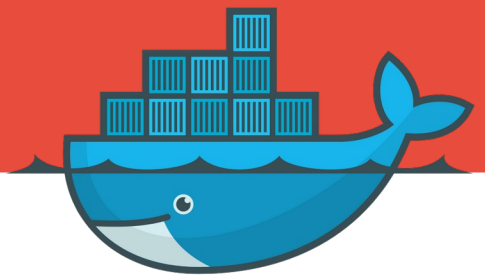
CONTAINERS

Prof. Dr. Dorival M. Machado Junior
Libertas Faculdades Integradas

CONTAINERS

CONTAINER



É uma aplicação ou ambiente (conjunto de softwares) encapsulado e isolado, no qual todas as dependências necessárias para sua execução, já estejam resolvidas.

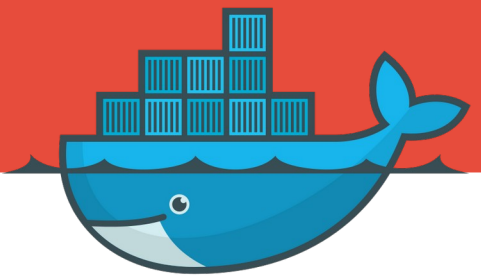
O container é portável, podendo “transitar” entre os ambientes de

- Developer → desenvolvimento de sistemas
- Homologação → teste de sistemas
- Produção → rodando “de verdade”

E entre as plataformas:

- Linux
- Windows
- Nuvem (Amazon AWS, Azure, Google Cloud, etc.)

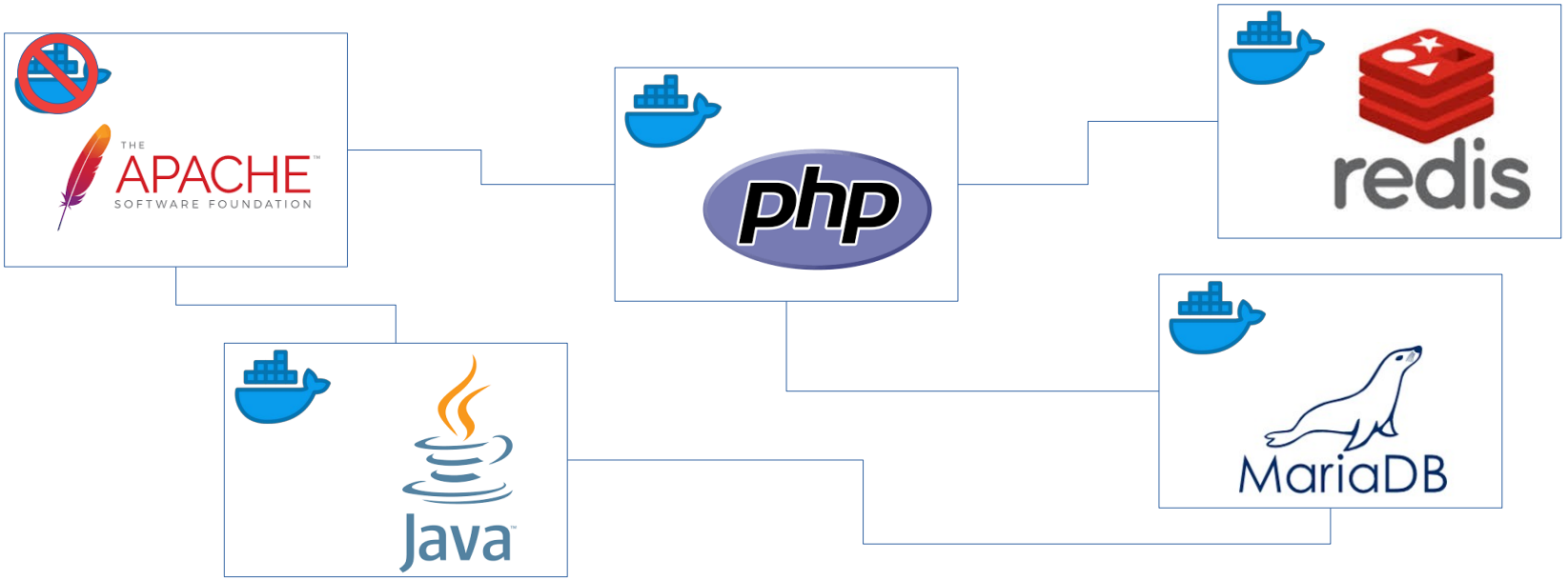
CONTAINER



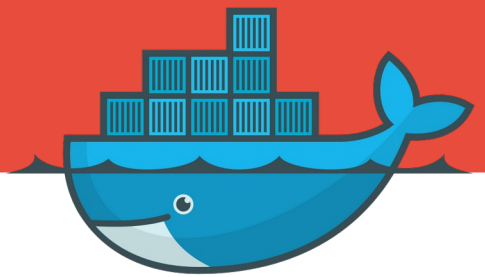
Existem várias implementações de container.

