Docker

Fernando Anselmo
http://fernandoanselmo.orgfree.com/wordpress/

Versão 1.5 em 1 de novembro de 2020

Resumo

Docker [1] veio para revolucionar a forma como é abordado o desenvolvimento e a implantação de aplicativos, em modo bem simples, é uma plataforma para construir e manter ambientes para a execução de sistemas distribuídos. Um projeto de código aberto que permite a criação de contêineres, a partir de imagens, leves e portáteis para diversas aplicações. Sua funcionalidade simplifica o uso dos LXC (LinuX Containers), que, basicamente, uma forma para isolamento de processo e sistemas (quase como uma virtualização), porém mais integrada ao Sistema Operacional. Os contêineres isolam o SO Base (host) e toda pilha de dependências da aplicação (bibliotecas, servidores, entre outros) com ganhos de performance.

1 Parte inicial

Provavelmente já ouviu falar sobre o Docker, quase todos os dias surgem notícias, por meio de inclusões nas redes sociais, em blogs ou eventos promovidos por diversas empresas do segmento de tecnologia. Possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um contêiner, e a partir desse momento, torna-se portável para qualquer outro sistema que contenha o Docker instalado.

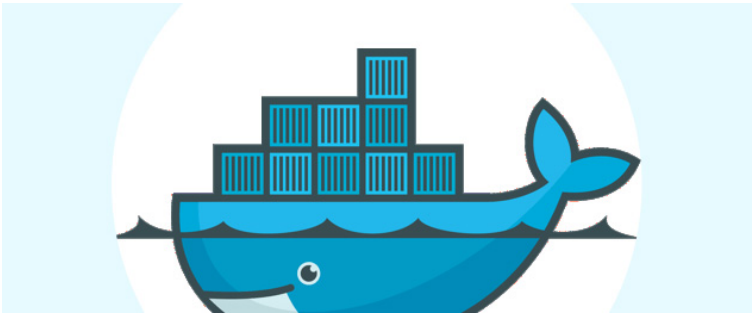

Figura 1: *Logo do Docker*

O Docker nasceu na empresa dotCloud, na época, fornecia hospedagem que utilizava LXC em quase todo seu ambiente. O Docker trabalha com um sistema de arquivos "empilháveis", denominado aufs, isso permite que a configuração do contêiner funcione de forma incremental, mais ou menos

como os “commits” do GIT. Docker é uma ferramenta que cria rapidamente ambientes isolados para desenvolver e implantar aplicativos. Trata-se de uma solução para profissionais de sistema desenvolverem, embarcarem, integrarem e executarem suas aplicações rapidamente.

Seu principal objetivo e proporcionar múltiplos ambientes isolados dentro do mesmo servidor, mas acessíveis externamente via tradução de portas. O conceito nada mais é do que isolar os recursos e as aplicações através de uma imagem (template), construir contêineres para otimizar deploy, performance, agilidade, entrega e principalmente o modo de compartilhar todos os recursos sejam físicos ou lógicos.

O Docker oferece um conjunto completo de ferramentas para o transporte de tudo o que constitui uma aplicação, seja sistemas ou máquinas (virtual ou física). Outra característica do Docker é a permissão para executar em qualquer sistema operacional baseado em Linux dentro de um contêiner com maior flexibilidade.

1.1 Imagens

As imagens são como “blocos de construção” que agem como ponto de partida para os contêineres. As imagens se tornam recipientes e os contêineres podem ser transformados em novas imagens otimizadas. Normalmente são imagens do host (como o Ubuntu), mas que podem ser altamente personalizadas para conter um SO básico (como o Alpine) juntamente com qualquer dependência que é instalada nela. Geralmente faz uso do UFS (Sistema de Arquivos Unix), e pode ser criada através do arquivo de configuração denominado “Dockerfile”.

Um contêiner não pode ser iniciado sem uma imagem. Através de uma imagem iniciada por um contêiner é possível gerar novas imagens, basta aplicar um “commit“ a cada mudança realizada.

Resumidamente:

- **Imagem**: é uma template que define a estrutura para os contêineres.
- **Contêiner**: simula o ambiente de execução como um todo, é uma “instância” da imagem.

1.2 O que é um Contêiner?

Docker usa o termo “Contêiner” para representar um ambiente em execução e que pode executar quase qualquer software; Seja este uma aplicação Web ou um serviço. O contêiner é tratado como o artefato, ou seja, o ambiente pode ser versionado e distribuído da mesma forma como é realizado com o código fonte de uma aplicação.

Os contêineres são compostos de namespaces e grupos de controle do Linux que fornecem isolamento de outros contêineres e do host. LXC é um tipo de virtualização, em nível de sistema operacional, que proporciona a execução de múltiplas instâncias isoladas de um determinado SO dentro de um único host. O conceito é simples e antigo, o comando “chroot” seu precursor mais famoso. Com o chroot é possível segregar os acessos de diretórios e evitar que o usuário tenha acesso à estrutura raiz (“/” ou root).

