Docker

Sumário

Docker

Por que usar Docker?

Docker tem sido um assunto bem comentado ultimamente, muitos artigos foram escritos, geralmente tratando sobre como usá-lo, ferramentas auxiliares, integrações e afins, mas muitas pessoas ainda fazem a pergunta mais básica quando se trata da possibilidade de utilizar qualquer nova tecnologia: “Por que devo usar isso?” Ou seria: “O que isso tem a me oferecer diferente do que já tenho hoje?”

Figura 1 Ilustração Docker

É normal que ainda duvidem do potencial do Docker, alguns até acham que se trata de um *hype*. Mas nesse capítulo pretendemos demonstrar alguns bons motivos para se utilizar Docker.

Vale frisar que o Docker não é uma “bala de prata” - ele não se propõe a resolver todos problemas, muito menos ser a solução única para as mais variadas situações.

Abaixo alguns bons motivos para se utilizar Docker:

1 – Ambientes semelhantes

Uma vez que sua aplicação seja transformada em uma imagem Docker, ela pode ser instanciada como container em qualquer ambiente que desejar. Isso significa que poderá utilizar sua aplicação no notebook do desenvolvedor da mesma forma que seria executada no servidor de produção.

A imagem Docker aceita parâmetros durante o início do container, isso indica que a mesma imagem pode se comportar de formas diferentes entre distintos ambientes. Esse container pode conectar-se a seu banco de dados local para testes, usando as credenciais e base de dados de teste. Mas quando o container, criado a partir da mesma imagem, receber parâmetros do ambiente de produção, acessará a base de dados em uma infraestrutura mais robusta, com suas respectivas credenciais e base de dados de produção, por exemplo.

As imagens Docker podem ser consideradas como implantações atômicas - o que proporciona maior previsibilidade comparado a outras ferramentas como Puppet, Chef, Ansible, etc - impactando positivamente na análise de erros, assim como na confiabilidade do processo de entrega contínua, que se baseia fortemente na criação de um único artefato que migra entre ambientes. No caso do Docker, o artefato seria a própria imagem com todas as dependências requeridas para executar seu código, seja ele compilado ou dinâmico.

2 – Aplicação como pacote completo

Utilizando as imagens Docker é possível empacotar toda sua aplicação e dependências, facilitando a distribuição, pois não será mais necessário enviar uma extensa documentação explicando como configurar a infraestrutura necessária para permitir a execução, basta disponibilizar a imagem em repositório e liberar o acesso para o usuário e, ele mesmo pode baixar o pacote, que será executado sem problemas.

A atualização também é positivamente afetada, pois a estrutura de camadas do Docker viabiliza que, em caso de mudança, apenas a parte modificada seja transferida e assim o ambiente pode ser alterado de forma mais rápida e simples. O usuário precisa executar apenas um comando para atualizar a imagem da aplicação, que será refletida no container em execução apenas no momento desejado. As imagens Docker podem utilizar tags e, assim, viabilizar o armazenamento de múltiplas versões da mesma aplicação. Isso significa que em caso de problema na atualização, o plano de retorno será basicamente utilizar a imagem com a tag anterior.

3 – Padronização e replicação

Como as imagens Docker são construídas através de arquivos de definição, é possível garantir que determinado padrão seja seguido, aumentando a confiança na replicação. Basta que as imagens sigam as melhores práticas de construção para que seja viável escalarmos a estrutura rapidamente.

Caso a equipe receba uma nova pessoa para trabalhar no desenvolvimento, essa poderá receber o ambiente de trabalho com alguns comandos. Esse processo durará o tempo do download das imagens a serem utilizadas, assim como os arquivos de definições da orquestração das mesmas. Isso facilita a introdução de um novo membro no processo de desenvolvimento da aplicação, que poderá reproduzir rapidamente o ambiente em sua estação e assim desenvolver códigos seguindo o padrão da equipe.

Na necessidade de se testar nova versão de determinada parte da solução, usando imagens Docker, normalmente basta a mudança de um ou mais parâmetros do arquivo de definição para iniciar um ambiente modificado com a versão requerida para avaliação. Ou seja: criar e modificar a infraestrutura ficou bem mais fácil e rápido.

4 – Idioma comum entre Infraestrutura e desenvolvimento

A sintaxe usada para parametrizar as imagens e ambientes Docker pode ser considerada um idioma comum entre áreas que costumeiramente não dialogavam bem. Agora é possível ambos os setores apresentarem propostas e contra propostas com base em um documento comum.

A infraestrutura requerida estará presente no código do desenvolvedor e a área de infraestrutura poderá analisar o documento, sugerindo mudanças para adequação de padrões do seu setor ou não. Tudo isso em comentários e aceitação de merge ou pull request do sistema de controle de versão de códigos.

5 – Comunidade

Assim como é possível acessar o github ou gitlab à procura de exemplos de código, usando o repositório de imagens do Docker é possível conseguir bons modelos de infraestrutura de aplicações ou serviços prontos para integrações complexas.

