Entendendo o Docker

Aprenda a falar Baleiês

Na União Soviética, o Baleiês fala você!

Quem sou eu?

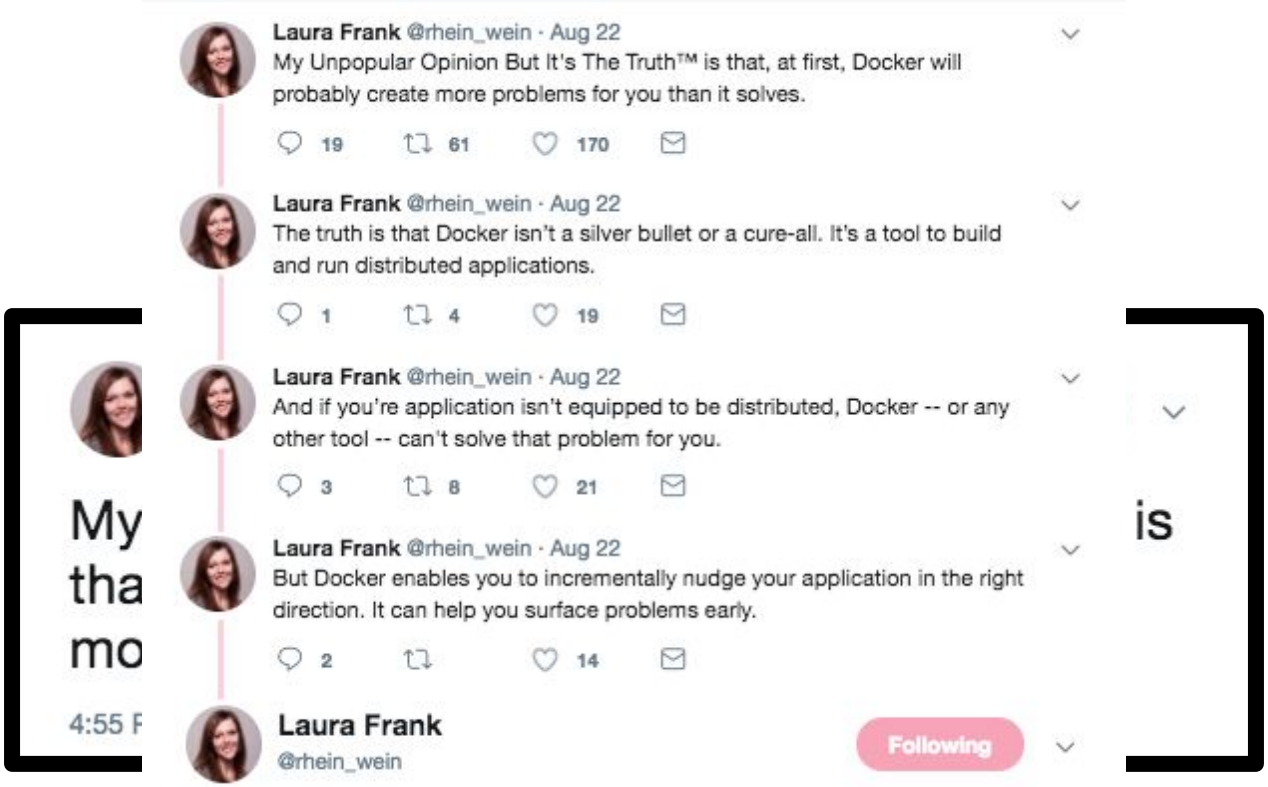
Formado em Eng Comp pela USP em 2008.

Já atuei como desenvolvedor Back End, Mobile, Líder Técnico, Gerente de Projetos e de Contas.

Sou sócio [e ex-CTO] do CasaeCafe. Atualmente sou Tech Lead na Arquivei.

Porque aprender a usar Docker?