Linha do tempo das tecnologias envolvidas:

- **chroot** 1979
- **apparmor** 1998

- **libvirt** 2005
- **cgroups** 2007
- **LXC** 2008
- **Docker** 2013

Características dos contêineres:

- Definir recursos como memória, rede, sistema operacional, aplicação ou serviço.
- Realizar testes, desenvolvimento e estudos.
- Utilizar em ambiente de produção.
- Controlar os recursos como CPU, memória e HD através dos parâmetros de configuração, que são passados ao iniciar um contêiner, ou durante sua execução.

O contêiner é construído através de namepspaces, cgroups, chroot entre outras funcionalidades do Kernel para construir uma área isolada para a aplicação.

Os contêineres podem ser utilizados através de seguintes redes:

- **Bridge**. Cada contêiner iniciado no Docker é associado a uma rede especifica, e essa é a rede padrão para qualquer contêiner que não foi explicitamente especificado.
- **None**. Isola o contêiner para comunicações externas, ou seja, não receberá nenhuma interface para comunicação externa. A sua única interface de rede IP será a localhost.
- **Host**. Entrega para o contêiner todas as interfaces existentes no docker host. De certa forma isso agiliza a entrega dos pacotes.

2 Instalação do Docker no Ubuntu

Nesta seção será instalado e configurado o Docker para o sistema operacional Ubuntu 18.04. A instalação pode ser realizada da seguinte forma:

1. Instalar: `$ sudo apt install docker docker.io`
2. Adicionar seu usuário ao grupo docker: `$ sudo usermod -aG docker [seu usuário]`
3. Reiniciar o computador: `$ sudo reboot`
4. Verificar a versão: `$ docker -v`

2.1 Testar a instalação com um Hello World

Com um simples comando é possível testar todo o ambiente:
`$ docker run hello-world`

Se tudo estiver correto a imagem será baixada, um contêiner criado e visualizaremos a seguinte a mensagem:

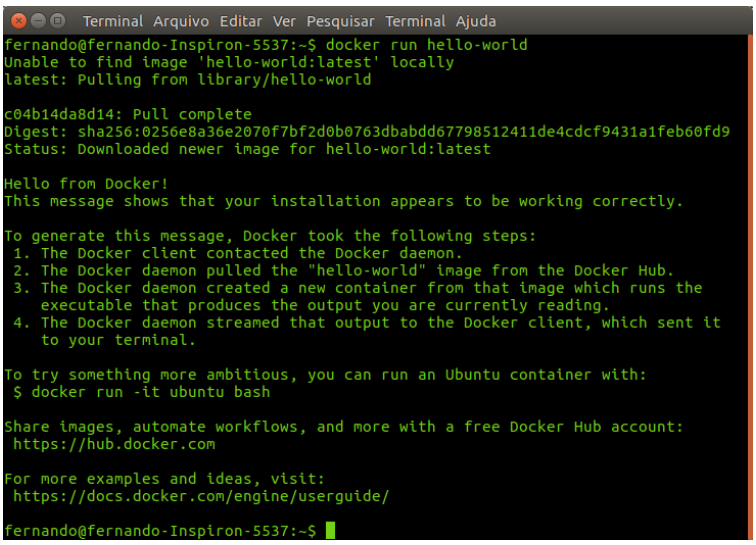

Figura 2: *Tela do Terminal com a execução do Contêiner*

3 Utilização do Docker

Antes de começar a utilizar o Docker é recomendável proceder um registro (gratuito) em seu repositório oficial de imagens. Acessar `https://hub.docker.com/` e proceder o registro.

3.1 Guia dos Comandos Básicos

Nos comandos a seguir, o termo [imagem] se refere ao nome ou ID da imagem, assim como o termo [contêiner] se refere ao nome ou ID do contêiner.

3.1.1 Comandos sobre o Docker

Obter informações sobre o Docker instalado:
`$ docker info`

Parar o serviço:
`$ sudo service docker stop`

Levantar o serviço:
`$ sudo service docker start`

Verificar o serviço:
`$ sudo service docker status`

Sobre a versão do Docker instalado:
`$ docker version`

3.1.2 Comandos sobre as Imagens

Verificar a existência de imagens:
```
$ docker images
```

Procurar uma imagem no repositório:
```
$ docker search [imagem]
```

Baixar uma imagem do repositório:
```
$ docker pull [imagem]
```

Construir uma imagem:
```
$ docker build -t [imagem] [caminho do dockerfile]
```

Subir uma imagem para o repositório:
```
$ docker push [imagem]
```

Verificar o histórico de criação de uma imagem:
```
$ docker image history [imagem]
```

Remover uma imagem:
```
$ docker rmi -f [imagem]
```

3.1.3 Comandos sobre os Contêiners

Criar um contêiner de uma imagem:
```
$ docker run --name [nome do Contêiner] [imagem]:[tag]
```

Criar um contêiner de uma imagem, em background e desviar para uma porta específica:
```
$ docker run -d -p [portahost]:[portacontiner] --name [contêiner] [imagem]
```

Iniciar um contêiner já criado:
```
$ docker start [contêiner]
```

Executar comando em um contêiner já iniciado:
```
$ docker exec [contêiner] [comando]
```