Um exemplo é o nginx como proxy reverso e mysql como banco de dados. Caso a aplicação necessite desses dois recursos, você não precisa perder tempo instalando e configurando totalmente esses serviços. Basta utilizar as imagens do repositório, configurando parâmetros mínimos para adequação com o ambiente. Normalmente as imagens oficiais seguem as boas práticas de uso dos serviços oferecidos.

Utilizar essas imagens não significa ficar “refém” da configuração trazida com elas, pois é possível enviar sua própria configuração para os ambientes e evitar apenas o trabalho da instalação básica.

O que é Docker?

De forma bem resumida, podemos dizer que o Docker é uma plataforma aberta, criada com o objetivo de facilitar o desenvolvimento, a implantação e a execução de aplicações em ambientes isolados. Foi desenhada especialmente para disponibilizar uma aplicação da forma mais rápida possível.

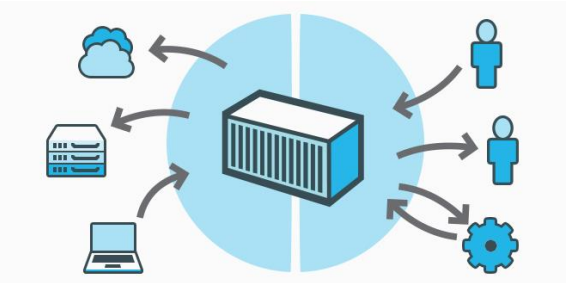


Figura 2

Usando o Docker, você pode facilmente gerenciar a infraestrutura da aplicação, isso agilizará o processo de criação, manutenção e modificação do seu serviço.

Todo processo é realizado sem necessidade de qualquer acesso privilegiado à infraestrutura corporativa. Assim, a equipe responsável pela aplicação pode participar da especificação do ambiente junto com a equipe responsável pelos servidores.

O Docker viabilizou uma "linguagem" comum entre desenvolvedores e administradores de servidores. Essa nova "linguagem" é utilizada para construir arquivos com as definições da infraestrutura necessária e como a aplicação será disposta nesse ambiente, em qual porta fornecerá seu serviço, quais dados de volumes externos serão requisitados e outras possíveis necessidades.

O Docker também disponibiliza uma nuvem pública para compartilhamento de ambientes prontos, que podem ser utilizados para viabilizar customizações para ambientes específicos. É possível obter uma imagem pronta do apache e configurar os módulos específicos necessários para a aplicação e, assim, criar seu próprio ambiente customizado. Tudo com poucas linhas de descrição.

O Docker utiliza o modelo de container para “empacotar” a aplicação que, após ser transformada em imagem Docker, pode ser reproduzida em plataforma de qualquer porte; ou seja, caso a aplicação funcione sem falhas em seu notebook, funcionará também no servidor ou no mainframe. Construa uma vez, execute onde quiser.

Os containers são isolados a nível de disco, memória, processamento e rede. Essa separação permite grande flexibilidade, onde ambientes distintos podem coexistir no mesmo host, sem causar qualquer problema. Vale salientar que o overhead nesse processo é o mínimo necessário, pois cada container normalmente carrega apenas um processo, que é aquele responsável pela entrega do serviço desejado. Em todo caso, esse container também carrega todos os arquivos necessários (configuração, biblioteca e afins) para execução completamente isolada.

Outro ponto interessante no Docker é a velocidade para viabilizar o ambiente desejado; como é basicamente o início de um processo e não um sistema operacional inteiro, o tempo de disponibilização é, normalmente, medido em segundos.

Virtualização a nível do sistema operacional

O modelo de isolamento utilizado no Docker é a virtualização a nível do sistema operacional, um método de virtualização onde o kernel do sistema operacional permite que múltiplos processos sejam executados isoladamente no mesmo host. Esses processos isolados em execução são denominados no Docker de container.

