: Debian, Ubuntu, RedHat, CentOS, Alpine e entre outros.



# Entendendo as camadas de uma imagem

A primeira camada da imagem é RW, ou seja uma camada que você pode escrever.

E as demais camadas ?

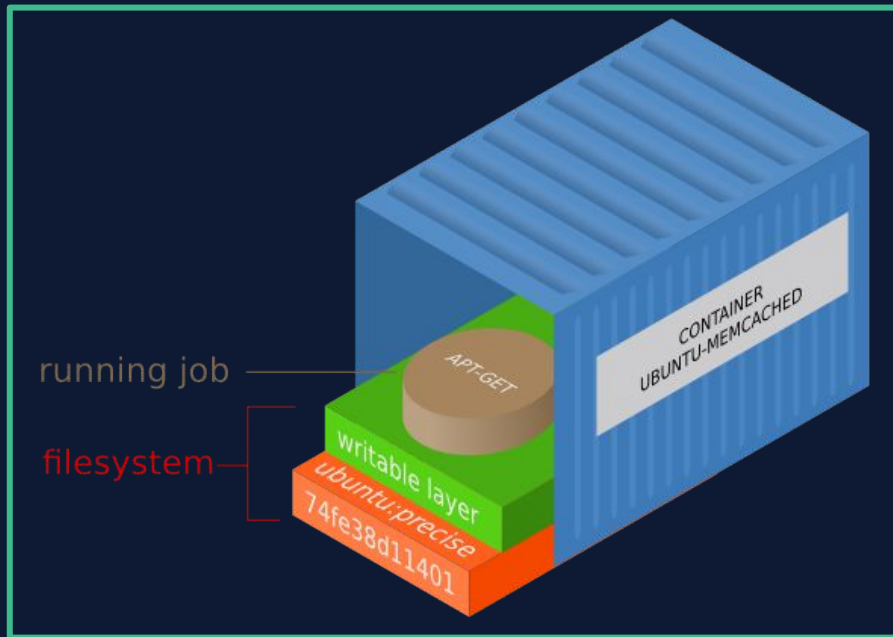
As demais camadas são somente RO, (Somente Leitura).

Legal mas onde você quer chegar com isso?

Se eu tiver um arquivo de 100MB, na camada mais baixa e remover este arquivo, ele não será removido.

Porque?

Se eu tiver 100 containers de 1GB utilizando a mesma imagem em um nó do cluster, quantas gigas estarei utilizando neste nó?



# Construindo uma imagem.

## Dockerfile

Para construir uma imagem é necessário realizar o [Build] através de um Dockerfile.  
Exemplo de Dockerfile:

```
FROM python:2.7.15-slim
WORKDIR /app
COPY crons /app/
RUN apk update --no-cache \
&& apk add tzdata \
&& ln -sf /usr/share/zoneinfo/America/Sao_Paulo /etc/localtime \
&& ln -sf /app/crons /etc/crontabs/root

ENTRYPOINT [ "crond" ]
CMD [ "-f" ]
```

```
$ docker image build -t cron:1.0 .
```





# CONSTRUINDO UM DOCKERFILE DE RESPOSTA!

# UTILIZE MULTI-LINE!

```
RUN apt-get update && \  
apt-get install git \  
python \  
java \  
e por aí vai...
```

Diferença de [&&] e [;].

- && Só executa se o comando anterior não retornar erro.
- ; Executa o comando e logo em seguida o próximo.

# LIMPE A CASA! 🧹

```
RUN apt-get update && \  
apt-get install -yq <pacote1> \  
<pacote2> \  
<pacote3> && \  
apt-get clean && rm -rf /var/lib/apt/lists/*
```

# LIMPE A CASA!

ENV MRAVERSION v1.1.0

RUN git clone

<https://github.com/intel-iot-devkit/mraa.git> && \

git checkout -b build \${MRAVERSION} && \

<some build steps> make install && \

cd .. && rm -rf mraa



# INSTALE SOMENTE O NECESSÁRIO

NADA DE INSTALAR O CURL, NETTOOLS, VIM,  
TELNET, E POR AÍ VAI... 🥲

Porque somente os pacotes necessários?