Iniciar uma sessão bash em um contêiner já iniciado:
```
$ docker exec -it [contêiner] bash
```

Listar todos os contêineres:
```
$ docker ps -a
```

Renomear um contêiner:
```
$ docker rename [nome antigo] [nome novo]
```

Ver os logs de um contêiner:
```
$ docker logs [contêiner]
```

Parar um contêiner:
```
$ docker stop [contêiner]
```

Remover um contêiner:

```
$ docker rm -f [contêiner]
```

Remover todos os contêineres parados:
```
$ docker system prune
```

Remover TODOS os contêineres
```
$ docker rm -f $(docker ps -a -q)
```

3.2 Sair e retornar de um contêiner

Por muitas vezes os contêineres são criados como ambientes (bash) no qual se pode executar comandos, mas pode ser necessário retornar ao host. No contêiner realizar a seguinte sequencia: `Ctrl+P+Q`. Para retornar ao ambiente do contêiner, utilizar o comando:
```
$ docker attach [imagem]
```

Para sair de vez do contêiner, realizar a seguinte sequencia: `Ctrl+D` ou com o comando `exit`.

3.3 Propriedade Volume

Volume é uma pasta dentro de um contêiner que está associada a outro que existe fora, ou seja, é um mapeamento para uma pasta existente na máquina host ou no dispositivo NFS remoto. O diretório que um volume mapeia existe independente de qualquer contêiner que o monte. Isso significa que se pode criar contêineres, gravar em volumes e, em seguida, destruí-los sem perder dados de aplicativos.

Os volumes são excelentes para compartilhar dados (ou estado) entre contêineres. É a maneira mais adequada para editar os fontes de uma aplicação dentro de um contêiner. Primeiro detalhe a realizar é verificar qual local que o contêiner usa para ler os códigos que desejamos manipular. Por exemplo, a pasta do Apache `/var/www/`. Caso o código local esteja na pasta `/home/usuario/codigo`, o comando do Docker para realizar mapeamento é o seguinte:
```
$ docker -d -v /home/usuario/codigo:/var/www imagem
```

Deste modo, tudo que for adicionado ou modificado na pasta `/home/usuario/codigo` do host docker, será refeito na pasta `/var/www` dentro do contêiner. **Importante**: alguns serviços específicos precisam que o serviço seja reiniciado em caso de mudança no código, nesse caso, basta parar o contêiner e reiniciar novamente.

4 Baixar e usar Imagens

Uma das grandes vantagens do Docker é o poder de instalar um software sem a preocupação de suas dependências, ou, no momento de sua desinstalação não deixar quaisquer resquícios de bibliotecas perdidas.

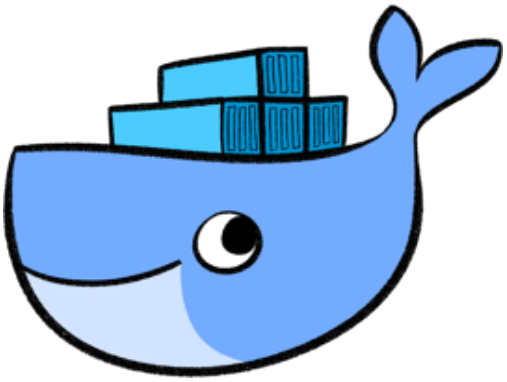

Figura 3: *Uso de imagens no Docker*

Nesta seção vamos verificar como podemos usar o Docker ao nosso favor com aplicativos prontos para o uso que são facilmente controlados pelo Docker. Essa é uma lista dos principais aplicativos que utilizo conteinerizados, não significa que são os melhores, únicos ou por ordem de importância, apenas alguns dos que utilizo.

4.1 MySQL

É um SGBD relacional, que utiliza a linguagem SQL. Atualmente é um dos SGBD mais populares.

Baixar a imagem oficial:
```
$ docker pull mysql
```

Criar uma instância do banco em um Contêiner:
```
$ docker run --name meu-mysql -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -d mysql
```

E nas próximas vezes, usamos para iniciar o MySQL:
```
$ docker start meu-mysql
```

Parar o MySQL:
```
$ docker stop meu-mysql
```

Entrar diretamente no gerenciador o comando:
```
$ docker exec -it meu-mysql sh -c 'exec mysql -u root -p'
```

Ou então:
```
$ docker exec -it meu-mysql bash
# mysql -u root -p
```

4.1.1 MySQL com o LiquiBase

É uma biblioteca independente, de código aberto para rastrear, gerenciar e aplicar alterações no esquema do banco de dados. Iniciado em 2006 para facilitar o rastreamento de alterações no banco de dados, especialmente em um ambiente ágil de desenvolvimento de software.

Como estamos com o banco MySQL no Docker, para realizarmos a conexão é exigido a seguinte parametrização:
```
$ liqui/liquibase
```

```
--driver=com.mysql.cj.jdbc.Driver
--classpath="/home/fernando/Aplicativos/libs/mysql-connector-java-8.0.11.jar"
--changeLogFile=db.changelog.xml
--url="jdbc:mysql://localhost:3306/teste?allowPublicKeyRetrival=TRUE&useSSL=FALSE"
--username=root
--password=root update
```

4.2 Apache Cassandra

É um projeto de sistema para banco de dados distribuído altamente escalável de segunda geração, que reúne a arquitetura do DynamoDB, da Amazon Web Services e modelo de dados baseado no BigTable, do Google.