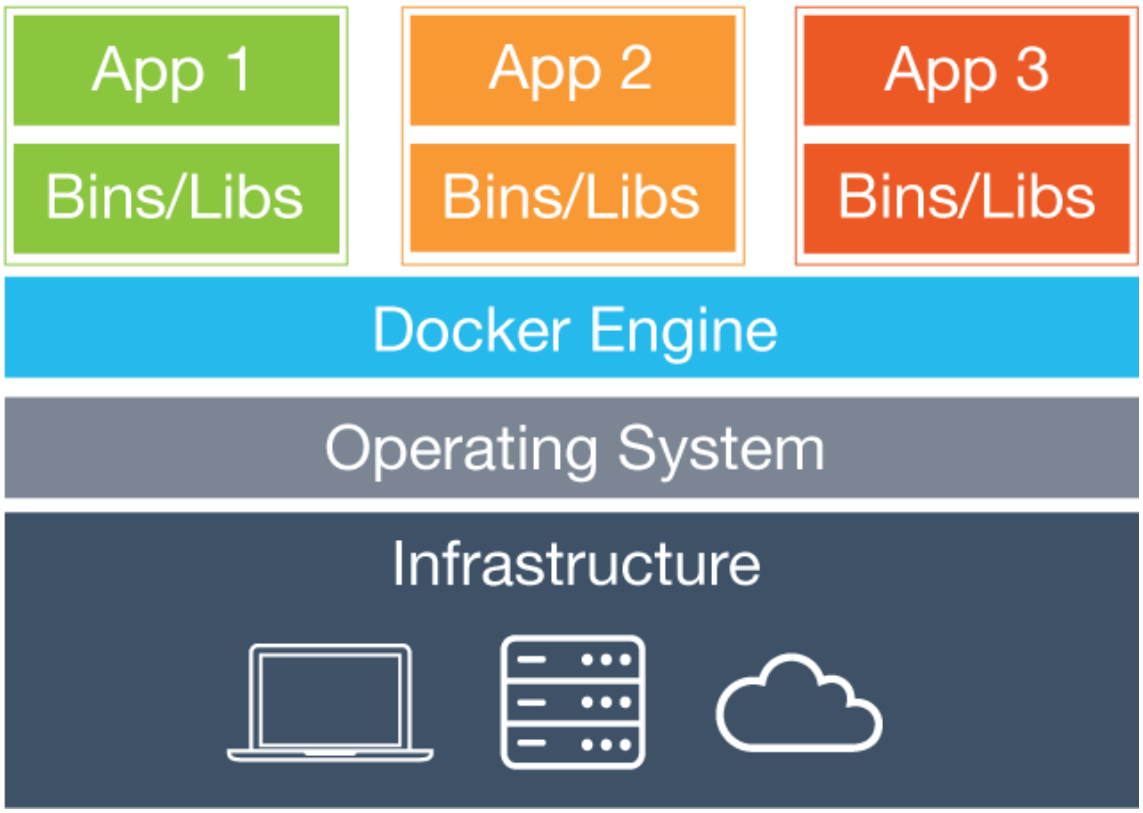


Figura 3

Para criar o isolamento necessário do processo, o Docker usa a funcionalidade do kernel, denominada de namespaces, que cria ambientes isolados entre containers: os processos de uma aplicação em execução não terão acesso aos recursos de outra. A menos que seja expressamente liberado na configuração de cada ambiente.

Para evitar a exaustão dos recursos da máquina por apenas um ambiente isolado, o Docker usa a funcionalidade cgroups do kernel, responsável por criar limites de uso do hardware a disposição. Com isso é possível coexistir no mesmo host diferentes containers sem que um afete diretamente o outro por uso exagerado dos recursos compartilhados.

Instalação

O Docker deixou de ser apenas um software para virar um conjunto deles: um ecossistema.

Nesse ecossistema temos os seguintes softwares:

- **Docker Engine:** É o software base de toda solução. É tanto o daemon responsável pelos containers como o cliente usado para enviar comandos para o daemon.
- **Docker Compose:** É a ferramenta responsável pela definição e execução de múltiplos containers com base em arquivo de definição.
- **Docker Machine:** é a ferramenta que possibilita criar e manter ambientes docker em máquinas virtuais, ambientes de nuvem e até mesmo em máquina física.

Não citamos o **Swarm** e outras ferramentas por não estarem alinhados com o objetivo desse livro: introdução para desenvolvedores.

Instalando no GNU/Linux

Explicamos a instalação da forma mais genérica possível, dessa forma você poderá instalar as ferramentas em qualquer distribuição GNU/Linux que esteja usando.

Docker engine no GNU/Linux

Para instalar o docker engine é simples. Acesse seu terminal preferido do GNU/Linux e torne-se usuário root:

```
su - root
```

ou no caso da utilização de sudo

```
sudo su - root
```

Execute o comando abaixo:

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

Aconselhamos que leia o script que está sendo executado no seu sistema operacional. Acesse esse link e analise o código assim que tiver tempo para fazê-lo.

Esse procedimento demora um pouco. Após terminar o teste, execute o comando abaixo:

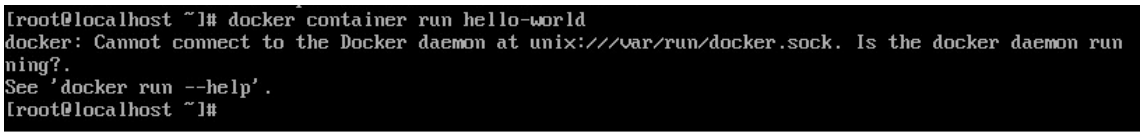
```
docker container run hello-world
```

Tratamento de possíveis problemas

Se o acesso à internet da máquina passar por controle de tráfego (aquele que bloqueia o acesso a determinadas páginas) você poderá encontrar problemas no passo do apt-key. Caso enfrente esse problema, execute o comando abaixo:

```
wget -qO- https://get.docker.com/gpg | sudo apt-key add -
```

Caso ocorra o erro ilustrado na imagem abaixo:



Execute o comando:

```
sudo service docker start
```

Instalando Docker compose com pip

O pip é um gerenciador de pacotes Python e, como o docker-compose, é escrito nessa linguagem, é possível instalá-lo da seguinte forma:

```
pip install docker-compose
```

Tratamento de possíveis problemas

Caso não tenha instalado o comando pip em seu computador, normalmente ele pode ser instalado usando seu sistema de gerenciamento de pacote com o nome python-pip ou semelhante.

