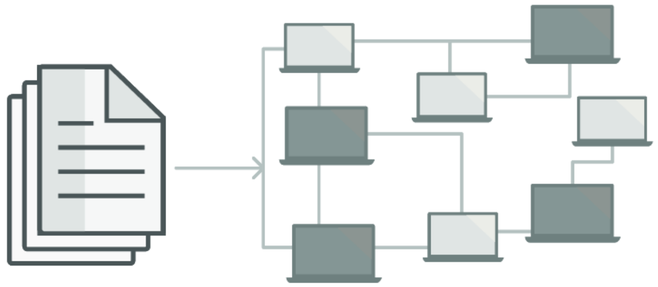
Isolamento de Ambiente

Prof. Renato Sampaio

Isolamento de Ambiente

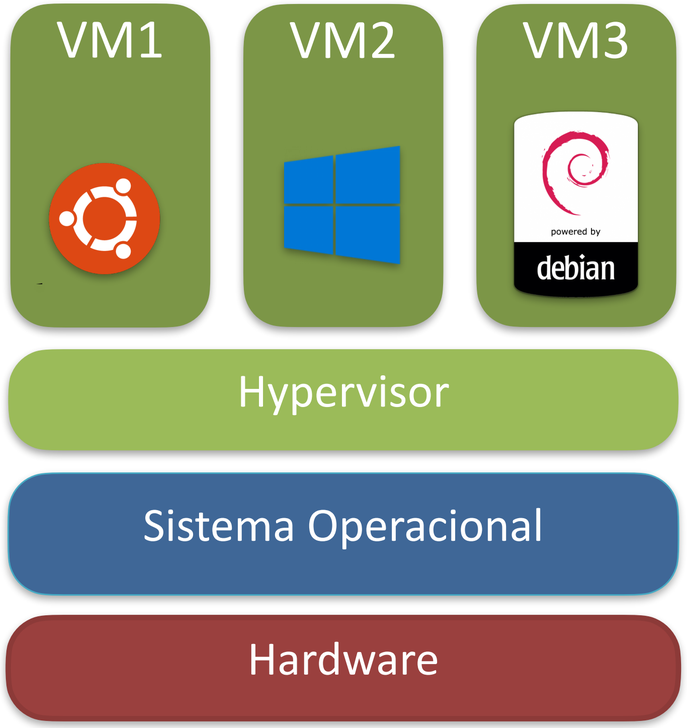
Objetivos:

- Reduzir acoplamento entre o ambiente de desenvolvimento e o ambiente de produção
- Ambiente igual para todos os desenvolvedores
  - Evitar a frase típica: “Funciona no meu computador”
- IaC - Infrastructure as Code
  - Manutenibilidade
  - Controle de versão de ambiente
  - Início com a computação em nuvem
    - IaaS - Infrastructure as Service
- Descrição e documentação do ambiente



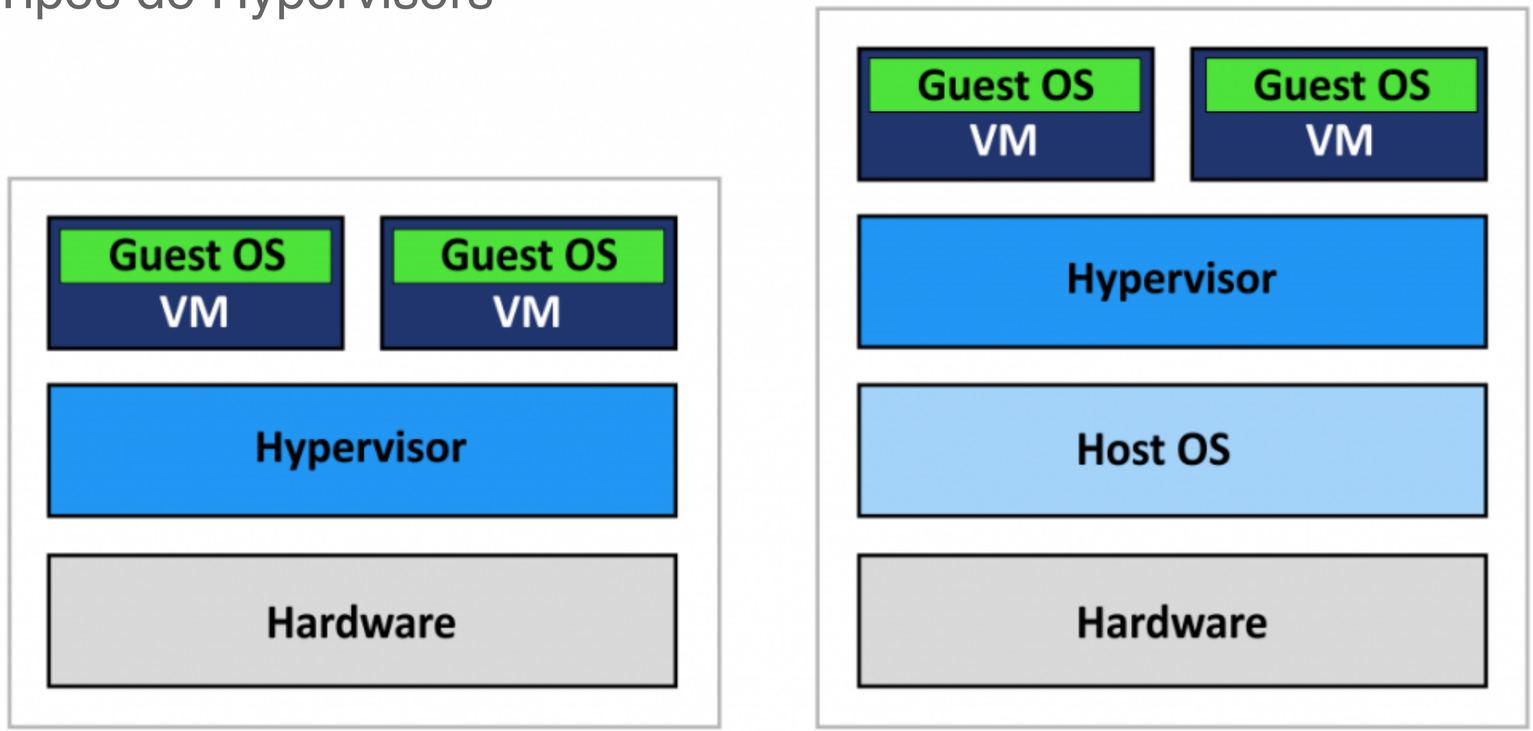
Virtualização

- Criação de Máquinas Virtuais (VM - Virtual Machine)
  - Abstração do HW por Software
- Tipos:
  - Full Virtualization
  - Paravirtualization
- Hypervisor: Programa que faz virtualização
  - NIC
  - Storage
  - Agents
  - Kernel Modules



