TIMO CABRAL

Ambiente de desenvolvimento e deploy com Docker e WordPress

Obtenha agora, resultados incríveis para seus projetos WordPress

TIMO CABRAL

Índice

Introdução

O processo de desenvolvimento, testes, implantação de projetos desenvolvido em wordpress é uma das fases mais importantes no desenvolvimento de projetos com essa plataforma, o que nem todos os desenvolviedores sabem, é que existem ferramentas e processos seguros para o desenvolvimento e implantação de projetos envolvendo a infraestrutura ao redor de uma aplicação Wordpress.

Para mostrar esses recursos e processos que envolvem a fase de desenvolvimento e implantação, escolhemos o Docker, uma ferramenta que vem revolucuionando a forma como equipes lidam com sofware. Mostraremos neste ebook como o Docker pode melhorar os resultados finais e simplificar o processo de desenvolvimento e infraestrutura dos seus projetos com WordPress.

Começamos apresentando o que é o Docker, sua história e seus casos de uso provendo um entendimento profundo da ferramenta e a sua aplicabilidade com o Wordpress. Depois partiremos para o mão na massa, fazendo desde o famoso “ola mundo” até a implantação de um container WordPress na nuvem.

Esperamos que, após ler este material você consiga modernizar e melhorar o seu processo de desenvolvimento em seus projetos pessoais ou em sua equipe, traçar estratégias e políticas para seu fluxo de trabalho diário que sejam seguras e ao mesmo tempo ágeis, desfrutando de toda a elegância e praticidade que o Docker pode prover.

Uma ótima leitura

TIMO CABRAL

Entendendo o que é o Docker

Todos nós já estivemos em situações onde é necessário compartilhar código de um projeto com outros desenvolvedores (ou muitos deles). Uma vez que enviamos nosso código para GitHub e eles tentam executá-lo, nada funciona. Quando perguntado qual era o problema tudo o que nos respodem é com essa frase familiar: **“a aplicação funciona no meu computador”.**

Quando dois desenvolvedores estão trabalhando no mesmo projeto este não parece ser um grande problema - eles se sentam juntos (ou quase), resolvem o problema e esperamos que eles documentem o problema. **Mas o que acontece quando há 10 desenvolvedores trabalhando no mesmo projeto?** Ou 20? As coisas podem sair do controle rapidamente.

O mesmo acontece com a produção. Quantas vezes você escreve algum código, testou localmente, mas quando implantado em produção você enfrentou **a tela branca da morte**? Agora imagine uma equipe de 10 desenvolvedores que implantam para o mesmo ambiente de produção. Em vez de nos preocuparmos com os detalhes do produto em si, **alguém tera de pagar as horas gastas com problemas de implantação e desenvolvimento.**

TIMO CABRAL

Possíveis culpados destes problemas

- Vamos dar um passo para trás e olhar para alguns possíveis culpados que causam esses problemas:
- plug-ins ou versões diferentes WordPress
- versões diferentes do PHP
- (Apache / Nginx) versões diferentes do servidor web
- versões diferentes do MySQL
- Sistema operacional completamente diferente
- Diferentes configurações de todos os acima (limites de memória / CPU, módulos instalados, php.ini)

Então o que é que Docker nos oferece? Uma coisa: **consistência. Ele garante que o ambiente será idêntico, não importa onde você executá-lo; Localmente, em 50 computadores diferentes, em um ambiente de CI (integração contínua) ou na produção – sua aplicação vai funcionar exatamente a mesma, porque ele vai usar o mesmo ambiente e código.**

Simplesmente por que o docker é capaz de empacotar em um container sua aplicação e todas as dependências e configurações baseado em uma tecnologia consagrada chamada LXC. O Docker apenas usa esta tecnologia de forma muito elegante provendo uma API para que sua vida fique melhor, seus cabelos não comecem a cair e a ficar brancos você ainda sendo tão novo.

Docker: O início

TIMO CABRAL

-**Lançado em março de 2013, Docker funciona com o sistema operacional para empacotar e executar software. Você pode pensar em Docker como um provedor de logística de software que irá poupar tempo e permite você se concentrar em atividades de alto valor.**

Você pode usar Docker para aplicativos de rede que vão desde servidores web, bancos de dados,servidores de email,editores de texto, compiladores, ferramentas de análise de rede e scripts; em alguns casos é utilizado para executar aplicações GUI, como navegadores web e software de produtividade.

Docker é uma ferramenta que ajuda a resolver problemas comuns como instalação, remoção atualização, distribuição e gerenciamento de software. É um software de código fonte aberto (assim como WordPress), o que significa que qualquer pessoa pode contribuir e extender suas funcionalidades.

Docker funciona como um programa de linha de comando, um daemon que roda em segundo plano, e usa uma abordagem logística para a solução de problemas de software comuns e simplifica a sua experiência de instalação, rodando, publicando e removendo software. Ele faz isso usando uma tecnologia UNIX chamada containers.

Docker vs Vms

Sem Docker, as empresas normalmente usam a virtualização de hardware (também conhecido como máquinas virtual) para proporcionar isolamento e garantir que ambientes de produção e desenvolvimento tenham menos diferenças possíveis.
As máquinas virtuais funcionam fornecendo um hardware virtual no qual um sistema operacional e outros programas podem ser instalados. Uma das desvantagens das Vms é que elas levam um longo tempo (muitas vezes minutos) para inicializar e requerem muito processamento da máquina que roda a Vm, porque elas rodam uma cópia inteira de um sistema operacional, além do software que você deseja usar.

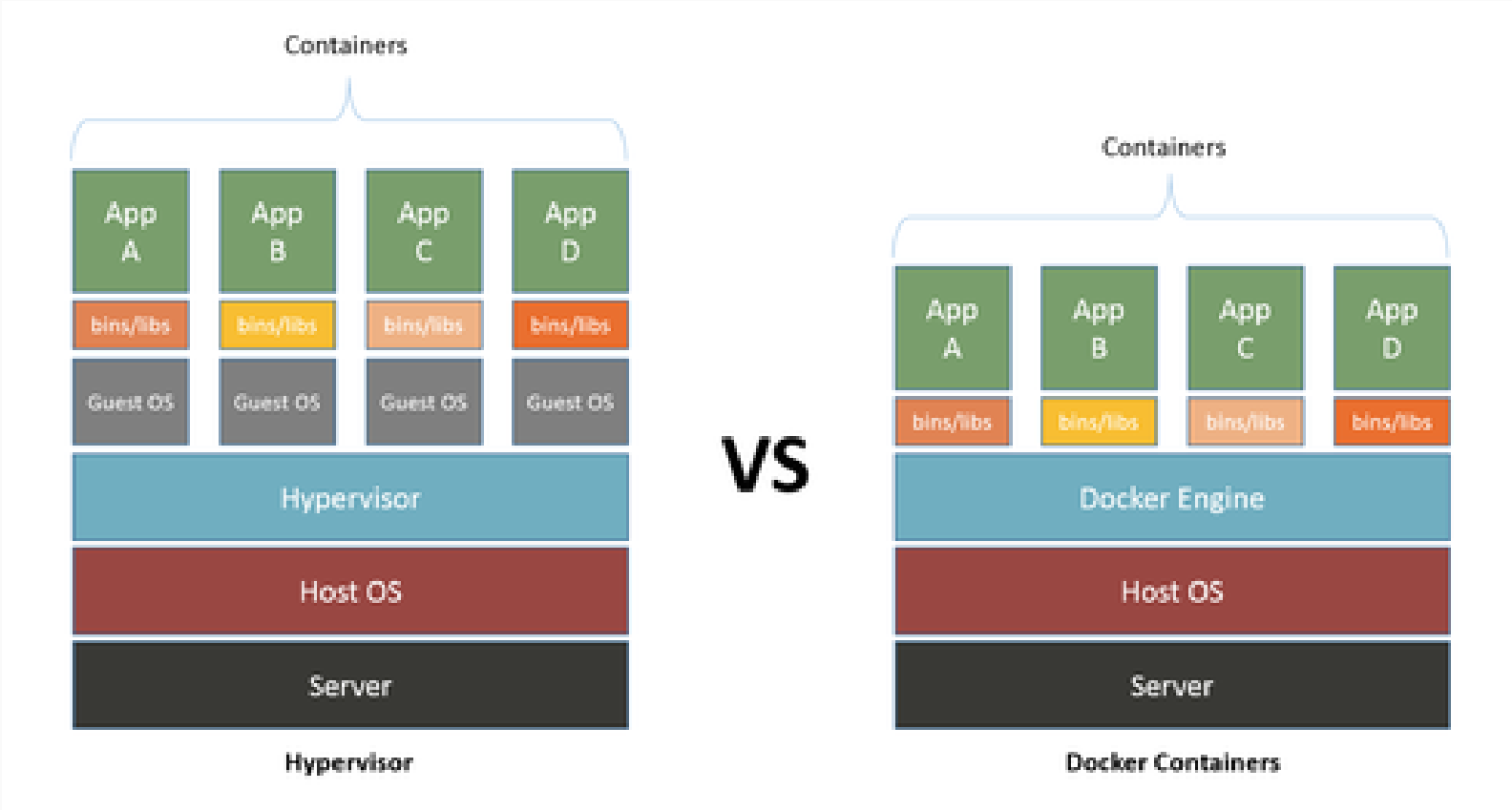
Ao contrário das máquinas virtuais, containers Docker não usam virtualização de hardware. Programas em execução dentro de containers Docker tem uma interface direta com o ambiente Linux hospedeiro. (a máquina que você está usando).
Porque não há nenhuma camada adicional entre o programa em execução no interior do container e o sistema operacional do computador, não há recursos desperdiçados, executando software redundante ou simulação de hardware virtual. Esta é uma distinção importante entre containers docker e máquinas virtuais.

