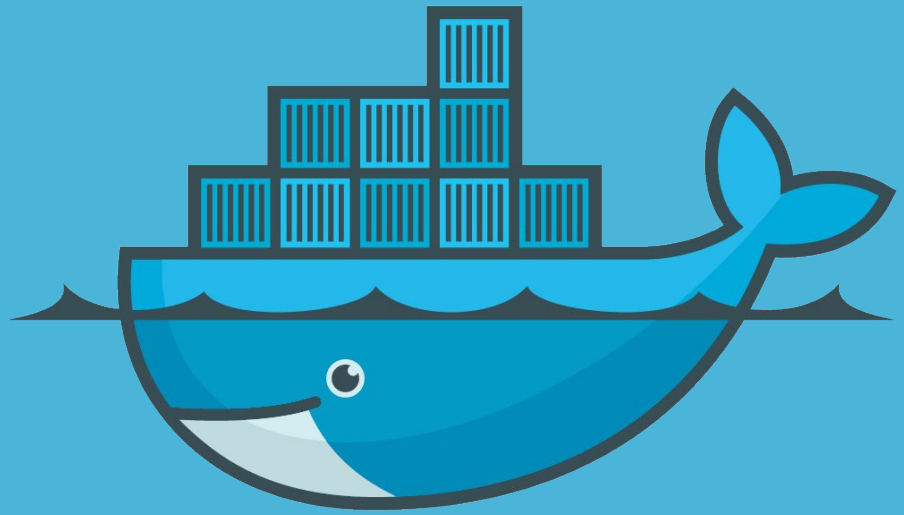


Docker Hands-On

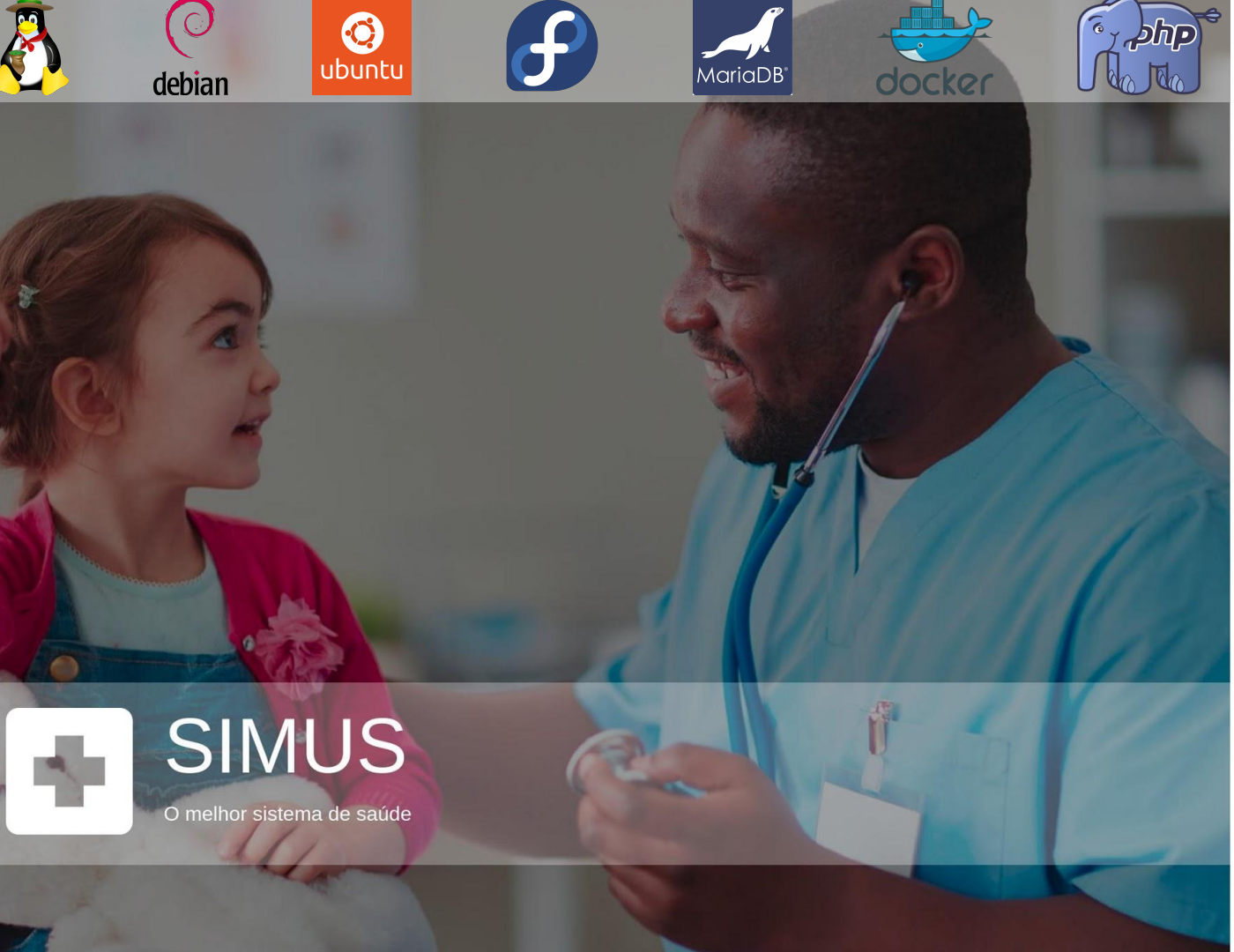
TONIN R. BOLZAN

Software Engineer ❤ DevOps

CTO na Metrosoft S/A - Santa Maria/RS

10 anos de experiência

https://bolzan.io



METROSOFT S/A

Respiramos Software Livre

Especialista em Software para Saúde

SUS / Privado

Milhares de cidadãos atendidos diariamente.

Operacional e Gestão

DEVOPS DAYS

FLISOL

FEMUG SM

Rock the Software Freedom Day

