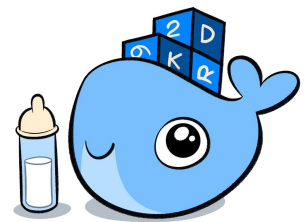


Canivete Suíço do Docker

Para iniciantes



Índice

- Apresentação
- Teoria
 - Virtualização
 - O que é Docker e como funciona?
 - Quem usa Docker? É confiável?
 - Vantagens

Índice

- Prática
- Kubernetes
- Referências / Contatos

Apresentação

- João Henrique de Oliveira Júnior
- Casado e feliz!
- Técnico em sistemas de informação (2009 - 2011)
- Bacharel em sistemas de informação (2012 - 2015)
- Bacharel mesmo! Entreguei o TCC (:
- ItFlex - Desenvolvedor de sistemas para serviços Linux
- Koder - Líder Técnico de desenvolvimento e desenvolvedor.
- Fã de starwars (A cada 10 textos que escrevo, os 10 tem gifs relacionados)
- Fã de Linux, tenho curso de administrador de sistemas Linux, e uso a mais de 9 anos
- Torço para o Cruzeiro e sou músico nas horas vagas (Violão clássico)
- Sou Cristão e escolhi seguir Jesus



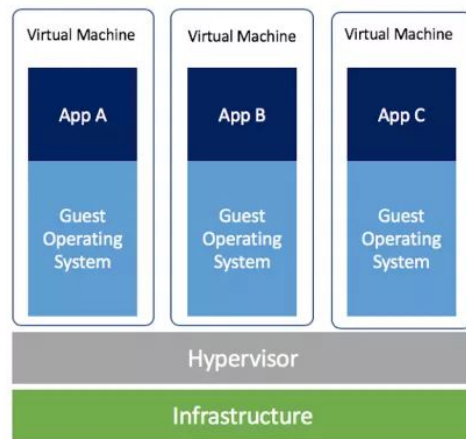
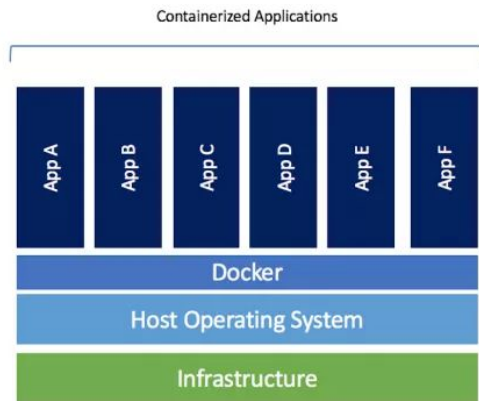
Virtualização

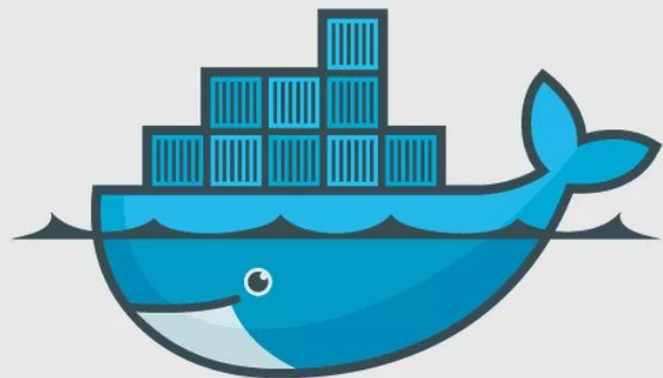
“Virtualização é uma tecnologia que permite criar serviços de TI valiosos usando recursos que tradicionalmente estão vinculados a um determinado hardware. Com a virtualização, é possível usar a capacidade total de uma máquina física, distribuindo seus recursos entre muitos usuários ou ambientes. - RedHat”

- **Vários serviços podem estar agindo de forma independente**
- **Uma máquina pode conter mais de um sistema operacional**
- **Tradicional exemplo: VirtualBox.**
- **No virtualbox um novo sistema operacional é instalado em cima do sistema hospedeiro**

O que é o Docker e como funciona?

- O Docker é uma forma de virtualização, desenvolvido inicialmente para Linux
- São virtualizados containers com serviços específicos
- Através de comandos, os containers podem se comunicar
- Compartilha o mesmo kernel e configurações base da máquina hospedeira





docker

A imagem do Docker representa exatamente essa forma de virtualizar. A baleia desempenha o papel de um navio carregando containers assim como se tem num porto qualquer. É como se fosse o docker servindo de base para várias aplicações distintas.