Docker não é uma tecnologia de virtualização. Em vez disso, ele ajuda você a usar um container que foi construído em seu sistema operacional.

TIMO CABRAL

Diferenças entre Docker e Hypervisor (Vm)

Containers usando hypervisor (máquinas virtuais) vs containers docker



TIMO CABRAL

Um pouco do funcionamento

Em geral, o sistema operacional é a interface entre todos os programas de usuário e o hardware que o computador está sendo executado. Você pode ver na figura 1.1 que a execução do Docker significa executar dois programas em espaço do usuário. O primeiro é o **daemon Docker**. Se instalado corretamente, este processo deve sempre estar em execução. O segundo é o **Docker CLI. Este é o programa que o Docker e os usuários interagem. Se você deseja iniciar, parar ou instalar o software, você poderá emitir um comando usando o programa Docker.**

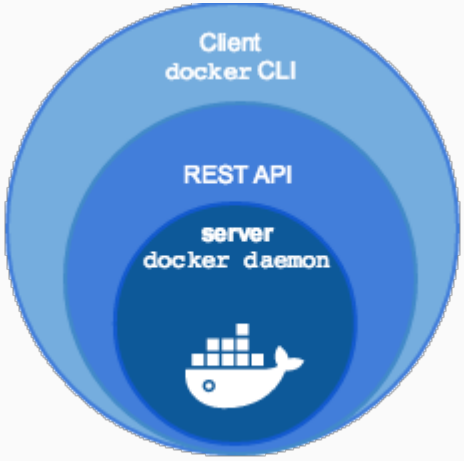
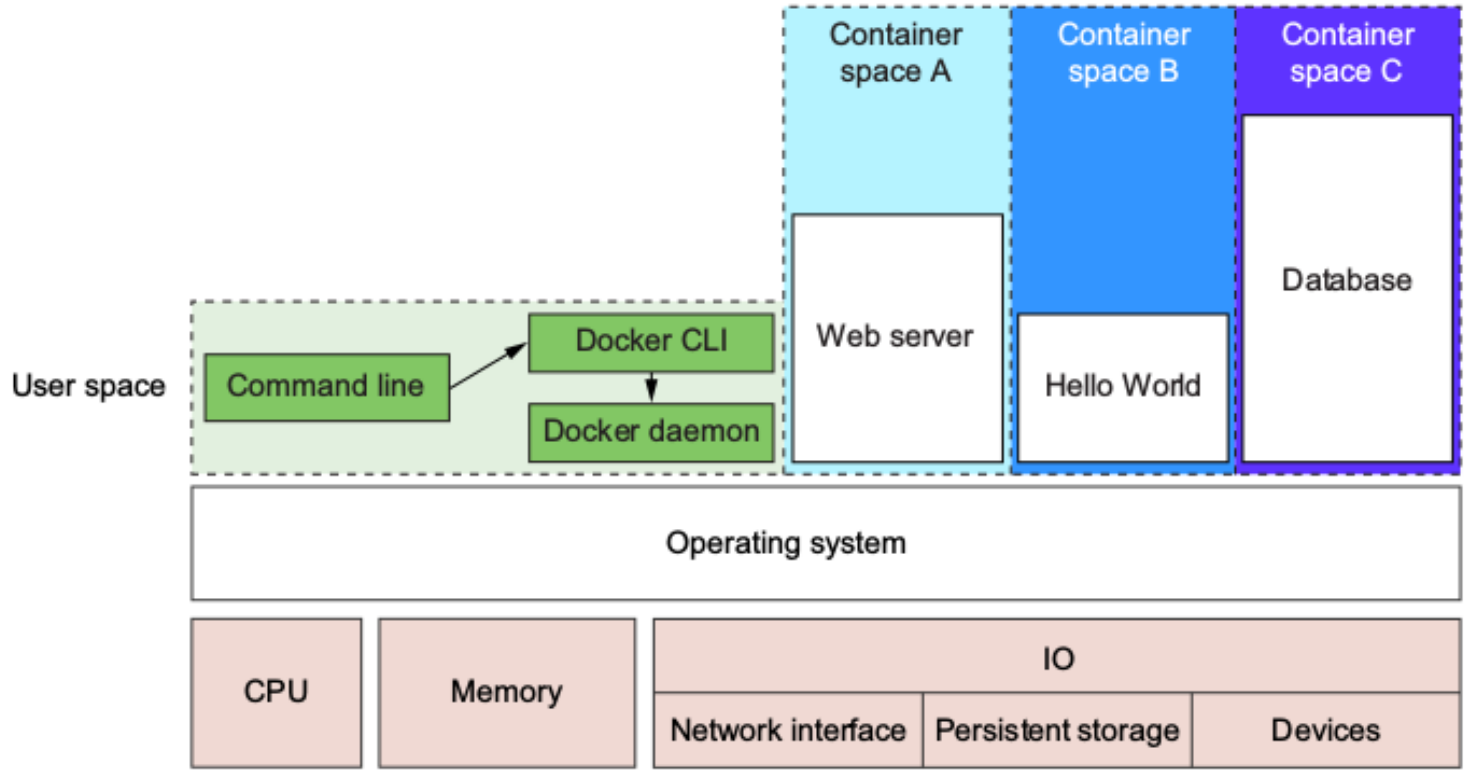


Figura 1.1 também mostra três containers em funcionamento. Cada um é executado como um processo filho do daemon Docker, envolvido com um container, **e o processo delegado está sendo executado em seu próprio sub-espaço de memória do espaço do usuário**.

Programas em execução dentro de um container pode acessar apenas a sua própria memória e recursos.

TIMO CABRAL

Arquitetura Docker



Docker Containers

TIMO CABRAL

Você pode pensar em um container Docker como um container físico. É uma caixa onde se pode armazenar e executar um aplicativo e todas as suas dependências. Assim como guindastes, caminhões, trens e navios podem facilmente trabalhar com containers (ou seja, lidar com grande complexidade de tipos de coisas sem necessariamente ter de conhecer ou lidar com o conteúdo interno dos mesmos) por isso Docker pode executar, copiar, e distribuir containers com facilidade. Incluindo maneiras de empacotar e distribuir software. O componente que preenche o papel de container de transporte é chamado de uma **Imagem.**

A idéia é fornecer uma camada de abstração abrangente que permite aos desenvolvedores "containerizar" ou "empacotar" qualquer aplicação e tê-lo executado em qualquer infra-estrutura. **O uso de containers aqui se refere mais a consistência, empacotamento padrão de aplicações em vez de se referir a qualquer tecnologia subjacente (uma distinção que será importante mais afrente). Containers Docker fornecem um padrão, uma forma consistente de embalar praticamente qualquer aplicação.**

