# *Docker*: o que é e para que serve

Luiz Sol e Marcos Vinicius

2019-03-11

# O problema

A pessoa *A* desenvolveu um *pipeline* de *Machile Leearning* em Python no seu computador pessoal.

O *setup* da pessoa *A* é:

- Majaro 17.1 (Arch Linux)
- Python 3.5
- PostgreSQL 10.3 (Banco de Dados)
- PyTorch 0.9
- Pandas 0.14

Agora a pessoa $B$ precisa revisar o código (ver se foi bem escrito e se funciona corretamente).

O *setup* da pessoa *B* é:

- Windows 10
- Python 3.7
- MySQL 5.7 (Banco de Dados)
- TensorFlow 0.9
- Pandas 0.21

E uma vez que o código for aprovado ele deverá ser colocado no servidor de produção $C$ (máquina que irá rodar de fato a aplicação).

O *setup* do servidor *C* é:

- RHEL 7.6 (Red Hat Linux)
- Python 3.6
- Cassandra 3.11 (Banco de Dados)
- TensorFlow 0.7
- Pandas 0.26

O Pesquisador *D* está tentando comparar a diferença de performance entre duas versões diferentes da JVM (Java), mas não quer ter que desinstalar e instalar cada versão para cada teste que precisar realizar.

E agora José?

Temos um problema de reproducibilidade e isolamento de ambientes de execução.

# Possíveis soluções:

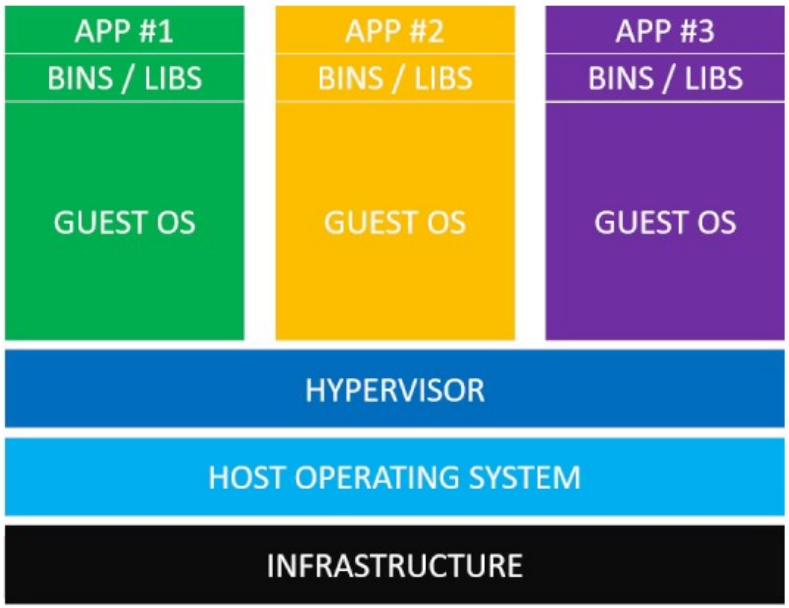
- **Solução 1**: Obrigar todos a utilizarem os mesmos softwares que estão no servidor de produção
- **Problemas da solução 1**:
  - Servidores tendem a utilizar versões antigas e estáveis de software, o que pode atrapalhar os pesquisadores
  - Nem sempre é possível usar para desenvolvimento o que se usa em produção (licenças, interface com o usuário, demanda computacional etc)
  - Restringir pesquisadores e desenvolvedores pode resultar na fuga de capital humano qualificado

- **Solução 2**: Utilizar *máquinas* virtuais que espelhem o setup de produção
- **Problemas da solução 2**:
  - Máquinas virtuais são grandes (~10GB) e consomem bastante memória (~6GB) por si só
  - Versionamento (controle de versões) de máquinas virtuais não é uma tarefa simples (arquivos binários)
  - A interação entre a máquina hopedeira e a máquina hóspede (máquina virtualizada) nem sempre é simples (arquivos, rede, *clipboard* etc)