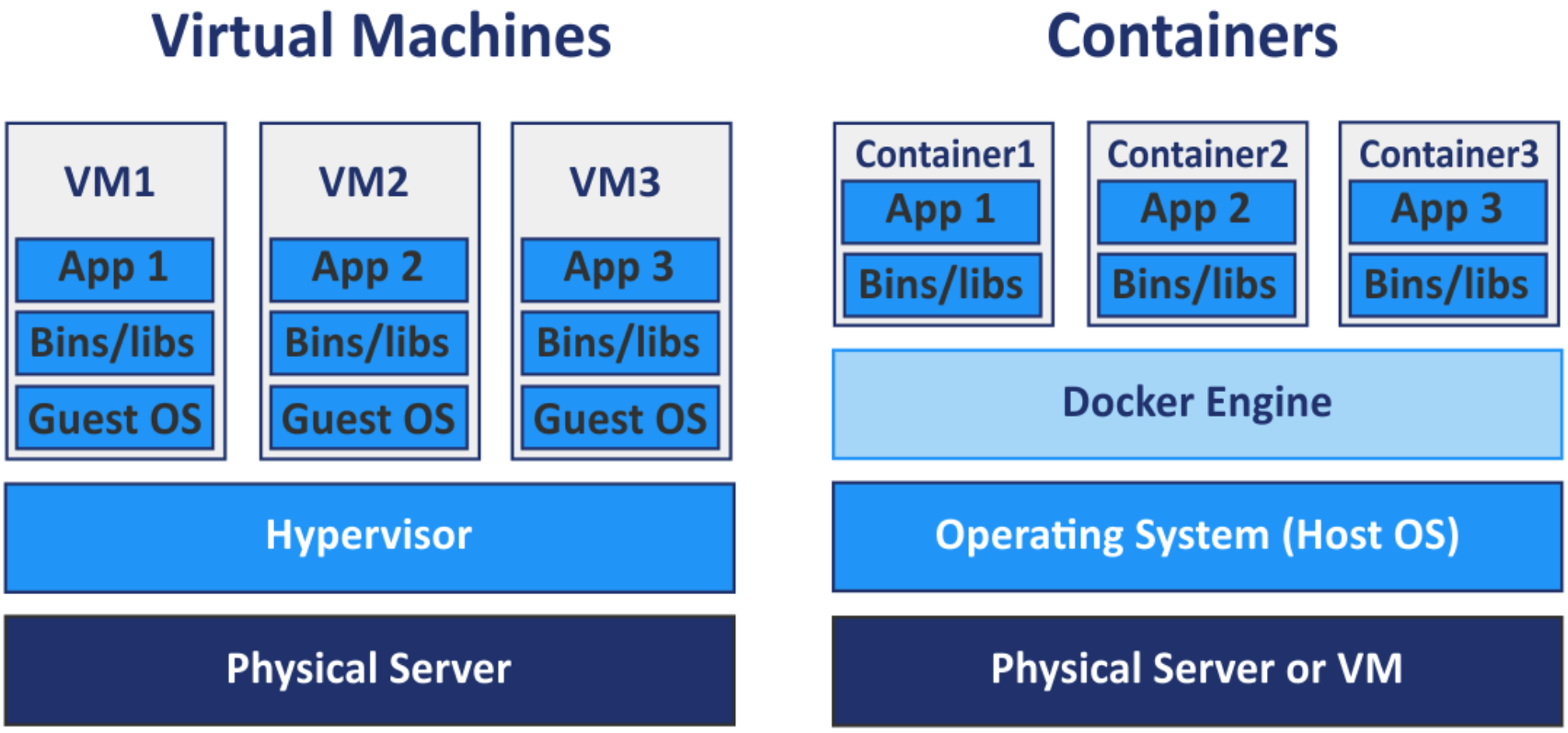
Docker é o mais usado

- Desenvolvido pelo Google
- Linguagem de alta performance Go (www.golang.org)