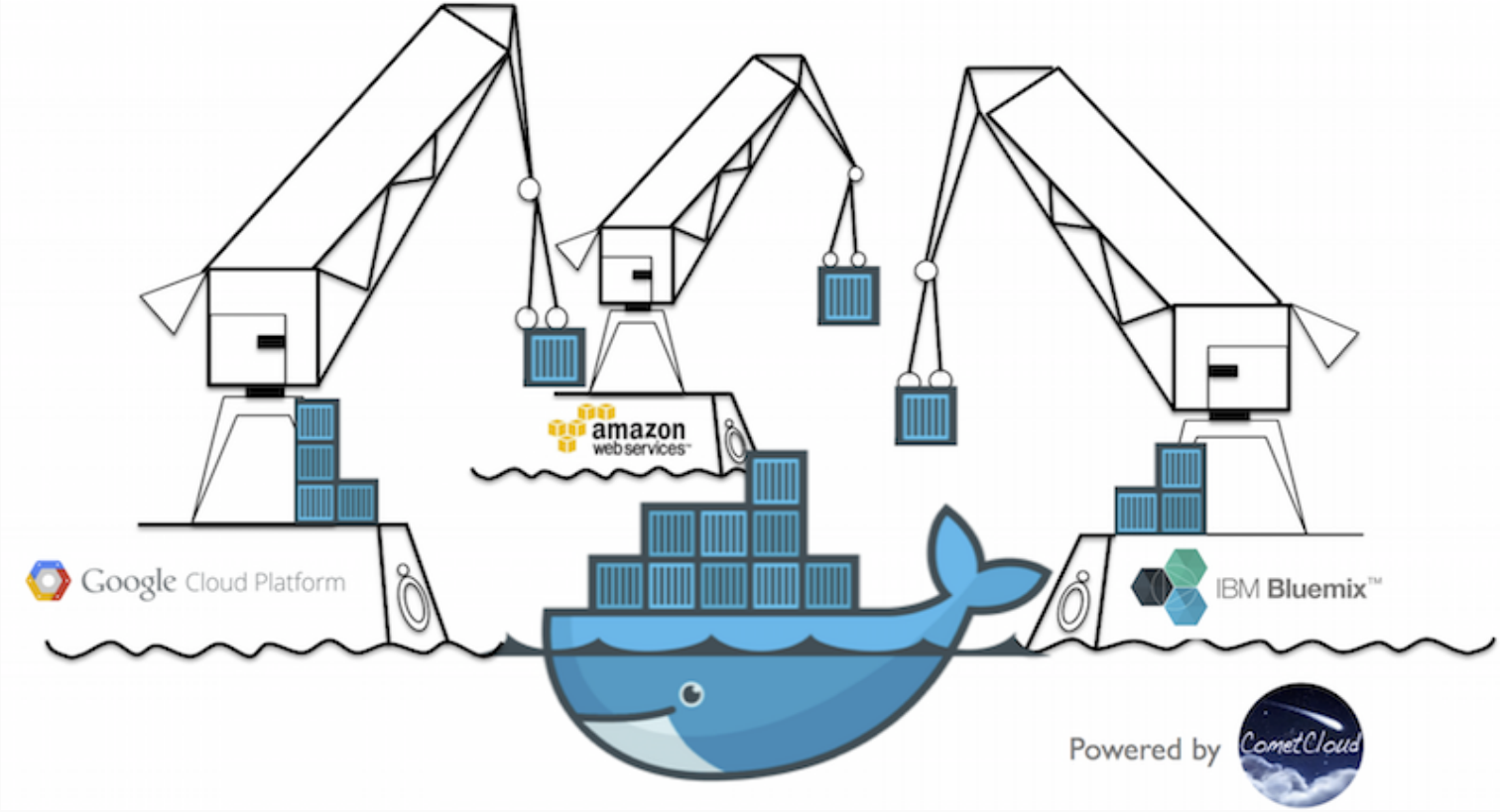
O Docker aproveita o LXC (Linux Containers), que engloba recursos Linux como **cgroups** e **namespaces** para isolamento de processos e controle de recursos, sendo que LXC já pode ser substituido por libcontainer.

A forma como lidar com esses containers, como falados anteriormente é mediante o uso de uma API, similar a uma **DSL** (linguagem específica de domínio) que é texto simples que você pode digitar em um terminal ou em arquivos chamados Dockerfile.

Docker e abstração

TIMO CABRAL

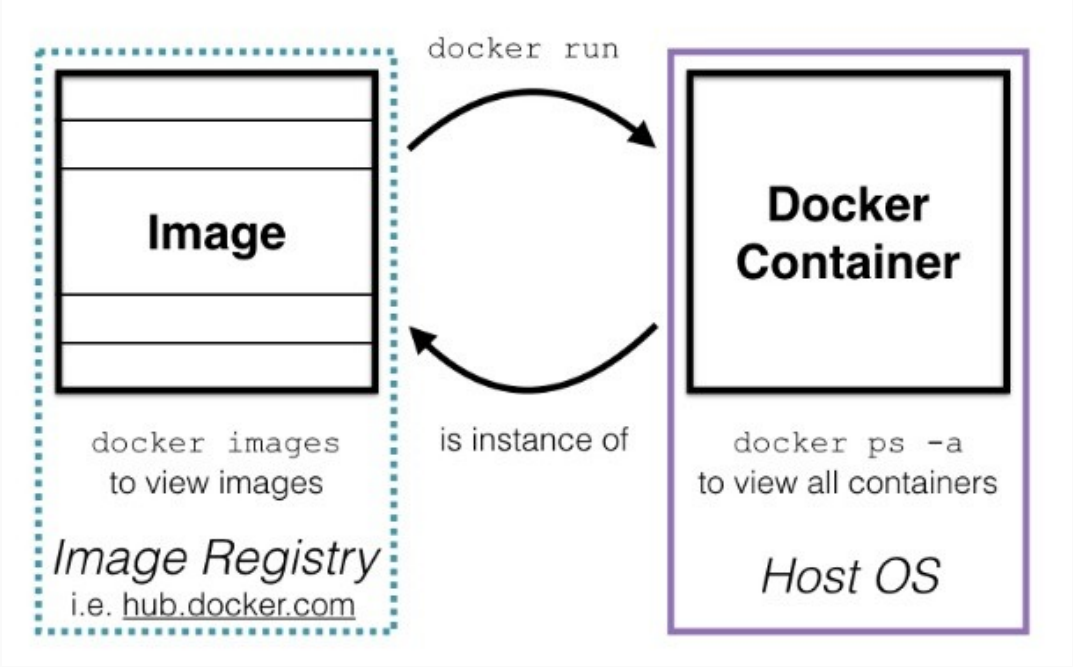
Docker fornece o que é chamado de uma abstração. **Abstrações permitem que você trabalhe com com coisas complicadas em termos simplificados.** Assim, no caso do Docker, em vez de se concentrar em toda as complexidades e especificidades associados à instalação de uma aplicação, todos nós precisamos considerar é o software que se deseja instalar. Como um guindaste que carrega um container em um navio, o processo de instalação de qualquer software com Docker é idêntico em qualquer contexto.



TIMO CABRAL

Imagens

Uma imagem Docker é um conjunto empacotado de todos os arquivos que devem estar disponíveis a um programa em execução dentro de um container. Você pode criar quantos containers de um imagem quiser. **Mas quando o fizer, containers que foram iniciadas a partir da mesma imagem não compartilham alterações ao seu sistema de arquivos. Quando você distribuir software com Docker, você distribui estas imagens, e os computadores que recebem criam containers deles**. As imagens são as unidades no ecossistema Docker.
Docker fornece um conjunto de componentes de infra-estrutura que simplificam a distribuição imagens Docker. **Esses componentes são registros e índices.** Você pode usar a infra-estrutura disponíveis ao público fornecida pela Docker Inc., outras empresas de hospedagem, ou o seu próprio **registros e índices.**



O que o docker resolve?

Hoje, sem Docker, aplicações são desenvolvidas de forma que fiquem espalhadas por todo o sistema de arquivos e acabam tornando o desenvolvimento de aplicações algo confuso e cheio de alterações e modicações que muitas vezes você não tem controle (e é quase impossível lembrar todos os detalhes que você modificou em um ambiente).

Docker mantém as coisas organizadas, isolando tudo com containers e imagens.

2 – Melhorar a portabilidade
Atualmente, Docker roda nativamente no Linux e vem com uma única máquina virtual para ambientes OS X e Windows. Esta convergência em Linux significa que o software rodando em recipientes Docker só precisam ser escritos uma vez contra um conjunto consistente de dependências.

Docker ajuda a minimizar qualquer curva de aprendizagem associada a adoção de novas tecnologias. Isso ajuda os desenvolvedores de software a entender melhor os sistemas que irão executar seus programas. Isso significa menos surpresas.