www.softwarefreedomday.org

Supported by

Google CANONICAL ubuntu linode.com fsfe

FREE SOFTWARE FOUNDATION FreeBSD LEMOTE creative commons

LINUX Joomla! LINUX JOURNAL UBUNTU user smart developer


Containers

2.

Containers são quase VMs

Vamos definir e separar as coisas

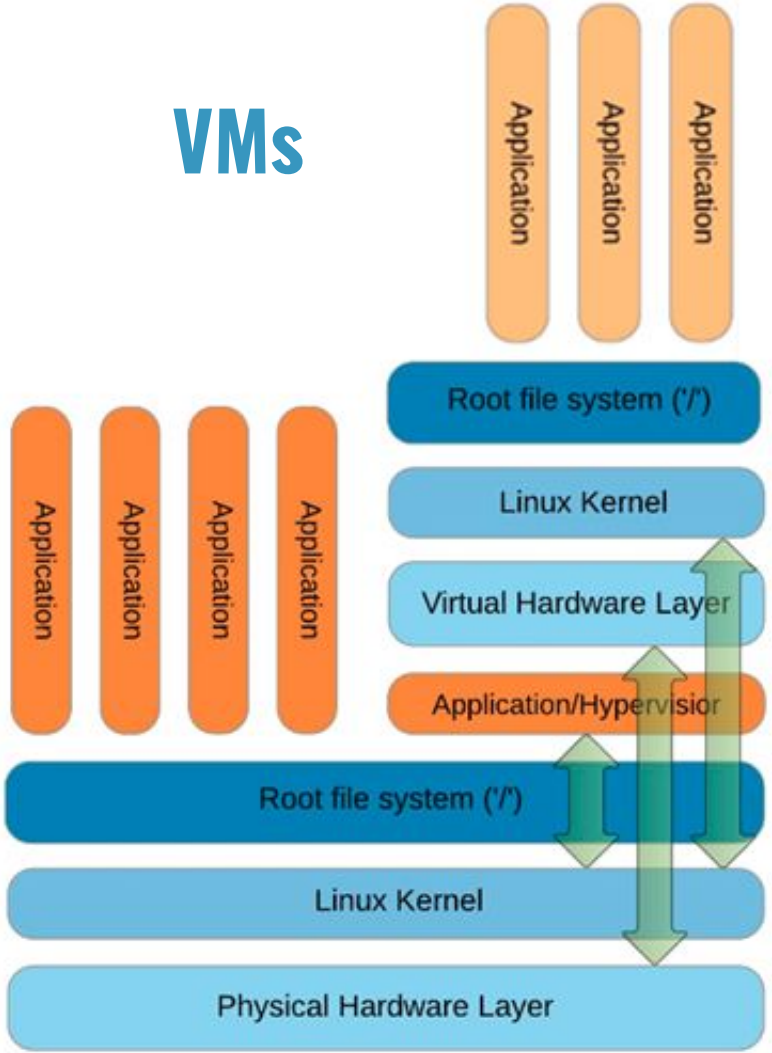
Containers não são VMs

- Não são virtualizados (hypervisor)
- São leves e rápidos
- Normalmente são efêmeros

