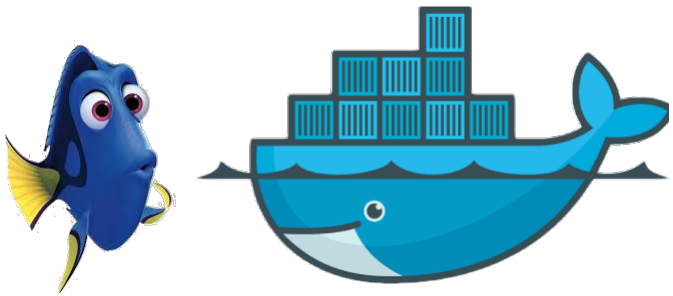
Como Falar Baleiês: Uma Introdução à Docker

Jonathas A. O. Conceição, Lucas Eduardo Bretana
Orientação: Profa. Dra. Dory

Universidade Federal de Pelotas
IPPD 2017/2



Dezembro - 2017 - Pelotas/RS

Sumário

Introdução

Docker

- Problema de desenvolver em sistemas heterogêneos;
- ”Ah, mas funcionou na minha máquina”;
- **Máquina Virtual**;
  - Sobrecusto de SO virtualizado;
  - Imagens mais pesadas.

Intodução

Pré-Requisitos

- IP e Portas;
- **VMs**;
- Editar arquivos de configuração, usando `vim` ou `emacs`;
- Familiaridade com dependências e build de código;
- Shell basics.

Breve explicação

Containers

- **Imagem**: código + OS + dependências + runtime + variaveis de ambiente + arquivos de configuração;
- Imagens são templates *read-only*, usando UFS;
- **Container**: uma instância em runtime de uma imagem;
- Os **Containers** rodam os aplicativos no `kernel` do SO hospedeiro;
  - Melhor perfomance;
  - Acesso direto ao recursos do SO;
  - Cada **Containers** roda em um processo separado.
    - Não usando mais memória do que o estritamente necessário.