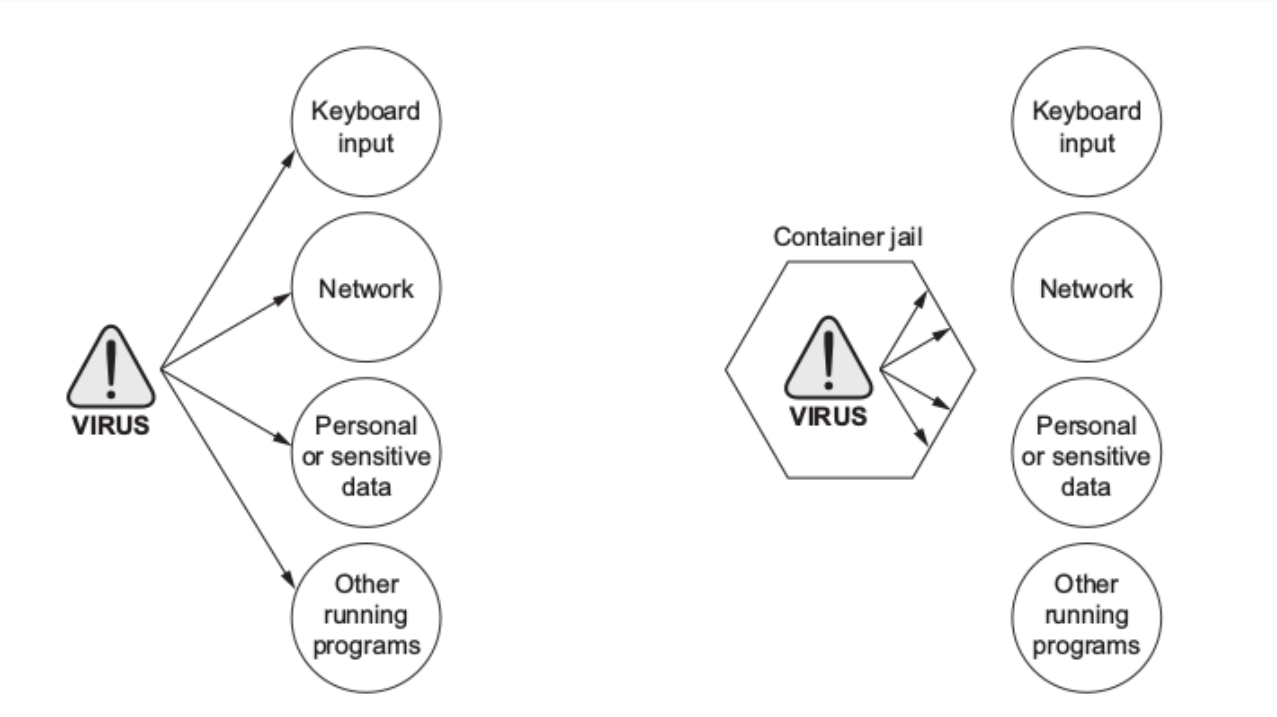
Proteger seu computador

Como a execução de software é um ponto crítico no desenvolvimento de aplicações, é prudente aplicar mitigações de risco.
Como celas físicas, qualquer coisa dentro de um container pode acessar apenas as coisas que estão restritas dentro deste container.

Containers limitam o âmbito de impacto de um programa que pode acessar recursos do sistema, **permitindo que falhas acontecam isoladamente sem que afetem todos os containers** (como um efeito em cascata).
O que isto significa para você ou sua empresa é que o escopo de qualquer ameaça à segurança associados com o funcionamento de uma determinada aplicação é limitada ao âmbito da aplicação em si.

Um exemplo seria se no seu web server rodando nginx estivesse um script malicioso, este script não poderia afetar o servidor de banco Mysql para se replicar.

Isolamento


Keyboard input
Network
Personal or sensitive data
Other running programs
VIRUS
Container jail
Keyboard input
Network
Personal or sensitive data
Other running programs
VIRUS

Importância do Docker

Docker fornece o que é chamado de uma **abstração. Abstrações permitem que você trabalhe com com coisas complicadas em termos simplificados.**

Docker é importante porque faz com com que os containers tornem-se disponíveis a todos de uma forma fácil. **Usando Docker para seu software ele salva tempo, dinheiro e energia.**
A segunda razão que Docker é importante é que não há pressão significativa da comunidade de software para adotar containers Docker. Este impulso é tão forte que empresas como a Amazon, Microsoft e Google têm trabalhado juntos para contribuir para o seu desenvolvimento e adotá-la em suas próprias ofertas de cloud. Estas empresas, que normalmente estão em desacordo, se uniram para apoiar um projeto de código aberto, em vez lançar suas próprias soluções.

Outra vantagem é para a remoção de aplicativos. Quando você quiser remover software, você simplesmente diz ao Docker qual o software você quer remover. Nenhum artefato remanescentes permanecerá porque eles estavam todos cuidadosamente contido e contabilizados. **Seu computador ficará tão limpo como era antes da instalação do software.**

Terceira razão que o Docker é importante é que ele tem feito para o computador o que as lojas de aplicativos fizeram para os dispositivos móveis. Elas fezem da instalação do software, compartimentação e remoção um processo muito simples. Melhor ainda, Docker faz isso a partir de plataforma cruzada e aberta.

Em quarto lugar, estamos começando ver a adoção de avançadas características de isolamento de sistemas operacionais de uma forma que qualquer usuário com conhecimentos técnicos razoáveis pode se beneficiar.

Instalação

TIMO CABRAL

Para instalações no Windows e demais sistemas operacionais
https://docs.docker.com/engine/installation/

Aqui mostraremos a instalação e exemplos usando ubuntu, mas todos os exemplos se aplicam uniformimente a outros sistemas operacionais (com diferenças somente aos caminhos do sistema de arquivos.

Instalação UBUNTU 14-10

Via apt-get

 # apt-get install -y docker.io

 # usermod -aG docker vagrant

#docker info

#docker -v

#docker version

#service docker restart

Última versão estável com wget

#wget -qO- https://get.docker.com/ | sh

Comandos básicos

#docker info

#docker -v

#docker version

#sudo service docker restart

Exemplo “Ola Mundo”

TIMO CABRAL

"Antes de começar, baixe e instale Docker para o seu sistema.
usando o Docker Toolbox.

Após a instalação abra o terminal Execute os comandos como usuário root
(vamos omitir o comando sudo)
Iremos instalar o primeiro container chamado hello world

 # docker run dockerinaction/hello_world

```
root@thymo:/home/thymo/test.bookexample#  docker run dockerinaction/hello_world
Unable to find image 'dockerinaction/hello_world:latest' locally
latest: Pulling from dockerinaction/hello_world

a3ed95caeb02: Pull complete
1db09adb5ddd: Pull complete
Digest: sha256:cfebf86139a3b21797765a3960e13dee000bcf332be0be529858fca840c00d7f
Status: Downloaded newer image for dockerinaction/hello_world:latest
hello world
root@thymo:/home/thymo/test.bookexample#
```

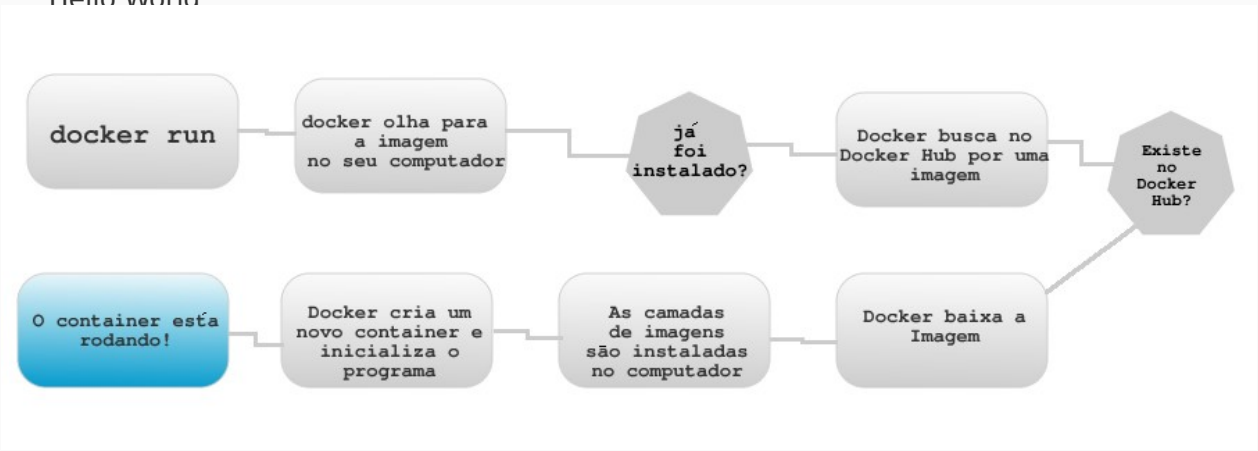
Comando Docker run

Depois de executar o comando, Docker brotará para a vida. Ele vai começar a baixar vários componentes e, eventualmente, irá imprimir "Hello World". Se você executá-lo novamente, ele só irá imprimir a mesma saida. Várias coisas estão acontecendo neste exemplo, e o próprio comando tem poucas partes distintas.