Virtualização

- Tipos de Hypervisors



Virtualização

- Tipos de Hypervisors

Hardware virtualization (hypervisors)	**Native**		Adeos · CP/CMS · Hyper-V · KVM (Red Hat Enterprise Virtualization) · LDoms / Oracle VM Server for SPARC · Logical Partition (LPAR) · LynxSecure · PikeOS · Proxmox VE · SIMMON · VMware ESXi (VMware vSphere · vCloud) · VMware Infrastructure · Xen (Oracle VM Server for x86 · XenServer) · XtratuM · z/VM
	Hosted	**Specialized**	Basilisk II · bhyve · Bochs · Cooperative Linux · DOSBox · DOSEMU · PCem · PikeOS · SheepShaver · SIMH · Windows on Windows (Virtual DOS machine) · Win4Lin
		Independent	Microsoft Virtual Server · Parallels Workstation · Parallels Desktop for Mac · Parallels Server for Mac · PearPC · QEMU · VirtualBox · Virtual Iron · VMware Fusion · VMware Player · VMware Server · VMware Workstation · Windows Virtual PC

Virtualização

- Produtos

Virtualização

- **IaC** - Infrastructure as Code
  - Automação da Configuração de Ambiente

- **Vagrant:** Gerência de VMs como código
  - Criar, Remover, Acessar, Configurar
  - Abstrai configurações de específicas de cada hypervisor

Virtualização

- **IaC** - Infrastructure as Code
  - Automação da Configuração de Ambiente

- **Vagrant:** Gerência de VMs como código
  - Criar, Remover, Acessar, Configurar
  - Abstrai configurações de específicas de cada hypervisor

HashiCorp
Vagrant

- **Não** é um virtualizador/hypervisor
- Ferramenta para gerência de VMs
  - Criar
  - Remover
  - Acessar
  - **Configurar**
- Boa ferramenta para trabalhar com máquinas virtuais
- Abstrai configurações de específicas de cada hypervisor

Comandos Básicos

`vagrant init`	- Criar um Vagrantfile
`vagrant up`	- Iniciar/Criar uma vm
`vagrant halt`	- Desligar uma vm
`vagrant destroy`	- Deletar uma vm
`vagrant provision`	- Configurar uma vm
`vagrant ssh`	- Acessar uma vm

Vagrantfile

```
Vagrant.configure("2") do |config|
  ## https://app.vagrantup.com/boxes/search
  config.vm.box = "ubuntu/trusty64"

end
```

Outras boxes

Nome da box escolhida

Box padrão do vagrant

Vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box_url = "https://mydomain.com"
  config.vm.box = "xpto"
end
```

Box de uma origem customizada

Vagrantfile

Encaminhamento de portas

```
config.vm.network "forwarded_port", guest: 80, host: 8080
```

IP de acesso a VM

```
config.vm.network "private_network", ip: "192.168.33.10"
```

Sincronização de pasta

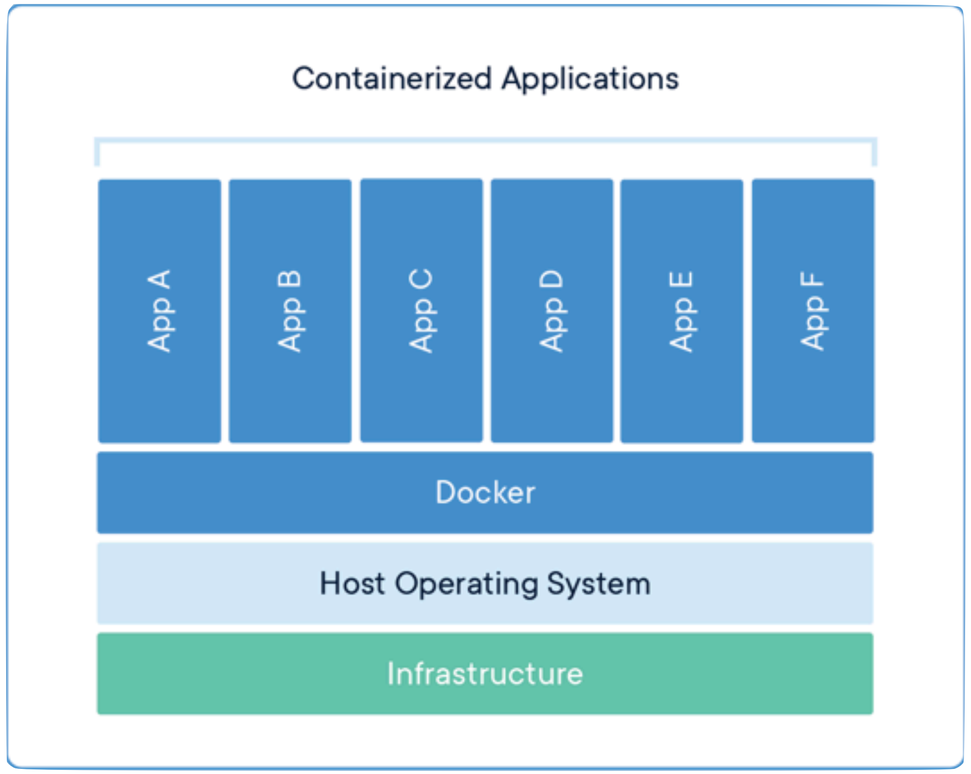
```
config.vm.synced_folder "../data", "/vagrant_data"
```

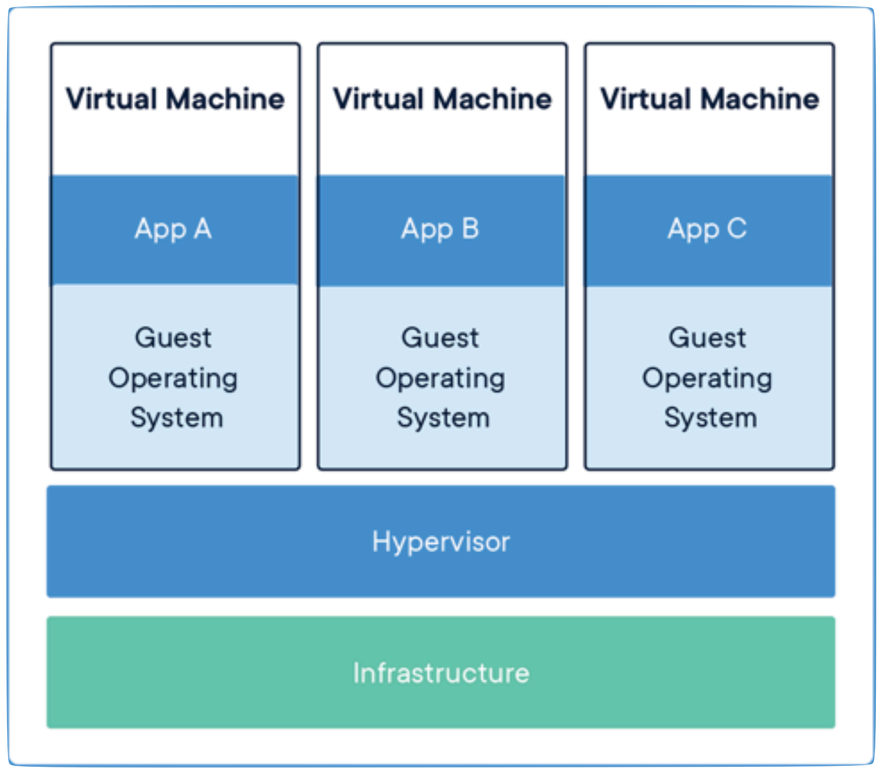
Vagrantfile