- Aumenta vulnerabilidades
- O tamanho da imagem
- Administração
- Padrão

# USE O RUN COM MODERAÇÃO!

```
RUN apt-get update && \  
apt-get install git \  
python \  
java \  
e por aí vai...
```

A cada RUN executado uma nova camada é criada,  
então use com moderação. 😊

# USE O **.DOCKERIGNORE**

Se não quer jogar “lixo” dentro do container utilize o **.dockerignore**, tem a mesma função do **.gitignore**.

# UTILIZE UM USUÁRIO NON-ROOT


Um exemplo é se você utilizar um webserver em container utilize o usuário `www-data/nginx` e assim por diante. Utilize um usuário diferente de root!

**CMD** é utilizado depois que o processo do container é “startado”.

Podemos utilizar das duas formas:

**CMD** [“nginx” , “parametro1” , “parametro2”] - **EXEC FORM**  
**CMD** nginx parametro1 parametro2 - **SHELL FORM**

# ENTRYPOINT

**ENTRYPOINT** é como se fosse o processo init do Linux, ele é utilizado para “segurar” o principal processo do container, se o processo morrer o container morre! 

Também podemos utilizar das duas formas!

**ENTRYPOINT** [“nginx” , ”parametro1” , “parametro2”] - EXEC  
**ENTRYPOINT** nginx parametro1 parametro2 - SHELL

# ENTRYPOINT E CMD NO MESMO DOCKERFILE

```
ENTRYPOINT ["nginx"]  
CMD ["-d"]
```

- Se eu coloco o **CMD** ele vira parâmetro do meu entrypoint!
- Se estivermos utilizando os dois juntos precisa ser no padrão EXEC FORM.

# SCRIPT DE START

```
ENTRYPOINT["python", "./script.py"]
```

Não tem problema utilizar script para solucionar um problema, mas sempre melhore!



# SHELL

SHELL [“comando”, “-parametro”]

O shell representa o [sh -c]

```
ENV app_dir /opt/app  
WORKDIR ${appdir}  
ADD . $workdir
```

Além de utilizar variáveis para dentro do container, utilize no Dockerfile pra ficar mais clean!

# LABELS

Utilize o label sempre que for inserir um rótulo no container.

```
LABEL "Mantainer"="Gerson Carneiro"
```

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

```
LABEL description="Este texto ilustra que os valores \  
das labels podem abranger várias linhas"
```

# QUAL UTILIZAR? COPY OU ADD?

Os dois fazem basicamente a mesma coisa, massssssss!

Com o copy eu posso copiar tudo que está no diretório, diretórios ou arquivos.

```
COPY ./app
COPY dir1 /app
COPY arq1 /app
```

O ADD eu consigo fazer tudo que o COPY faz, porém consigo copiar o conteúdo de um arquivo empacotado ou um arquivo em uma URL.

```
ADD ./app
ADD artefato.tar /app
ADD http://dns.com/arquivo /app
```



```
FROM alpine
ARG HTTP_PROXY
RUN echo $HTTP_PROXY
```

```
$ docker image --build-arg HTTP_PROXY=http://10.20.30.2:1234 -t app:1.0 .
```

# MULTI-STAGE

```
FROM golang:alpine AS build
ADD ./src
RUN cd /src && go build -o teste
```

```
FROM alpine
WORKDIR /app
COPY --from=build /src/teste /app
ENTRYPOINT ./opa
```

É uma maneira de você fazer uma “pipeline” dentro do seu Dockerfile.

Se retirar o cache na hora do build, pode ter certeza que irá demorar muito mais para baixar a imagem.

No Docker podemos ou não utilizar o cache!

Forçar para não utilizar o cache.

```
$ docker image build -t cron:1.0 . --no-cache
```

# UTILIZE IMAGENS PEQUENAS

Sempre que possível utilize imagens pequenas!

alpine  
ubuntu-slim  
busybox

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	2c891f81f76d	About a minute ago	21.4MB
ubuntu	latest	a9e51dc72612	17 minutes ago	212MB



# HABILITE BUILDKIT E O MODO DEBUG!

## Windows

Clique com o botão direito do mouse no ícone do Docker na barra de tarefas do Windows e clique em Settings > Docker Engine.