Em primeiro lugar, você usa o comando **run** para iniciar um novo container. Este único comando desencadeia uma sequência (mostrado na figura 1.6) **que instala, roda, e para um program dentro de um container.**
Em segundo lugar, o programa vai ser executado em um container chamado dockerinaction/hello_world . trata-se do repositório (ou imagem). **Por enquanto, você pode pensar o nome do repositório como o nome do programa que pretende instalar ou executar.**

A primeira vez que você roda o comando, Docker tem que descobrir se dockerinaction/hello_world já está instalado. Se é incapaz de localizá-lo no seu computador (porque é a primeira execução do comando), Docker faz uma chamada para Docker Hub.

Docker Hub é um registro público fornecido pela Docker Inc. Ele responde ao comando run executado no seu terminal onde dockerinaction/hello_world pode ser encontrado, e Docker inicia um download. Uma vez instalado, Docker cria um novo container e executa um único comando. O comando é um simples:echo "Hello World"



TIMO CABRAL

Instalando uma pilha LEMP do Zero:01

Após o exemplo Hello World, vamos criar um novo container contendo uma imagem do Nginx e iremos imprimir um Hello World só que em um brower e iremos ver mais detalhes de como o Docker funciona. Para você acompanhar todos os exemplos colocados aqui, entre no repositório git do Livro:

https://github.com/timocabral/wordpress-docker-ebook

Criando as pastas de trabalho

#mkdir example-docker-02
#cd example-docker-02

Agora que você tem uma compreensão básica do que o Docker é, vamos tentar construir uma stack LEMP básica (Linux, Nginx, MySQL, PHP), que vamos usar para o nosso site WordPress daqui para frente. Lembrando que instalaremos tudo passo a passo.

No seu diretório de trabalho, crie um arquivo chamado index.html

#nano index.html

<h1>Ola mundo nginx<h1/>

Execute
#docker run --name example-ebook-02 -v /home/thymo/example-docker-02:/usr/share/nginx/html -P -d nginx

Execulte
docker ps

Procure pelo container example-ebook-02 e copie o endereço Port 0.0.0.0:32775 (aparecerá outro número já que é executado randomicamente e cole no navegador ou use curl, você verá “ola mundo nginx”

Instalando uma pilha LEMP do Zero:02

TIMO CABRAL

```
root@thymo:/home/thymo/test.bookexample#  nano index.html
root@thymo:/home/thymo/test.bookexample# ls
index.html
root@thymo:/home/thymo/test.bookexample# docker run --name example-ebook-02 -v /home/thymo/test.bookexample:/usr/shar
e/nginx/html -P -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx

43c265008fae: Already exists
e4c030a565b1: Pull complete
685b7631c1ce: Pull complete
Digest: sha256:dedbce721065b2bcfae35d2b0690857bb6c3b4b7dd48bfe7fc7b53693731beff
Status: Downloaded newer image for nginx:latest
78f369956515850962d1389bb65064544facdf16d759bc3e82530e02f63de30a
root@thymo:/home/thymo/test.bookexample# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                                NAMES
78f369956515        nginx               "nginx -g 'daemon off"   25 seconds ago      Up 23 seconds       0.0.0.0:3276
9->80/tcp, 0.0.0.0:32768->443/tcp   example-ebook-02
root@thymo:/home/thymo/test.bookexample# curl 0.0.0.0:32769
<h1>Ola mundo docker nginx</h1>
```

Instalando uma pilha LEMP do Zero:03

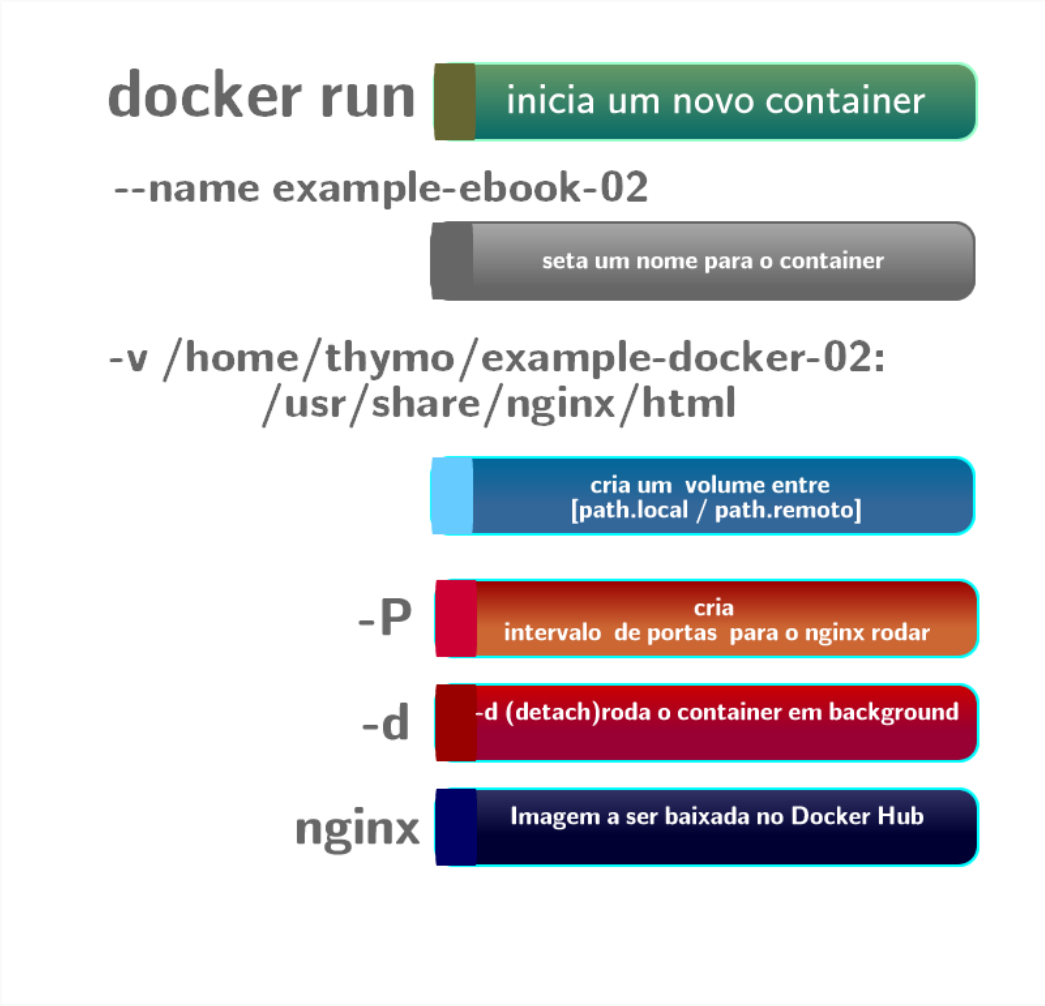
TIMO CABRAL

O que acabou de acontecer?
Embora possa parecer um pouco de mágica em primeiro lugar, é realmente muito simples; Docker baixa a imagem (nginx - a última palavra do comando), em seguida, **roda um container com essa imagem com dois parâmetros especiais**: É montado o nosso diretório do projeto $(pwd) (ou o patch absoluto) chamando seu diretório de trabalho atual) em / usr / share / nginx / html e o sistema convidado também mapeou a porta do servidor com o comando **-P que liga cada porta exposta a uma porta aleatória sobre o host hospedeiro.** O intervalo de portas estão dentro de um intervalo de portas temporárias definida por / proc / sys / net / ipv4 / ip_local_port_range. Use o sinalizador **-p para mapear explicitamente uma única porta ou intervalo de portas.**

O comando -v monta um volume com seu path de arquivos (/home/seu-usuario/seus-arquivos/) que será montado como um diretório de trabalho do container, ou seja, você seleciona um diretório local que será montado de forma similar a um dispositivo usb que será acessado pelo container, permitindo que edite, exclua, adicione arquivos em seu container de forma tão transparente como se fossem arquivos locais.

Instalando uma pilha LEMP do Zero:04

O que acabou de acontecer?



Instalando uma pilha LEMP do Zero:05

TIMO CABRAL

A principal diferença entre o uso de volumes e copiar arquivos é que você costuma montar volumes em desenvolvimento, mas vai precisar de arquivos copiados para a imagem de distribuição (produção). **A montagem de volumes no desenvolvimento economiza uma quantidade significativa de tempo, porque você não tem que reconstruir a imagem para cada mudança - Docker troca temporariamente o diretório de origem sobre a imagem com o que você tem localmente.**

Por que isso é impressionante? Porque nós não precisamos instalar nada na nossa máquina de desenvolvimento, tudo o necessário para executar tudo o que deseja executar é fornecida por imagens! Isso significa que você não precisa instalar Nginx, PHP e MySQL no computador de desenvolvimento, que é uma coisa boa, todos estes processos ocupam recursos valiosos se não desligá-los - e na maior parte não, eles apenas rodam em background.