```
config.vm.provider "virtualbox" do |vb|
  vb.name = "gcs"
  vb.gui = false
  vb.memory = "1024"
  vb.cpus = 2
end
```

provider == hypervisor

Containers vs VMs





Tamanho: MBs
Inicialização: “Instantânea (ms)“
Armazenamento volátil

Tamanho: GBs
Inicialização: “Lenta (segundos)“
Armazenamento persistente

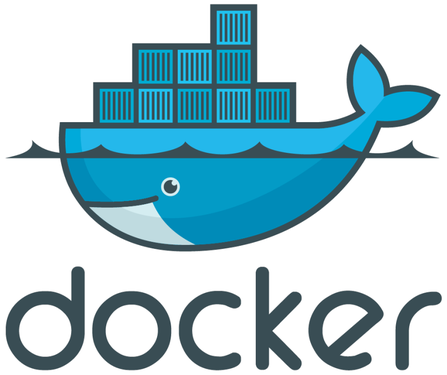
Ref. https://www.docker.com/resources/what-container

Container

- ***Não é uma VM leve!***
- Unidade de software auto-contida.
- Isola uma aplicação ou um conjunto de aplicações do S.O.
- Contém todas as dependências necessárias para rodar determinado processo.
- O que está dentro do container?
  - Código do software a ser executado
  - Arquivos de configuração
  - Processos
  - Gerência de rede
  - Dependências (bibliotecas)
  - Parte essencial do S.O. para rodar a aplicação.

Container

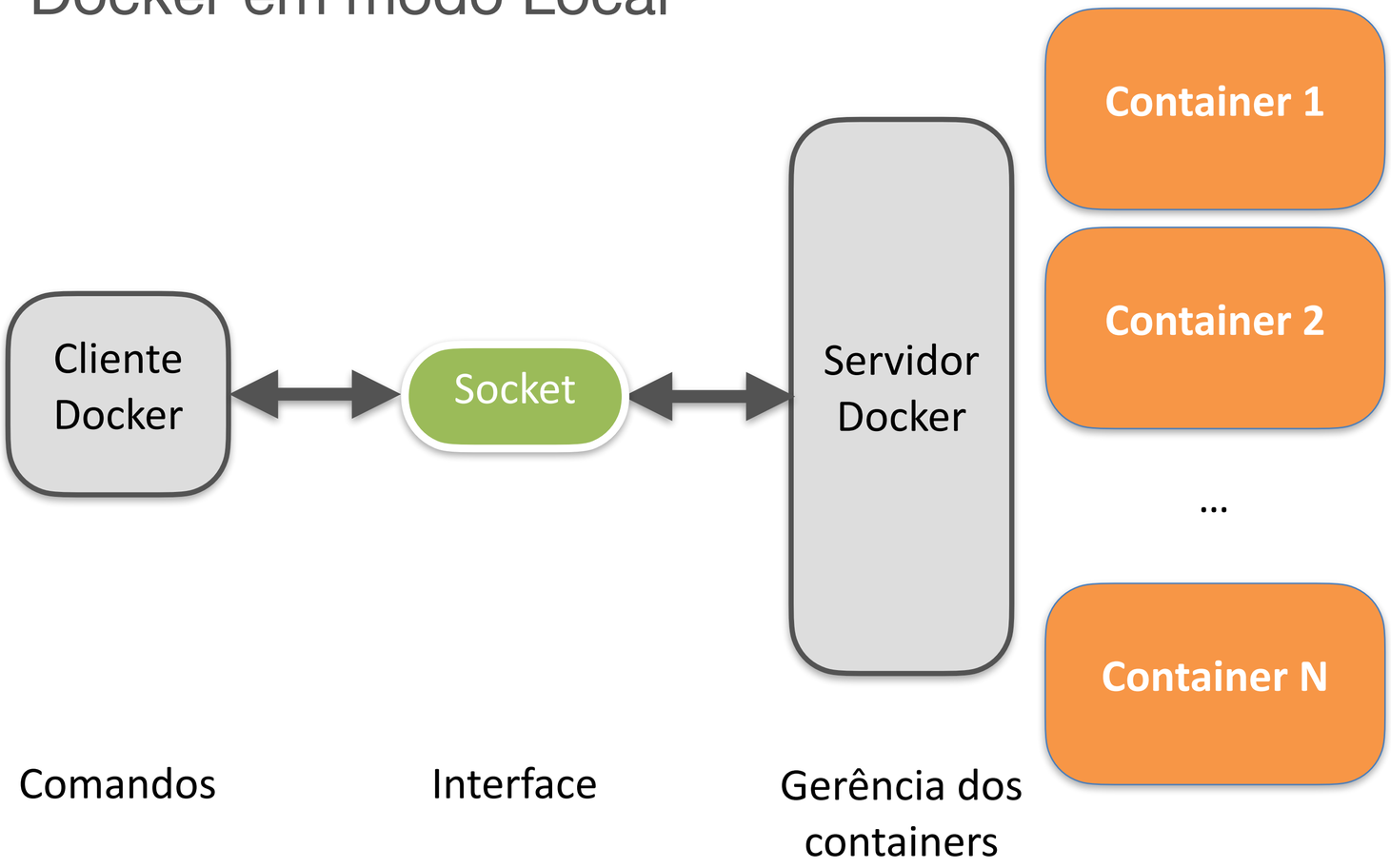
- Utiliza uma coleção de recursos dos sistemas operacionais para isolar e gerenciar aplicações.
- Recursos do S.O. utilizados:
  - ***Cgroups***: agrupa processos em espaços isolados
  - ***Namespaces***: gerenciador de redes internas
  - "***Copy-on-write***": para criar imagens



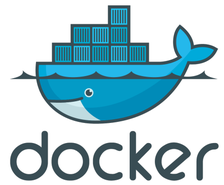
- ***Popularizou a tecnologia de Containers!***
  - Empresa
  - Programa em modo cliente (comando)
  - Serviço que gerencia containers e faz a interface com o sistema operacional
  - Programa que contrói containers
  - Serviço que hospeda um repositório de containers

Docker

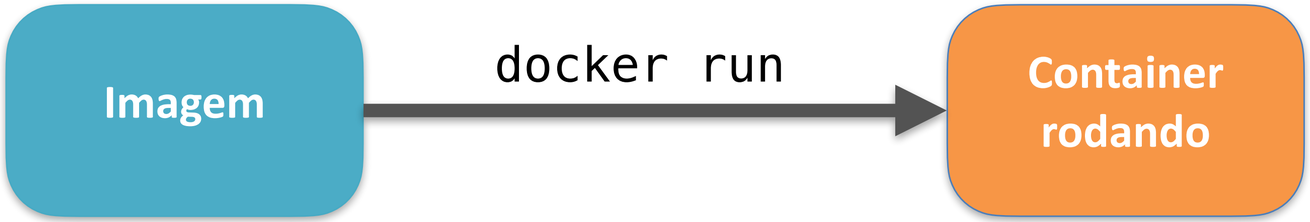
- Docker em modo Local



Docker Image



- Imagem (parte necessária do S.O. para rodar a aplicação)



Docker Flow

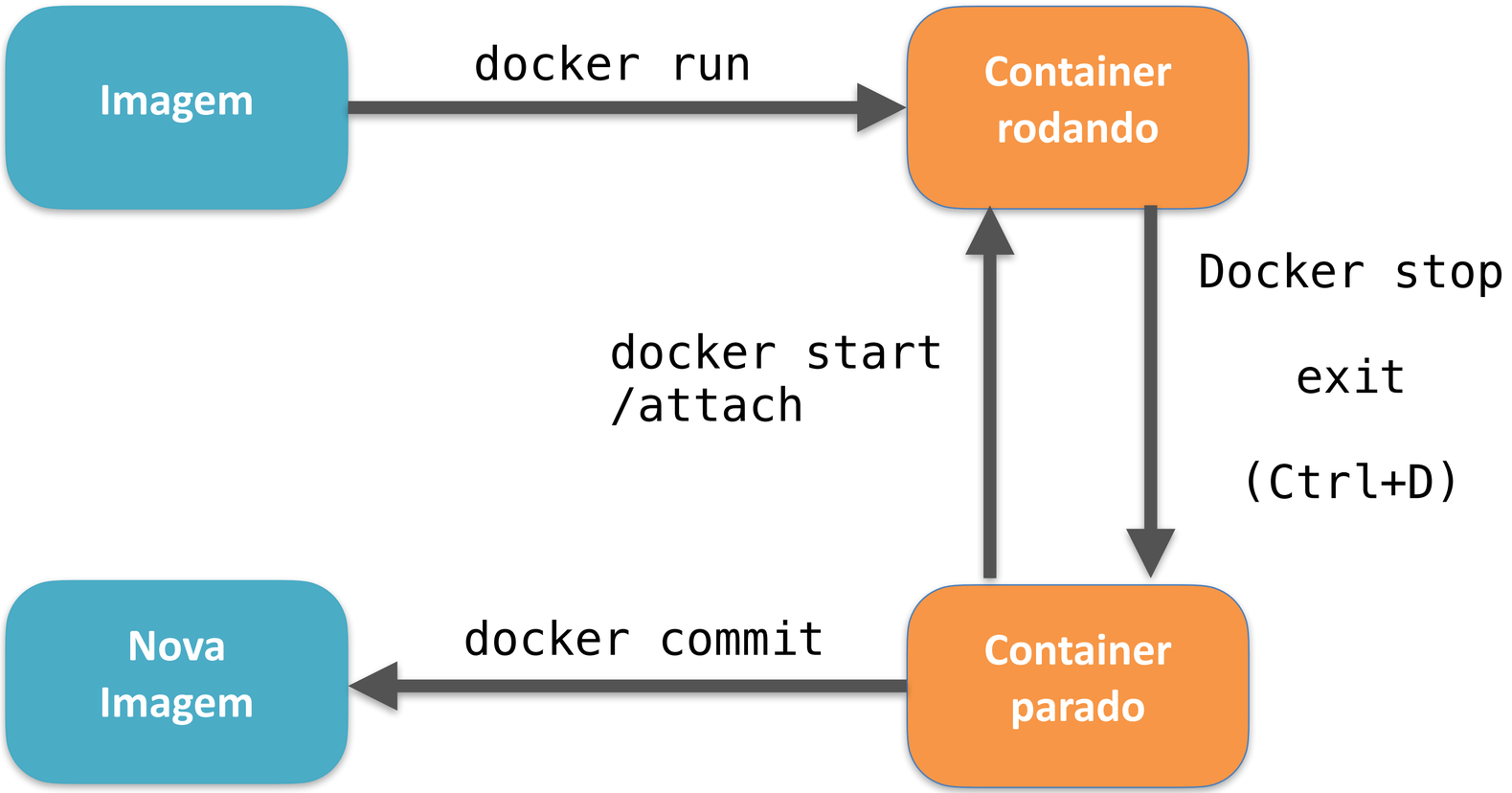
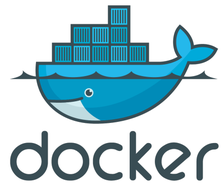


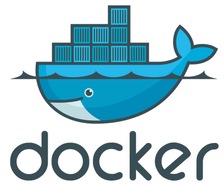
Imagem
docker run
Container rodando
Docker stop
exit
(Ctrl+D)
docker start /attach
Nova Imagem
docker commit
Container parado

Docker - Comandos