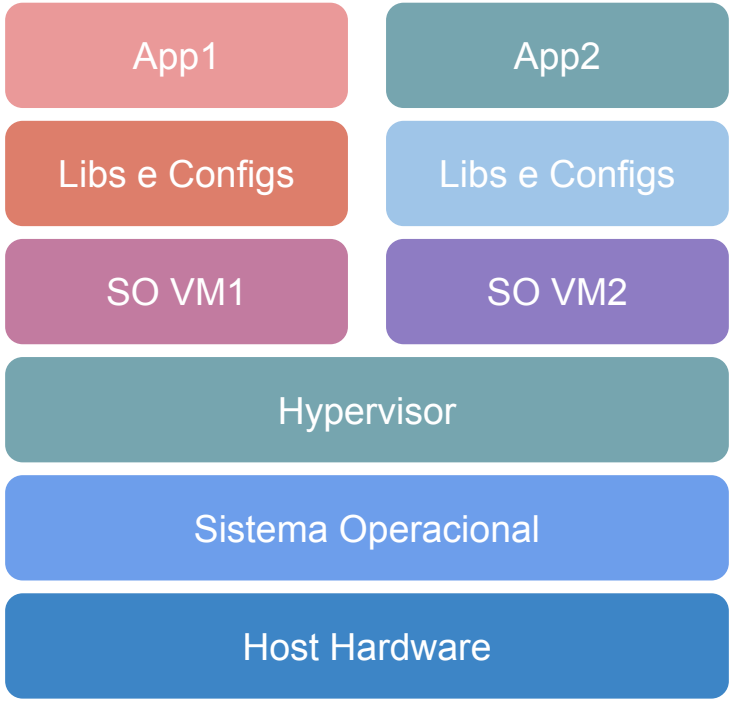

CENSURADO


Laura Frank @rhein_wein · Aug 22
My Unpopular Opinion But It's The Truth™ is that, at first, Docker will probably create more problems for you than it solves.
19 61 170
Laura Frank @rhein_wein · Aug 22
The truth is that Docker isn't a silver bullet or a cure-all. It's a tool to build and run distributed applications.
1 4 19
Laura Frank @rhein_wein · Aug 22
And if you're application isn't equipped to be distributed, Docker -- or any other tool -- can't solve that problem for you.
3 8 21
Laura Frank @rhein_wein · Aug 22
But Docker enables you to incrementally nudge your application in the right direction. It can help you surface problems early.
2 14
Laura Frank
@rhein_wein
Following
So start with the bare minimum! Just get your app running in a container! No need to blow away everything and start new. Just evolve.
4:58 PM - 22 Aug 2017

Virtualização

Como funciona uma Máquina Virtual?

- Quais as vantagens de utilizar uma VM ao invés de uma máquina física?
- Em quais situações a VM é melhor?
- Como uma VM provê essas vantagens?



Camadas de um sistema virtualizado com VM

Dificuldades

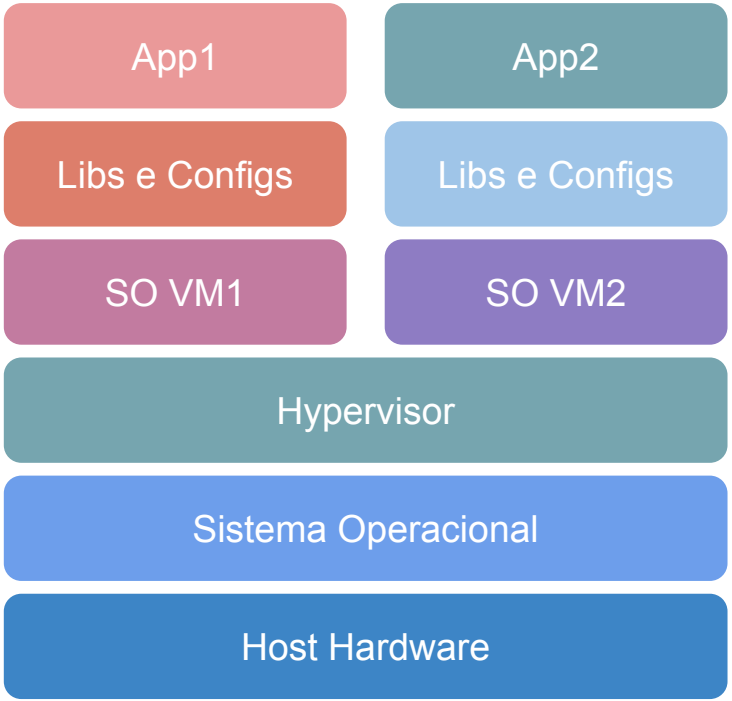
- Overhead de **processamento** e consumo de **memória**
- Tempo de subida de alguns **minutos**
- **Imagens grandes**, pois possuem toda a instalação do sistema operacional
- Baixo **reuso**
- Execução de **diversos processos** simultâneos, dificultando debug
    - Exemplo: Máquina Virtual instável
- Dificuldade de **orquestrar** muitas VMs