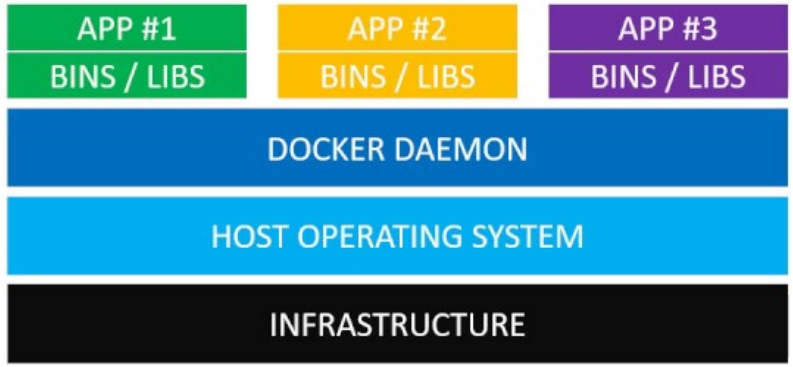
# Docker ao resgate

O *Docker* tenta solucionar este problema criando "máquinas virtuais" compactas, flexíveis e reutilizáveis.

O *Docker* utiliza o próprio *kernel* (a "base") do sistema operacional para executar as aplicações das máquinas virtualizadas.




Virtual Machines




Docker Containers

Os pricipais conceitos do *Docker* são:

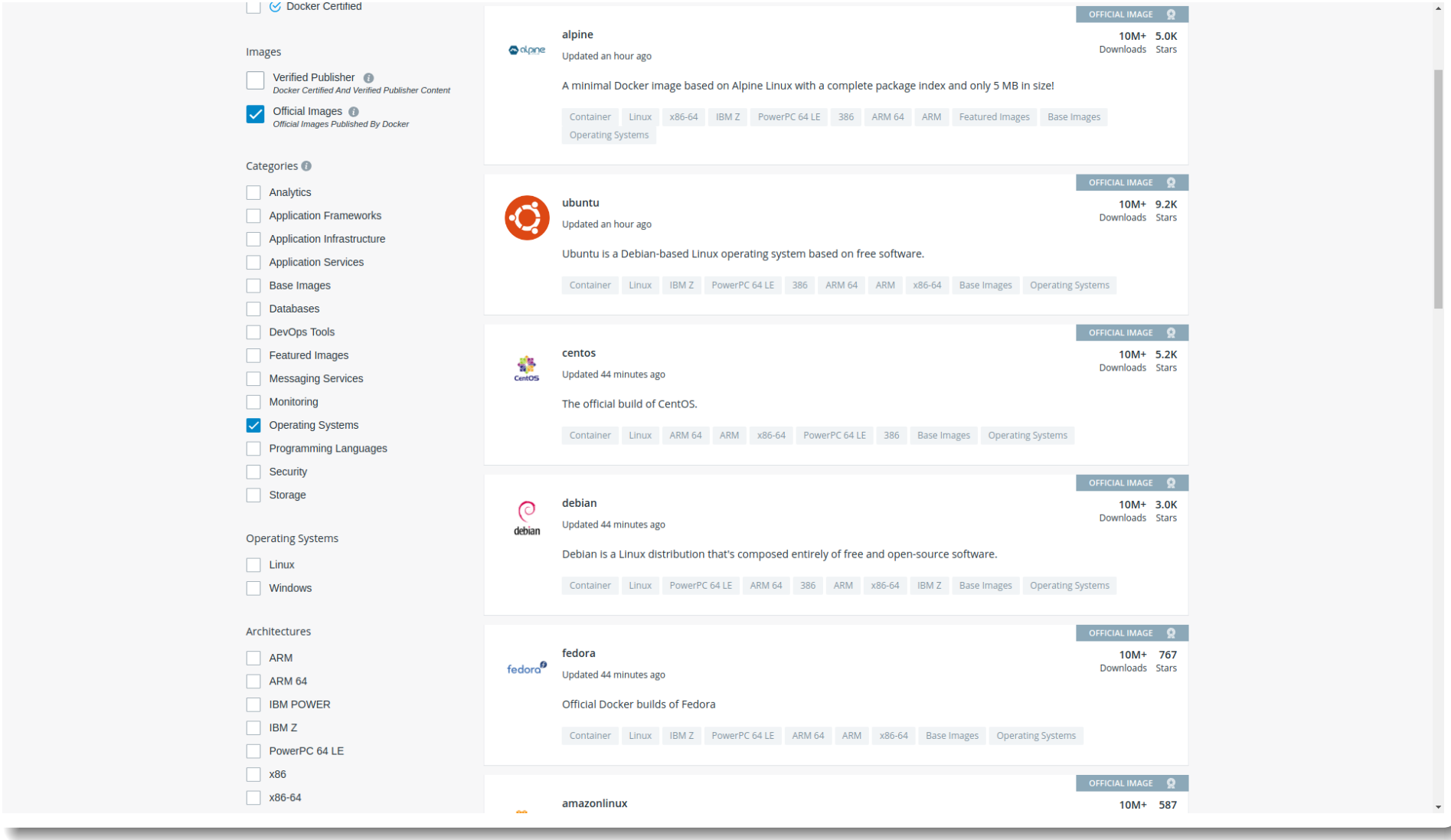
- **Imagens**: instruções de como construir os containers
- ***Containers***: as "máquinas virtuais" já construídas a partir dos containers
- **Volumes**: pastas no computador hospedeiro que o container irá utilizar para armazenar arquivos permanentemente
- **Redes**: as sub-redes das quais os containers farão parte