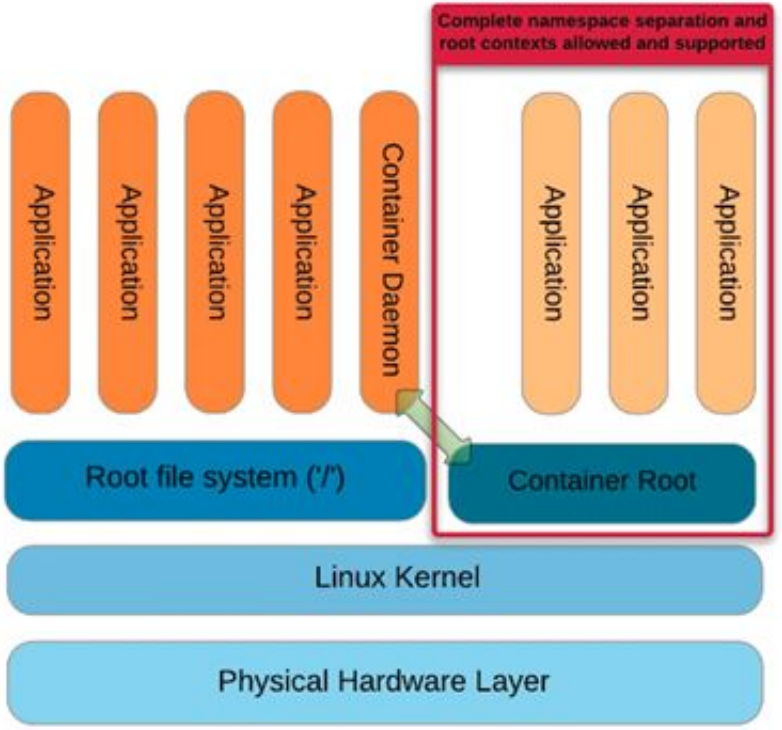
Mas são parecidos

- Empacotamento e “Isolamento”
- Um S/O dentro de outro S/O

VMs



Containers



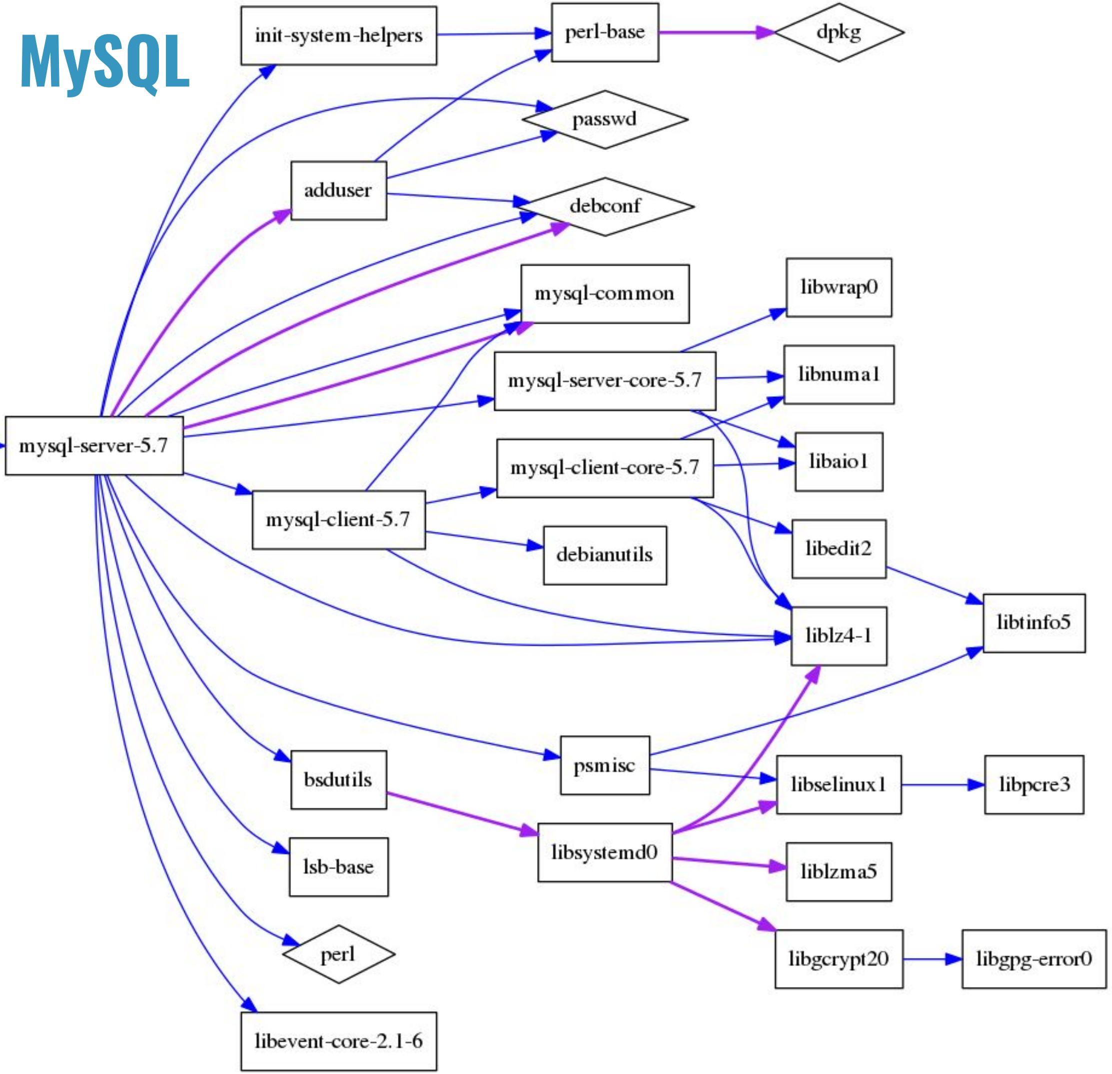
Mas por que eu preciso disso?