Baixar a imagem oficial:
```
$ docker pull cassandra
```

Criar uma instância do banco em um Contêiner:
```
$ docker run --name meu-cassandra -d cassandra
```

Acessar o administrador:
```
$ docker run -it --link meu-cassandra:cassandra --rm cassandra sh -c
'exec cqlsh "$CASSANDRA_PORT_9042_TCP_ADDR"'
```

4.3 Postgres

É um SGBD relacional, desenvolvido como projeto de código aberto.

Baixar a imagem oficial:
```
$ sudo docker pull postgres
```

Criar uma instância do banco em um Contêiner:
```
$ docker run --name meu-post -e POSTGRES_PASSWORD=postgres -d -p 5432:5432 postgres
```

Chamar o PSQL:
```
$ docker exec -it meu-post psql -U postgres --password
```

Criar uma tabela:

```
1  postgres=# create database teste;
2
3  postgres=# \connect teste;
4
5  postgres=# create table cidade(
6  postgres(# id int not null,
7  postgres(# nome varchar(80),
8  postgres(# local point,
9  postgres(# primary key (id));
10
11 postgres=# insert into cidade values (1, 'San Francisco', '(-194.0, 53.0)');
12
13 postgres=# \d cidade;
```

Sair do PSQL:

```
postgres=# \q
```

4.4 MongoDB

É um SGBD orientado a documentos livre, de código aberto e multiplataforma, escrito na linguagem C++. Classificado como um programa para banco de dados NoSQL, o MongoDB usa documentos semelhantes a JSON com esquemas.

Baixar a imagem oficial:
`$ docker pull mongo`

Criar uma instância do banco em um Contêiner:
`$ docker run --name meu-mongo -p 27017:27017 -d mongo`

Acessar o administrador do MongoDB:
`$ docker exec -it meu-mongo mongo admin`

```
> show dbs
> use local
> show collections
> exit
```

4.5 TensorFlow

É uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. Contempla um sistema para criação e treinamento de redes neurais para detectar, decifrar e correlações entre padrões, análogo à forma como humanos aprendem e raciocinam.

Baixar a imagem oficial:
`$ docker pull tensorflow/tensorflow`

Criar a primeira vez em um Contêiner:
`$ docker run -it --name meu-tensor -p 8181:8888 tensorflow/tensorflow`

Anotar o número do Token e lembrar que a porta é 8181.

Iniciar novamente:
`$ docker start meu-tensor`
`$ docker exec -it meu-tensor bash`
`# jupyter notebook list`

4.6 Apache NiFi

Apache Nifi é utilizado na construção e fluxos de dados, é uma solução flexível e escalável para automatizar o fluxo de dados entre sistemas de software.

Baixar a imagem oficial:
```
$ docker pull apache/nifi
```

Criar um Contêiner:
```
$ docker run --name meu-nifi -p 8080:8080 -d apache/nifi
```

4.7 Pentaho

Pentaho é um software de código aberto para inteligência empresarial, desenvolvido em Java. A solução cobre as àreas de ETL, reporting, OLAP e mineração de dados.

Baixar uma imagem criada pelo Wellington Marinho:
```
$ docker pull wmarinho/pentaho
```

Criar um contêiner:
```
$ docker run --name meu-pentaho -p 8080:8080 -d wmarinho/pentaho
```

Acessar a imagem:
```
$ docker exec -it meu-pentaho bash
```

Podemos ver aonde está o Pentaho:
```
# echo $PENTAHO_HOME
```

Ou podemos acessá-lo na URL `http://localhost:8080` (usuário:admin e senha:password).

4.8 Airflow

Apache Airflow é uma plataforma de modo que programaticamente criar, agendar e monitorar *workflows*, que são definidos como código, e podem ser mantidos, versionados, testados e colaborados.

Baixar a imagem oficial:
```
$ docker pull apache/airflow
```

Criar um contêiner:
```
$ docker run -d -p 8080:8080 -e LOAD_EX=y
--name meu-airflow puckel/docker-airflow
```

Para adicionar uma consulta "Ad hoc"devemos configurar a conexão em: **Go to Admin** ▷ **Connections**, editar o arquivo **postgres_default** e inserir os seguintes valores (equivalente em airflow.cfg/docker-compose*.yml):

- Host: postgres
- Schema: airflow
- Login: airflow
- Password: airflow

E conseguiremos executar as seguintes URLs:
Airflow: `http:\\localhost:8080`
Flower: `http:\\localhost:5555`

4.9 Apache Superset

Aplicação de código aberto para visualizar dados e gerar *dashboards* interativos. Airbnb, Twitter e Yahoo são algumas das empresas que o utilizam. Possui uma excelente integração SQL através do **SQLAlchemy** e com o **Druid.io**.