O que é o Docker e como funciona?

“Mas isso não é sobre infra?”

Sim, e daí?


Nos tempos que vivemos, faz muito sentido a aproximação das equipes e empresas de desenvolvimento com as de infraestrutura, pois a infraestrutura e a aplicação são ligadas intimamente. Além de tudo isso, você é obrigado a saber no mínimo, o setup básico dos seus sistemas.

Você conhece DevOps? Desafio aprender sobre o assunto (:

Quem usa Docker? É confiável?

- Servidores de aplicações de forma geral
- Times de desenvolvimento
- Visa
- PayPal
- BBC news
- The New York Times
- Uber
- Spotify
- Shopify
- Koder Tecnologia (#nois)

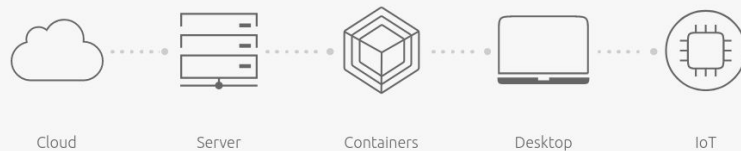
Site do ubuntu




Watch our Kubernetes webinars ›

A series of on-demand webinars to learn more about Charmed's Distribution of Kubernetes.

Ubuntu is an open source software operating system that runs from the desktop, to the cloud, to all your internet connected things




Site do Fedora



fedora
MAGAZINE


Aproveite ao m



Performing storage management tasks in Cockpit

Performing storage management tasks in Cockpit

September 6, 2019



how to build
FEDORA CONTAINERS

How to build Fedora container images

September 4, 2019

Vantagens

- Seu sistema rodando em qualquer lugar
- Terceirização de serviços simplificada em maioria dos casos
- Cada projeto pode ter sua própria configuração, não exigindo instalações diversas na máquina host
- Depuração de erros com o mesmo ambiente do servidor de produção
- Fácil integração das equipes de desenvolvimento com as equipes de infraestrutura
- Atualização de versão do projeto

Prática



Prática

- Instalando o Docker (Debian derivados)

- `sudo apt update`
- `sudo apt install docker docker-compose -y`
 - `-y` * Aceitação prévia do que será instalado
- `sudo groupadd docker`
 - Adiciona um novo grupo ao sistema, chamado docker
- `sudo usermod -aG docker $USER`
 - Adiciona seu usuário ao grupo docker
- `newgrp docker`
 - Acessa o grupo docker nesse terminal (Para você não precisar sair da sessão)
- `docker`
 - Acessa o docker

Prática

- **Hello-World do Docker**
 - `docker run hello-world`
 - Docker procurou um container chamado hello-world
 - Como não foi encontrado, ele fez download de sua base (DockerHub - Veremos no final)
 - Iniciou a execução desse container, que fez seu papel e desligou de forma automática
- **Experimentando alguns comandos do Docker**
 - `docker images`
 - Imagens que estão disponíveis de forma local no sistema
 - `docker ps`
 - Containers que estão em execução
 - **OBS:** Ambos contém um id, serão chamados no estudo de `id_container`, e `id_img`

Prática

- Experimentando alguns comandos do Docker

- `docker run (nome da imagem, id_img)`
 - Executa o container
 - Processo igual ao HelloWorld.
 - Efetua Download caso a imagem não esteja no sistema
 - `-it` -> Ativa o modo interativo
 - `docker run -it (nome da imagem, id_img)` (Comando para executar)
 - `docker run -it ubuntu /bin/bash`
- `ctrl + d`
 - Sai do container e elimina o processo
- `ctrl + p + q`
 - Sai do container e mantém em background

Prática

- Experimentando alguns comandos do Docker
 - `docker attach id_container`
 - Entra no container rodando em background
 - `docker rmi id_img`
 - remove uma imagem do docker
 - `docker rmi --force id_img`
 - remove uma imagem mesmo que ela tenha dependências