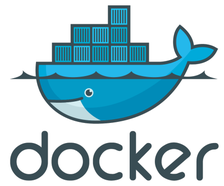
- **RUN** # criar um container a partir de uma imagem
  - docker **run** <imagem> <comando>
- **START** # inicial um container parado
  - docker **start** <id|name>
- **STOP** # parar um container
  - docker **stop** <id|name>
- **EXEC** # executar um comando dentro de um container que esteja rodando
  - docker **exec** –ti <id|name> <comando> (executa um processo dentro de um container que esteja funcionando)
  - Exemplo: docker exec –ti nome_docker bash

Docker - Limites



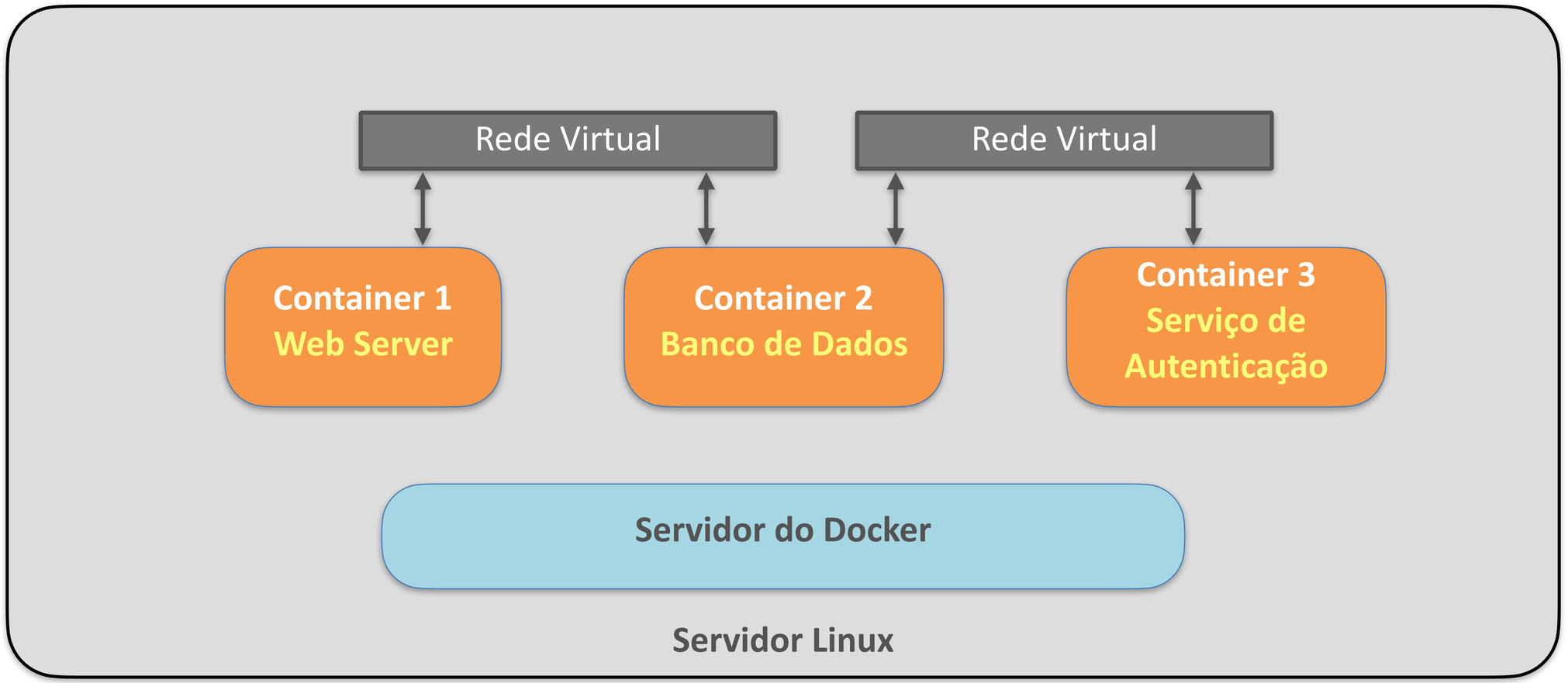
- Limites de memória
  - `docker run **--memory** maximum-allowed-memory command`
- Limites de CPU
  - `docker run **--cpu-shares** (relativo a outros containers)`
  - `docker run **--cpu-quota** (limite geral)`

Docker - Boas práticas



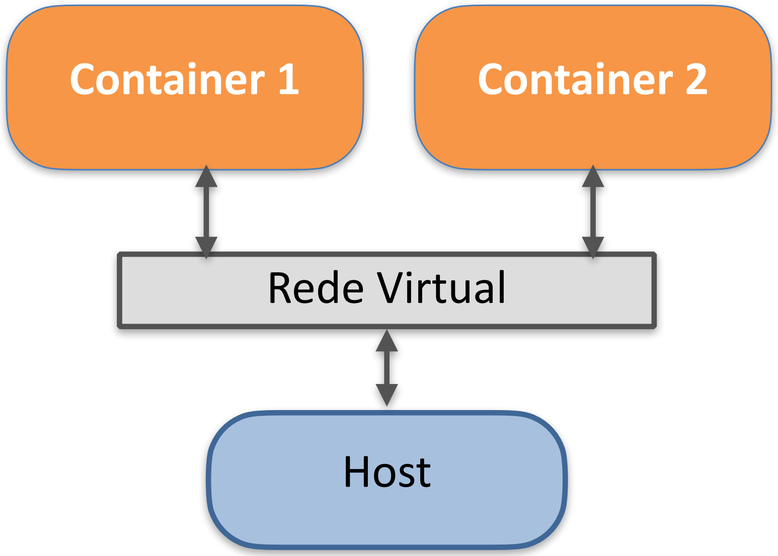
- Não carregar dependências ao iniciar um container (as dependências podem não mais existir)
- Não deixar arquivos importantes em containers parados.
- Não deixar senhas em um Dockerfile.
- Incluir instaladores em seu projeto (ao longo dos anos os mesmos instaladores podem não estar mais disponíveis)
- Construa sempre uma imagem do zero a partir de um Dockerfile
- Evitar “*Golden Images*”
- Labels
- Mantenedores (Dokerfile)

Docker - Rede

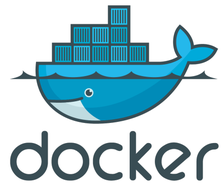

Rede Virtual
Rede Virtual
Container 1
Web Server
Container 2
Banco de Dados
Container 3
Serviço de
Autenticação
Servidor do Docker
Servidor Linux

Docker - Portas

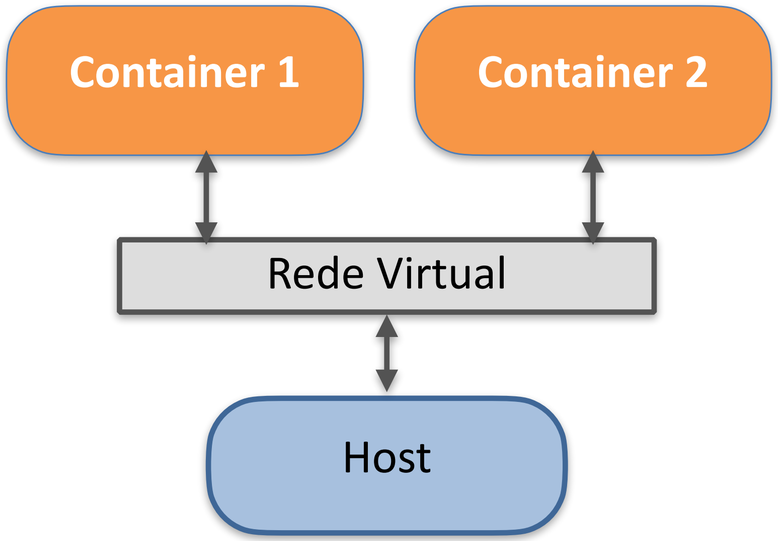
- `docker run -it --name my_ubuntu -p 8003:8000 ubuntu bash`



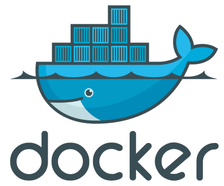
Docker - Redes Privadas



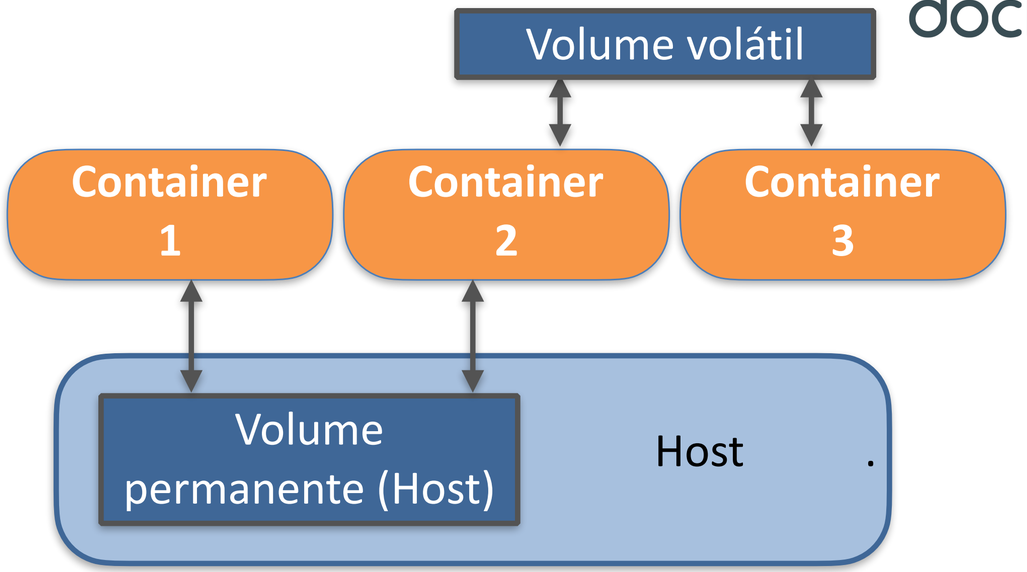
- `docker **network create** rede_exemplo`
- `docker run --rm -ti --net=rede_exemplo --name servidor ubuntu bash`
- `docker run --rm -ti --link servidor --net=rede_exemplo --name cliente ubuntu bash`



Docker - Discos Virtuais