LinuX Containers

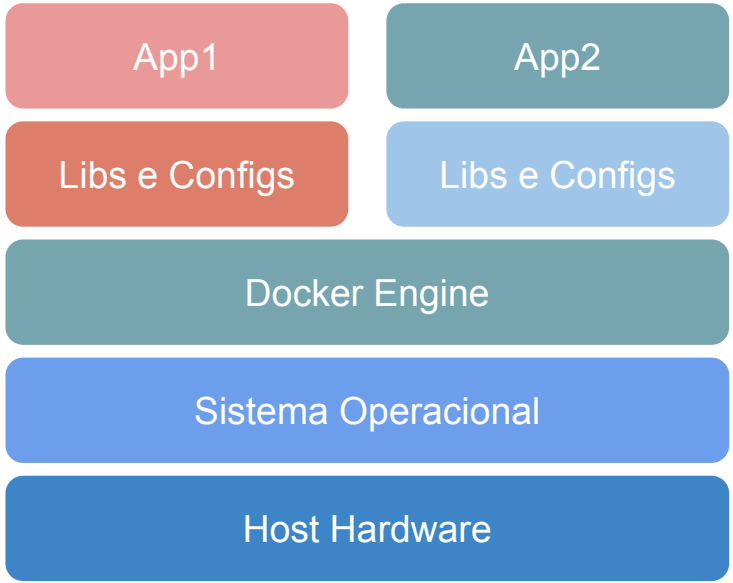
- Compartilhamento de Kernel
- Isolamento de Árvore de Processos
- Isolamento de pastas ~ *chroot*
- Isolamento de consumo de consumo de recursos ~ *crgroup*
- Isolamento de rede

Docker

- Facilitador para utilização de Linux Containers
- Toolset para gerenciamento de containers
- Grande ecossistemas



Máquina Virtual



Docker

Container vs Imagem

Imagem

- Template do sistema de arquivo base para o container
  - Contém todos os arquivos necessários para iniciar containers
- Criação a partir de build do Dockerfile [~~ou commit de um container~~]
- Armazenamento em repositórios de um Registry
  - Versionamento de imagens a partir de tags
- Permite variações
  - Diferentes SOs(Ex: Apline)