TIMO CABRAL

Instalando uma pilha LEMP do Zero:06: shell

Em algumas situações (como verificar parâmetros e arquivos de configuração em seu container) seja necessário acessar-lo diretamente como você faria acessando um servidor remoto com SSH (no caso de docker não há SSH envolvido). Execute o comando:

 # docker run -it nginx /bin/sh

Normalmente, em sistemas baseados em Debian / Ubuntu, a configuração Nginx está localizada em /etc/nginx e felizmente esta imagem usa a mesma estrutura exata, por isso, se você executar **$ ls -al /etc/nginx/conf.d/ (dentro da imagem)** , você vai realmente ver um único arquivo lá, chamado default.conf, que você irá substituir em breve. Mas primeiro, vamos investigar de que há, de fato, na imagem docker.

 # cat /etc/nginx/conf.d/default.conf


```
root@thymo:/home/thymo/test.bookexample# docker run -it nginx /bin/sh
# ls -al /etc/nginx/conf.d
total 12
drwxr-xr-x 2 root root 4096 Oct 21 22:31 .
drwxr-xr-x 3 root root 4096 Oct 21 22:31 ..
-rw-r--r-- 1 root root 1097 Oct 11 15:56 default.conf
#
```

Instalando uma pilha LEMP do Zero:04: conf

TIMO CABRAL

#crie um arquivo chamado nginx.conf no seu diretório de trabalho com o seguinte conteúdo.

```
server {
  server_name _;
  listen 0.0.0.0:80;

  root   /var/www/html;
  index  index.php index.html;

  access_log /dev/stdout;
  error_log /dev/stdout info;

  location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
    expires max;
    log_not_found off;
  }

  location / {
    try_files $uri $uri/ /index.php?$args;

  }

  location ~ [^/]\.php(/|$)  {
    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
    fastcgi_intercept_errors on;
    include fastcgi_params;
    fastcgi_pass php-app:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
  }
}
```

TIMO CABRAL

Instalando uma pilha LEMP do Zero:04: DockerFile

Após a criação do arquivo de configuração do nginx, você ira criar um arquivo chamado Dockerfile (sem extensão). Este arquivo é um pouco similar a um arquivo de shell scrip, onde você pode colocar vários comandos em ordem.

FROM nginx:1.9-alpine

MAINTAINER Tomaz Zaman

RUN mkdir -p /var/www/html

WORKDIR /var/www/html

COPY nginx2.conf /etc/nginx/conf.d/default.conf

COPY index.php /var/www/html

Após a criação do arquivo, você ira criar o container com o seguinte comando

#docker build -f Dockerfile -t example-ebook-02 .

-f : significa o nome do arquivo de configuração (no caso Dockerfile)
-t : opcional, é uma forma de setar uma tag (no caso example-ebook-02)

Observação: não se esqueça do . (ponto) No final da declaração.

Instalando PHP-FPM 01

TIMO CABRAL

A imagem PHP-FPM

É uma prática costumaz criar imagens Docker que tentam executar o Nginx e PHP-FPM dentro do mesmo recipiente, o que não é uma boa prática do ponto de vista do desempenho e segurança. Lembre-se também do **princípio de responsabilidade única** que afirma que uma dependência de software deve ter somente uma responsabilidade bem clara e definida.

O que devemos ter são duas imagens: uma para Nginx (onde o web server é a responsabilidade) e uma para PHP-FPM (onde a análise e interpretação dos scripts PHP é sua responsabilidade).

Crie um novo Dockerfile chamado **Dockerfile.php-fpm** e coloque o seguinte código nele:

```
FROM php:7.0-fpm
# install the PHP extensions we need
RUN apt-get update && apt-get install -y libpng12-dev libjpeg-dev && rm -rf /var/lib/apt/lists/* \
&& docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr \
&& docker-php-ext-install gd mysqli opcache
VOLUME .:/var/www/html/
```

Instalando PHP-FPM 02

TIMO CABRAL

Após a criação do Dockerfile.php-fpm crie um arquivo php chamado index.php com o seguinte conteúdo

<?php

phpinfo();

?>

E execute

```
root@thymo:/home/thymo/test.bookexample# docker build -t example-docker-lemp --file Dockerfile.php-fpm .
Sending build context to Docker daemon 6.656 kB
Step 1 : FROM php:7.0-fpm
 ---> 19c540b69303
Step 2 : RUN apt-get update && apt-get install -y libpng12-dev libjpeg-dev && rm -rf /var/lib/apt/lists/* && docker-p
hp-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr && docker-php-ext-install gd mysqli opcache
 ---> Using cache
 ---> 84113e8137a5
Step 3 : VOLUME .:/var/www/html/
 ---> Using cache
 ---> 555028b07e57
Step 4 : COPY index.php /var/www/html
 ---> b23d2e946b0a
Removing intermediate container ccdc91de42dd
Successfully built b23d2e946b0a
root@thymo:/home/thymo/test.bookexample#
```

Instalando PHP-FPM 03

Com ambas as nossas imagens prontas, é hora de testar a conexão entre eles. Em primeiro lugar, trazer a imagem PHP-FPM (porque é uma dependência para a nossa configuração Nginx) e executá-lo em segundo plano (que é o que o -d significa):

docker run -v $(pwd):/var/www/html --name php-app -d example-docker-lemp

Como esse comando não retornar nada, execute $ docker ps para certificar-se de que o container está em execução. Observe a identificação "CONTAINER", porque você vai precisar para pará-lo digitando $ f0b621891979 (no caso coloque aqui o id do seu container) com docker {seu-id} stop (altere a última corda com a sua ID). E, de fato, é executado:

docker run -it --link php-app:example-docker-lemp -p 8886:80 example-ebook-02

Instalando PHP-FPM 02

TIMO CABRAL

Após a criação do Dockerfile.php-fpm crie um arquivo php chamado index.php com o seguinte conteúdo

<?php

phpinfo();

?>

E execute

```
root@thymo:/home/thymo/test.bookexample# docker build -t example-docker-lemp --file Dockerfile.php-fpm .
Sending build context to Docker daemon 6.656 kB
Step 1 : FROM php:7.0-fpm
 ---> 19c540b69303
Step 2 : RUN apt-get update && apt-get install -y libpng12-dev libjpeg-dev && rm -rf /var/lib/apt/lists/* && docker-p
hp-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr && docker-php-ext-install gd mysqli opcache
 ---> Using cache
 ---> 84113e8137a5
Step 3 : VOLUME .:/var/www/html/
 ---> Using cache
 ---> 555028b07e57
Step 4 : COPY index.php /var/www/html
 ---> b23d2e946b0a
Removing intermediate container ccdc91de42dd
Successfully built b23d2e946b0a
root@thymo:/home/thymo/test.bookexample#
```

Instalando PHP-FPM 04

TIMO CABRAL

```
docker build -f Dockerfile -t example-ebook-02 .
```

constroi a imagem com nginx baseado no Dockerfile

```
docker build -t example-docker-lemp --file Dockerfile.php-fpm .
```

constroi a imagem com php baseado no Dockerfile.php.fpm (tag:example-docker-lemp)

```
docker run -v $(pwd):/var/www/html
--name php-app -d example-docker-lemp
```