Containers vs VMs

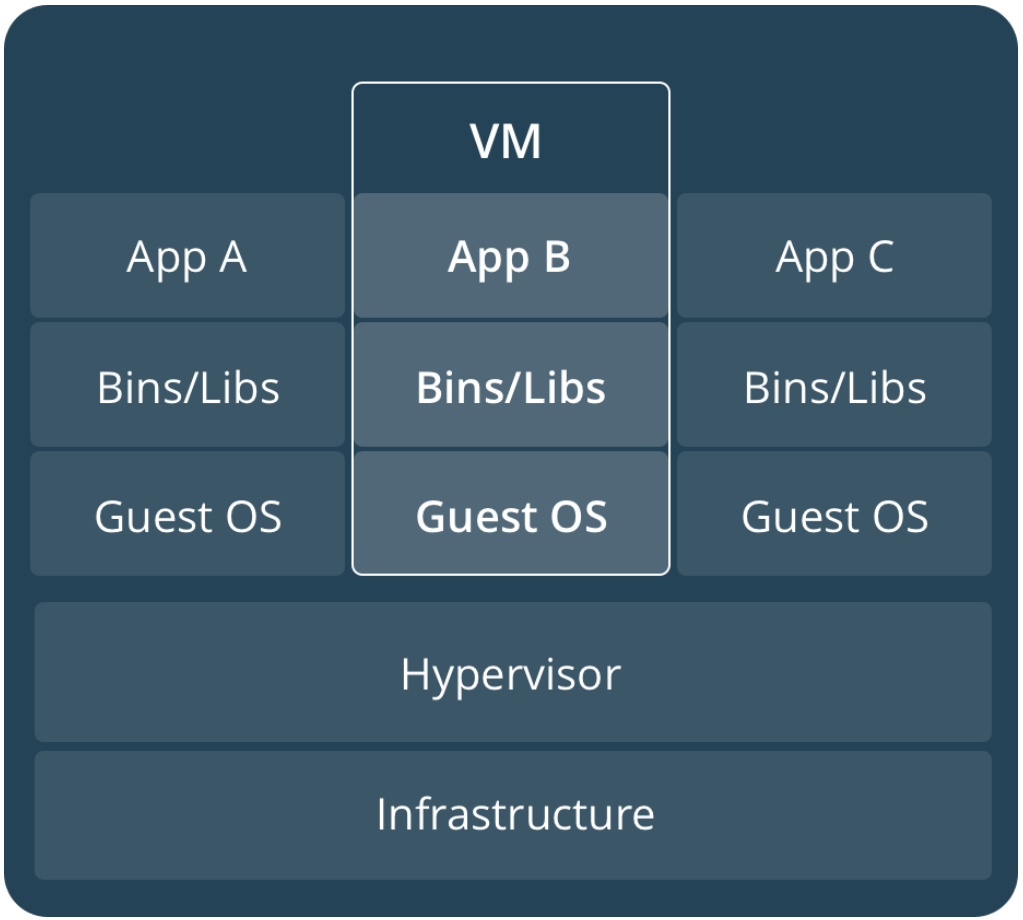
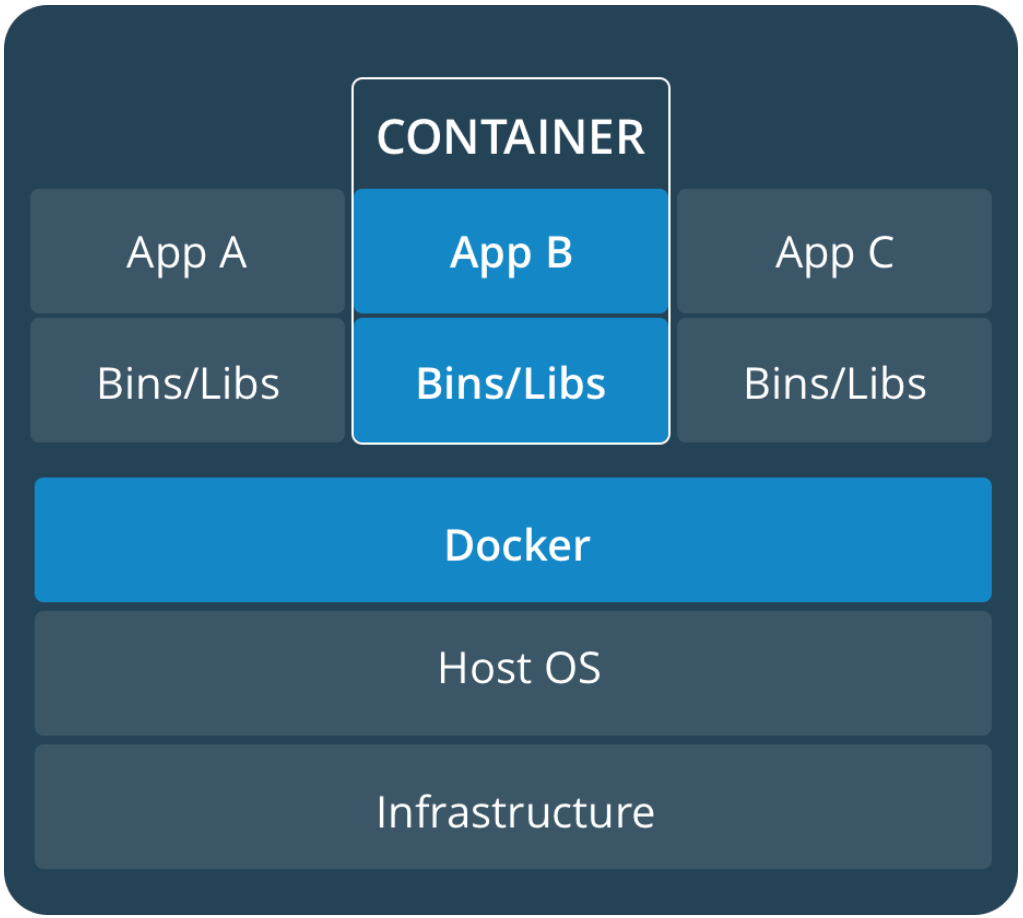


Figura: **VM** diagram

VMs

- Vários `Guests` SO;
- Isso traz um uso intensivo de recursos;
- Cada imagem de **VM** é pesada e contém tudo.