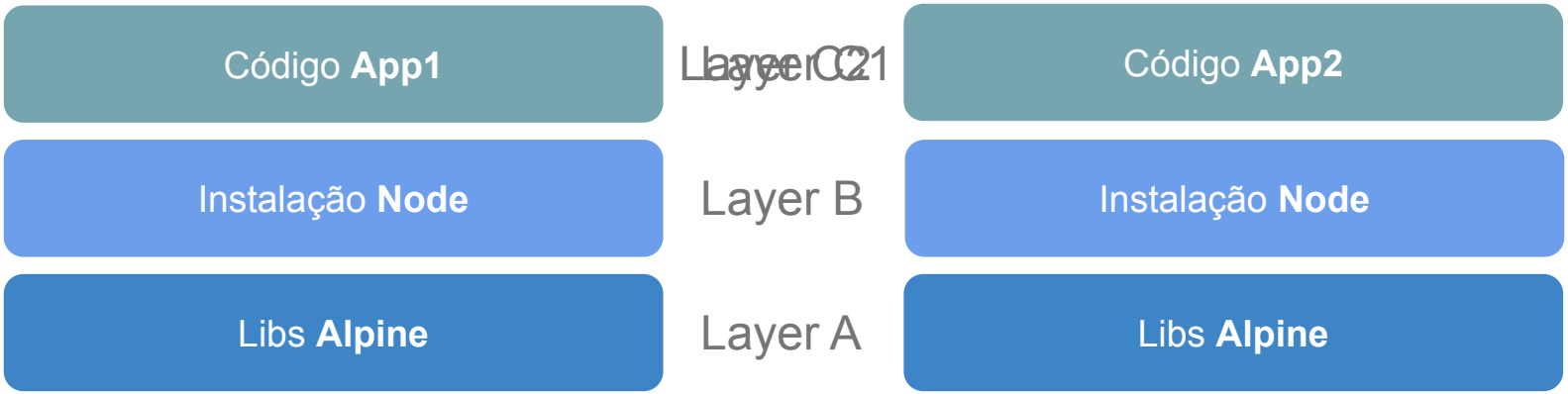
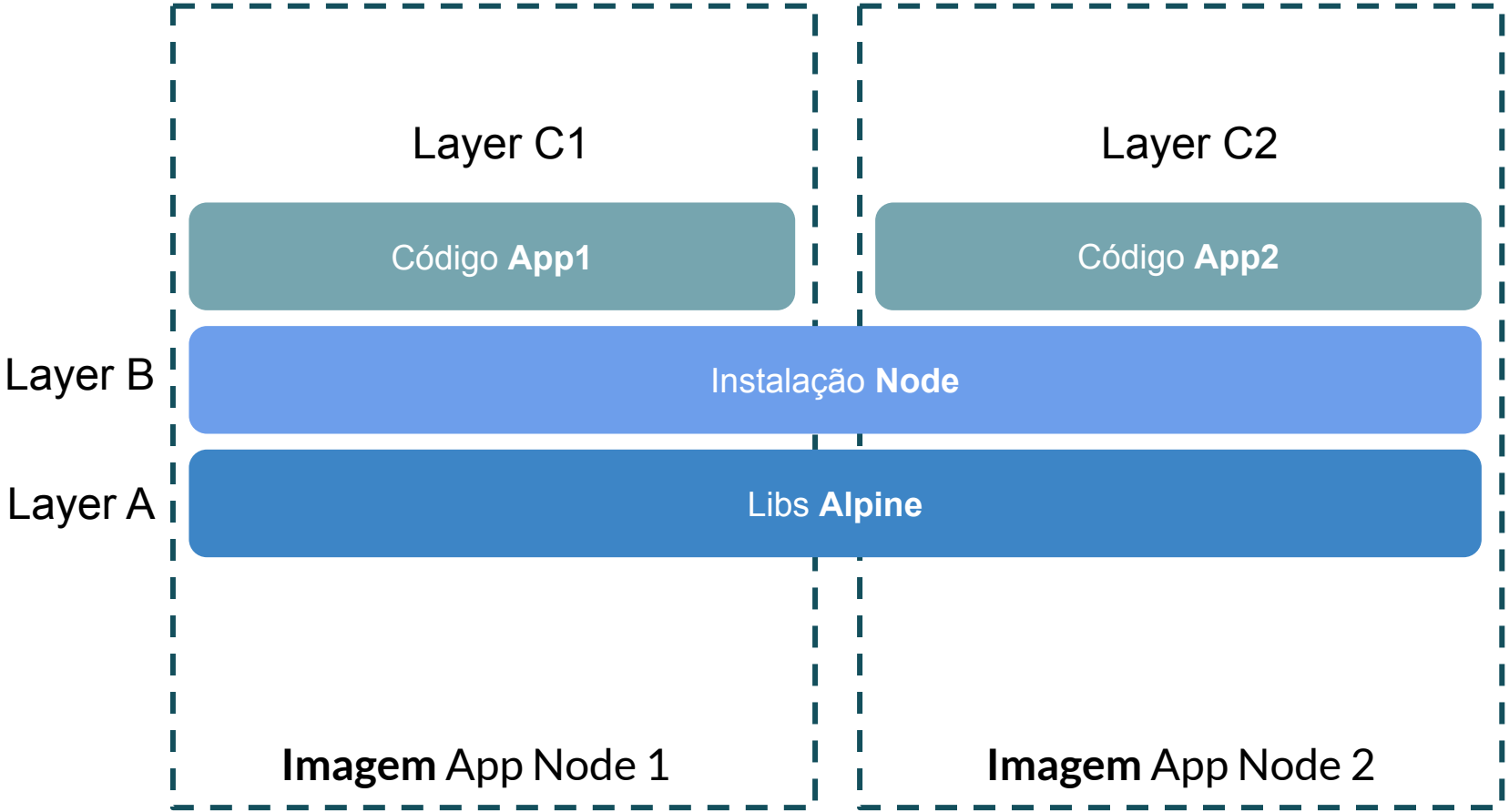


Imagem App Node 1

Imagem App Node 2


Layer C1
Layer C2
Código App1
Código App2
Layer B
Instalação Node
Layer A
Libs Alpine
Imagem App Node 1
Imagem App Node 2