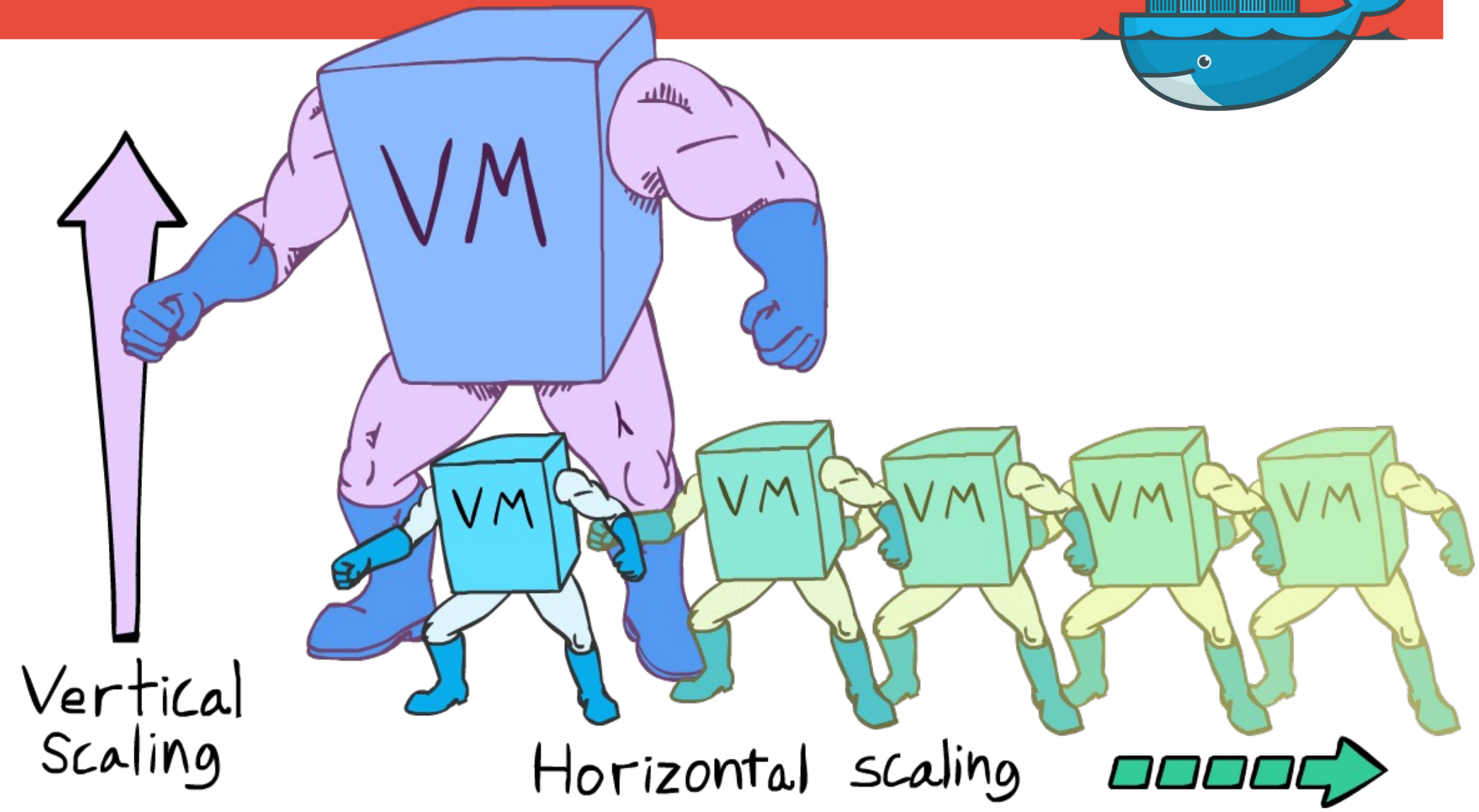


CONTAINER

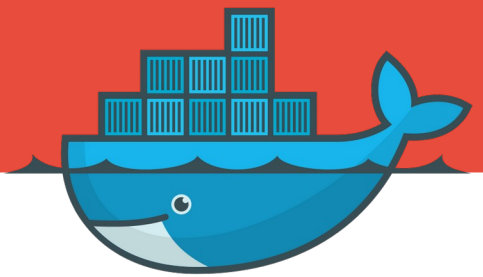



Virtual Machines
VM1
App 1
Bins/libs
Guest OS
VM2
App 2
Bins/libs
Guest OS
VM3
App 3
Bins/libs
Guest OS
Hypervisor
Physical Server
Containers
Container1
App 1
Bins/libs
Container2
App 2
Bins/libs
Container3
App 3
Bins/libs
Docker Engine
Operating System (Host OS)
Physical Server or VM