Baixar a imagem configurada por amancevice:
```
$ docker pull amancevice/superset
```

Criar o contêiner:
```
$ docker run --name meu-superset -d -p 8088:8088 amancevice/superset
```

Iniciar o contêiner:
```
$ docker exec -it meu-superset superset-init
```

Verificar o IP no Conteiner do Postgres:
```
$ docker inspect meu-post | grep IP
```

No meu caso mostrou:
```
IPAddress:  172.17.0.3
```

Montar a conexão:
```
URI: postgresql+psycopg2://usuario:senha@172.17.0.3:5432/banco
```

4.10 SonarQube

É uma plataforma de código aberto para inspeção contínua da qualidade deste, para executar revisões automáticas com análise estática como forma de encontrar problemas, erros e vulnerabilidades de segurança que pode ser usado em mais de 20 linguagens de programação.

Baixar a imagem oficial:
```
$ docker pull sonarqube
```

Criar o contêiner:
```
$ docker run -d --name meu-sonar -p 9000:9000 -p 9092:9092 sonarqube
```

Acessar o SonarQube na URL `http:\\localhost:9000` com usuário e senha admin—admin

5 Criar uma imagem

Mas as vezes pode ser que a imagem disponível não seja adequada as necessidades que precisamos, é nesse momento que sentiremos a necessidade de criarmos nossas próprias imagens. Para isso precisamos entender a estrutura de um arquivo **Dockerfile**.

Dockerfile é um arquivo em formato texto que contém todos os comandos necessários para montar uma imagem Docker. Também é possível obter o Dockerfile de uma determinada imagem, modificar o que deseja e criar uma nova. Isso pode demorar um pouco mais, mas essa imagem será muito mais adequada e se obtém um total controle sobre esta, o que seria bem mais difícil no modelo de efetuar alterações em um contêiner.

Seus comandos principais são:

- **FROM**. Imagem base, é usada com nome da distribuição (Debian, Ubuntu), para não ocorrer a necessidade em se criar toda estrutura. Obrigatoriamente o primeiro comando.
- **LABEL**. Ajudar na organização das imagens por projeto, registrar informações de licenciamento, auxiliar na automação ou por quaisquer outros motivos.
- **MAINTAINER**. Especifica o autor da imagem.
- **RUN**. As instruções que serão executadas para criação da imagem.
- **ENV**. Para atualizar uma variável de ambiente PATH para o software que o contêiner instala. É útil para fornecer variáveis de ambiente necessárias especificadas para os serviços.
- **ADD e COPY**. Embora sejam funcionalmente semelhantes, de um modo geral, a COPY é preferida. Isso é porque é mais transparente que ADD. COPY suporta a cópia básica de arquivos locais no contêiner, enquanto ADD tem alguns recursos (como a extração de tar local-only e o suporte de URL remoto) que não são imediatamente óbvios. Consequentemente, o melhor uso para ADD é a auto-extração local do arquivo do tar na imagem.
- **ENTRYPOINT**. Especifica o que será executado ao iniciar o contêiner. Age como precedente à sintaxe CMD, ou seja, caso o ENTRYPOINT seja "top", o CMD pode ser "-b", que nesse caso executa o top em modo batch. Uma vez que o ENTRYPOINT não seja especificado, e um CMD seja usado, o ENTRYPOINT padrão é "/bin/sh -c".
- **EXPOSE**. Porta do contêiner que o Docker deve escutar, essa informação é utilizada para interconexão entre contêineres, ao utilizar links. EXPOSE não define qual porta será exposta para o hospedeiro ou tornar possível o acesso externo para portas do contêiner em questão. Para expor essas, utiliza-se em tempo de inicialização da imagem a flag "-p" ou "-P".
- **CMD**. Usada para executar um software contido pela imagem, juntamente com quaisquer argumentos, resumidamente o que deve ser executado quando o contêiner subir. Obrigatoriamente o último comando.

Por exemplo, esses são os passos necessários para se criar uma imagem para o banco MongoDB.