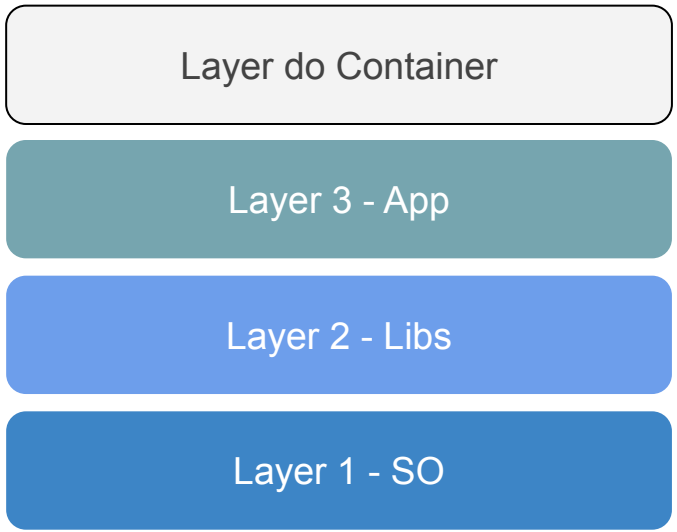
Imagens e Containers

- **Imagem** é o template com o qual um **Container** é criado
  - Receita e Bolo
- Todo container é criado a partir de uma imagem
- Múltiplos contaneirs podem ser baseados na mesma imagem
- Containers: Layers de uma imagem + Layer RW
- Sistema de arquivo mais comum é o AUFS (Another Union File System)
  - Copy on Write

Container

- Processo pai com sua árvore de processos.
- Sistema de arquivos isolado (chroot)
- Encapsulamento das dependências mínimas
  - Idealmente apenas um processo importante rodando
- Ambiente leve
- Criado a partir de uma imagem