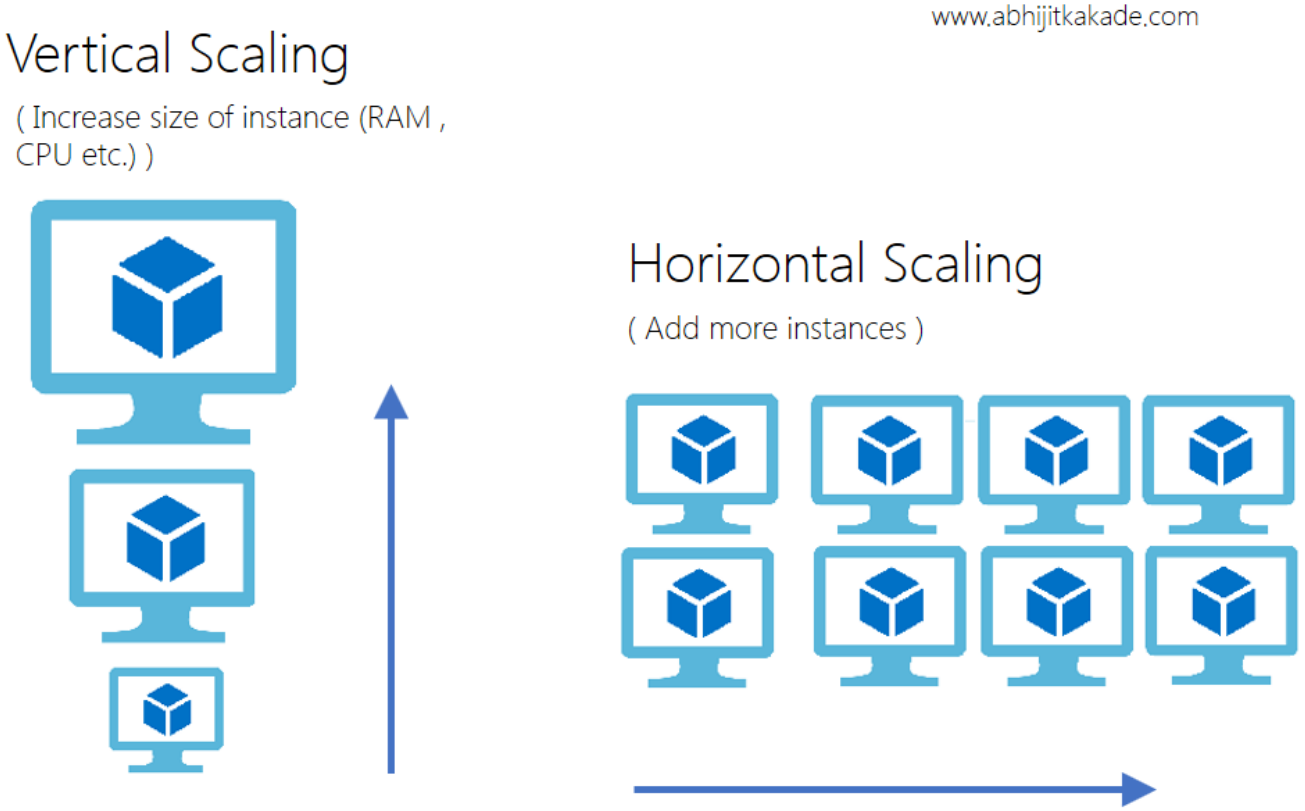
CONTAINER



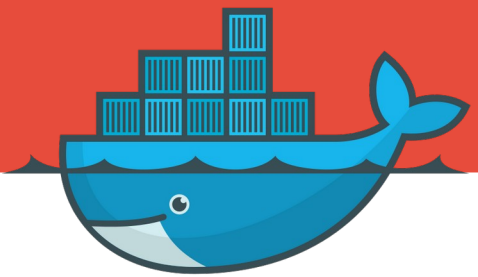
CONTAINER



Escalonamento horizontal: Uma vez configurado, pode ser replicado quantas vezes forem necessárias.



ARQUITETURA DO DOCKER



Docker Container → recipiente que contém todos os arquivos necessários

Docker Daemon → processo do docker

Docker client → cliente que interage com o daemon (envio de comandos ao docker)

Docker image → template (dados e metadados para execução do container)

Docker engine → motor para criação do container

Docker registry → coleção de imagens. Pode ser público ou privado

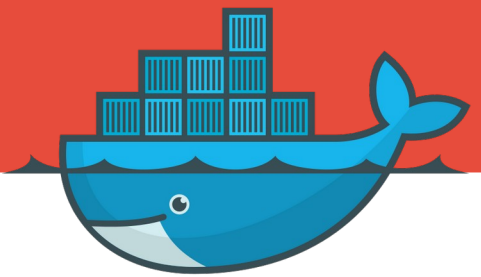
Docker hub → É um “repositório” usado para baixar ou hospedar imagens. (SaaS)

Docker file → arquivo texto com comandos para criação de novas imagens.

Docker Composer → definição de aplicações usando vários containers.

Docker Swarm → ferramenta para agrupamento de conteiners docker.