Docker machine no GNU/Linux

Instalar o docker machine é simples. Acesse o seu terminal preferido do GNU/Linux e torne-se usuário root:

```
su - root
```

ou no caso da utilização de sudo

```
sudo su - root
```

Execute o comando abaixo:

```
$ curl -L
https://github.com/docker/machine/releases/download/v0.10.0/docker-machine-
`uname -s`-`uname -m` > /usr/local/bin/docker-machine && \
chmod +x /usr/local/bin/docker-machine
```

Para testar, execute o comando abaixo:

```
docker-machine version
```

Obs.: O exemplo anterior utiliza a versão mais recente no momento desta publicação. Verifique se há alguma versão atualizada consultando a documentação oficial.

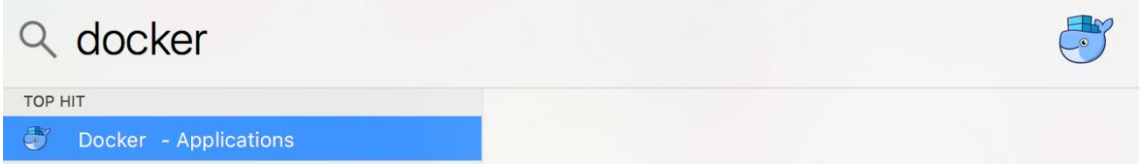
Instalando no MacOS

A instalação das ferramentas do Ecossistema Docker no MacOS pode ser realizada através de um único grande pacote chamado Docker for Mac.

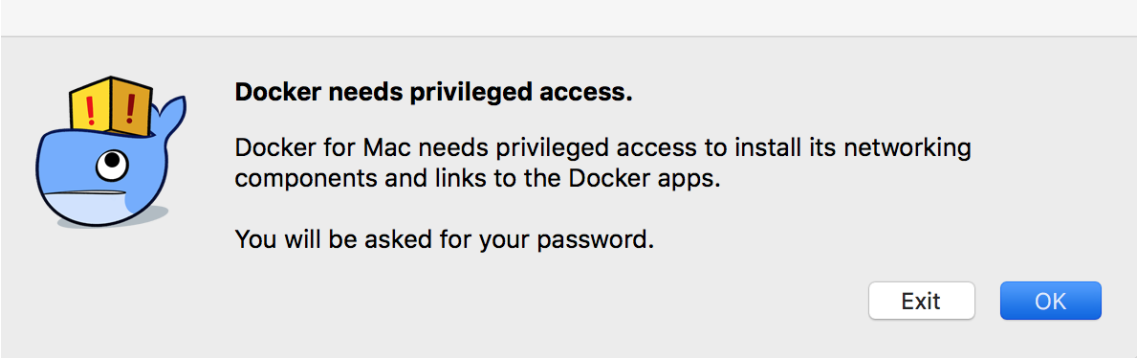
Você pode instalar via brew cask com o comando abaixo:

```
brew cask install docker
```

Para efetuar a configuração inicial, você deve executar o aplicativo Docker:



Na tela seguinte selecione a opção Ok.



Será solicitado seu usuário e senha para liberar a instalação dos softwares. Preencha e continue o processo.

Será solicitado seu usuário e senha para liberar a instalação dos softwares. Preencha e continue o processo.

Para testar, abra um terminal e execute o comando abaixo:

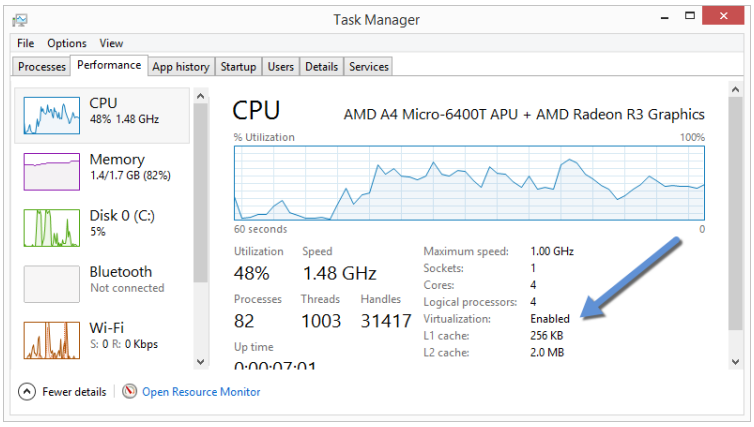
```
brew cask install docker
```

Instalando no Windows

A instalação das ferramentas do Ecossistema Docker no Windows é realizada através de um único grande pacote, que se chama Docker Toolbox.