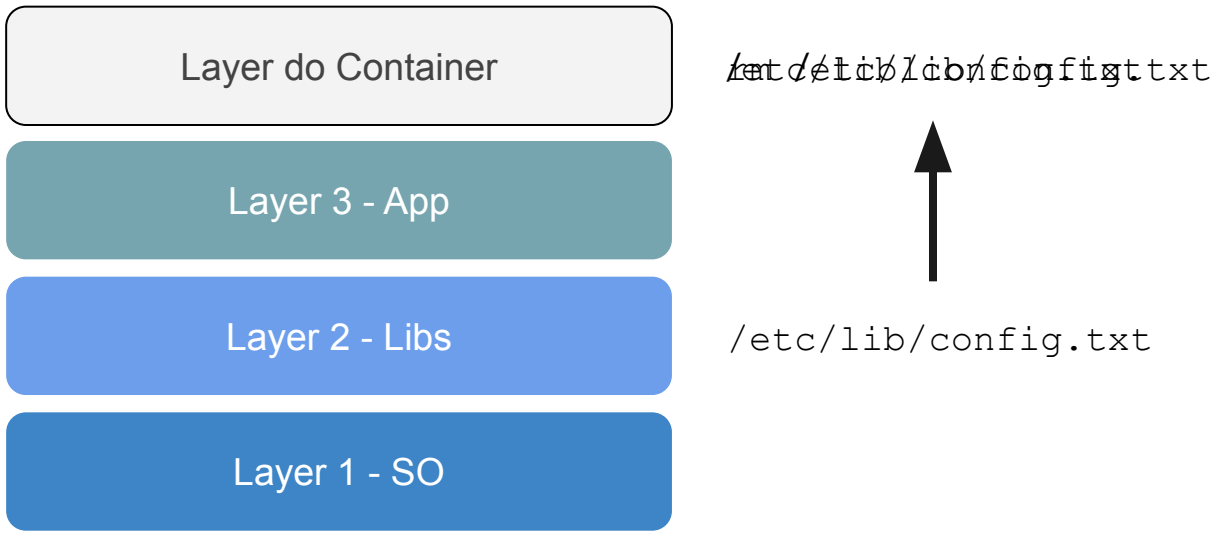
Copy on Write

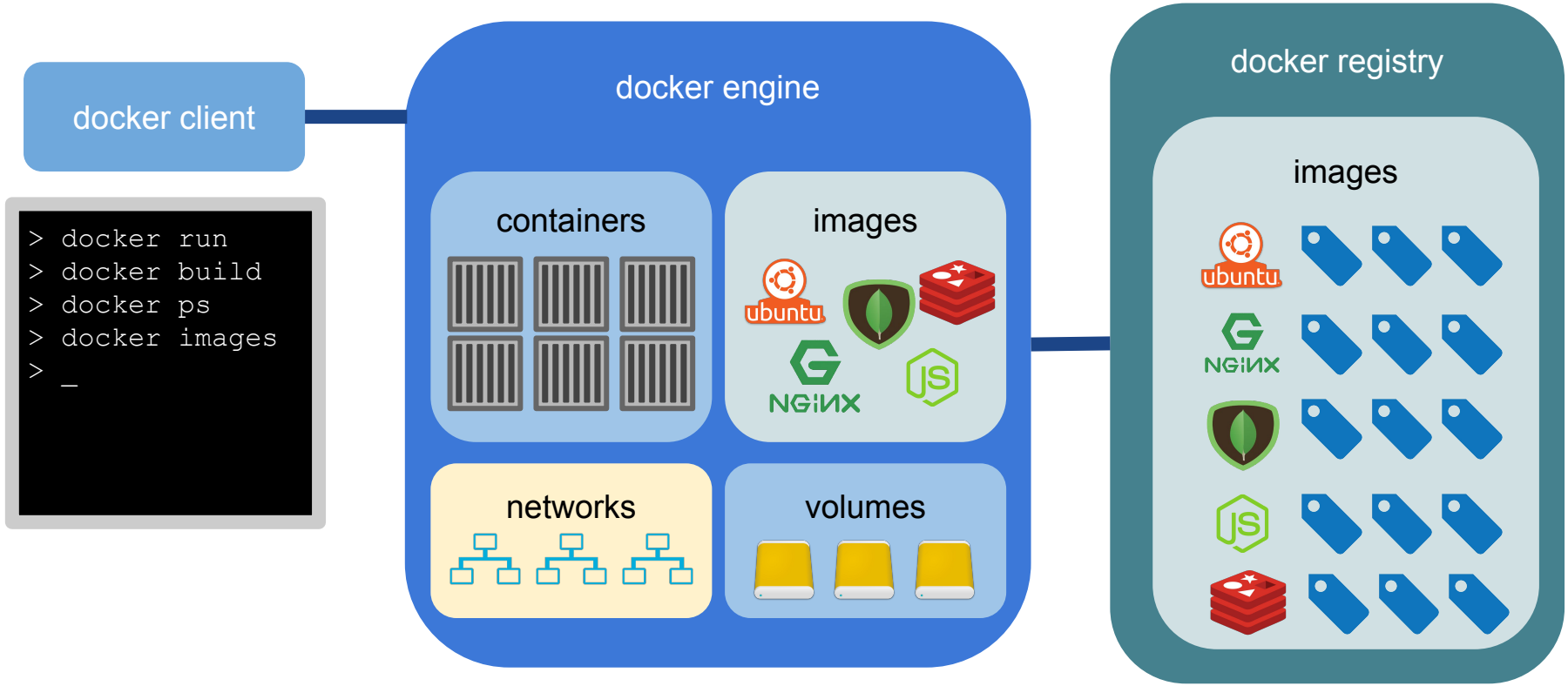


Apaga o arquivo `/etc/lib/config.txt`

Define o arquivo `/etc/lib/config.txt`

Copy on Write




docker client
> docker run
> docker build
> docker ps
> docker images
> _
docker engine
containers
images
networks
volumes
docker registry
images

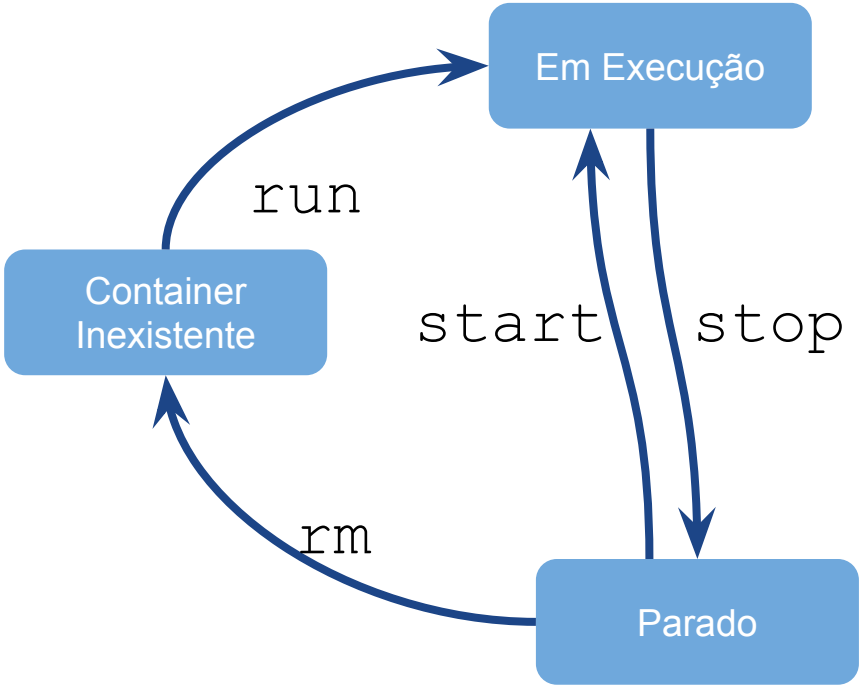
PRESS START

Olá, Oceano!!!

```
> docker container run hello-world
```

O que aconteceu aqui?