Prática

- **Vamos criar um container CentOS**
- **Não vem da família Debian, a única coisa que eles tem em comum é o shell e o kernel.**
- **É da família RHEL (Red Hat Enterprise Linux) - Uma gigante dos servidores**
- **Criando um container Centos**
 - `docker run -it centos /bin/bash`
 - `cat /etc/os-release`
 - `vi`
 - `:q!`
 - `yum update` | Brevemente será apenas `dnf update`
 - `ctrl + d`
 - `docker run -it centos cat /etc/os-release`

Prática

- **Vamos criar um container Debian**
- **Pai do Ubuntu, Kali, PureOs e avô legítimo do Mint, Xubuntu, Kubuntu, Budgie, Kylin, etc...**
- **Criando um container do vovô Debian**
 - `docker run -it -p 3000:80 debian:jessie /bin/bash`
 - `-p` define a porta
 - `cat /etc/os-release`
 - `apt update && apt upgrade -y`
 - `apt install -y apache2 php5`
 - `service apache2 start`
 - **Acesse seu localhost:3000**

Server rodando na porta 3000!



Prática

- **Criando um container Debian**
 - `ctrl + d`
 - `docker run -it -p 3000:80 debian:jessie /bin/bash`
 - `service apache2 start`

Apache sumiu... Docker nem era tão bom assim



Prática

- **Criando um container Debian**

- `apt update && apt install -y apache2 php5`
- `ctrl + p + q`
- `docker ps`
- `docker commit id_container meudebian:1.0`
- `docker stop id_container`
- `docker run -it -p 3000:80 id_img /bin/bash`
- `service apache2 start`
- `Acesse localhost:3000`
- `ctrl + d`
- `docker run -d -it -p 3000:80 meudebian:1.0 /bin/bash`
- `docker exec container_id service apache2 start`

Prática

- Seu Debian está salvo e sempre que você precisar, esse serviço estará ali
- Quanto o Debian está consumindo de memória?
 - `docker stats container_id`
 - `docker stats ->` Mostra todos os containers.
- Quero saber mais!
 - `docker inspect container_id`
 - `docker inspect container_id | egrep "IPAddress"`
 - `curl ipaddress:80`
 - `docker stop container_id`
 - `docker start container_id`
 - `docker kill container_id`

Prática

- Os containers conversam?
- Certamente que sim! Não faz sentido levantarmos serviços como servidor web e servidor de banco de dados, sem que eles tenham integração, não é?
- Os containers podem se comunicar através da rede
- “Linkando” containers:
 - `docker run -it -d --name centos-machine centos /bin/bash`
 - A tag `--name`, gera um nome a sua escolha para ser reconhecido na rede
 - `docker run -it --name debian-machine --link centos-machine debian /bin/bash`
 - `ping centos-machine`
 - `ping debian-machine`