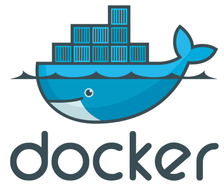


- **Volume**
  - Permanentes
  - Efêmeros



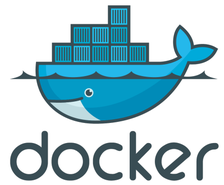
- Comandos
  - `docker run -ti -v <pasta_local>:<pasta_compartilhada> <imagem> <comando>`
  - `docker run -ti -v /users/renato/pasta:pasta_interna ubuntu bash`

Docker - Volume Compartilhado



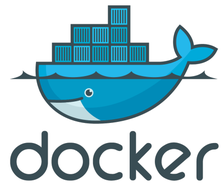
- Volume **efêmero** compartilhado entre os *containers* que estão rodando no momento.
- São destruídos depois que todos os *containers* são encerrados.
- ***Container 1***
  - `docker run -ti <pasta_compartilhada> imagem comando`
  - `docker run -ti pasta_interna ubuntu bash`
- ***Container 2***
  - `docker run -ti -volumes-from <container_1> imagem comando`

Docker - Repositórios



- **Registries**
  - Gerenciam e distribuem imagens
  - Busca, Upload, Download, etc.
- **Exemplos:**
  - Local
  - Docker Hub
  - Amazon Elastic Container Registry (ECR)
  - Google Container Registry (GCR)
  - Azure Container Registry (ACR)
  - Private Docker Registry
  - Gitlab Container Registry

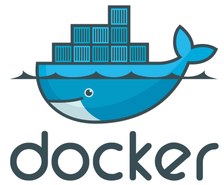
Docker - Repositórios



- **Registry - Comandos**
  - docker **login**
  - docker **search**
  - docker **pull**
  - docker **push**

- **Arquivamento local**
  - docker **save**
  - docker **load**

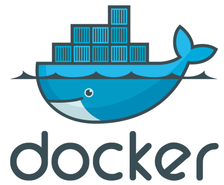
Dockerfile



- IaC - Infrastructure as Code
  - O Dockerfile descreve a receita de como construir uma imagem de um container.
  - ***Cuidado!*** o Dockerfile não é um ***shell script***
  - Cada linha do Dockerfile é executada separadamente e gera uma nova imagem;
  - O estado de uma linha para a próxima não é preservado.
  - A execução de cada linha do Dockerfile é salva em estágios e, caso não haja mudança desde a última ***build***, a linha é pulada (*cache*).

Ref. https://docs.docker.com/engine/reference/builder/

Dockerfile

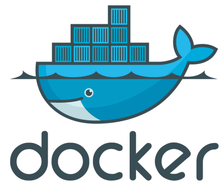


- Comando para construir a imagem