Docker Container Lifecycle



Modos de Execução

-d: execução em background (detached)
-it: iterativo
--rm: remover ao parar

Exemplos

bash no *Ubuntu*

Executar *NGINX*

Rodar o *MongoDB*

```
> docker container run -it ubuntu
bash


> docker container run -p 8080:80 -d
nginx


> docker container run --name c_mongo
-d -v $PWD/data/db -d mongo
> docker exec -it c_mongo mongo
```

Aprendizado

- Para acessar portas de dentro do container, precisamos mapeá-las no HOST
- O container deve ser stateless, pois a qualquer momento pode ser reiniciado
- Os arquivos do container devem ser mapeados para um volume no HOST
- Podemos usar comandos exec para "entrar" no container

```
$ docker container run

$ docker container exec
```

Multiplos containers

Juntando tudo! :)

Vamos fazer um serviço **REST** em **Node.js** que escreve e lê documentos no **MongoDB**

Código no Repositório:

https://github.com/rafapg/docker_semcomp2019

Passo a Passo #1 _MongoDB

Precisamos inicializar o *MongoDB*, que será nosso banco de Dados.

Nosso container tem algumas especificidades:

- Deve ser *detached*
- Precisamos mapear o *volume* para não perder os dados
- É conveniente dar um *nome* para o container

```
$ docker run --name c_mongo -v $PWD:/data/db
    --rm -d mongo
```

Passo a Passo #2 _Dependências

Agora vamos ao serviço *Node.js*.

- Código no repositório deve ter sido clonado
- Agora vamos instalar as dependências do projeto. Para isso precisamos de ter o *npm* instalado... Será mesmo???

```
$ docker container run --rm -it -v $PWD:/usr/src/app
    -w /usr/src/app node npm install
```

Passo a Passo #3 _Server Rest

Como diria o Waze: *"Tudo pronto? Vamos!"*

Vamos inicializar o nosso servidor. O que precisamos levar em conta?

- O servidor deve ser acessível via alguma porta (p.ex. 8080)
- Mapeamento do diretório do projeto para dentro do container (*volume*)
- Definir a pasta de trabalho que o *Node.js* vai utilizar

```
$ docker container run --rm --name node_server
    -v $PWD:/usr/src/app  -w /usr/src/app
     -d -p 8080:3000 node npm run start
```

Passo a Passo #4 _Erro de Conexão com DB

Putz... Deu ruim!

MongoError: failed to connect to server [c_mongo:27017] on first connect

- Temos que configurar o IP do Container *c_mongo* no Mongoose

```
$ docker container inspect c_mongo -f
"{{".NetworkSettings.Networks.bridge.IPAddress"}}"
```

- Na string de conexão do arquivo *server.js* do projeto

```
mongoose.connect('mongodb://[IP]/opensanca);
```

Passo a Passo #5 _Teste dos Serviços

Vamos testar as funcionalidades.

- Criar alguns Minicursos
- Listar todos os Minicursos
- Apagar um Minicurso
- Obter um Minicurso específico

IP do Mongo *Hard Coded :(*

Em um cenário hipotético que temos diversos serviços acessando o *MongoDB*, imagine o caos que deve ser configurar todos para o IP do container *MongoDB* do ambiente.

Não parece muito prático, Né?!

Vamos linkar os containers

O *docker engine* faz a função de *DNS* para os containers. Isso quer dizer que podemos, ao invés de acessar o container pelo IP, chamá-lo diretamente pelo seu nome, contanto que realizemos o link.

Como isso ficaria?

```
$ docker run --name node_server --link c_mongo -v
$PWD:/usr/src/app -w /usr/src/app -d -p 8080:3000 node
npm run start
```

Dicas e Malemolências

Como tirar proveito das funcionalidades de um container:

- Mapeamento de Volumes (pasta host <-> container)
- Copia de arquivos para dentro do container e vice-versa
- Exposição de Portas (host <-> container)
- Utilização de um container por outros containers

```
> docker container [comando]
```

`ps` - Listar (`-a` para todos)

`stop` - Parar

`start` - Iniciar

`restart` - Reiniciar

`rm` - Remover (`-f` para forçar)

`logs` - Ver logs (`-f` para tail)