Qualquer que seja VMs ou Container

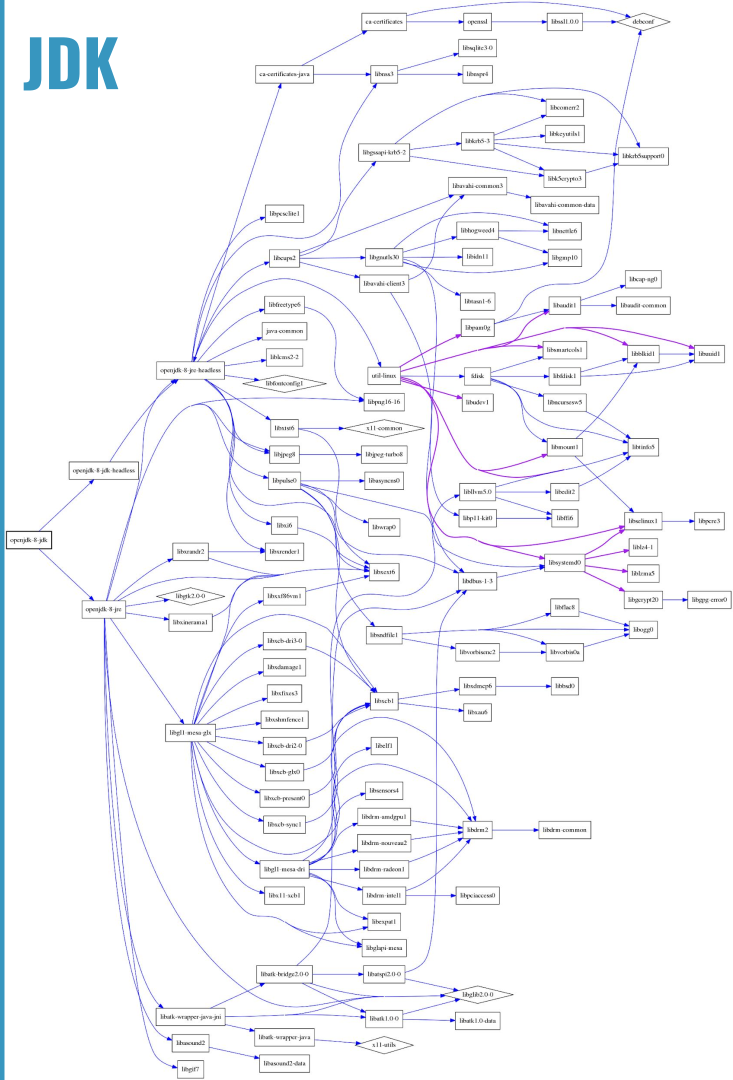
Não é mais fácil “só” instalar o programa

- WampServer
- MAMP
- LAMP / LEMP

MySQL


init-system-helpers
perl-base
dpkg
passwd
adduser
debconf
mysql-common
libwrap0
mysql-server-core-5.7
libnuma1
mysql-server-5.7
mysql-client-core-5.7
libaio1
mysql-client-5.7
debianutils
libedit2
libtinfo5
liblz4-1
psmisc
bsdutils
libselinux1
libpcre3
libsystemd0
lsb-base
liblzma5
perl
libgcrypt20
libgpg-error0
libevent-core-2.1-6

JDK



Não é mais fácil instalar o programa?