Criar uma pasta para abrigar o arquivo:
`$ mkdir mongod`

Entrar na pasta:
`$ cd mongod`

Criar um arquivo chamado **Dockerfile** com o seguinte conteúdo:

```
FROM ubuntu:yakkety
MAINTAINER Fernando Anselmo <fernando.anselmo74@gmail.com>

# Adiciona o usuario e o grupo
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb

RUN apt-get update \
  && apt-get install -y --no-install-recommends \
    numactl \
  && rm -rf /var/lib/apt/lists/*

# Adiciona o grab gosu
```

```
ENV GOSU_VERSION 1.7
RUN set -x \
  && apt-get update && apt-get install -y --no-install-recommends ca-certificates wget
    && rm -rf /var/lib/apt/lists/* \
  && wget -O /usr/local/bin/gosu
    "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg
    --print-architecture)" \
  && wget -O /usr/local/bin/gosu.asc
    "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg
    --print-architecture).asc" \
  && export GNUPGHOME="$(mktemp -d)" \
  && gpg --keyserver ha.pool.sks-keyservers.net --recv-keys
    B42F6819007F00F88E364FD4036A9C25BF357DD4 \
  && gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu \
  && rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc \
  && chmod +x /usr/local/bin/gosu \
  && gosu nobody true \
  && apt-get purge -y --auto-remove ca-certificates wget

# Importar a chave
RUN apt-key adv --keyserver ha.pool.sks-keyservers.net --recv-keys
    492EAFE8CD016A07919F1D2B9ECBEC467F0CEB10

ENV MONGO_MAJOR 3.0
ENV MONGO_VERSION 3.0.14
ENV MONGO_PACKAGE mongodb-org

RUN echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/$MONGO_MAJOR main" >
    /etc/apt/sources.list.d/mongodb-org.list

RUN set -x \
  && apt-get update \
  && apt-get install -y \
    ${MONGO_PACKAGE}=$MONGO_VERSION \
    ${MONGO_PACKAGE}-server=$MONGO_VERSION \
    ${MONGO_PACKAGE}-shell=$MONGO_VERSION \
    ${MONGO_PACKAGE}-mongos=$MONGO_VERSION \
    ${MONGO_PACKAGE}-tools=$MONGO_VERSION \
  && rm -rf /var/lib/apt/lists/* \
  && rm -rf /var/lib/mongodb \
  && mv /etc/mongod.conf /etc/mongod.conf.orig

RUN mkdir -p /data/db /data/configdb \
  && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]

EXPOSE 27017
CMD ["mongod"]
```

Criar a imagem:
`$ docker build -t mongod .`

Veja mais referencias sobre essas construções[2].

5.1 Trabalhar com Imagens derivadas do Alpine

Alpine é uma distribuição construída com musl libc e BusyBox, que é conhecida como canivete suíço, pois combina versões minúsculas de muitos utilitários comuns no UNIX em um único pequeno executável.

Algumas características do Alpine:

1. Enquanto que a menor imagem do Docker ocupa cerca de 130 MB de espaço em disco, a Alpine precisa de no máximo 8 MB, isso faz com que, mesmo com a montagem de todo o ambiente, nunca terá o mesmo tamanho de uma imagem tradicional, isso deixa o ambiente mais limpo e simples para gerenciar.
2. Possui apenas o necessário para o funcionamento da aplicação, se precisar de mais alguma biblioteca, deve ser instalada.
3. Não existe a preocupação em desativar ou remover lixos, pois estes não ocorrem.
4. Foi desenvolvida com o objetivo de proporcionar uma maior segurança, e para garantir isso, seus desenvolvedores se preocuparam em aprimorar os recursos de segurança do Kernel como **grsecurity/PaX**, além disso, todos os binários foram compilados em executáveis independente de posição, isso previne alguns uso problemas relacionados a *buffer overflow* ou tipos de ataques tais como estouro de pilha.

Vamos realizar um exemplo prático para entendermos melhor esse conceito.

Criar um arquivo chamado **app.js** com o seguinte conteúdo:

```
var http = require('http');
http.createServer(function(req,res) {
 res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
 res.end('Node.js no Docker com Alpine');
}).listen(8080);
```

Criar um arquivo chamado **package.json** com o seguinte conteúdo:

```
{
 "name": "docker_web_app",
 "version": "1.0.0",
 "description": "Node.js on Docker",
 "author": "First Last <first.last@example.com>",
 "main": "app.js"
}
```

Criar um arquivo chamado **Dockerfile** com o seguinte conteúdo:

```
FROM alpine:3.1
# Update
RUN apk add --update nodejs
# Cria a pasta da app
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
# Instala as dependencias da app
COPY package.json /usr/src/app/
RUN npm install
```

```
# copia a app
COPY . /usr/src/app
EXPOSE 8080
CMD ["node", "/usr/src/app/app.js"]
```

Para criar a imagem:
`$ docker build -t alpteste .`

Para executar a imagem:
`$ docker run -p 8080:8080 -d alpteste`

Verificar o contêiner em execução:
`$ docker ps`

Acessar a seguinte URL `http://localhost:8080`, e será mostrada a mensagem:
`Node.js no Docker com Alpine`.

E para interromper o contêiner:
`$ docker stop [id do contêiner]`

A imagem oficial do Node.js pode ser obtida em: `https://hub.docker.com/_/node/`.

5.2 Manipular Imagens no Repositório

O **Docker Hub** possui o objetivo de construir um repositório de Imagens e pode ser usado gratuitamente e de forma pública, através da seguinte URL `https://hub.docker.com/` (também é possível pagar uma conta para hospedar imagens particulares).