O Docker Toolbox funciona apenas em versões 64bit do Windows e somente para as versões superiores ao Windows 7.

É importante salientar também que é necessário habilitar o suporte de virtualização. Na versão 8 do Windows, é possível verificar através do Task Manager. Na aba Performance clique em CPU para visualizar a janela abaixo:

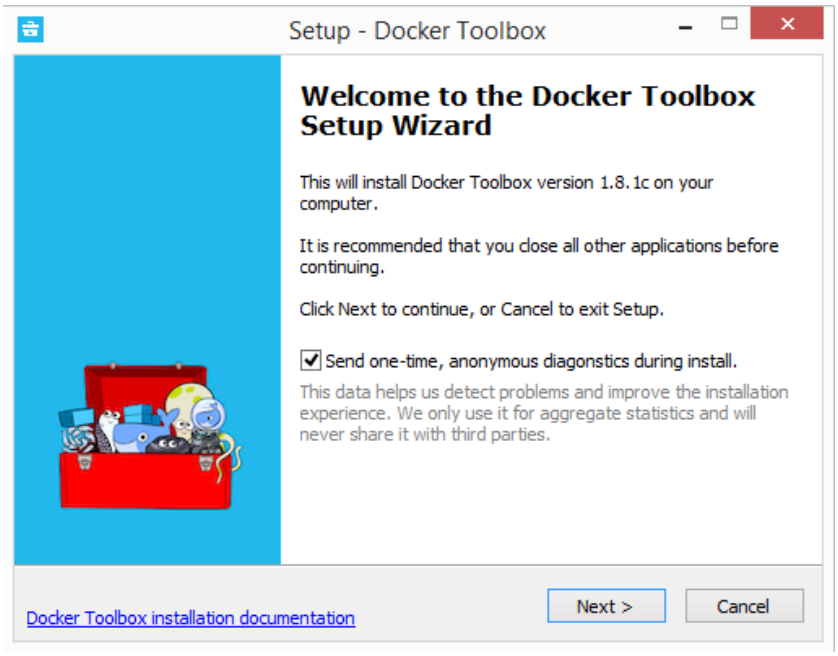


Para verificar o suporte a virtualização do Windows 7, utilize esse link para maiores informações.

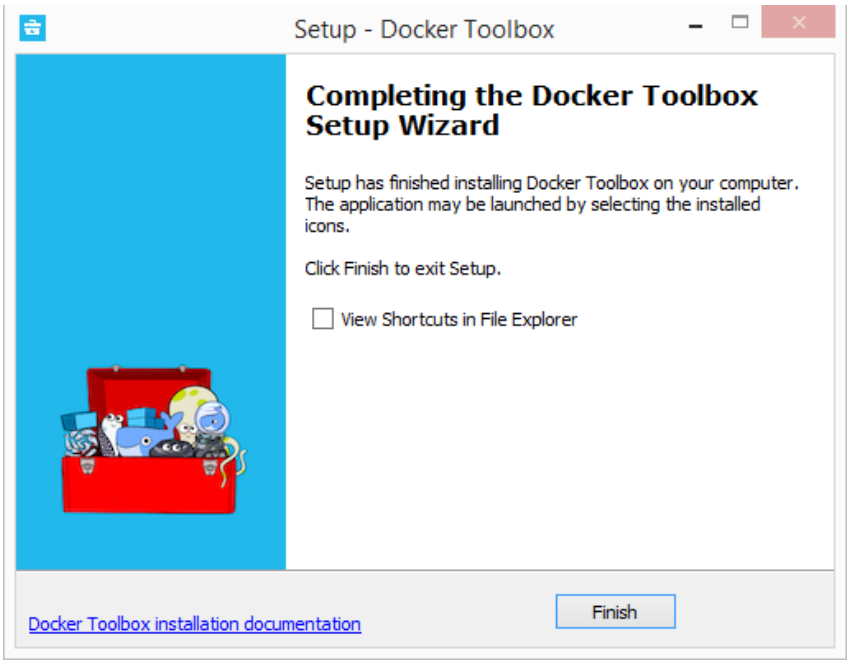
Instalando o Docker Toolbox

Acesse a página de download do Docker toolbox e baixe o instalador correspondente ao Windows.

Após duplo clique no instalador, verá essa tela:



Apenas clique em Next.



Por fim, clique em Finish.

Para testar, procure e execute o software Docker Quickstart Terminal, pois ele fará todo processo necessário para começar a utilizar o Docker.

Nesse novo terminal execute o seguinte comando para teste:

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

Comandos básicos

Para utilização do Docker é necessário conhecer alguns comandos e entender de forma clara e direta para que servem, assim como alguns exemplos de uso.

Não abordaremos os comandos de criação de imagem e tratamento de problemas (troubleshooting) no Docker, pois têm capítulos específicos para o detalhamento.