INSTALAÇÃO DO DOCKER 1/2



 # apt-get update

 # apt-get install docker.io

 # dpkg -l | grep docker ← conferir instalação

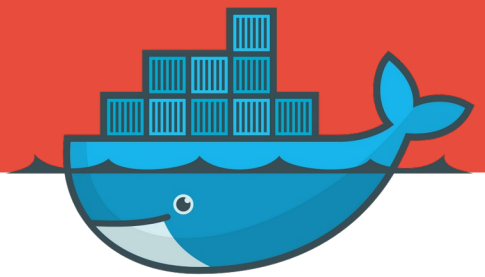
Ou

$ **snap install docker**

 # ln -s /snap/docker/384/bin/docker /usr/bin/docker

$ **snap list** ← conferir instalação

INSTALAÇÃO DO DOCKER 2/2



Conferir se o usuário comum faz parte do grupo Docker:

$ **groups**

Ou

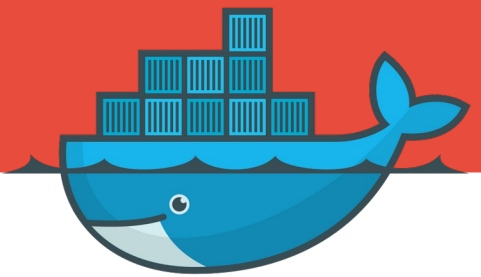
$ **cat /etc/group**

SE o usuário não fizer parte, faça:

gpasswd -a $USER docker

Encerre todas as sessões e acesse o sistema novamente

INICIALIZAÇÃO DO DOCKER



Inicializando o docker (efeito não permanente)

 # /etc/init.d/docker start

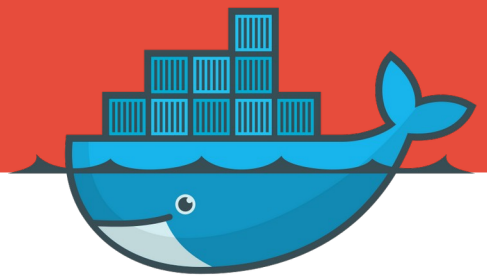
Ou

 # systemctl start docker

Habilitando o docker para execução na inicialização do sistema operacional hospedeiro.

systemctl enable docker

DOCKER-CLI



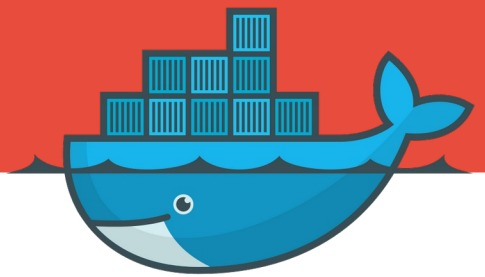
Sintaxe padrão

$ docker [opções] [comando] [argumentos]

$ docker version [opções] [comando] [argumentos]

$ docker.machine [opções] [comando] [argumentos]

DOCKER-CLI



Conhecendo argumentos de cada comando

$ docker [comando] --help

Exemplo:

$ docker start --help

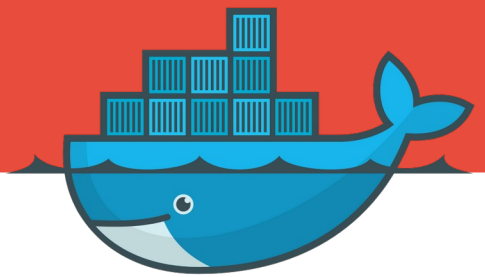
Usage: docker start [OPTIONS] CONTAINER [CONTAINER...]

Start one or more stopped containers

Options:

```
  -a, --attach               Attach STDOUT/STDERR and forward signals
      --checkpoint string    Restore from this checkpoint
      --checkpoint-dir string  Use a custom checkpoint storage directory
      --detach-keys string   Override the key sequence for detaching a container
  -i, --interactive          Attach container's STDIN
```

Comandos básicos



$ docker info

$ docker version

$ docker search ← procura containers no Hub

$ docker pull [container] ← baixa imagem

$ docker ps -l ← exibe os containers em execução

$ docker ps -a ← exibe todos os containers

$ docker images ← lista todas as imagens baixadas

$ docker run [imagem] ← executa o container da imagem

$ docker ps -aq ← lista o ID de todos os containers

$ docker stats [container] ← listar estatísticas de uso

$ docker inspect [container] ← inspetor do container

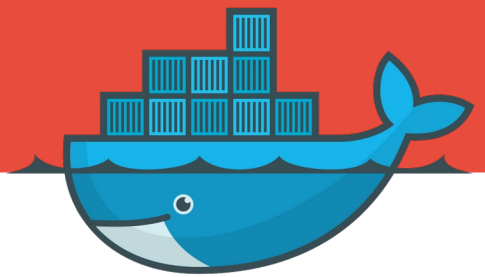
$ docker stop [container] ← para a execução do container