# Imagens

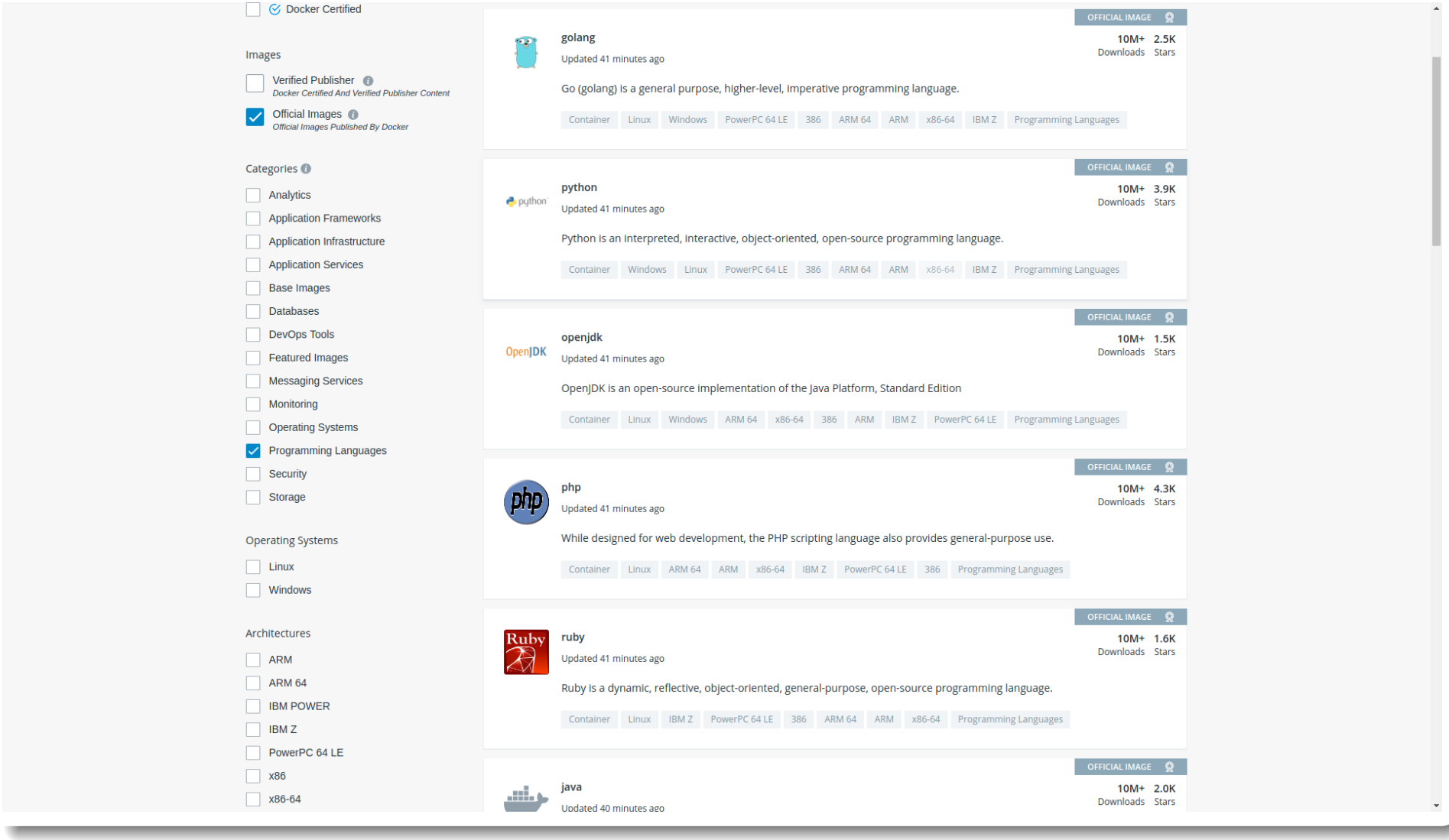
São as “plantas” que serão utilizadas para construir os *containers*

