

PROJETO 3: DOCKER

Danilo Borges Dutra¹, Patrícia Duarte da Silva¹

¹Departamento de Ciência da Computação – Universidade Federal de Goiás (UFG)

dbd235@yahoo.com, patricia.duarte.dds@gmail.com

1. Tarefa 1: Orientação

a) Explicar o que é o docker e porque este pode ser considerado um sistema distribuído.

Docker é a principal empresa que faz desenvolvimento de tecnologias para Containers. Foi criada com o objetivo de facilitar o desenvolvimento, a implantação e a execução de aplicações em ambientes isolados. Todo processo é realizado sem necessidade de qualquer acesso privilegiado à infraestrutura corporativa. O Docker também disponibiliza uma nuvem pública para compartilhamento de ambientes prontos, que podem ser utilizados para viabilizar customizações para ambientes específicos. Se usar o Docker em servidores de produção, vai ter um alto ganho de escalabilidade e performance (mesma característica de um sistema distribuído).

b) Diferencie containers de máquinas virtuais.

Containers: São uma forma de virtualização a nível de sistema operacional, que permite rodar múltiplos sistemas isolados em um único sistema operacional real. Ou seja, consegue compartilhar o mesmo kernel do sistema operacional. Você divide responsabilidades, isolando os processos de cada ferramenta de desenvolvimento. E isso garante que nenhuma ferramenta, se intrometa no funcionamento da outra.

Máquinas virtuais: Em uma máquina virtual pode ser usadas vários recursos e ferramentas rodando no mesmo sistema operacional.

c) Instale o docker no Linux (máquina virtual). Distribuição recomendada: Ubuntu 16.04 LTS. Descreva detalhadamente os passos de instalação.

O Docker foi instalado no Ubuntu 18.04.3 LTS. Foi dado os seguintes passos no Terminal, para a sua instalação:

Primeiramente, foi digitado o comando *sudo passwd root* para definir a senha do usuário root da minha máquina, em seguida definimos uma senha, e por fim entramos no terminal como root com o comando *su root*.

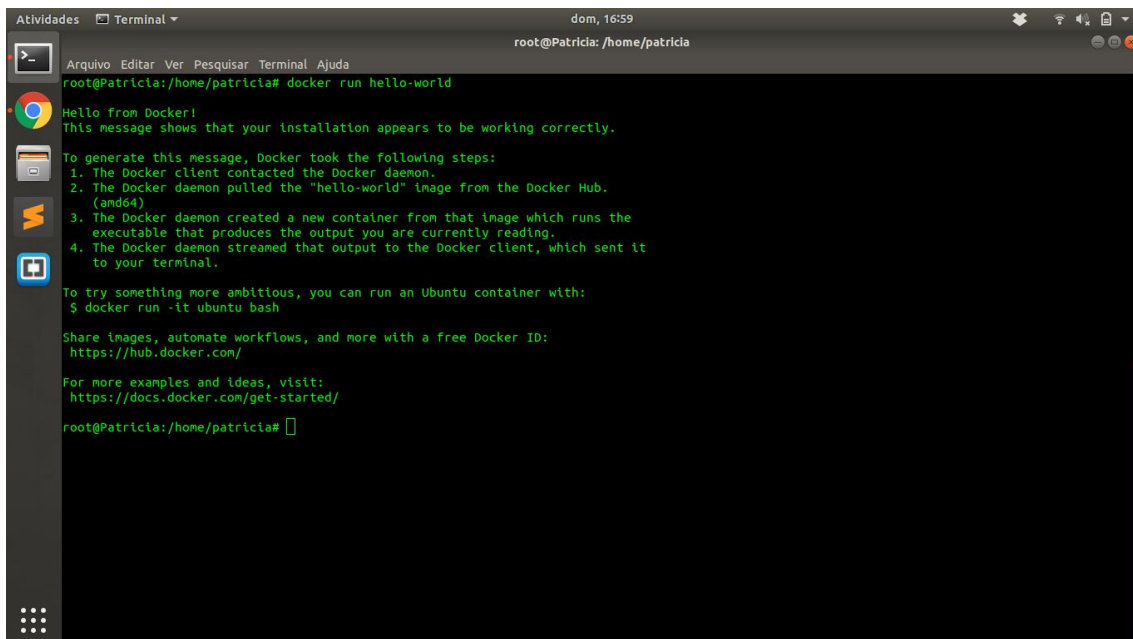
Como usuário root, foi dado somente o comando para a instalação do docker: *apt-get install docker.io*

d) Confirme sua versão do docker e execute o primeiro aplicativo (hello-world).

Com o comando *version docker*, identificamos que a versão do docker é a 19.03.4.