```
docker build -t <nome_da_imagem_resultante> .
```

– “." Local do Dockerfile

Dockerfile - Camadas



Dockerfile

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

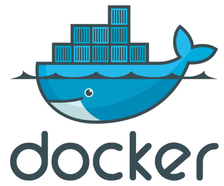
build

```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
 ---> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
 ---> Running in 5090fde23e44
 ---> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
 ---> Running in 2987746b5fba
 ---> 86c81d89b023
Removing intermediate container 2987746b5fba
Step 4 : COPY package.json /usr/src/app/
 ---> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
 ---> Running in 31ee9721cccb
 ---> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
 ---> 995a21532fce
Removing intermediate container a3b7591bf46d
```

Run

Ref. https://medium.com/@jessgreb01/digging-into-docker-layers-c22f948ed612

Dockerfile - Camadas



Dockerfile

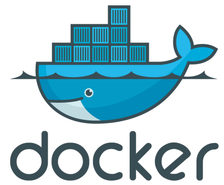
```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

build

```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
 ---> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
 ---> Running in 5090fde23e44
 ---> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
 ---> Running in 2987746b5fba
 ---> 86c81d89b023
Removing intermediate container 2987746b5fba
Step 4 : COPY package.json /usr/src/app/
 ---> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
 ---> Running in 31ee9721cccb
 ---> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
 ---> 995a21532fce
Removing intermediate container a3b7591bf46d
Step 7 : EXPOSE 8080
 ---> Running in fddb8afb98d7
 ---> e9539311a23e
```