Em seguida clique em *Apply & Restart*, para aplicar as configurações!

## Linux

\$ vim /etc/docker/daemon.json

```
{
  "experimental": false,
  "debug": true,
  "features": {"buildkit": true}
}
```

Salve e saia do arquivo: *ESC :wq*

Reinicie o serviço do Docker:  
\$ systemctl restart docker.service

# VISUALIZANDO O BUILD

Vantagens da utilização do Buildkit:

- Visualizar o tempo do build
- Visualizar por camadas

Vantagens de ativar o modo debug:

- Caso dê problemas irá informar se problema de proxy, conexão ou qualquer outro problema.

```
gerson@gerson-dell /app/workspace/ master ± docker image build -t report-python:1.0 .
[+] Building 3.6s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 552B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/python:2.7.18-alpine3.11
=> [1/4] FROM docker.io/library/python:2.7.18-alpine3.11@sha256:724d0540eb56ffaa6dd770aa13c3bc7dfc829dec561d87cb36b2f5b9ff8a760a
=> [internal] load build context
=> => transferring context: 13.11kB
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY files crons /app/
=> [4/4] RUN apk update --no-cache && apk add tzdata && ln -sf /usr/share/zoneinfo/America/Sao_Paulo /etc/localtime && ln -sf /app/crons /etc/crontabs/root
=> exporting to image
=> => exporting layers
=> => writing image sha256:3b0e908e7f7af5a039abc5aca5bbfb8e8b5c66a0392251b288ca8251346e3719
=> => naming to docker.io/library/report-python:1.0
```

# DEPOIS DESSA CALL

## Cenário real

Antes

```
FROM adoptopenjdk/openjdk11:jdk-11.0.6_10-ubuntu-slim

ENV LANG C.UTF-8

appdynamics
RUN mkdir -p /opt/appdynamics/

COPY /appdynamics/* /opt/appdynamics/

ENV /
ENV /
ENV /
ENV /
ENV /
ENV /

#Entrega de certificados na imagem
RUN apt-get update && apt-get install ca-certificates && rm -rf /var/cache/apk/*

RUN mkdir -p /usr/local/share/ca-certificates/
COPY /usr/local/share/ca-certificates/
COPY /usr/local/share/ca-certificates/
COPY /usr/local/share/ca-certificates/
COPY /usr/local/share/ca-certificates/

RUN update-ca-certificates

RUN
RUN
RUN
RUN

#var de caminho .jar
local
ARG JAR_FILE=/itau-rda-application/target/*.jar
COPY $JAR_FILE app.jar

COPY ${JAR_FILE} app.jar

EXPOSE 8080

local
CMD ["java", "-jar", "-Dspring.profiles.active=local", "app.jar"]
ENTRYPOINT exec java $JAVA_OPTS -jar app.jar
```

Depois

```
FROM adoptopenjdk/openjdk11:jdk-11.0.6_10-ubuntu-slim

ENV LANG C.UTF-8

appdynamics
RUN apt-get update && \
apt-get install ca-certificates && \
rm -rf /var/cache/apk/* && \
mkdir -p /opt/appdynamics && \
mkdir -p /usr/local/share/ca-certificates/

ADD /appdynamics/* /opt/appdynamics/

ENV
ENV
ENV
ENV
ENV

#Entrega de certificados na imagem
ADD cert.usr /usr/local/share/ca-certificates/
RUN update-ca-certificates && \
rm -rf /var/cache/apk/*

ARG JAR_FILE=*.jar
COPY ${JAR_FILE} app.jar

EXPOSE 8080
ENTRYPOINT exec java $JAVA_OPTS -jar app.jar
```

Whaomi



# Gerson Carneiro

SRE / Esferas



[gerson.carneiro@zup.com.br](mailto:gerson.carneiro@zup.com.br)



[\(11\) 99748-1905](tel:(11)99748-1905)



[www.zup.com.br](http://www.zup.com.br)