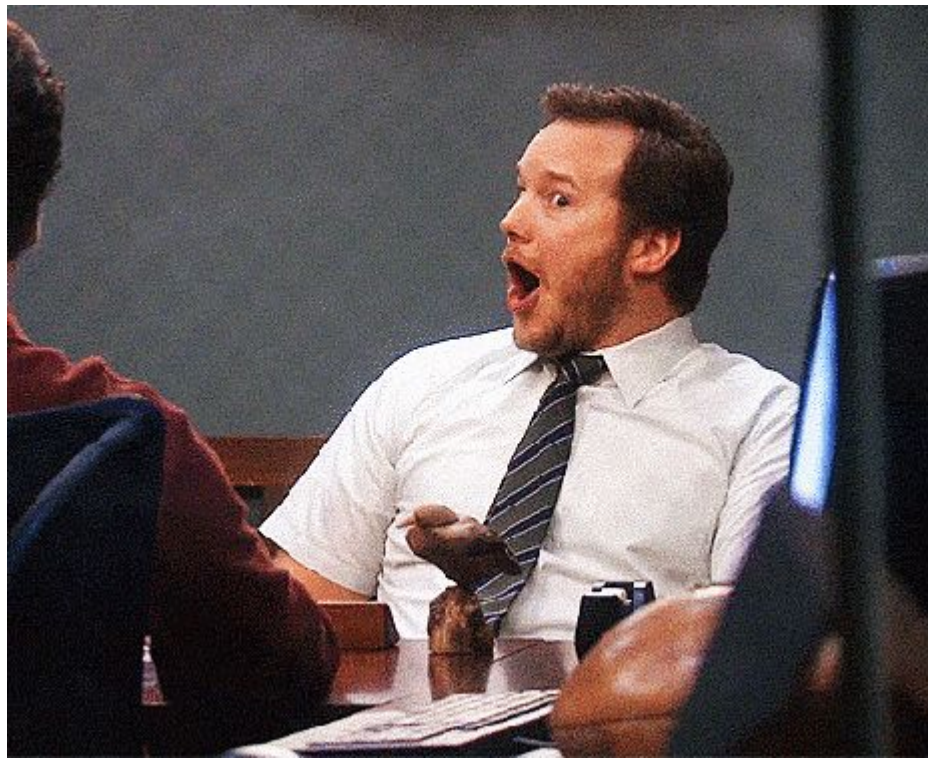
Prática

- **Interagindo com o docker da máquina Host**

- Na máquina local: `cd ~/`
- `touch teste.txt`
- `echo "olá, vim do host" >> teste.txt`
- `cat teste.txt`
- `docker ps`
- `docker cp ~/teste.txt id_container:/teste.txt`
- `docker exec id_container cat teste.txt`
- `docker exec id_container ls -la | egrep "*teste*"`

Prática

- **Interagindo com o docker da máquina Host**
 - `docker kill id_container`
 - `cd ~/`
 - `mkdir teste`
 - `docker run -d -it -v ~/teste:/teste container_id /bin/bash`
 - `docker ps`
 - `docker exec container_id ls -a | egrep “*teste*”`
 - `docker exec container_id ls teste`
 - `mv teste.txt teste`
 - `docker exec container_id ls teste`



Prática

- **Até aqui você já sabe:**
 - Criar imagens
 - Rodar containers
 - Conectar containers via rede
 - Inserir conteúdo no container
 - Compartilhar arquivos com o container
 - Expor uma porta na máquina local
 - Ver detalhes sobre o container
 - Ver o consumo do container

Prática

Estamos quase prontos para aprender sobre **Dockerfile! Mas antes precisamos esclarecer algo sobre desempenho.**

Rodar ao mesmo tempo muitos containers, apesar de ser mais rápido e eficiente do que virtualizar da forma convencional, pode causar problemas de desempenho (embora raro). É preciso aprender a controlar o quanto nossos containers irão consumir, mas isso é possível?



Prática

- `docker run -it -d --memory="256m" debian /bin/bash`
- `docker ps`
- `docker stats`
- `--memory-reservation="50m"`
 - Essa instrução indica que se os 256m forem atingidos, ele pode estender por mais 50m no máximo, porém o container fará máximo esforço para não passar dos 256m

Prática – O Dockerfile

- É importante você saber que ainda teríamos várias coisas do Docker para ver
 - Fazer login no dockerhub
 - Exportar/Importar imagens
 - etcs...
- Dockerfile é uma maneira de construir suas imagens
- É um tipo de script que o Docker é capaz de interpretar
- Tem sua própria sintaxe
- O comando que lê esse arquivo é o ***docker build***
- O nome precisa ser Dockerfile *(Na verdade não, mas é preciso configurar para que não seja. Quase todo mundo vai usar esse nome mesmo)*

Prática

- **Crie um arquivo chamado Dockerfile em algum diretório qualquer para nosso experimento**
- **Comandos principais**
 - **FROM -> De que imagem você está se baseando para construir a sua imagem**
 - **COPY -> Copia arquivo para a container**
 - **RUN -> Executa um comando no container**
 - **VOLUME -> Compartilha um diretório manipulável**
 - **WORKDIR -> Define o diretório padrão quando você acessa o container**
 - **EXPOSE -> Expõe uma porta para ser acessada**
 - **CMD -> Define um comando que sempre vai rodar como você inicializar seu container**
 - **USER -> Define o usuário default que vai ser logado ao inicializar o container**
 - **ENTRYPOINT -> Define um comando padrão para inicializar com o container**



Jota-info Servidor apache2 + mysql + php7.0 em Docker

710eb98 16 days ago

[1 contributor](#)