São contruídas a partir de outras imagens que estão disponíveis no repositório público do Docker (entitulado Docker Hub)

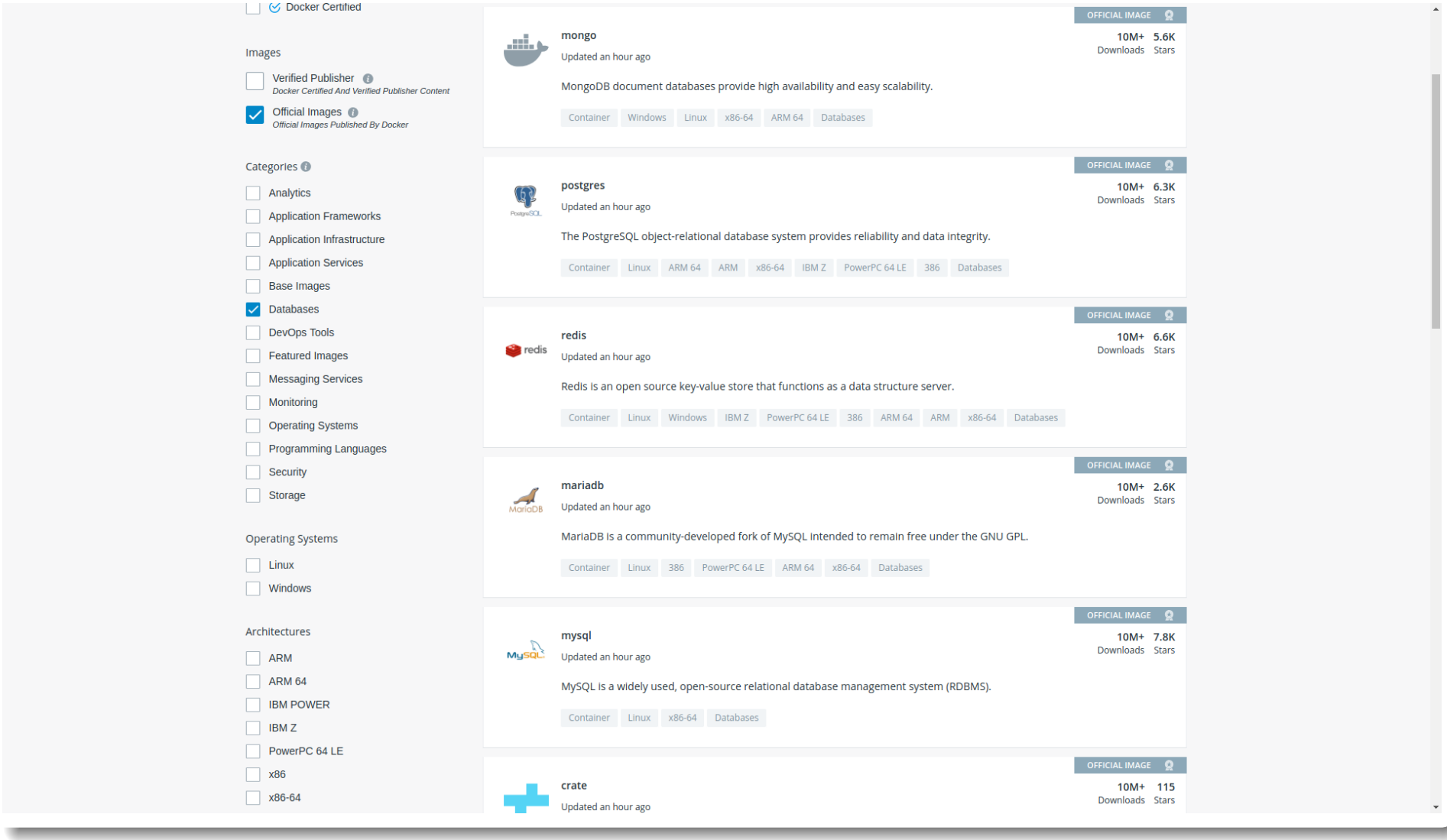
# Imagens de sistemas operacionais “puros”