Executando um container

Para iniciar um container é necessário saber a partir de qual imagem será executado. Para listar as imagens que seu Docker host tem localmente, execute o comando abaixo:

```
docker image list
```

As imagens retornadas estão presentes no seu Docker host e não demandam qualquer download da nuvem pública do Docker, a menos que deseje atualizá-la. Para atualizar a imagem basta executar o comando abaixo:

```
docker image pull python
```

Usamos a imagem chamada python como exemplo, mas caso deseje atualizar qualquer outra imagem, basta colocar seu nome no lugar de python.

Caso deseje inspecionar a imagem que acabou de atualizar, basta usar o comando abaixo:

```
docker image inspect python
```

O comando inspect é responsável por informar todos os dados referentes à imagem.

Agora que temos a imagem atualizada e inspecionada, podemos iniciar o container. Mas antes de simplesmente copiar e colar o comando, vamos entender como ele realmente funciona.

```
docker container run <parâmetros> <imagem> <CMD> <argumentos>
```

Os parâmetros mais utilizados na execução do container são:

Parâmetro	**Explicação**
-d	Execução do container em background
-i	Modo interativo. Mantém o STDIN aberto mesmo sem console anexado
-t	Aloca uma pseudo TTY
--rm	Automaticamente remove o container após finalização (Não funciona com -d)
--name	Nomear o container
-v	Mapeamento de volume
-p	Mapeamento de porta
-m	Limitar o uso de memória RAM
-c	Balancear o uso de CPU

Segue um exemplo simples no seguinte comando:

```
docker container run -it --rm --name meu_python python bash
```

De acordo com o comando acima, será iniciado um container com o nome meu_python, criado a partir da imagem python e o processo executado nesse container será o bash.

Vale lembrar que, caso o CMD não seja especificado no comando docker container run, é utilizado o valor padrão definido no Dockerfile da imagem utilizada. No nosso caso é python e seu comando padrão executa o binário python, ou seja, se não fosse especificado o bash, no final do comando de exemplo acima, ao invés de um shell bash do GNU/Linux, seria exibido um shell do python.

Mapeamento de volumes

Para realizar mapeamento de volume basta especificar qual origem do dado no host e onde deve ser montado dentro do container.

```
docker container run -it --rm -v "<host>:<container>" python
```

O uso de armazenamento é melhor explicado em capítulos futuros, por isso não detalharemos o uso desse parâmetro.

Mapeamento de portas

Para realizar o mapeamento de portas basta saber qual porta será mapeada no host e qual deve receber essa conexão dentro do container.

```
docker container run -it --rm -p "<host>:<container>" python
```

Um exemplo com a porta 80 do host para uma porta 8080 dentro do container tem o seguinte comando:

```
docker container run -it --rm -p 80:8080 python
```

Com o comando acima temos a porta 80 acessível no Docker host que repassa todas as conexões para a porta 8080 dentro do container. Ou seja, não é possível acessar a porta 8080 no endereço IP do Docker host, pois essa porta está acessível apenas dentro do container que é isolada a nível de rede, como já dito anteriormente.

Gerenciamento dos recursos

Na inicialização dos containers é possível especificar alguns limites de utilização dos recursos. Trataremos aqui apenas de memória RAM e CPU, os mais utilizados.

Para limitar o uso de memória RAM que pode ser utilizada por esse container, basta executar o comando abaixo:

```
docker container run -it --rm -m 512M python
```

Com o comando acima estamos limitando esse container a utilizar somente 512 MB de RAM.

Para balancear o uso da CPU pelos containers, utilizamos especificação de pesos para cada container, quanto menor o peso, menor sua prioridade no uso. Os pesos podem oscilar de 1 a 1024.

Caso não seja especificado o peso do container, ele usará o maior peso possível, nesse caso 1024.

Usaremos como exemplo o peso 512:

```
docker container run -it --rm -c 512 python
```

Para entendimento, vamos imaginar que três containers foram colocados em execução. Um deles tem o peso padrão 1024 e dois têm o peso 512. Caso os três processos demandem toda CPU o tempo de uso deles será dividido da seguinte maneira:

- O processo com peso 1024 usará 50% do tempo de processamento
- Os dois processos com peso 512 usarão 25% do tempo de processamento, cada.

Verificando a lista de containers

Para visualizar a lista de containers de um determinado Docker host utilizamos o comando docker container ls.

Esse comando é responsável por mostrar todos os containers, mesmo aqueles não mais em execução.

```
docker container ls <parâmetros>
```

Os parâmetros mais utilizados na execução do container são:

Parâmetro	**Explicação**
-a	Lista todos os containers, inclusive os desligados
-l	Lista os últimos containers, inclusive os desligados
-n	Lista os últimos N containers, inclusive os desligados
-q	Lista apenas os ids dos containers, ótimo para utilização em scripts

Gerenciamento de containers

Uma vez iniciado o container a partir de uma imagem é possível gerenciar a utilização com novos comandos.