NÃO

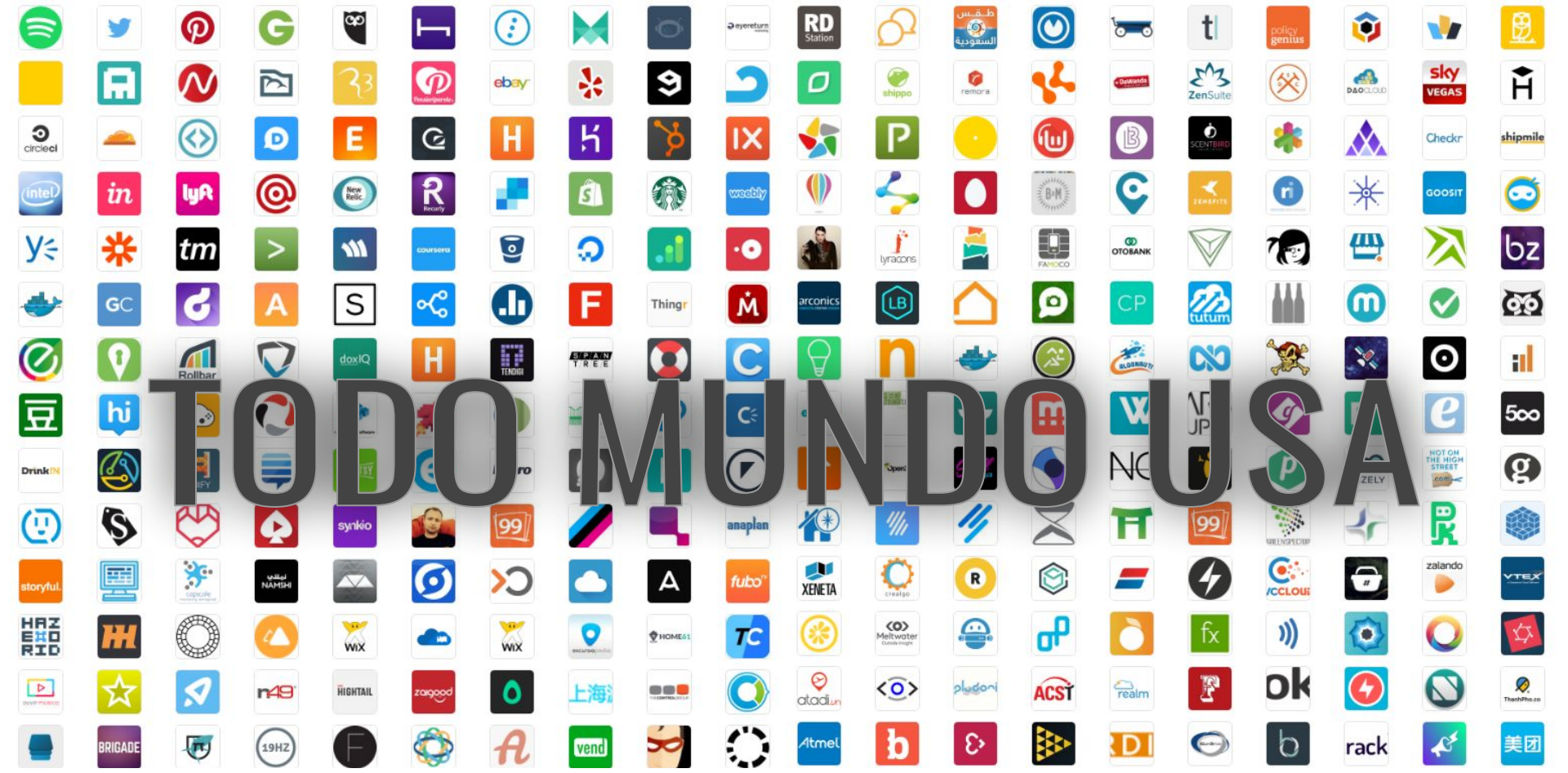
- Você “suja” todo o seu sistema
- Fica limitado a uma versão do software
- Cada S/O tem sua forma de instalação
- Difícil de reproduzir

VMs e Containers == Benefícios

- Segurança
- Independência
- Empacotamento
- Isolamento
- Limitação de recursos
- **Reprodutibilidade**
- **Agilidade**

Porém, containers:

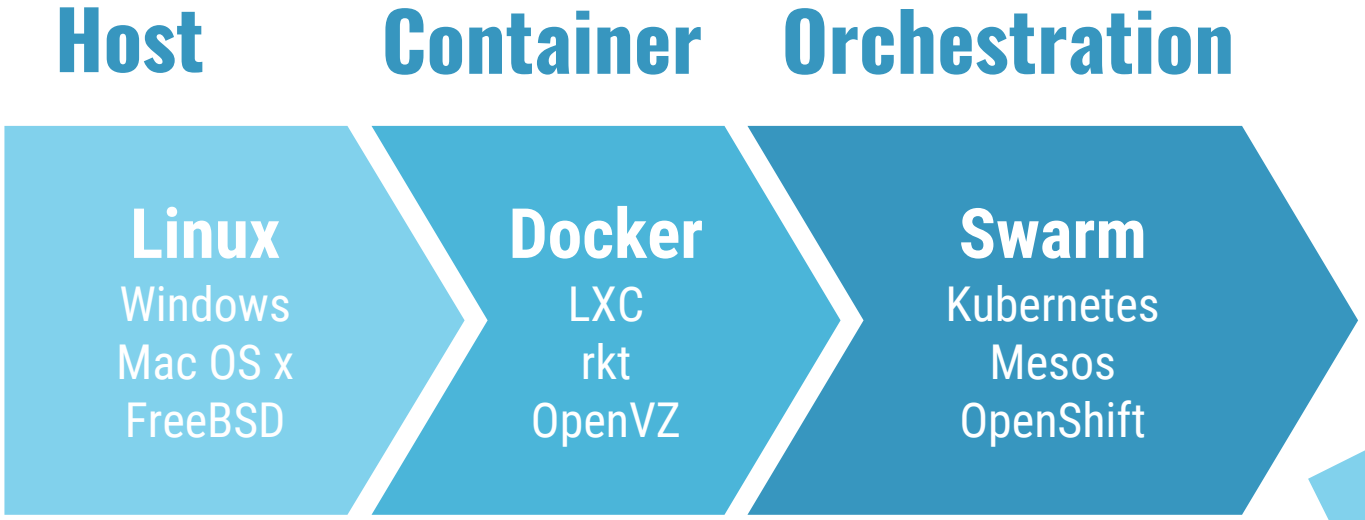
- **São leves, rápidos e pequenos**


TODO MUNDO USA
+Documentação
+Comunidade
+Estabilidade
+Confiança

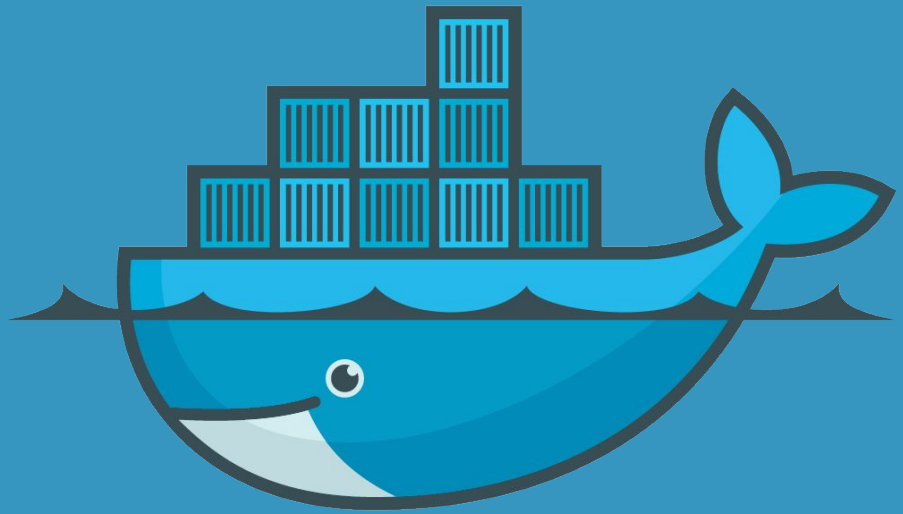
3.