# Imagens de sistemas operacionais com linguagens pré-instaladas



# Imagens de sistemas operacionais com bancos de dados pré-instalados



Ex: Se eu quiser construir uma imagem para um continer Linux que irá fazer *webscrapping* utilizando Celery eu posso criar a imagem:

- A partir da imagem de um Linux puro (Alpine, Ubuntu etc)
- A partir da imagem oficial do Python (que por sua vez foi criada a partir da imagem de um Linux puro)
- A partir de uma imagem que já tenha Celery instalado e configurado

## Trechos do `Dockerfile` do *MySQL*

```
# Usando outra imagem como ponto de partida
FROM debian:stretch-slim # ...
# Executando comandos para instalar pacotes
RUN apt-get update && # ...
# Definindo variáveis de ambiente
ENV GOSU_VERSION 1.7 # ...
# Expondo uma das pastas para o hospedeiro
VOLUME /var/lib/mysql # ...
# Copiando arquivos para a imagem
COPY config/ /etc/mysql/ # ...
# Expondo a porta 3306 para o hospedeiro
EXPOSE 3306 33060 # ...
# Executando o deamon do MySQL
CMD ["mysqld"]
```

Fonte: MySQL Docker File

Esses comandos serão executados toda vez que um novo container de MySQL for construído.

Observações importante: **o sistema de arquivo dos *containers* é volátil!**

(Isso ficará mais claro quando falarmos sobre volumes)

# *Containers*

*Containers* são instâncias de imagens.

- Um *container* é gerado a partir de uma imagem
- Um hospedeiro pode executar vários *containers* simultaneamente

Analogia: a imagem é o molde e os *containers* são os objetos criados a partir desse molde

# Volumes

Lembram-se que o sistemas de arquivos das *imagens* são efêmeros?

Vamos fazer um teste.

Vamos criar uma instância (*container*) a partir da imagem oficial do *Ubuntu*:

```
[luiz@SPWSPRD37L Downloads]$ docker create ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
6cf436f81810: Pull complete
987088a85b96: Pull complete
b4624b3efe06: Pull complete
d42beb8ded59: Pull complete
Digest: sha256:7a47ccc3bbe8a451b500d2b53104868b46d60ee8f5b35a24b41a86077c650210
Status: Downloaded newer image for ubuntu:latest
54c6c11aeac6e0226ea2aa70c2d5b73eac514c27c9773df77980eea438dbe5f7
[luiz@SPWSPRD37L Downloads]$ docker run -it -d ubuntu bash
80def1a3ad4993b6a62c0527cd9874a0499ba04357b3d3c687930f66b6ab5b93
[luiz@SPWSPRD37L Downloads]$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
TS                  NAMES
80def1a3ad49        ubuntu              "bash"              9 seconds ago       Up 7 seconds
                    friendly_galileo
```

Agora que o *container* está rodando vamos acessar o container, criar um arquivo o arquivo `/root/file.txt` e depois parar o container.

```
[luiz@SPWSPRD37L Downloads]$ docker exec -it ubuntu shell
Error: No such container: ubuntu
[luiz@SPWSPRD37L Downloads]$ docker exec -it 80def1a3ad49 shell
OCI runtime exec failed: exec failed: container_linux.go:344: starting container process caused "exec:
\"shell\": executable file not found in $PATH": unknown
[luiz@SPWSPRD37L Downloads]$ docker exec -it 80def1a3ad49 bash
root@80def1a3ad49:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@80def1a3ad49:/# cd root/
root@80def1a3ad49:~# ls
root@80def1a3ad49:~# echo "Here comes a string..." >> file.txt
root@80def1a3ad49:~# ls
file.txt
root@80def1a3ad49:~# pwd
/root
root@80def1a3ad49:~# exit
exit
[luiz@SPWSPRD37L Downloads]$ docker stop 80def1a3ad49
80def1a3ad49
[luiz@SPWSPRD37L Downloads]$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
TS                  NAMES
```