$ docker start [container] ← inicia a execução do container

$ docker rm [container] ← apaga o container

$ docker image rmi [id_imagem] ← apaga uma imagem

Comandos básicos



$ docker run

-i → interagir com o container

-t → associa o terminal atual com o do container

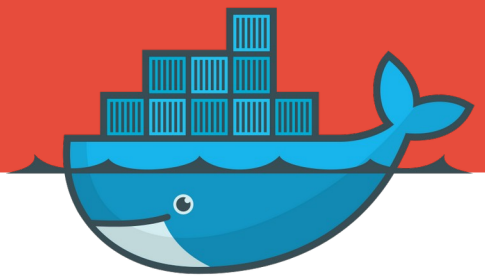
--name → atribui nome ao container

-p 8080:80 → mapeia a porta 8080 do host à porta 80 do container

-d → executa o container em segundo plano

-v /pasta/host:/pasta/container → mapeia “/diretório/host” dentro do container em “/diretório/container”

Exercício



Ex.1: criando e saindo do container

$ **docker run --name cobaia -it debian /bin/bash**

container# **exit** ← o comando exit na saída, mata o processo do container (mas não "mata" o container)

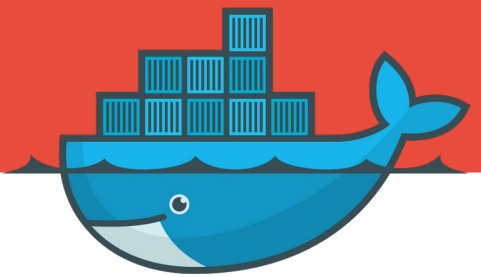
Ex.2: criando e mantendo o container rodando

$ **docker run --name cobaia2 -d -it debian /bin/bash**

$ **docker attach [id-container]** ← acessando o container

Container# **<ctrl+p> + <ctrl+q>** ← sequência para "deattach" (sair sem finalizar o processo)

Exercício



Ex.3: executando um comando dentro do container sem estar dentro dele

$ **docker exec [id-container] ls -l**

$ **docker exec [id-container] mkdir /cobaia1234**

$ **docker exec [id-container] ls -l**

Ex.4: acessando o container e criando arquivo dentro do diretório criado

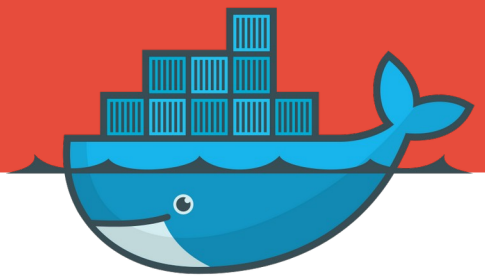
$ **docker attach [id-container]**

container# **touch /cobaia1234/obabao.txt**

container# **<ctrl+p> + <ctrl+q>**

$ **docker exec [id-container] ls -l /cobaia1234**

Exercício



Ex.5: exercício usando mapeamento de portas

$ **docker run -d --name web1 -p 8080:80 httpd**

$ **docker run -d --name web2 -p 8081:80 nginx**

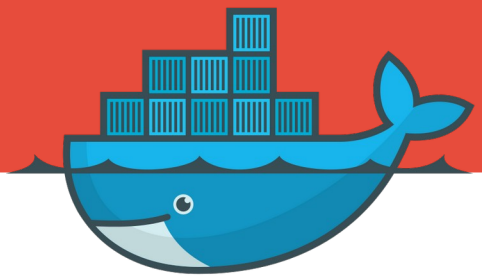
$ **docker run --name web3 -p 8082:80 nginx**

Pelo browser acesse os endereços:

http://localhost:8080

http://localhost:8081

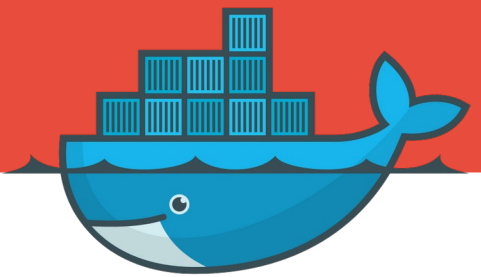
Exercício



Ex.6: crie um container debian, mapeando a pasta home do usuário atual

```
$ docker run -it -d -v /home/madruga:/madruga --name cobaiaX debian
$ docker attach cobaiaX
container# ls -l /madruga
```

