

Docker no Ambiente de Desenvolvimento

Gustavo Morini¹, William Roberto Pelissari¹, Anderson Burnes¹

¹Faculdade Alfa – Umuarama – PR – Brasil

gustavommh@gmail.com, wrpelissari@gmail.com, burnes@professorburnes.com

Abstract. *This article describes some advantages of using Docker in the development environment, its main commands, and how to set up a working environment for web development, making different technologies communicate to form a complete environment completely independent of the operating system. Once installed the tool can be used directly by the operating system terminal using basic commands for container management. The tool proved to be functional and matched expectations for the proposed work.*

Resumo. *Este artigo descreve algumas vantagens de se utilizar Docker no ambiente de desenvolvimento, seus principais comandos e como montar um ambiente funcional para o desenvolvimento web, fazendo tecnologias distintas se comunicarem formando um ambiente completo e totalmente independente do sistema operacional. Assim que instalada a ferramenta pode ser utilizada diretamente pelo terminal do sistema operacional utilizando comandos básicos para o gerenciamento dos containers. A ferramenta se mostrou funcional e correspondeu as expectativas para o trabalho proposto.*

1. Introdução

Compreender as principais características para se criar um ambiente de desenvolvimento virtualizado, customizável com *Docker* e *containers*. Analisar como gerenciar os *containers* utilizando *Docker*, montar um ambiente de desenvolvimento Web funcional, configurar *containers* individuais interligados entre si, demonstrar o funcionamento de um projeto rodando dentro do ambiente virtual.

O ambiente de desenvolvimento se mostra muitas vez bem específico para cada tipo de projeto.

Um ambiente convencional precisar ser totalmente refeito sempre que é preciso realizar uma manutenção na máquina hospedeira, dependendo da quantidade de elementos utilizados no ambiente de desenvolvimento essa pode ser uma tarefa árdua.

Configurar apenas uma vez este ambiente e compartilha-lo para toda a equipe de trabalho funcionando exatamente igual em todas as máquinas, independente de sistema operacional, vem a ser uma das justificativas de se usar *Docker* e *containers*.

A virtualização com *Docker* chega com o intuito de simplificar esta tarefa, mantendo um nível profissional de segurança, confiabilidade e padronização, sendo para um único desenvolvedor ou uma grande equipe, facilmente customizável e replicável.

2. Apresentação

Segundo (Gomes & Souza, 2015), *Docker* é uma ferramenta que possibilita gerenciar aplicações distintas dentro de ambientes isolados denominados *containers*. É uma alternativa para desenvolvedores criarem com rapidez e facilidade seu ambiente de desenvolvimento e realizar todas as tarefas rotineiras, inclusive testes com total segurança.

A principal funcionalidade é proporcionar ambientes isolados dentro da mesma máquina, independente de sistema operacional, disponível localmente ou remotamente através de mapeamento de portas. (Henrique Rocha Silva & — Licenciatura Orientador André Costa Drummond, 2017), descreve que um contêiner *Docker* basicamente, consiste em manter tudo o que é necessário para a aplicação ser executada, independente do sistema operacional nativo, compartilhando apenas o kernel da máquina hospedeira. Dentro do *container* existe o sistema operacional, configurações do usuário e metadados.

Quando o *Docker* inicia um *container* ele cria uma nova instância à partir da imagem original que é somente leitura e nessa nova instância a aplicação pode ser executada de maneira segura com permissões de leitura e escrita.

A figura 1 demonstra a diferença entre a virtualização tradicional e a utilizando *containers*. Nota-se que a virtualização exige duas camadas a mais, o *Hypervisor* que é o programa de virtualização como *virtualbox*, *virtual machine* e o *Guest OS*, que é o sistema operacional completo instalado dentro da *vm*. O *container* já possui os requisitos do sistema operacional sendo independente para executar uma aplicação específica.