Reiniciando o *container*, acessando-o e procurando pelo arquivo temos uma surpresa:

```
[luiz@SPWSPRD37L Downloads]$ docker run -it -d ubuntu bash
20b1f1799d605b85a9c02cb5a746b739f8c7d54a724e8c2c8f48a56ceecc5ca8
[luiz@SPWSPRD37L Downloads]$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
TS                  NAMES
20b1f1799d60        ubuntu              "bash"              9 seconds ago       Up 7 seconds
                    optimistic_goldwasser
[luiz@SPWSPRD37L Downloads]$ docker exec -it 20b1f1799d60 bash
root@20b1f1799d60:/# cd root/
root@20b1f1799d60:~# ls
root@20b1f1799d60:~# ????
```

Os containers são como arquivos binários de programas que não podem ser modificados permanentemente.

E agora? Vamos ter que deixar nosso container de banco de dados rodando eternamente para não perder os dados?

Não.

Vamos usar ***Volumes***!

Volumes são pastas do sistema hospedeiro que “conectamos” (montamos) ao sistema de arquivo do container, dessa forma quando ele escrever nessas pastas ele estará escrevendo no sistema de arquivos da máquina hospedeira.

*Talk is cheap, show me the code!*

Vamos criar outro container, dando um nome a ele (para facilitar nossa vida) e montando a pasta `~/Downloads/volume_do_ubuntu` da máquina hospedeira na pasta `/root` do container e repetir o nosso experimento

```
[luiz@SPWSPRD37L Downloads]$ docker commit 20b1f1799d60 nova_imagem
sha256:4dc217e439e98e04b761d3813f4319d69f080714bc6ccafbf6e218fae6d9aa4c
[luiz@SPWSPRD37L Downloads]$ docker run -it -v "$PWD/volume_do_ubuntu/":/root nova_imagem bash
root@b718de8ab764:/# cd /root/
root@b718de8ab764:~# ls
root@b718de8ab764:~# echo "I hope this works..." >> file.txt
root@b718de8ab764:~# ls
file.txt
root@b718de8ab764:~# exit
[luiz@SPWSPRD37L Downloads]$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
TS                  NAMES
20b1f1799d60        ubuntu              "bash"              21 minutes ago      Up 21 minutes
                    optimistic_goldwasser
[luiz@SPWSPRD37L Downloads]$ docker stop 20b1f1799d60
20b1f1799d60
[luiz@SPWSPRD37L Downloads]$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              POR
TS                  NAMES
```

```
[luiz@SPWSPRD37L Downloads]$ docker run -it -v "$PWD/volume_do_ubuntu/":/root nova_imagem bash
root@d546966b9fab:/# cd /root/
root@d546966b9fab:~# ls
file.txt
root@d546966b9fab:~# cat file.txt
I hope this works...
root@d546966b9fab:~# :)
```

É isso que fazemos com containers de bancos de dados: os arquivos utilizados pelas aplicações de banco de dados são armazenados em volumes no sistema hospedeiro.

```
[luiz@blade luizsol.com]$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                              NAMES
d04b763eeada        nginx               "nginx -g 'daemon of…"   24 seconds ago      Up 22 seconds       0.0.0.0:80->80/tcp                 backend_nginx_1
7d0bfff8796c        backend-django      "/bin/sh -c 'rm -f /…"   25 seconds ago      Up 24 seconds       25/tcp, 0.0.0.0:5555->5555/tcp     backend-django
ca622f964d46        postgres:11-alpine  "docker-entrypoint.s…"   27 seconds ago      Up 25 seconds       5432/tcp                           backend_backend-db_1
[luiz@blade luizsol.com]$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
backend-django      latest              50eedad19182        9 days ago          1.45GB
postgres            11-alpine           db3d433d155d        13 days ago         70.8MB
python              3.6-stretch         e0a418687f6c        2 weeks ago         922MB
nginx               latest              f09fe80eb0e7        3 weeks ago         109MB
[luiz@blade luizsol.com]$ docker volume ls
DRIVER              VOLUME NAME
local               backend_backend-db-datavolume
local               backend_backend-redis-datavolume
[luiz@blade luizsol.com]$ docker network ls
NETWORK ID          NAME                    DRIVER              SCOPE
3b1f5a653940        backend_containers_net  bridge              local
5f463b598e69        bridge                  bridge              local
2547e57953da        host                    host                local
76473a36eaeb        none                    null                local
[luiz@blade luizsol.com]$
```

teste