`inspect` - Inspecionar

`exec` - Executar comandos

Let´s bake a Image

Nossa primeira imagem

Comandos do Dockerfile

```
FROM <imagem>[:<tag>]

RUN <comando_shell>

COPY <path_host> <path_container>

EXPOSE <porta>

ENV <chave> <valor>

WORKDIR <path_container>

ENTRYPOINT <comando_shell>
```

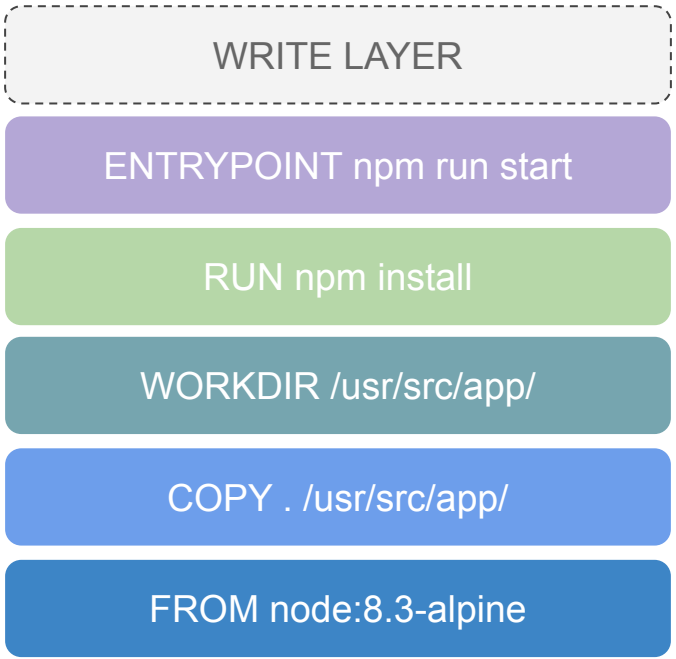
Uma imagem do Serviço *Node.js*

Quando não funciona de primeira

Imagem Serviço *Node.js*

1. Imagem a partir de uma versão estável do node
    a. Versões do alpine deixam a imagem mais leve
2. Copiar o código fonte para o *filesystem*
3. Setar o diretório de trabalho
4. Instalar as dependências
    a. Se possível, garantir que não são usadas dependências velhas
5. Definir o comando de inicialização do container

Layers Serviço *Node.js*



Docker Compose

Docker Compose

Compose é a ferramenta do *Docker* que permite iniciar um ambiente com diversos serviços interdependentes através de um arquivo de configuração com apenas **UM** comando

https://docs.docker.com/compose/compose-file/

Compondo nosso Ambiente

Analisando o comando para executar o *MongoDb*:

```
$ docker container run --name c_mongo -v
    $PWD/data:/data/db --rm -d mongo
```

- Nome: c_mongo
- Volume: ./data -> /data/db
- Imagem: mongo:latest

Compondo nosso Ambiente

Analisando o comando para executar o *NodeServer*:

```
$ docker container run --name node_server -d --link
c_mongo -p 8080:3000 node-server:0.0.1
```

- Nome: node_server
- Link: c_mongo
- Depende: Mongo em execução
- Porta: 8080->3000
- Imagem: node-server:0.0.1

docker-compose.yml

```yaml
version: '3'
services:
  c_mongo:
    image: mongo:latest
    volumes:
      - ./node-rest-example/data:/data/db

  node_server:
    image: opensanca/minicurso-server:devconf19
    ports:
      - 8080:3000
    links:
      - c_mongo
    depends_on:
      - c_mongo
```

Executando o Ambiente

Para iniciar o ambiente:

```
> docker-compose up -d
```

Para ver os logs:

```
> docker-compose logs -f
```

Para remover o ambiente:

```
> docker-compose down
```

E o que mais?

- Redes no docker
- Volumes
- Inspecionar containers
- Clusters
  - Swarm
  - Kubernetes

Bora Revisar?

- VM vs Docker
- Imagem vs Container
- Layers de um container
- Repositorio e Registry
- Docker Client + Engine + Registry
- Container Lifecycle
- Parâmetros de Execução
  - Volumes
  - Portas
  - Modo (detached, iterativo)
- Links
- Dockerfile
- Compose

```
docker inspect -f "{{.icmc.SemComp.2019.Questions}}" minicurso
```

Obrigado!!!

mantenha contato comigo!!!!

rafapg

fb.com/rafael.girolineto

@rafakareka

rafapg

rafapg.85@gmail.com