Para executar o primeiro aplicativo (hello-world), primeiro foi baixada a imagem

hello-world com o comando *docker pull hello-world*, em seguida executamos o nosso primeiro hello world com o seguinte comando, e tivemos o seguinte resultado:

A terminal window titled 'Terminal' with a menu bar (Arquivo, Editar, Ver, Pesquisar, Terminal, Ajuda) and a status bar (dom, 16:59, root@Patricia: /home/patricia). The command 'root@Patricia:/home/patricia# docker run hello-world' has been executed. The output is green text on a black background, showing a 'Hello from Docker!' message and a list of steps taken by Docker to run the container. It also provides links to Docker Hub and documentation.

```
root@Patricia:/home/patricia# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

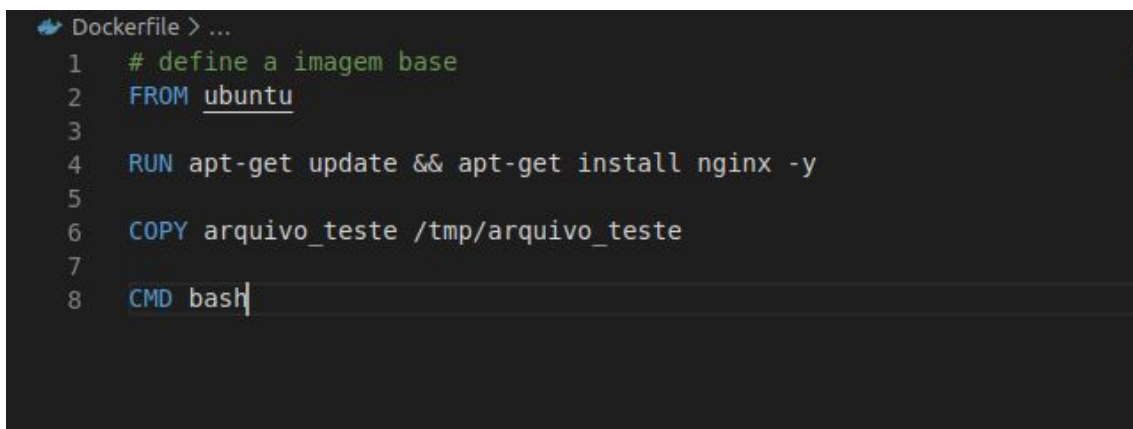
root@Patricia:/home/patricia#
```

(Fonte: Autoria própria)

2. Tarefa 2: Containers

a) Defina um container com um Dockerfile

Foi criado o arquivo chamado *Dockerfile* com nenhuma extensão. E nele foi definida as instruções a seguir:

A screenshot of a text editor showing the content of a Dockerfile. The file has a title bar 'Dockerfile > ...'. The content consists of eight lines of instructions for building a Docker image based on Ubuntu, installing nginx, copying a test file, and setting the command to bash.

```
Dockerfile > ...
1  # define a imagem base
2  FROM ubuntu
3
4  RUN apt-get update && apt-get install nginx -y
5
6  COPY arquivo_teste /tmp/arquivo_teste
7
8  CMD bash
```

Em seguida foi dado os passos:

- Foi criado o arquivo_teste com o comando: *touch arquivo_teste*
Dentro do terminal root foi dado:
- *docker build .*
- *docker build -t primeira_imagem:1.0 .*
- *docker run -ti ubuntu*, e será executado o sistema operacional

ubuntu.(Executando em foreground)

b) Teste seu ambiente usando a interface bridge ou ainda, crie um canal de comunicação específico para sua máquina virtual (ex. um barramento host only entre a máquina virtual e o host para testes).

- Primeiramente foi baixado o host docker com o seguinte comando no root:

```
base=https://github.com/docker/machine/releases/download/v0.16.0 && curl -L  
$base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine && sudo mv  
/tmp/docker-machine /usr/local/bin/docker-machine && chmod +x  
/usr/local/bin/docker-machine
```

- Posteriormente, foi dado o seguinte comando para executar o host docker:

```
docker-machine create --driver virtualbox linuxtips
```

- Foi dado o seguinte comando para entrar no ambiente do virtualbox:

```
docker-machine env linuxtips
```

- Através do resultado do comando anterior, foi dado o seguinte comando no terminal:

```
eval $(docker-machine env linuxtips)
```

- Em seguida teve o seguinte comando e resultado para testar o container do host docker:

```
root@Patricia:/home/patricia# docker run busybox echo "Rodando um container"  
Unable to find image 'busybox:latest' locally  
latest: Pulling from library/busybox  
0f8c40e1270f: Pull complete  
Digest: sha256:1303dbf110c57f3edf68d9f5a16c082ec06c4cf7604831669faf2c712260b5a0  
Status: Downloaded newer image for busybox:latest  
Rodando um container  
root@Patricia:/home/patricia#
```

c) Estude o aplicativo app.py. Descreva sua função.