Exercício



Ex.7: crie um container MariaDB e torne-o acessível de fora do container

$ **docker run --name mariacobaia -e MARIADB_ROOT_PASSWORD=senha -d mariadb**

$ **docker exec -it mariacobaia bash**

container# mysql -u root

container# exit

container# apt-get update

container# apt-get install vim

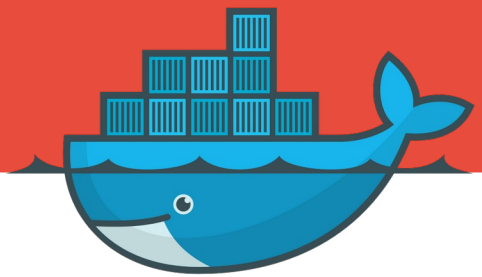
container# vi /etc/mysql/my.cnf ← conferir #bind-address no my.cnf

$ docker inspect mariacobaia -f '{{range .NetworkSettings.Networks}} {{.IPAddress}}{{end}}'

Ou

$ docker inspect mariacobaia | grep IPAddress

Exercício



Ex.8: usando VOLUMES

Volumes são áreas específicas para salvar dados. O uso deste recurso facilita a portabilidade. Os volumes nada mais são do que diretórios (use o comando inspect para ver a localização)

Comando “docker volume” cria diretórios que somente o root pode manipular.

$ **docker volume ls**

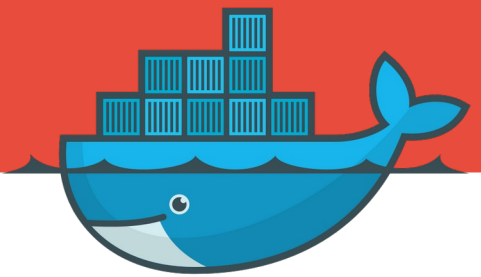
$ **docker volume create MEU-VOLUME** (Se o volume especificado não existir, ele será criado)

$ **docker volume inspect MEU-VOLUME**

Exemplo:

$ docker run -d --name SERVIDOR_COBAIA -v meu-volume:/app nginx
$ docker exec -it SERVIDOR_COBAIA /bin/bash
container# cd /app
container# touch teste1
container# touch teste2
container# touch teste3
Em seguida, alterne para root e encontre no S.O., os arquivos criados dentro do container

Exercício



Ex.9: usando VOLUMES com MariaDB

Diretório oficial do MariaDB para salvar arquivos de banco de dados é **/var/lib/mysql**

Criando um container MariaDB

$ **docker run -d –name -MARIA -p 3306:3306 -e MARIADB_ROOT_PASSWORD=123 mariadb**

Acessando o container do MariaDB

$ **docker exec -it MARIA bash**

container# **mysql -u root -p**

OU $ **docker exec -it MARIA mysql -u root -p123**

Dica: https://hub.docker.com/_/mariadb

Exercício

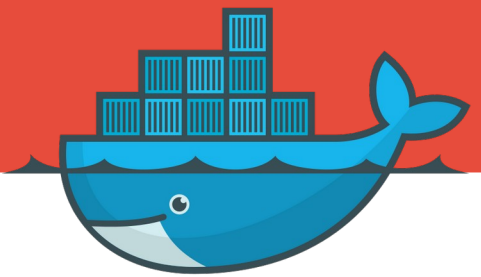
Ex.9: usando VOLUMES com MariaDB (CONTINUAÇÃO)

Dentro do mysql, criando um banco cobaia

mysql>

```
create database TOBOLINO;
create table pet(name varchar(20),dono varchar(20),especie varchar(20));
show tables;
desc pet;
insert into pet values('xaninho','Zezao','gato');
insert into pet values('tiu','Mimosa','cachorro');
```

Exercício



Ex.10: criando container de banco usando VOLUME

$ **docker run --name MARIAx -v /home/libertas/cobaia:/var/lib/mysql -p 4406:3306 -e MARIADB_ROOT_PASSWORD=123 -d mariadb**

Entre no container e cria o banco com uma tabela;

Confira o resultado no diretório real;

Mate o container recém criado

Crie um novo container utilizando-se do banco e tabela utilizado no container anterior;

$ **docker run --name MARIAy -v /home/libertas/cobaia:/var/lib/mysql -p 5506:3306 -e MYSQL_ROOT_PASSWORD=123 -d mariadb**

Exercício

Ex.11: apagando containers

Apagando todos os containers em um determinado status (created, restarting, running, removing, paused, exited, or dead)

$ **docker rm $(docker ps -a -q -f status=exited)**

Dica de uso para filtrar containers → https://docs.docker.com/engine/reference/commandline/ps/

Apagando todos os containers parados (diferente de “up”)

$ **docker container prune**

Apagando containers (instância) em execução:

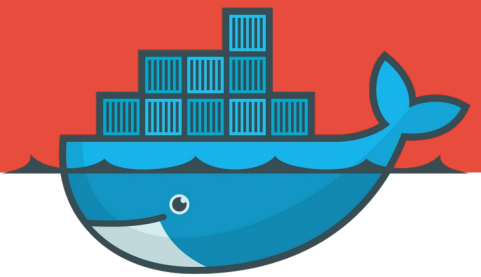
$ **docker stop [container-id ou nome]**

$ **docker rm [container-id ou nome]**

Apagando imagens (somente não utilizadas em nenhuma instância)

$ **docker rmi [nome do repositório]**

Exercício



Ex.12: Dockerfile

Dockerfile é um arquivo de texto simples com definições para criação de uma imagem personalizada.

1)Criar um diretório vazio e dentro dele o arquivo Dockerfile também vazio:

- $ **mkdir ~/chapolin**
- $ **touch ~/chapolin/Dockerfile**

2)Abrir o Dockerfile e colocar as personalizações:

- $ **nano ~/chapolin/Dockerfile**

Exercício

Ex.12: Dockerfile (2)

BOAS PRÁTICAS:

```
RUN comando1 && \
    comando 2 && \
    comando 3
```

```
# Conteúdo de Dockerfile
FROM  alpine
```
← definição da imagem que será usada como base

```
RUN apk update
RUN apk add vim
RUN apk add curl
```
← comandos a serem executados dentro do container (personalizações)

Em seguida, execute o comando para construção do container:

$ **docker build -t madruga/alpine-smarter:1.0 .**

-t especificação de TAGs de nome: desenvolvedor/nome_do_container:versão

$ Confirmar a imagem recém-criada:

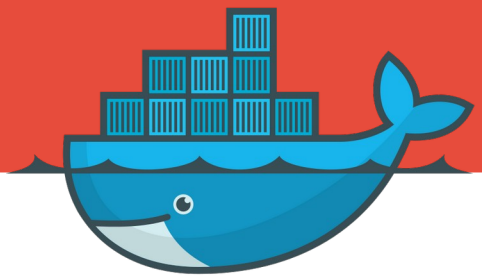
$ **docker images**

$ **docker run it madruga/alpine-smarter:1.0**

Exercício de reforço:

Crie uma imagem da distribuição “debian” com o pacote “vim” e “oneko” já instalados. Em seguida, execute uma instância de container.

Exercício



Ex.13: Dockerfile (3)

Boas práticas de Dockerfile

FROM todo Dockerfile inicia com esta tag, que serve para especificar o container base

COPY copia diretórios e arquivos para a imagem

ADD copia diretórios e arquivos para a imagem, incluindo de URLs.

ENV define variáveis de ambiente para o container

RUN executa comandos dentro do container (ex.: comandos que seriam executados no terminal)

VOLUME define um diretório no host para armazenamento persistente de dados do container. Ressalta-se que volumes não são destruídos quando o container é removido.

USER comando que precise de algum usuário específico (NÃO É MUITO USUAL)

WORKDIR diretório de trabalho para execução de comandos. Pode ser especificado diversas vezes.

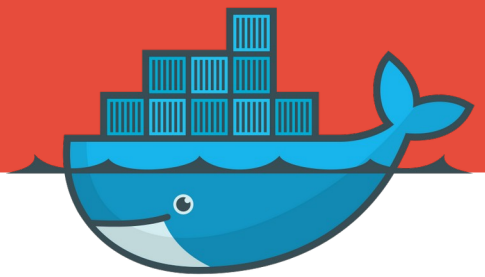
EXPOSE especificação das portas que serão expostas (públicas)

CMD especifica que comando será executado pela imagem

ENTRYPOINT especifica um comando principal a ser executado pela imagem. Ex.: ENTRYPOINT git CMD --help

ONBUILD especifica instruções para quando a sua imagem for utilizada como base em outro Dockerfile. Ex.: ONBUILD COPY . /usr/src/app; ou ONBUILD RUN /usr/src/app/mybuild.sh

Exercício



Ex.14: Dockerfile (4)

Dockerfile para criação de uma imagem usando MariaDB:

```
FROM mariadb

ENV MARIADB_ROOT_PASSWORD=tangamandapio
ENV MARIADB_USER=madruga
ENV MARIADB_PASSWORD=tangamandapio

RUN apt-get update && \
    apt-get install vim -y

WORKDIR .

RUN mkdir /aux
COPY tobolino.sql /aux

EXPOSE 4409
```

Executando um container (instância da imagem recém criada)

$ **docker build -t mariacobaia_x:1.0 .**
$ **docker run –name MariaCobaia -d mariacobaia**
$ **docker exec -it MariaCobaia bash**
container# **mysql -u root -p**