Tools Overview

Vamos as ferramentas

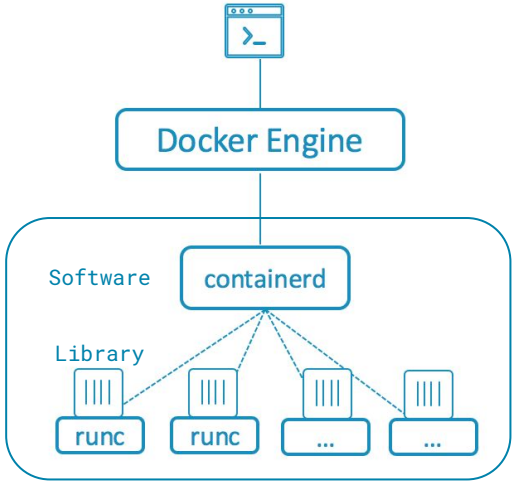


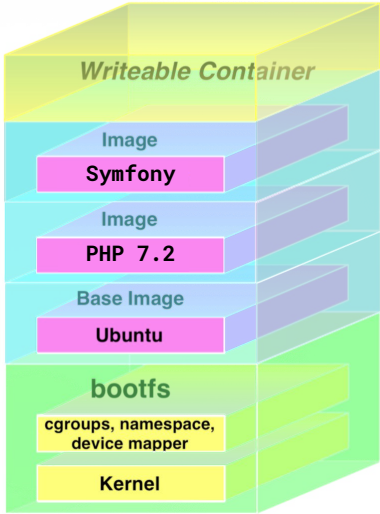

```
root@pc:~# apt install docker.io docker-compose
```



DOCKER

Conceitos

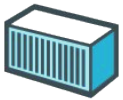




Build

» **Docker CLI**
» Executável de linha de comando

» **Docker Engine**
» Daemon que roda como serviço



Ship

» **Container runtime**
» Responsável por “rodar” o container

» **Container image**
» Formato de armazenamento em camadas

Run

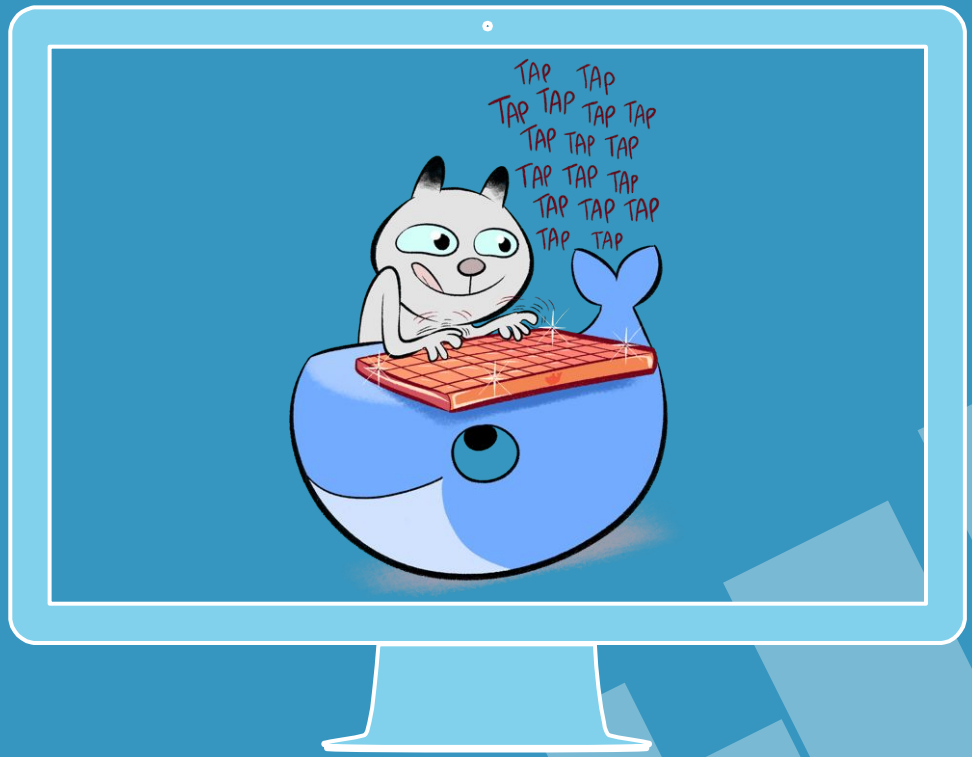
» **Repositório**
» https://hub.docker.com

» **Dockerfile**
» Receita de bolo para construção de uma imagem

» **Volumes**
» Camada de persistência para conteúdo dinâmico

» **Docker Compose**
» YML para descrição de vários containers

Docker
Hands-on



CLI - Faz download e roda uma aplicação docker

$ docker run --rm -ti debian bash

$ docker run --rm metrosoftsa/php-dev:latest php -v

```
docker run    :> Manda rodar o container
--rm          :> Limpa tudo depois de rodar
-t            :> tty
-i            :> interactive
Container     :> <uri>/<nome>/<imagem>:<tag>
```