Neste exemplo *app.py* redis é o nome do host do container redis na rede do aplicativo. A porta padrão para Redis é 6379. O loop de repetição básico da função `get_hit_count` permite tentar a solicitação várias vezes se o serviço redis não estiver disponível. Isso é útil na inicialização enquanto o aplicativo fica on-line, mas também o torna mais resistente se o serviço Redis precisar ser reiniciado a qualquer momento durante a vida útil do aplicativo. Em um cluster, isso também ajuda a lidar com quedas momentâneas de conexão entre nós.

d) Execute o app (em foreground e background).

Foi dado os seguintes comandos para o background:

- *docker-compose up*
- *apt install docker-compose*
- *docker-compose up*

Enquanto o comando *docker-compose up* estava executando no terminal, no navegador foi digitado *localhost:5000* para ver a aplicação em execução.

Na página tem a seguinte mensagem: Hello World! I have been seen 1 times.

Quando recarrega a página, vai aumentando a quantidade de times.

A execução foreground foi executado acima.

e) Crie sua conta docker e mande sua imagem para um repositório (Tag/Publish the image).

Primeiramente, foi criada a conta no docker. Em seguida foi criado um repositório chamado *testing*. Já no terminal da máquina foi dado o comando *docker login* e entramos com login e senha.

Foi dado o comando: *docker tag 6f63d037b592 patriciaduartedds/testing:alpine*

Em seguida o push para subir o container pro docker hub: *docker push patriciaduartedds/testing*

f) Execute a imagem a partir do repositório (Pull and run).

Foi dado o comando *docker run patriciaduartedds/redis*

3. Tarefa 3

a) Explique o que são serviços.

Os serviços são realmente apenas "contêineres em produção". Um serviço executa apenas uma imagem, mas codifica a maneira como a imagem é executada - quais portas deve usar, quantas réplicas do contêiner devem ser executadas para que o serviço tenha a capacidade necessária e em breve.

b) Crie seu primeiro arquivo docker-compose.yml.

version: "3"

services:

web:

replace username/repo:tag with your name and image details

image: username/repo:tag

deploy:

replicas: 5

resources:

limits:

cpus: "0.1"

memory: 50M

restart_policy:

condition: on-failure

ports:

- "4000:80"

networks:

- *webnet*

networks:

webnet:

c) Execute seu primeiro aplicativo de balanceamento de carga.

Foi dado os seguintes comandos:

- *docker stack deploy -c docker-compose.yml getstartedlab*
- *docker service ps getstartedlab_web*
- *docker container ls -q*

d) Demonstre como realizar a “escalabilidade” do aplicativo (scale the app)

Foi dado os seguintes comandos:

```
root@Patricia:/home/patricia/composetest# docker stack deploy -c docker-compose.yml getstartedlab
Updating service getstartedlab_web (id: wk35wizc2pxs3uu9w1ssh2a9j)
root@Patricia:/home/patricia/composetest# docker container ls -q
```

e) Desabilite o seu app e o swarm (take down).

```
root@Patricia:/home/patricia/composetest# docker stack rm getstartedlab
Removing service getstartedlab_web
Removing network getstartedlab_webnet
root@Patricia:/home/patricia/composetest# docker swarm leave --force
Node left the swarm.
```

4. Tarefa 4

Swarm é um orquestrador de containers desenvolvido pela própria equipe de desenvolvimento do Docker e integrado nativamente nas novas versões. Assim como o kubernetes, é utilizado para criar e gerenciar clusters de containers.

Um *swarm cluster* agrupa nós de uma mesma aplicação. Os nós conectados podem ser classificados como worker ou manager. Para garantir níveis de redundância do serviços, pelo menos dois nós managers devem serem utilizados.

Para a utilização do swarm basta iniciá-lo, vide que já mesmo já vem integrado na instalação do próprio Docker, Para ativação do um *swarm*, foi executado o seguinte comando para iniciar o processo do *swarm* junto ao sistema do Docker:

- *sudo docker swarm init.*

Como resultado do comando, foi gerado um código para que outras máquinas possam se conectar ao *swarm*. Também foi mostrado que a máquina atual em que o comando foi executado agora trabalha como um manager., conforme é mostrado na figura abaixo:

```
danbd@note: ~  
danbd@note:~$ sudo docker swarm init  
Swarm initialized: current node (e8fe8wvljw54oloj1r8z8duaw) is now a manager.  
  
To add a worker to this swarm, run the following command:  
  
    docker swarm join --token SWMTKN-1-05uarvlovejh46ysv95frp70dyk353ti6ikilqacr4jnv8njoy0-162fzur8aw  
k5yl5yy12h1f7zq 192.168.0.3:2377  
  
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Para verificar o status de funcionamento do *swarm* pode-se utilizar o comando abaixo:

- `sudo docker node ls`