Containers vs VMs



Containers

- `Kernel` único;
- **Containers** contém o executável e suas dependências;
- Os processos rodam como processos nativos;
  - $ *docker ps*
- Roda em qualquer lugar.

Dockerfile

Iniciando um novo app

- Normalmente: instala todo o `enviroment`, bibliotecas externas e desenvolver;
  - Ambiente deve estar ”em ordem”.
- Agora: podemos baixar uma imagem portátil pronta e desenvolver e em cima de tal;
- Sem instalação!

Dockerfile

- Usado pra definir o que vai no ambiente do **Container**;
- Recursos como rede e disco ficam isolados deste ambiente;
  - Portas de acesso;
  - Arquivos que vão ”entrar”pro **Container**;
  - Uma vez configurado, roda sempre igual.
- É uma das maneiras de se criar a sua imagem.

Nuvem

Docker Hub

- Serve de repositório para as imagens;
- Imagens oficiais das distribuições;
- Por padrão as imagens são adicionadas como públicas;
- Existe limite de repositórios privados.

Docker Cloud

- Vai além do Docker Hub;
- Oferece funcionalidades para *build* e testes de projetos.

Serviço

Service

- A verdadeira maneira de rodar um *container* na produção é na forma de um *service*;
- Docker te permite ainda facilitar o processo de levantar os containers;
- Para isso vamos subir na hierarquia de uma aplicação distribuída, o *service*;
- Diferentes partes de uma aplicação são chamadas de *service*;
- Instância uma imagem e define configurações da mesma, além de definir o número de réplicas a serem rodadas;
    - 1 service $\leftrightarrow$ N containers

Docker-compose

- Um arquivo YAML que define o comportamento do *container* em produção;
- Dentro do *service* serão criadas tantas *tasks* quanto definido no número de réplicas;

```
1   version: "3"
2   services:
3     web:
4       image: username/repo:tag
5       deploy:
6         replicas: 5
7         resources:
8           limits:
9             cpus: "0.1"
10            memory: 50M
11        restart_policy:
12          condition: on-failure
13      ports:
14        - "80:80"
15      networks:
16        - webnet
```

Mão na massa

Exemplo de aplicação

- *Docker* baseado no **go:1.8** e **Redis**;
- Bibliotecas de dependências do projeto;

Ferramentas e configurações extras

Ferramentas

- *Swarns*, usando para agrupar máquinas (físicas ou virtuais) em *clusters* e facilitar a administração;
- *Stacks* são utilizados para agrupar os diferentes *services* que compõe uma aplicação e especificar condições de relacionamento entre eles;
- *Docker Cloud* pode ser utilizado para automatização de testes dos containers antes de entrarem para produção;
- API de comunicação direta com o *deamon* do Docker;

Configurações

- Configurações de rede permite que seja criado uma próprio modelo de comunicação, e.g., entres nodos;
- *Volumes* a persistência de dados não deve ser feito dentro do próprio *container*, pelo alto custo das novas imagens;
- *tmpfs* é uma estratégia pra poder ter dados que não ficam nem em *volumes* acessíveis à maquina hospedeira, nem são escritos sobre a imagem;
- A documentação oferece um suporte enorme para todo o processo de criação, desenvolvimento e *deploy* de aplicativos.

Trabalho proposto

Tutorial básico

- **https://docs.docker.com/get-started/**;

Ambiente de trabalho

- Montar seu ambiente de trabalho em uma imagem *Docker*;
- Baseado na distribuição de escolha;
- Ferramentas básicas de programação:
  - Compilador de C e *debugger* (GCC, GDB, ...);
  - Ferramentas básicas de automação e edição (Make, VIM, Emacs, ...);
  - ...

Mais informações:

https://docs.docker.com/

Mais informações:

E-mail: {jadoliveira, lebretana}@inf.ufpel.edu.br