Run

Dockerfile - Camadas

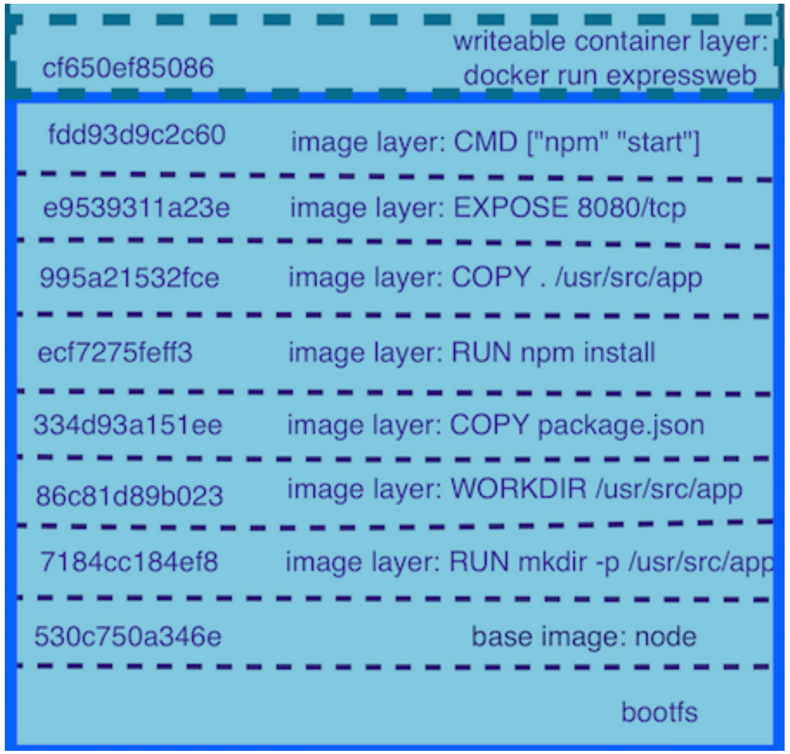



```
app

cies
/src/app/
```

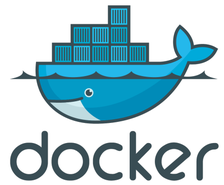
build

```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
 ---> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
 ---> Running in 5090fde23e44
 ---> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
 ---> Running in 2987746b5fba
 ---> 86c81d89b023
Removing intermediate container 2987746b5fba
Step 4 : COPY package.json /usr/src/app/
 ---> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
 ---> Running in 31ee9721cccb
 ---> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
 ---> 995a21532fce
Removing intermediate container a3b7591bf46d
```