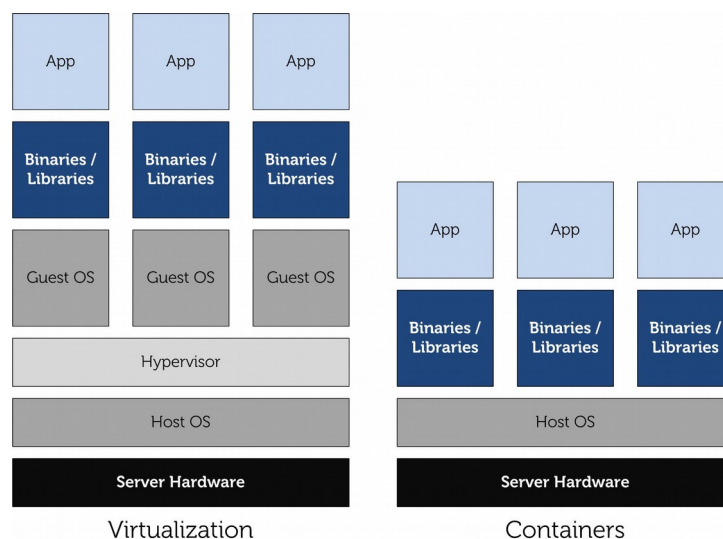


Figura 1. Virtualização vs Containers

3. Problematização

Quando pensamos em desenvolvimento web, precisamos de um servidor que interprete os códigos, além de um banco de dados onde seja possível popular os dados da aplicação. Geralmente é utilizado um ambiente local antes de passar para o ambiente de produção na nuvem. A instalação desses recursos pode ser um tanto trabalhoso, devido a utilizar várias tecnologias distintas, que trabalhem juntas.

4. Metodologia

Para a realização do estudo é necessário a instalação da ferramenta *Docker* e *Docker-Compose*, que auxilia na orquestração dos *containers*. Para os sistemas Windows e Mac existe um instalador pronto, já no caso dos sistemas baseados em Linux é possível instalar via terminal seguindo orientações do próprio site do desenvolvedor. Também foi utilizado uma aplicação de exemplo desenvolvida em *PHP*, com ajuda do *framework Codeigniter*, tendo uma conexão em *MariaDB* e utilizando o *PHPmyAdmin* para gerenciamento do banco de dados.

Montar com *Docker*, um ambiente de desenvolvimento, rodando virtualmente os programas necessários para o aplicativo funcionar, de fácil manutenção e compartilhamento para outras pessoas da equipe, sendo recriado com apenas um comando.

5. Resultados

Assim que instalado a ferramenta *Docker* é possível utilizar seus comandos diretamente no terminal do sistema, a (tabela 1) referência a função e seu respectivo comando no terminal, primeiramente vamos verificar se a instalação foi realizada corretamente exibindo a versão instalada no sistema. Assim podemos então criar o *container* à partir de uma imagem base, no caso o banco de dados *MariaDB*, neste comando se define o modo de execução, nome e senha de *root* para o novo *container*.

Quando se utiliza a instrução *run* sempre criará um novo *container*, após a criação podemos listá-los para mais detalhe, é possível acessá-lo e rodar comandos específicos dentro deles, inspecionar a fundo mostrando todas as informações referentes a ele com a instrução *inspect*.

Tabela 1. Comandos Docker.

Função	Comando do terminal
Verificar Instalação	<code>docker --version</code>
Criar um <i>container</i> à partir de uma imagem	<code>docker run -d --name teste-db -e MARIADB_ROOT_PASSWORD=mypass mariadb</code>
Verifica <i>containers</i> em execução	<code>docker container ls</code>
Acessar terminal do <i>container</i>	<code>docker exec -it teste-db bash</code>
Acessar o mysql dentro do <i>container</i>	<code>mysql -u root -p</code>
Inspecionar o <i>container</i> em execução	<code>docker inspect teste-db</code>
Parar o <i>container</i>	<code>docker container stop teste-db</code>
Iniciar novamente o <i>container</i> criado	<code>docker container start teste-db</code>
Listar todos os <i>containers</i> existentes na máquina	<code>docker container ls -a</code>
Remover o <i>container</i>	<code>docker container rm teste-db</code>

Este são os passos básicos para criação e gerenciamento de um *container* com *Docker*, poderíamos criar mais de um contêiner e interligá-los, mas seria algo muito verboso que não tem a necessidade, ainda mais que a proposta é facilitar a experiência do desenvolvedor, assim vamos utilizar para criar o ambiente completo o *Docker-compose*, que à partir de uma lista de comandos interpreta e passa para o *Docker* executá-los de maneira simples.