roda a imagem php em background, cria um volume

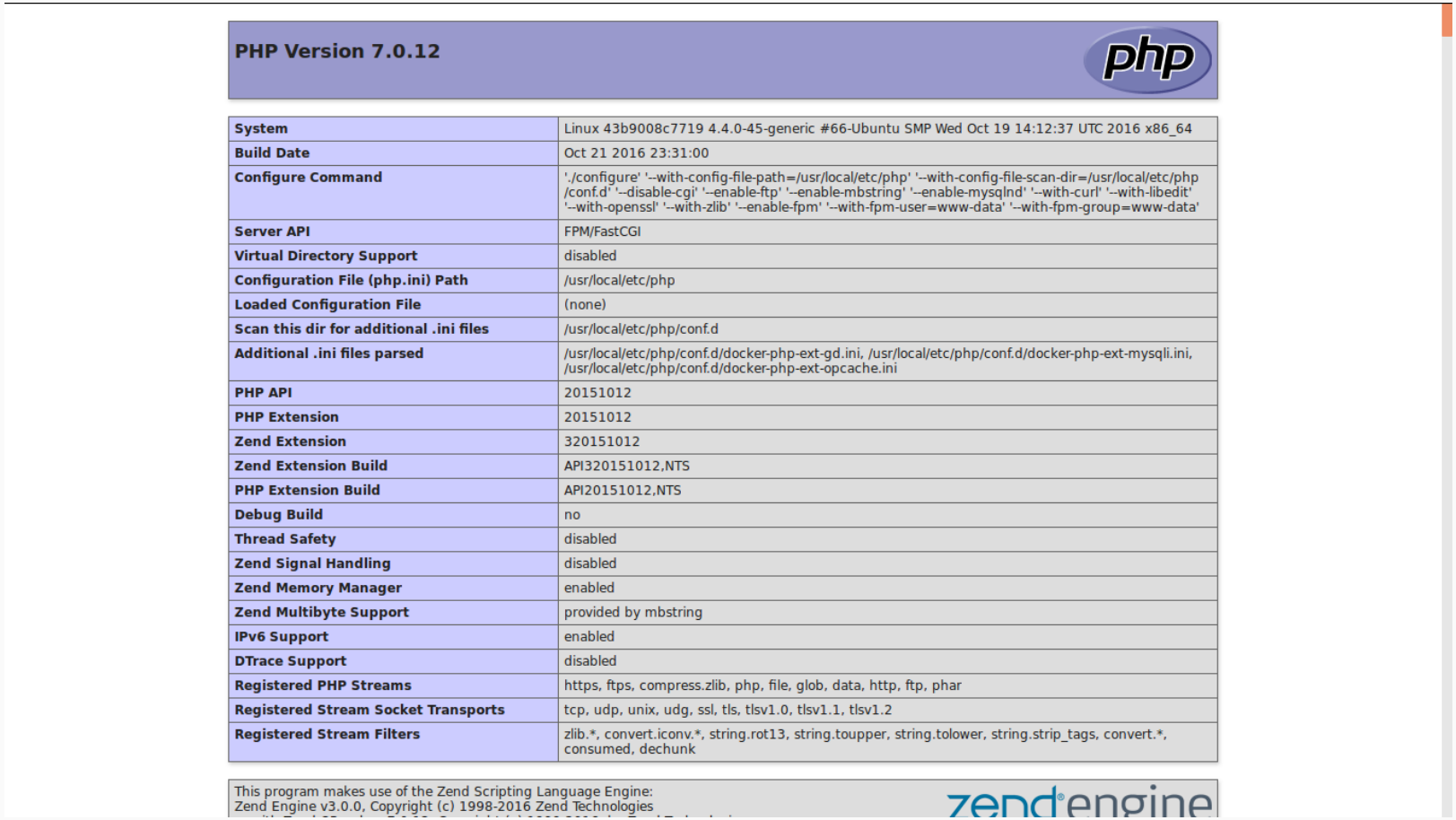
```
docker run -it --link php-app:example-docker-lemp
               -p 8886:80 example-ebook-02
```

roda as duas imagens fazendo um link entre nginx:php na porta 8886

Instalando PHP-FPM 05

TIMO CABRAL

Você irá visualizar a seguinte imagem no navegador



PHP Version 7.0.12

System	Linux 43b9008c7719 4.4.0-45-generic #66-Ubuntu SMP Wed Oct 19 14:12:37 UTC 2016 x86_64
Build Date	Oct 21 2016 23:31:00
Configure Command	'./configure' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--disable-cgi' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--enable-fpm' '--with-fpm-user=www-data' '--with-fpm-group=www-data'
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-mysqli.ini, /usr/local/etc/php/conf.d/docker-php-ext-opcache.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

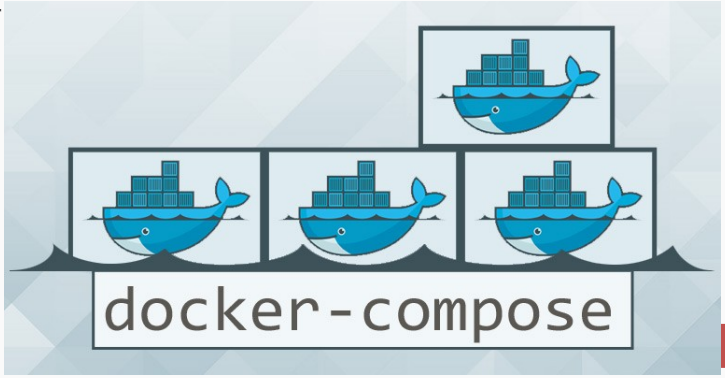
This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies

Compondo comandos

TIMO CABRAL

Antes de continuarmos, vamos verificar alguns dos nossos comandos que tivemos para executar a fim de construir nossas imagens e executar nossos recipientes. Mais importante, vamos investigar quais partes estão se repetido

1 - Ambas as imagens requerem um volume externo a ser montado em / var / www / html
2 - Ambas as imagens expõe certas portas (80 para Nginx, 9000 para PHP-FPM)
3 - Ambas as imagens são construidas a partir de um Dockerfile
4 - Nginx se baseia em PHP-FPM para estar funcionando (como my-php hostname)



Para facilitar nossas vidas, **Docker compose** é ferramenta que nos permite colocar todas essas configurações em um arquivo, a fim de tornar os comandos mais curto e fáceis de compreender. A ferramenta é chamada de docker-composer e requer um arquivo chamado docker-compose.yml presente no diretório do seu projeto, antes de tudo, instale :

https://docs.docker.com/compose/install/

 # curl -L "https://github.com/docker/compose/releases/download/1.8.1/docker-compose-$(uname -s)-$(uname -m)" > /usr/local/bin/docker-compose

 # chmod +x /usr/local/bin/docker-compose

 # docker-compose --version

Executando Wordpress 01

Antes de seguir em frente exclua todas as imagens e containers para evitar conflitos entre as imagens e containers que você criou .

- Deletando todos os containers

#docker rm $(docker ps -a -q)

- Deletando todas as imagens

#docker rmi $(docker images -q)
Após a criação do docker-compose.yml baixe a versão do WordPress em portguês e descompacte em seu diretorio de diretório de trabalho do docker

Baixe o Wordpress

#wget https://br.wordpress.org/wordpress-4.6.1-pt_BR.zip

#unzip wordpress-4.6.1-pt_BR.zip
Movendo a pasta wordpress para o seu diretório de trabalho.
#mv wordpress/* /home/thymo/example.book.1/

Renomeie o wp-config-sample.php para wp-config.php e adicione

/** MySQL hostname */

define('DB_HOST', 'examplebook1_my-mysql_1');

Mais adiante, iremos explicar este parâmetro, pois ele é essencial para que o WordPress se comunique com o Mysql.

Lembre-se de colocar os campos de senhas, usuários iguais aos que estão no docker-compose.yml

Crie seu docker-compose.yml

Executanto Wordpress 02

Compreendendo os volumes de dados do Docker

Uma grande confusão em relação ao docker é o sistema de arquivos que são temporários por padrão. Se você iniciar uma imagem Docker você vai se deparar com um container que em sua superfície se comporta de forma similar como uma máquina virtual. **Você pode criar, modificar e excluir arquivos para o conteúdo da sua preferência. Mas se você der um stop ou reiniciá-lo, todas as alterações serão perdidas:** qualquer arquivo que você tenha excluído anteriormente estará de volta e os novos arquivos ou edições que você fez não estarão presentes. **Isso ocorre porque as imagens Docker são mais parecidas com templates do que com imagens no contexto de máquinas virtuais.**

Imagens Docker são os arquivos que compõem um aplicativo. Por exemplo, se você fizer o download da imagem do Docker Nginx a partir do Docker Hub (o Public Docker Registry), você estará baixando uma imagem que contém uma distribuição Linux mínima e os binários que o próprio Nginx precisa para executar. **Um contêiner é como outra camada que fica sobre esta imagem sem modificar a própria imagem original.** Se você estiver familiarizado com o conceito de snapshots no VirtualBox ou outras ferramentas de virtualização, isso funciona da mesma forma. Para salvar suas alterações você realmente tem que criar uma nova imagem Docker que inclui seus arquivos recém-alterados.

Executanto Wordpress 03

Isso é bom se você estiver interessado em criar novas imagens Docker, mas e se você quiser apenas fazer algo como usar uma imagem pública do Nginx para criar sua própria página da web? Tendo que copiar suas páginas cada vez que você inicia seu container. É aqui que os volumes de dados se aplicam. Os volumes do Docker fornecem um local separado para armazenar dados. Você pode dizer ao Docker para usar qualquer pasta como um contêiner e, a partir daí, qualquer arquivo gravado nesse local será persistido.

compartilhamento de dados entre o host e o contêiner Docker

Sempre que criar as pastas de volume lembre-se em setar as permições corretamente para que o nginx possa acessar os arquivos corretamente.