29 lines (23 sloc) | 1.37 KB

[Raw](#)[Blame](#)[History](#)

```
1 FROM debian
2
3 RUN apt-get update && apt-get install -y --no-install-recommends apt-utils && \
4     apt-get upgrade -y && \
5     apt-get install -y apt-transport-https ca-certificates curl software-properties-common gpg apache2 apache2-utils wget && \
6     wget -q https://packages.sury.org/php/apt.gpg -O- | apt-key add - && \
7     echo "deb https://packages.sury.org/php/ stretch main" | tee /etc/apt/sources.list.d/php.list
8
9 RUN apt-get update && apt-get install -y php7.0 php7.0 libapache2-mod-php7.0 php7.0-mysql php-common php7.0-cli php7.0-common ph
10
11 RUN echo "<Directory /var/www/>" >> /etc/sysctl.conf && \
12     echo "Options Indexes FollowSymLinks" >> /etc/sysctl.conf && \
13     echo "AllowOverride All" >> /etc/sysctl.conf && \
14     echo "Require all granted" >> /etc/sysctl.conf && \
15     echo "</Directory>" >> /etc/sysctl.conf && \
16     rm /var/www/html/index.html && \
17     a2enmod php7.0 && \
18     sed -i -e 's/display_errors Off/display_errors On/g' /etc/php/7.0/apache2/php.ini && \
19     sed -i -e 's/display_startup_errors Off/display_startup_errors On/g' /etc/php/7.0/apache2/php.ini && \
20     sed -i -e 's/log_errors Off/log_errors On/g' /etc/php/7.0/apache2/php.ini && \
21     sed -i -e 's/track_errors Off/track_errors On/g' /etc/php/7.0/apache2/php.ini && \
22     service apache2 start
23
24
25 VOLUME /var/www/html/projeto
26 WORKDIR /var/www/html
27 EXPOSE 80
28 CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```



Pratique você mesmo!

- Gere um Dockerfile rodando Debian Jessie, com apache2 e php5
- Após você terminar você pode testar com esse comando:
 - `docker build -t debian:1.0 .`
 - `-t` -> vem de “tag”, veja que é possível versionar
 - `.` -> significa que você está executando o build dentro do diretório atual

Docker compose

- Docker compose é um comando a parte do docker
- Facilitador para manusear containers
- Tem sua linguagem própria, mas é muito simplista
- Seu arquivo, parecido com o Dockerfile, se chamará sempre docker-compose.yml

Docker compose

- **Comandos**

- **docker-compose build**
- **docker-compose up**
- **docker-compose down**
- **docker-compose exec**
- **docker-compose start**
- **docker-compose stop**
- **docker-compose top**

Vamos fazer esse servidor juntos



Dockerhub

- É de onde buscamos as imagens que usamos como base
- Podemos ser contribuintes também
- Estude os comandos específicos para subir seus containers
- <https://hub.docker.com>

Kubernetes

- É um sistema opensource para ajudar a escalar as suas aplicações em containers
- Maioria das distribuições Linux conhecidas possuem suporte
- Você pode testar de forma local algumas das funcionalidades do Kubernetes
- É possível fazer deploy automático também através da ferramenta
- Site principal: <https://kubernetes.io/>

Referências

<https://docs.docker.com/>

<https://medium.com/free-code-camp/a-beginners-guide-to-docker-how-to-create-your-first-docker-application-cc03de9b639f>

<https://medium.com/free-code-camp/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b>

https://www.youtube.com/watch?v=0xxHiOSJVe8&list=PLf-O3X2-mxDkiUH0r_BadgtELJ_qyrFJ

Meus contatos

E-mail da empresa: joao.henrique@koder.com.br

E-mail pessoal: j.h.o.junior12@gmail.com

Github: <https://github.com/jota-info>

Medium (minhas postagens): <https://medium.com/@joao.henrique>

Linkedin: <http://tiny.cc/ga4ccz>

Minha empresa

SOFTWARE HOUSE

Desenvolvimento de plataformas sob demanda.

LOJAS VIRTUAIS

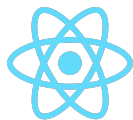
Desenvolvimento de lojas virtuais com base em Magento.

APLICATIVOS

Desenvolvimento para Android, iOS, PWA

KODER

COMO FAZEMOS?



QUER SABER MAIS?



<http://www.koder.com.br/blog>



contato@koder.com.br



Minha empresa

Site: <http://www.koder.com.br>

Blog: <http://www.koder.com.br/blog>

E-mail: contato@koder.com.br

KODER

SISTEMAS ESPECÍFICOS

Desenvolvimento de plataformas sob demanda.

LOJAS VIRTUAIS

Desenvolvimento de lojas virtuais com base em Magento.

APLICATIVOS

Desenvolvimento para Android, iOS, PWA

COMO FAZEMOS?