Dentro de uma pasta vazia criar um arquivo “*docker-compose.yml*” este será responsável por subir nossos serviços, o arquivo depende de uma endentação correta para compreender em qual camada estão os comandos e como interpretá-los.

```

1 | version: '3'
2 | services:
3 |
4 |   db:
5 |     image: bitnami/mariadb
6 |     environment:
7 |       - MARIADB_ROOT_PASSWORD=mypass
8 |
9 |   admin:
10 |    image: phpmyadmin/phpmyadmin
11 |    ports:
12 |      - 8080:80
13 |    environment:
14 |      - PMA_HOST=db
15 |    links:
16 |      - db
17 |
18 |   app:
19 |     image: bitnami/codeigniter:3
20 |     ports:
21 |       - 80:8000
22 |     volumes:
23 |       - './app'
24 |     links:
25 |       - db

```

Figura 2. Estrutura do arquivo *docker-compose.yml*.

Figura 2 mostra como a organização do arquivo `docker-compose.yml` é feita, a indentação é fundamental para este tipo de arquivo, através dela que define como cada comando deve ser interpretado, inicia com a versão dos comandos, após os serviços a serem criados.

Db é o nome do serviço, *image* é qual imagem o *Docker* vai procurar para espelhar e criar o *container db*, *environment* são as variáveis globais que são passadas no momento da instanciação, no projeto define a senha de *root* para *mypass*.

Próximo serviço *admin* referencia a imagem do *phpmyadmin*, realizando um mapeamento de portas para não haver conflito com o *app*, sua variável global é *PMA_HOST* que define o nome do *host* que irá se conectar, percebe que é o mesmo nome do serviço de banco de dados criado acima, *links* apenas define que irá se conectar com o serviço *db*.

Já o serviço *app* é a nossa aplicação, utilizando a imagem do *framework* que já tem o ambiente pré-configurado, define a porta a ser utilizada local e virtual, *volumes* é muito importante, define onde irão ficar os arquivos localmente sendo espelhados dentro do *container*, e novamente diz para se conectar com o serviço *db*.

No mesmo diretório é criada a pasta *myapp* nela colocar os arquivos da aplicação. Dentro da aplicação existe o arquivo de configuração do banco de dados chamado *database.php* onde devemos preencher os dados de acordo com o serviço *db* do arquivo *docker-compose.yml*.

Neste momento é possível iniciar os serviços descritos no *docker-compose.yml*, seguindo a (tabela 2) é possível acompanhar a função e cada comando executado no terminal, vamos iniciar os *containers*, onde uma primeira execução é feita buscando na nuvem as imagens referenciadas. A primeira execução pode demorar de acordo com a conexão e a quantidade de imagens a serem baixadas, após isso a iniciação é feita em poucos segundos. Após listar os *containers* exibindo detalhes como nome, id e portas onde estão rodando.

O *docker-compose* já inicia, para e remove todos os *containers* de uma só vez com apenas um comando.

Tabela 2. Comandos Docker.

Função	Comando terminal
Iniciar todos os <i>containers</i>	<i>docker-compose up -d</i>
Listar todos os <i>containers</i>	<i>docker-compose ps</i>
Parar todos os <i>containers</i>	<i>docker-compose stop</i>
Remover todos os <i>containers</i>	<i>docker-compose down</i>

Na lista de *containers* é possível observar seu nome, o comando interno que está sendo executado, o estado atual e em qual porta está rodando. O *phpmyadmin* pode ser acessado na URL: *http://localhost:8080*, Figura 03 e a aplicação URL: *http://localhost*, Figura 04.