Caso deseje desligar o container basta utilizar o comando docker stop. Ele recebe como argumento o ID ou nome do container. Ambos os dados podem ser obtidos com o docker ls, explicado no tópico anterior.

Um exemplo de uso:

```
docker container stop meu_python
```

No comando acima, caso houvesse um container chamado meu_python em execução, ele receberia um sinal SIGTERM e, caso não fosse desligado, receberia um SIGKILL depois de 10 segundos.

Caso deseje reiniciar o container que foi desligado e não iniciar um novo, basta executar o comando docker start:

```
docker container start meu_python
```

Vale ressaltar que a ideia dos containers é a de serem descartáveis. Caso você use o mesmo container por muito tempo sem descartá-lo, provavelmente está usando o Docker incorretamente. O Docker não é uma máquina, é um processo em execução. E, como todo processo, deve ser descartado para que outro possa tomar seu lugar na reinicialização do mesmo.

Criando sua própria imagem no Docker

Antes de explicarmos como criar sua imagem, vale a pena tocarmos em uma questão que normalmente confunde iniciantes do docker: “Imagem ou container?”

Qual a diferença entre Imagem e Container?

Traçando um paralelo com o conceito de orientação a objeto, a imagem é a classe e o container o objeto. A imagem é a abstração da infraestrutura em estado somente leitura, de onde será instanciado o container.

Todo container é iniciado a partir de uma imagem, dessa forma podemos concluir que nunca teremos uma imagem em execução.

Um container só pode ser iniciado a partir de uma única imagem. Caso deseje um comportamento diferente, será necessário customizar a imagem.

Anatomia da imagem

As imagens podem ser oficiais ou não oficiais.

Imagens oficiais e não oficiais

As imagens oficiais do docker são aquelas sem usuários em seus nomes. A imagem “ubuntu:16.04” é oficial, por outro lado, a imagem “nuagebec/ubuntu” não é oficial. Essa segunda imagem é mantida pelo usuário nuagebec, que mantém outras imagens não oficiais.

As imagens oficiais são mantidas pela empresa docker e disponibilizadas na nuvem docker.

O objetivo das imagens oficiais é prover um ambiente básico (ex. debian, alpine, ruby, python), um ponto de partida para criação de imagens pelos usuários, como explicaremos mais adiante, ainda nesse capítulo.

As imagens não oficiais são mantidas pelos usuários que as criaram. Falaremos sobre envio de imagens para nuvem docker em outro tópico.

Nome da imagem

O nome de uma imagem oficial é composto por duas partes. A primeira, a documentação chama de "repositório" e, a segunda, é chamada de "tag". No caso da imagem "ubuntu:14.04", ubuntu é o repositório e 14.04 é a tag.

Para o docker, o "repositório" é uma abstração do conjunto de imagens. Não confunda com o local de armazenamento das imagens, que detalharemos mais adiante. Já a "tag", é uma abstração para criar unidade dentro do conjunto de imagens definidas no "repositório".

Um "repositório" pode conter mais de uma "tag" e cada conjunto repositório:tag representa uma imagem diferente.

Execute o comando abaixo para visualizar todas as imagens que se encontram localmente na sua estação, nesse momento:

```
docker image list
```

Como criar imagens

Há duas formas de criar imagens customizadas: com commit e com Dockerfile.

Criando imagens com commit

É possível criar imagens executando o comando commit, relacionado a um container. Esse comando usa o status atual do container escolhido e cria a imagem com base nele.

Vamos ao exemplo. Primeiro criamos um container qualquer:

```
docker container run -it --name containercriado ubuntu:16.04 bash
```

Agora que estamos no bash do container, instalamos o nginx:

```
apt-get update
apt-get install nginx -y
exit
```

Paramos o container com o comando abaixo:

```
docker container stop containercriado
```

Agora, efetuamos o commit desse container em uma imagem:

```
docker container commit containercriado meuubuntu:nginx
```

No exemplo do comando acima, containercriado é o nome do container criado e modificado nos passos anteriores; o nome meuubuntu:nginx é a imagem resultante do commit; o estado do containercriado é armazenado em uma imagem chamada meuubuntu:nginx que, nesse caso, a única modificação que temos da imagem oficial do ubuntu na versão 16.04 é o pacote nginx instalado.

Para visualizar a lista de imagens e encontrar a que acabou de criar, execute novamente o comando abaixo:

```
docker image list
```

Para testar sua nova imagem, vamos criar um container a partir dela e verificar se o nginx está instalado:

```
docker container run -it --rm meuubuntu:nginx dpkg -l nginx
```

Se quiser validar, pode executar o mesmo comando na imagem oficial do ubuntu:

```
docker container run -it --rm ubuntu:16.04 dpkg -l nginx
```

Vale salientar que o método commit não é a melhor opção para criar imagens, pois, como verificamos, o processo de modificação da imagem é completamente manual e apresenta certa dificuldade para rastrear as mudanças efetuadas, uma vez que, o que foi modificado manualmente não é registrado, automaticamente, na estrutura do docker.