```
danbd@note: ~  
danbd@note:~$ sudo docker node ls  
ID                                HOSTNAME      STATUS      AVAILABILITY      MANAGER STATUS      ENGINE VERSION  
e8fe8wvljw54oloj1r8z8duaw *      note          Ready       Active             Leader               19.03.4  
danbd@note:~$
```

Como resultado do comando, será listado todos *swarm* e seus respectivos status de funcionamento e outras informações relevantes.

Criação de um cluster

Para a criação de um cluster local de teste foi utilizado o docker machine, configurado com uma integração com o Virtualbox para a criação dos nós com a base do Docker. Para isto, primeiramente foi utilizado o seguinte comando para a instalação do docker-machine:

- `base=https://github.com/docker/machine/releases/download/v0.16.0 && curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine && sudo mv /tmp/docker-machine /usr/local/bin/docker-machine && chmod +x /usr/local/bin/docker-machine`

Com o Docker Machine devidamente instalado, foram utilizados os seguintes comandos para a criação das máquinas virtuais no Docker Machine:

- `docker-machine create --driver virtualbox node1`
- `docker-machine create --driver virtualbox node2`

O comando indica o drive de conexão, neste caso com o Virtualbox o nome que será dado ou nó. O resultado pode ser conferido na imagem abaixo para o primeiro nó criado:

```
danbd@note: ~  
danbd@note:~$ docker-machine create --driver virtualbox node1  
Creating CA: /home/danbd/.docker/machine/certs/ca.pem  
Creating client certificate: /home/danbd/.docker/machine/certs/cert.pem  
Running pre-create checks...  
(node1) Image cache directory does not exist, creating it at /home/danbd/.docker/machine/cache...  
(node1) No default Boot2Docker ISO found locally, downloading the latest release...  
(node1) Latest release for github.com/boot2docker/boot2docker is v19.03.5  
(node1) Downloading /home/danbd/.docker/machine/cache/boot2docker.iso from https://github.com/boot2docker/boot2docker/releases/download/v19.03.5/boot2docker.iso...  
(node1) 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%  
Creating machine...  
(node1) Copying /home/danbd/.docker/machine/cache/boot2docker.iso to /home/danbd/.docker/machine/machines/node1/boot2docker.iso...  
(node1) Creating VirtualBox VM...  
(node1) Creating SSH key...  
(node1) Starting the VM...  
(node1) Check network to re-create if needed...  
(node1) Found a new host-only adapter: "vboxnet0"  
(node1) Waiting for an IP...  
Waiting for machine to be running, this may take a few minutes...  
Detecting operating system of created instance...  
Waiting for SSH to be available...  
Detecting the provisioner...  
Provisioning with boot2docker...  
Copying certs to the local machine directory...  
Copying certs to the remote machine...  
Setting Docker configuration on the remote daemon...  
Checking connection to Docker...  
Docker is up and running!  
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env node1
```

Com uma forma de visualizar as máquinas já criadas e suas respectivas informações, como o endereço IP, o comando *docker-machine ls* listar os nós virtuais criados, com conforme é mostrado na figura abaixo:

```
danbd@note: ~  
danbd@note:~$ docker-machine ls  
NAME      ACTIVE   DRIVER      STATE     URL                         SWARM   DOCKER   ERRORS  
node1     -        virtualbox  Running   tcp://192.168.99.101:2376   -        v19.03.5  
node2     -        virtualbox  Running   tcp://192.168.99.100:2376   -        v19.03.5
```

f) Inicialize o swarm e adicione nós.

Não teve como executar.

g) Instale (deploy) seu app no swarm cluster.

Não teve como executar.

h) Acesse o seu cluster.

Não teve como executar.

i) Finalize e reinicie (cleanup and reboot).

Não teve como executar.

TAREFA 5: Stacks

a) Adicione um novo serviço e reinstale (redeploy).

Não teve como executar.

b) Demonstre a persistência de dados (persist the data).

Não teve como executar.

TAREFA 6: questões de instalação (deployment)

Verifique e relate as opções de instalação disponíveis para o Docker. Durante esta prática de laboratório, quais opções você testou?

Não teve como executar.

6. Referências

Como Instalar e Configurar o Docker no Ubuntu 18.04. **Hostinger Blog**, 2019. Disponível em: <<https://www.hostinger.com.br/tutoriais/install-docker-ubuntu>>. Acesso em: 23/10/2019. **Docker Docs**. Disponível em: <<https://docs.docker.com/v18.09/>> Acesso em:19/11/2019.