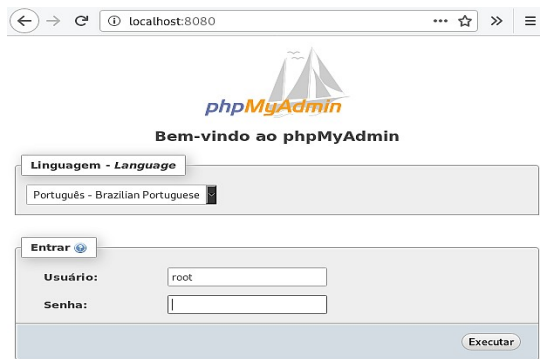


Figura 3. *http://localhost:8080*.

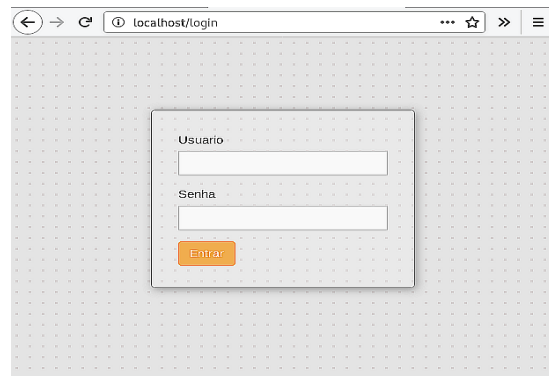


Figura 4. *http://localhost*.

Este é o ambiente completo rodando em *Docker*, todas as alterações feitas na pasta *myapp* serão repassadas para o *container* interpretando internamente e devolvendo na URL definida no arquivo *docker-compose.yml*. Assim foi concluído o ambiente de desenvolvimento virtual com *Docker*.

6. Considerações finais

A ferramenta é sim funcional, necessita o conhecimento de alguns comandos básicos, com um gerenciamento simples, com pouco estudo é possível criar e compartilhar ambientes complexos em apenas um arquivo central, executou com perfeição o trabalho proposto, mantém um sistema operacional limpo, livre de aplicações que não utilizamos no dia a dia, como ferramentas de testes, que podem ser criadas e apagadas rapidamente sem qualquer risco de conflito com outros aplicativos instalados. Mantém um nível de segurança alto, podendo manter informações dentro e fora do *container*, criar redes específicas isoladas da rede padrão. Por ser virtual e necessitar de adicionais para funcionar corretamente consome mais recursos da máquina host, tanto em disco quanto em processamento do que um ambiente local, mas menos que uma máquina virtual convencional que recria todo o sistema operacional do zero.

7. Referências

Gomes, R., & Souza, R. (2015). Docker – Infraestrutura como código, com autonomia e replicabilidade.

Henrique Rocha Silva, F., & — Licenciatura Orientador André Costa Drummond, C. (2017). Avaliação de Desempenho de Contêineres Docker para Aplicações do Supremo Tribunal Federal. Retrieved from http://bdm.unb.br/bitstream/10483/17796/1/2017_FlavioHenriqueSilva_tcc.pdf

Docker – Plataforma de gerenciamento de *containers*. Disponível em:

<<https://www.docker.com/>>. Acesso em: 01 jun. 2019.

Docker-Compose – Ferramenta auxiliar para *Docker*. Disponível em:

<<https://docs.docker.com/compose/>>. Acesso em: 01 jun. 2019.

MariaDB – Banco de dados relacional. Disponível em:

<<https://mariadb.org/>>. Acesso em: 29 jun. 2019.

PHP – Linguagem interpretada livre. Disponível em:

<<https://www.php.net/>>. Acesso em: 29 jun. 2019.

PhpMyAdmin – Gerenciador web de banco de dados. Disponível em:

<<https://www.phpmyadmin.net/>>. Acesso em: 29 jul. 2019.