Criando imagens com Dockerfile

Quando se utiliza Dockerfile para gerar uma imagem, basicamente, é apresentada uma lista de instruções que serão aplicadas em determinada imagem para que outra imagem seja gerada com base nas modificações.

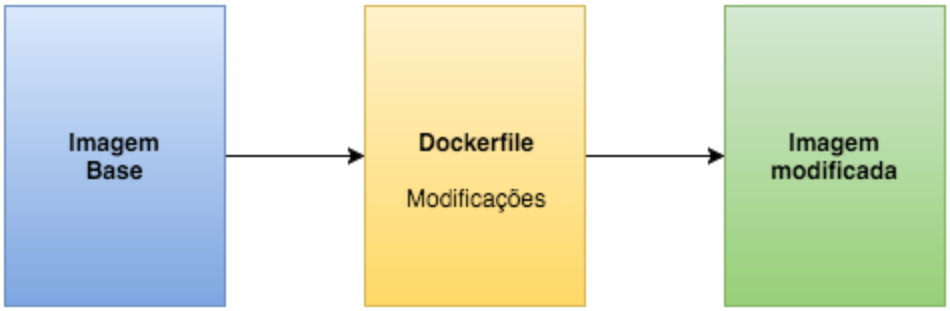


Figura 4

Podemos resumir que o arquivo Dockerfile, na verdade, representa a exata diferença entre uma determinada imagem, que aqui chamamos de base, e a imagem que se deseja criar. Nesse modelo temos total rastreabilidade sobre o que será modificado na nova imagem.

Voltemos ao exemplo da instalação do nginx no ubuntu 16.04.

Primeiro crie um arquivo qualquer para um teste futuro:

```
touch arquivo_teste
```

Crie um arquivo chamado Dockerfile e dentro dele o seguinte conteúdo:

```
FROM ubuntu:16.04
RUN apt-get update && apt-get install nginx -y
COPY arquivo_teste /tmp/arquivo_teste
CMD bash
```

No arquivo acima, utilizamos quatro instruções:

FROM para informar qual imagem usaremos como base, nesse caso foi ubuntu:16.04.

RUN para informar quais comandos serão executados nesse ambiente para efetuar as mudanças necessárias na infraestrutura do sistema. São como comandos executados no shell do ambiente, igual ao modelo por commit, mas nesse caso foi efetuado automaticamente e, é completamente rastreável, já que esse Dockerfile será armazenado no sistema de controle de versão.

COPY é usado para copiar arquivos da estação onde está executando a construção para dentro da imagem. Usamos um arquivo de teste apenas para exemplificar essa possibilidade, mas essa instrução é muito utilizada para enviar arquivos de configuração de ambiente e códigos para serem executados em serviços de aplicação.

CMD para informar qual comando será executado por padrão, caso nenhum seja informado na inicialização de um container a partir dessa imagem. No exemplo, colocamos o comando bash, se essa imagem for usada para iniciar um container e não informamos o comando, ele executará o bash.

Após construir seu Dockerfile basta executar o comando abaixo:

```
docker image build -t meuubuntu:nginx_auto .
```

Tal comando tem a opção “-t”, serve para informar o nome da imagem a ser criada. No caso, será meuubuntu:nginx_auto e o “.” ao final, informa qual contexto deve ser usado nessa construção de imagem. Todos os arquivos da pasta atual serão enviados para o serviço do docker e apenas eles podem ser usados para manipulações do Dockerfile (exemplo do uso do COPY).

A ordem importa

É importante atentar que o arquivo Dockerfile é uma sequência de instruções lidas do topo à base e cada linha é executada por vez. Se alguma instrução depender de outra instrução, essa dependência deve ser descrita mais acima no documento.

O resultado de cada instrução do arquivo é armazenado em cache local. Caso o Dockerfile não seja modificado na próxima criação da imagem (build), o processo não demorará, pois tudo estará no cache. Se houver alterações, apenas a instrução modificada e as posteriores serão executadas novamente.

A sugestão para melhor aproveitar o cache do Dockerfile é sempre manter as instruções frequentemente alteradas mais próximas da base do documento. Vale lembrar de atender também as dependências entre instruções.

Um exemplo para deixar mais claro:

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install nginx
RUN apt-get install php5
COPY arquivo_teste /tmp/arquivo_teste
CMD bash
```

Caso modifiquemos a terceira linha do arquivo e, ao invés de instalar o nginx, mudarmos para apache2, a instrução que faz o update no apt não será executada novamente, e sim a instalação do apache2, pois acabou de entrar no arquivo, assim como o php5 e a cópia do arquivo, pois todos eles são subsequentes a linha modificada.

Como podemos perceber, de posse do arquivo Dockerfile, é possível ter a exata noção de quais mudanças foram efetuadas na imagem e, assim, registrar as modificações no sistema de controle de versão.

Licença

Bibliografia

- https://github.com/gomex/docker-para-desenvolvedores