$ docker version

$ docker login

$ docker ps

$ docker images

$ docker volume ls

$ docker build

$ docker push

$ docker pull

$ docker run

$ docker system prune --volumes -af

https://docs.docker.com/engine/reference/commandline/docker

Dockerfile

```
FROM metrosoftsa/php-dev:latest
ENV  APP_ENV development
COPY . /srv
RUN  composer install -d /srv --no-dev --no-ansi --no-progress
CMD  ["php",  "/srv/index.php"]
```

Esses arquivos são receitas para uma imagem

“FROM” estende uma imagem anterior

build - Roda e empacota a partir das definições do Dockerfile

```
$ docker build -t tonin/php:latest -f Dockerfile .
$ docker run --rm tonin/php:latest
```

push - Envia para o repositório docker hub (padrão)

```
$ docker push tonin/php:latest
$ docker pull tonin/php:latest
```

save & **load** - Realiza o dump e restore da imagem OCI

```
$ docker save -o php.tar tonin/php:latest
$ docker load -i php.tar
```

Docker

Compose

Docker Compose - Arquivo YML e CLI

```yaml
version: "3"
services:
  redis:
    image: redis:latest
    networks:
      - default

  web:
    build: .
    ports:
      - 8080:80
    volumes:
      - ./:/srv:rw
      - log:/var/log:rw
    environment:
      - APP_DEBUG: true
    depends_on:
      - redis
    networks:
      - default

volumes:
  log: {}

networks:
  default:
```

```yaml
Versão de ref. do yml
Descrição dos serviços
  Nome do container 1
    Imagem de repositório: tag
    Definição de rede
      - alias de rede

  Nome do container 2
    Local de build do Dockerfile
    Portas a serem mapeadas
      - <host>:<container>
    Volumes a serem mapeados
      - <host>:<container>:rw
      - <alias>:<container>:rw
    Variaveis de ambiente
      - Nome da variavel: <valor>
    Prioridade de inicialização
      - Nome do container
    Definição de rede:
      - alias de rede

Descrição dos volumes
  Nome do volume referenciado

Descrição das redes
  Nome da rede: configuração
```

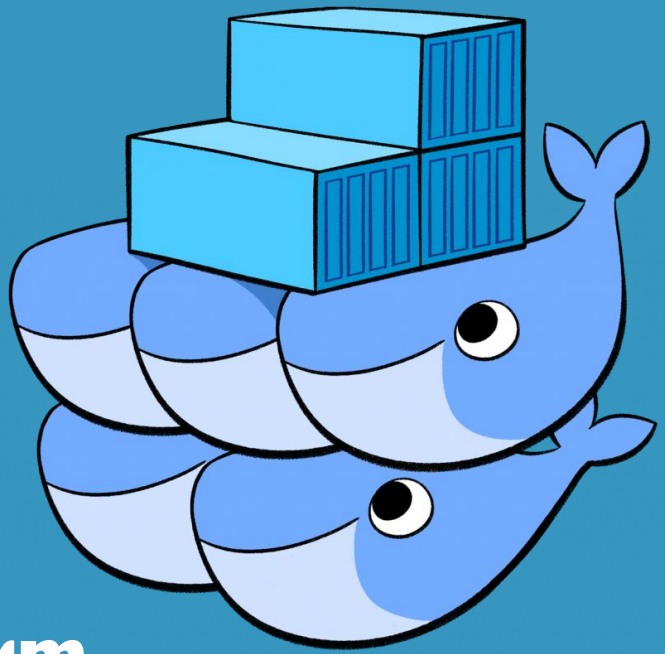
```
$ docker-compose version
$ docker-compose build
$ docker-compose pull
$ docker-compose up -d
$ docker-compose down
```

4.

Local vs. Produção

Qual a diferença?

Em produção você tem muitos containers



Orquestração

Docker Swarm

Orquestração

- Automação
- Padronização
- Agilidade
- Controle
- Monitoramento
- **Escalabilidade**

Escalabilidade Pokémon GO

1X
Target Traffic

5X
Worst Case Estimate

50X
Actual Traffic

Google Cloud

https://bit.ly/2dasyr4

Fluxo de Trabalho

Dockerfile && build

Estende um imagem base do repositório e inclui suas peculiaridades e gera sua própria imagem.

docker-compose

Usado em desenvolvimento e testes.

push || save

Gerar a imagem final e envia para um repositório ou servidor.

pull || load

Carrega a imagem em outro computador ou servidor.

docker run -d ...
docker-compose up -d

docker stack deploy -c stack.yml app

- docker service update

Executa os containers utilizando a ferramenta mais adequada para cada situação.

Obrigado

Slides: https://bit.do/tchelinuxsm18