Criar uma pasta para hospedar os arquivos, e nesta um arquivo em Python chamado **app.py** com o seguinte conteúdo:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
   return "Hello Beautiful and Strange World!"

if __name__ == "__main__":
   app.run(host="0.0.0.0")
```

Criar o arquivo **Dockerfile** para construir a imagem:

```
FROM python:3.6.1-alpine
RUN pip install flask
CMD ["python","app.py"]
COPY app.py /app.py
```

Criar a imagem:
`$ docker image build -t python-hello-world .`

Criar o contêiner:
`$ docker run -p 5001:5000 -d python-hello-world`

Acessar a URL `http://localhost:5001`. Se tudo estiver OK podemos subir a imagem para o repositório.

Realizar o login na conta:
`$ docker login`

Criar uma tag para a imagem com seu nome de usuário (usarei o meu para exemplificar):
`$ docker tag python-hello-world fernandoanselmo/python-hello-world`

Subir a imagem para o repositório:
`$ docker push fernandoanselmo/python-hello-world`

Para realizar qualquer alteração basta construir a imagem novamente e executar o comando **push**.

6 Docker Compose

Compose é uma ferramenta para definir e executar aplicativos Docker com vários contêineres. Seus comandos básicos são:

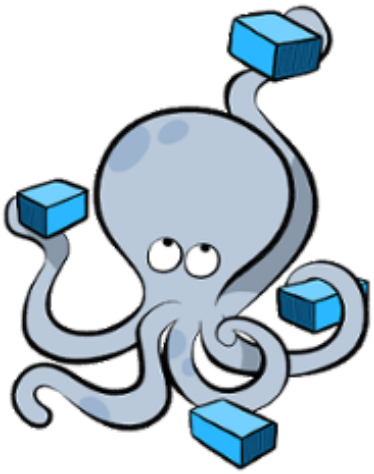

Figura 4: *Logo da Docker Compose*

Instalar o Docker Compose:
`$ sudo apt install docker-compose`

Construir uma composição de contêineres:
`$ docker-compose build`

Remover uma composição de contêineres:
`$ docker-compose rm`

Subir ou Descer uma composição de contêineres:
`$ docker-compose (up | down)`

Iniciar ou Parar uma composição de contêineres:

```
$ docker-compose (start | stop)
```

Executar uma composição de contêineres não iniciada:
```
$ docker-compose run -d
```

Executar uma composição de contêineres já iniciada:
```
$ docker-compose exec
```

Ver o log de uma composição de contêineres:
```
$ docker-compose logs (-f)
```

6.1 Caso PostgreSQL - Varias Instâncias

PostgreSQL é um sistema de gerenciamento de banco de dados relacional de objetos (ORDBMS) com ênfase em extensibilidade e conformidade com padrões. Lida com cargas de trabalho que variam de pequenos a grandes aplicativos voltados para a Internet (ou para armazenamento de dados) com muitos usuários simultâneos. Criar uma pasta chamada **teste**.

Nesta pasta, criar o arquivo **docker-compose.yml**:

```
version: '2'
 services:
  db:
   image: postgres
   restart: always
   environment:
    POSTGRES_PASSWORD: postgres
    POSTGRES_USER: postgres
   ports:
    - 5432:5432
   adminer:
    image: adminer
    restart: always
    ports:
     - 8080:8080
   client:
    image: postgres
    depends_on:
     - db
    command: psql -U postgres --password -h db

   db-legacy:
    image: postgres:9.5
    restart: always
    environment:
     POSTGRES_PASSWORD: postgres
     POSTGRES_USER: postgres
    ports:
     - 5533:5432
```

Levantar os contêineres:
```
$ docker-compose up
```

Para manipular, abrir uma nova aba do terminal na mesma pasta (Ctrl+Shift+T)

```
$ docker-compose ps
$ docker-compose run client
# create database teste;
# \connect teste;
# create table base(id serial not null, nome varchar(50), primary key (id));
# \d
# \dS+
# \d base
# \q
$ docker exec -it teste_db-legacy_1 psql -U postgres --password
```

E para encerrar:

```
$ docker-compose down
```

6.2 Caso NGINX

Nginx é um servidor web rápido, leve, e com inúmeras possibilidades de configuração para melhor performance.

Criar a seguinte estrutura de arquivos:

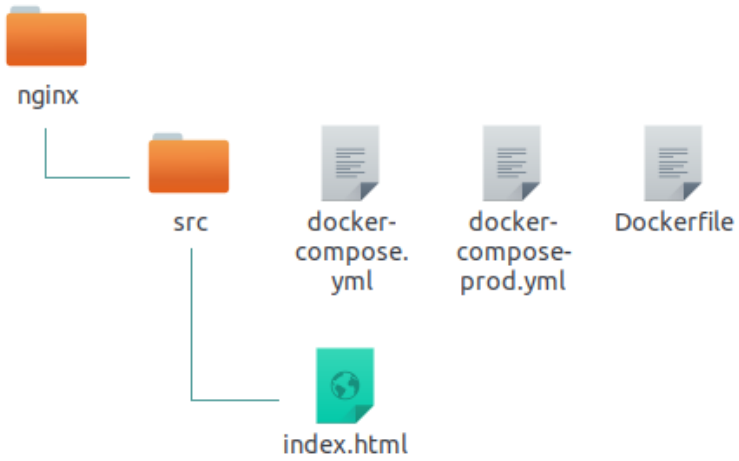

Figura 5: *Estrutura dos Arquivos*

Listagem do arquivo **src/index.html**:

```
<html>
 <body>
  <h1>Hello World</h1>
 </body>
</html>
```

Listagem do arquivo **Dockerfile**:

```
FROM nginx
COPY src /usr/share/nginx/html
```

Listagem do arquivo **docker-compose.yml**:

```
version: '2'
services:
 app:
  build: .
  image: app:1.0.0
  volumes:
   - ./src:/usr/share/nginx/html
  ports:
   - 8080:80
```

Listagem do arquivo **docker-compose-prod.yml**:

```
version: '2'
services:
 app:
  build: .
  image: app:1.0.0
  ports:
   - 80:80
```

Subir o Docker Compose para construir a estrutura:
`$ docker-compose up --build`

Acessar a URL `http://localhost:8080`

Para manipular, abrir uma nova aba do terminal na mesma pasta (Ctrl+Shift+T):
`$ docker-compose down`
`$ docker-compose -f docker-compose-prod.yml up --build`

Acessar a URL `http://localhost`

Encerrar o Docker Compose:
`$ docker-compose down`

6.3 Caso Node.js

Node.js é um interpretador de código JavaScript com o código aberto, focado em migrar o Javascript do lado do cliente para servidores.

Criar uma pasta e executar o seguinte comando:
`$ npm init`

Em seguida preparar o projeto com o Express.js:
`$ npm install express --save`

Modificar o arquivo **package.json** para:

```
{
 "name": "docnode",
 "version": "1.0.0",
 "description": "Exemplo do Node no docker",
 "main": "index.js",
 "scripts": {
```

```
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "fernandoanselmo",
 "license": "MIT",
 "dependencies": {
  "express": "^4.16.3"
 }
}
```

Criar o arquivo **index.js** com a seguinte codificação:

```
var express = require('express');
var app     = express();

app.get('/', function(req,res){
 res.send('Hello World!');
});

var port = 3000;
app.listen(port,function(){
 console.log('Listening on port:' + port);
});
```

Criar o arquivo **Dockerfile** com a seguinte codificação:

```
FROM node:6.5.0

WORKDIR /app

RUN npm install nodemon -g

COPY index.js /app/index.js
COPY package.json /app/package.json
RUN npm install

EXPOSE 3000
```

E o arquivo **docker-compose.yml** com a seguinte codificação:

```
db:
 image: mongo
 ports:
  - "27017:27017"
 restart: always
web:
 build: .
 volumes:
  - ./:/app
 ports:
  - "3000:3000"
 links:
  - db
 command: nodemon /app/server.js
```

Observe que no arquivos juntamos o banco MongoDB, deste modo teremos um esqueleto completo

para futuros projetos.

Subir os contêineres:
`$ docker-compose up --build`

Acessar a URL `http://localhost:3000`

Encerrar a composição:
`$ docker-compose down`

7 Conclusão

O Docker também estabelece um padrão para o empacotamento de soluções e os contêineres incluem tudo o que é necessário para executar os processos dentro deles, para que não seja necessário instalar dependências adicionais no host. Ou seja, uma vez criada a estrutura do ambiente esta pode ser facilmente replicada, usada como referência para a criação de novas estruturas. Docker fornece blocos construtivos fundamentais necessários às implantações de contêineres distribuídos através da filosofia PODA (*Package Once Deploy Anywhere*).

Foi citado nesta o que vem a ser Docker e suas vastas possibilidades, porém não se deve utilizá-lo para tratar o contêiner como uma máquina virtual. É apenas um serviço, nada mais que um processo de "host hospedeiro". O contêiner não pode ter uma vida longa (uptime), como ele é um processo do host hospedeiro, ele precisa ser encerrado e iniciado quando possível. Não armazene dados dentro do , utilize sempre o parâmetro volumes para armazenar os dados dinâmicos. E por fim, assegure que o host hospedeiro possui os recursos necessários de segurança para acesso nos contêineres.

Devido às propriedades de isolamento dos contêineres, podemos programar muitos deles em um único host sem nos preocuparmos com dependências conflitantes. Deste modo, economizaremos custos do servidor. Através do empacotamento dos componentes, das aplicações em seus próprios contêineres e da sua escalabilidade horizontal torna-se um simples processo de iniciar ou desligar múltiplas instâncias de cada componente.

Sou um entusiasta do mundo **Open Source** e novas tecnologias. Qual a diferença entre Livre e Open Source? Livre significa que esta apostila é gratuita e pode ser compartilhada a vontade. Open Source além de livre todos os arquivos que permitem a geração desta (chamados de arquivos fontes) devem ser disponibilizados para que qualquer pessoa possa modificar ao seu prazer, gerar novas, complementar ou fazer o que quiser. Os fontes da apostila (que foi produzida com o LaTex) está disponibilizado no GitHub [5]. Veja ainda outros artigos que publico sobre tecnologia através do meu Blog Oficial [3].

Referências

[1] Página Oficial do Docker
`https://www.docker.com/`

[2] Construção de Imagens
`https://docs.docker.com/engine/reference/builder/`

[3] Fernando Anselmo - Blog Oficial de Tecnologia
`http://www.fernandoanselmo.blogspot.com.br/`

[4] Encontre essa e outras publicações em
https://cetrex.academia.edu/FernandoAnselmo

[5] Repositório para os fontes da apostila
https://github.com/fernandoans/publicacoes