Run



Dockerfile



- Construindo um Dockerfile

  FROM <image> # define a imagem de origem

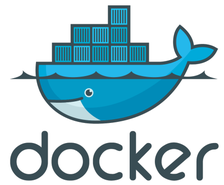
  RUN <comando> # define o comando a ser executado no shell

  ADD <arquivos> # Adiciona arquivos a um local específico no container

  CMD <comando>

  ENV <variável> # configura variáveis de ambiente

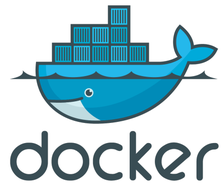
Dockerfile - **FROM**



- Imagem inicial de referência
- Geralmente a primeira linha do Dockerfile

```
FROM  ubuntu:14.04
FROM  Debian:latest
FROM  alpine
```

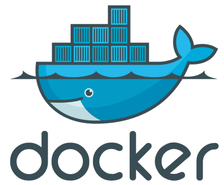
Dockerfile - **RUN**



- Executa um comando a partir do shell

```
RUN  unzip   pacote.zip    /opt/pacote/

RUN apt-get update -qq && apt-get install -y \
    build-essential \
    nodejs \
    mysql-client
```

Dockerfile - **ADD**



- Adiciona arquivos locais:

```
ADD arquivo.txt /arquivo.txt
```

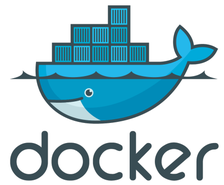
- Adiciona conteúdo de pacotes zip:

```
ADD pacote.tar.gz /pasta/
```

- Adiciona arquivos em URLs:

```
ADD https://dl.repo.com/projeto.rpm /projeto/
```

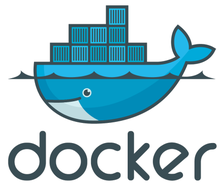
Dockerfile - **ENV**



- Configura variáveis de ambiente pela duração do Dockerfile e na imagem resultante:

```
ENV DB_HOST=db.prod.projeto.com
ENV DB_PORT=5522
```

Dockerfile - **ENTRYPOINT** e **CMD**



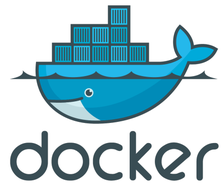
- O **CMD** especifica um comando **padrão** a ser executado sempre que o container for executado.
- Caso o usuário passe um outro comando ao executar o container, este será sobrescrito.

```
CMD [“/usr/bin/wc","--help"]
CMD echo “Hello world!”
```

- O **ENTRYPOINT** é utilizado sempre que se deseja que um comando padrão seja executado com argumentos, ou seja, permite usar o container como um binário onde se passam somente os argumentos

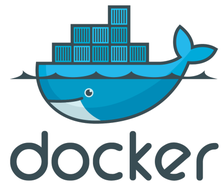
```
ENTRYPOINT ["mysql"]
```

Dockerfile - **Exemplo**



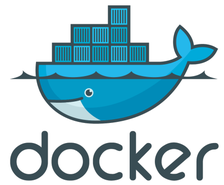

```
FROM python:3.6

RUN apt-get update && \
    apt-get install -y postgresql \
                       postgresql-client \
                       libpq-dev\
                       cron


WORKDIR /smi-master

COPY . /smi-master

RUN pip install --no-cache-dir -r requirements.txt
```

Dockerfile - **Otimização de Espaço**



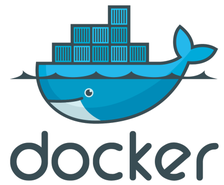

```
FROM ubuntu:18.04
RUN apt-get update && \
    apt-get install -y curl
RUN curl https://fga.unb.br | wc -c > fga-size
CMD echo Tamanho da Home da FGA; cat fga-size
```

Dockerfile - **Otimização de Espaço**



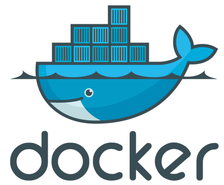

```
FROM ubuntu:18.04 as builder
RUN apt-get update && \
    apt-get install -y curl
RUN curl https://fga.unb.br | wc -c > fga-size

FROM alpine
COPY --from=builder /fga-size /fga-size
CMD echo Tamanho da Home da FGA; cat fga-size
```

Orquestração



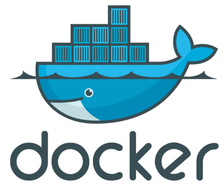
- Gerenciar um grupo de containers para funcionarem juntos;
- Permitir que os containers se comuniquem dinamicamente;
- Iniciar e reiniciar containers em caso de falhas;
- Alocação de recursos

Orquestração



- **Docker Compose**
  - Orquestração em máquina local
  - Adequado para teste e desenvolvimento
  - Gerencia containers, volumes, redes, etc.

Docker-compose (Exemplo)




```
version: '3.5'

services:
    master:
        container_name: master
        build: .
        env_file: dev-env
        command: ["sh", "scripts/start.sh"]
        ports:
            - 8001:8001
        volumes:
            - .:/smi-master
        depends_on:
            - db

    db:
        container_name: postgres_master
        image: postgres:9.6
        ports:
            - 8008:8008
        env_file: dev-env
        volumes:
            - master-pg-data:/var/lib/postgresql/data

volumes:
    master-pg-data: {}
```