Em seu arquivo docker-compose.yml você irá visualizar a linha (todo o trecho do arquivo será colocado mais a frente).

```
version: '2'

services:

  nginx:

    build: .

    volumes:

      - ./my.volume:/var/www/html
```

Toda a instrução **volume: diz que my.volume é o seu diretório de trabalho local do seu host, equanto :/var/www/html serão “montados** “ os seus arquivos.

Crie agora o seu docker-compose.yml

```
version: '2'
services:
  nginx:
    build: .
    volumes:
      - "/home/thymo/ex.ebook:/var/www/html"
    ports:
      - "8887:80"
    links:
      - php-app
  php-app:
    build:
      context: .
      dockerfile: Dockerfile.php-fpm
    ports:
      - "9000:9000"
    volumes:
      - "/home/thymo/ex.ebook:/var/www/html"
    links:
      - my-mysql
  my-mysql:
    image: mysql:5.7
    volumes:
      - "./.data/db:/var/lib/mysql"
    restart: always
```

TIMO CABRAL

```
environment:
      MYSQL_ROOT_PASSWORD: wp
      MYSQL_DATABASE: wp
      MYSQL_USER: wp
      MYSQL_PASSWORD: wp
  phpmyadmin:
    image: corbinu/docker-phpmyadmin
    links:
      - my-mysql
    ports:
      - 8181:80
    environment:
      MYSQL_ROOT_PASSWORD: wp
      MYSQL_DATABASE: wp
      MYSQL_USER: wp
      MYSQL_PASSWORD: wp
```

```
root@thymo:/home/thymo/ex.ebook# docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
             STATUS              PORTS                                 NAMES
daaf61b47e2f        exebook_nginx                  "nginx -g 'daemon off"   About a
n hour ago   Up About an hour     443/tcp, 0.0.0.0:8887->80/tcp   exebook_nginx_1
53efb39ac666        corbinu/docker-phpmyadmin      "/bin/sh -c phpmyadmi"   About a
n hour ago   Up About an hour     0.0.0.0:8181->80/tcp            exebook_phpmyad
min_1
3d7b41c74a2f        exebook_php-app                "php-fpm"                About a
n hour ago   Up About an hour     0.0.0.0:9000->9000/tcp          exebook_php-app
_1
05f87b3bfb52        mysql:5.7                      "docker-entrypoint.sh"   About a
n hour ago   Up About an hour     3306/tcp                        exebook_my-mysq
l_1
```

Execulte
docker-compose build

docker-compose up

Preste atenção na saida do comando up, pegue o nome do container mysql e coloque na variável define(“DB_HOST”) no arquivo wp-config.php para você poder se conectar sem problema ao banco.

TIMO CABRAL

Executando Wordpress 04

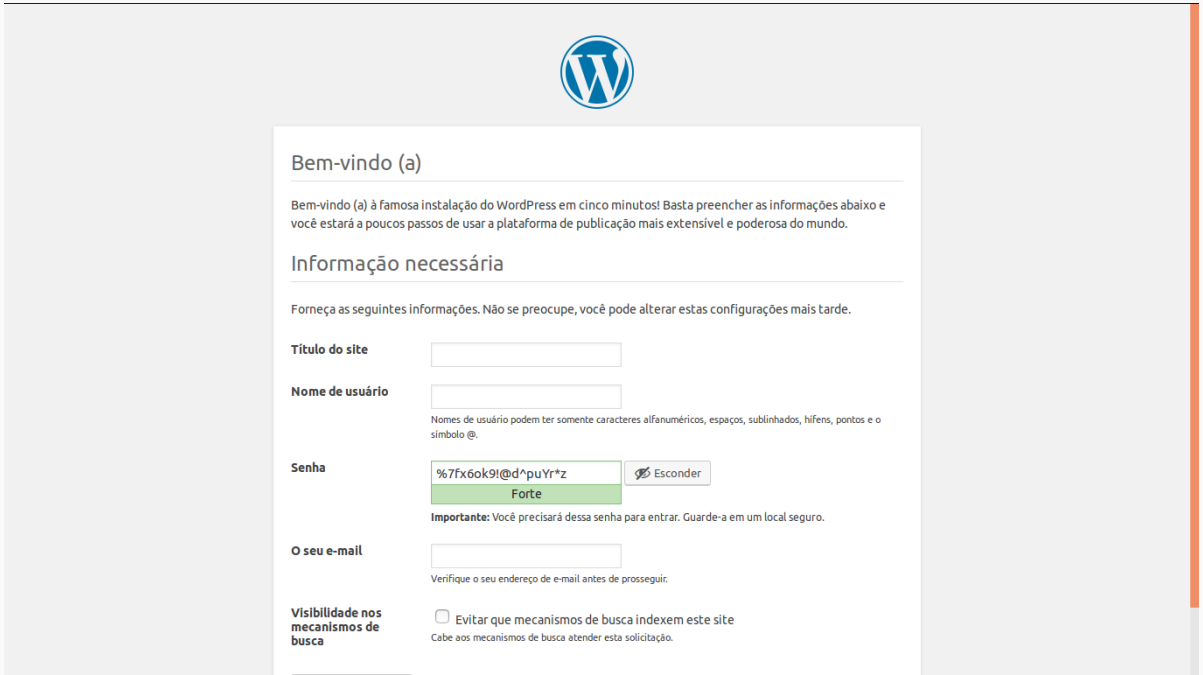
Após criado o docker-compose.yml você irá executar o comando
#docker-compose build

E para subir todas as imagens integradas use:

#docker-compose up

Para a documentação completa do docker compose acesse:
https://docs.docker.com/compose/

Você irá visualizar a seguinte tela se tudo ocorrer com sucesso:



Deploy com Heroku

TIMO CABRAL

O Heroku é uma plataforma poliglota que permite construir, executar e dimensionar aplicativos de maneira semelhante em todas as linguagens, e agora, ela disponibiliza (de forma gratuita) a possibilidade de usar containers docker.

Embora esta funcionalidade ainda estejam em beta, você pode implantar aplicações php e wordpress (mas lembre-se, por sua conta e risco).

Fiquem em alerta no seu email por que estamos preparando um material para implantar containers docker no Amazon Web Services

Crie um conta em heroku.com

Instalando o heriku CLI.
Debian/ubuntu
wget -O- https://toolbelt.heroku.com/install-ubuntu.sh | sh

Versão do heroku
heroku version

Criando a aplicação

cd ~/myapp

heroku create

Instalando plugin para suporte docker

heroku plugins:install heroku-container-registry

Logando-se com o container registry

heroku container:login

O modelo para registrar e enviar imagens para o heroku é como se segue

docker tag <image> registry.heroku.com/<app>/<process-type>

docker push registry.heroku.com/<app>/<process-type>

Criando a tag no docker contendo o id da imagem que será enviada e o enderço do container registry no heroku

docker tag imagem-id registry.heroku.com/sua-app/web

Enviando a imagem

docker push registry.heroku.com/sua-app/web

Abrindo a aplicação

Heroku open -a sua-app

Documentação
https://devcenter.heroku.com/articles/container-registry-and-runtime

Conclusão

Embora tentemos abordar os aspectos mais importantes do Docker, ainda há um enorma quantidade de conhecimentos e assuntos associados a esta ferramenta na implantação e desenvolvimento com esta ferramenta.

Neste Ebook tentamos esclarecer os pontos iniciais de entendimento do Docker, bem como o uso prático dos comandos mais comuns e até ferramentas mais avançadas como o uso do Docker compose e Heroku em conjunto com uma pilha WordPress.

Esperamos que este ebook seja útil para você como pontapé inicial em sua jornada para melhorar sua vida e facilitar o seu processo de desenvolvimento bem como economizar horas e horas debugando problemas de configuração, horas estas que você poderia estar se preocupando com as regras do seu negócio, ou simplesmente aproveitando mais tempo com a família e vida pessoal.

Software é o que lidamos 24 horas por dia, são os nossos alicerces e sabemos que software é sempre um componente crítico de um negócio (e nos dão algumas noites em claro).

A minha intenção com este material e outros que estão sendo construidos, tem por base fundamentar alicerces para que software se torne menos complexo,mais legal e divertido e você possa otimizar tempo e aproveitar mais a vida.

Obrigado e até a próxima!

Timo Cabral.

TIMO CABRAL

Aprimoramento para sua carreira para você alavancar de verdade e dar um show de conhecimento

Acesse!
timocabral.com/materiais

Contato:
timocabral@timocabral.com

Cursos
timocabral.com/cursos