

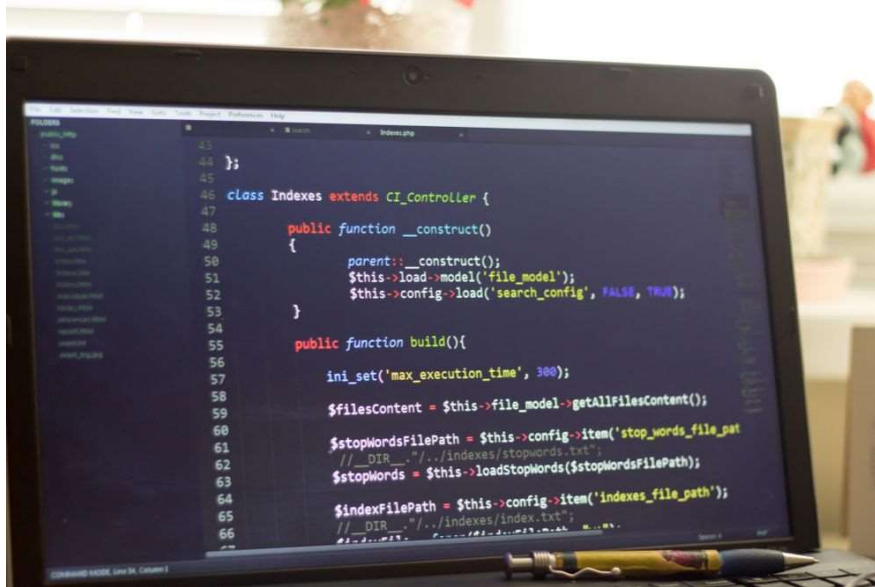
NÃO PODE FALTAR

## SIMULANDO SISTEMAS DISTRIBUÍDOS COM DOCKER

Caique Silva Pereira

### O QUE É DOCKER?

O Docker é uma tecnologia de software que fornece contêineres, é uma das principais plataformas de containerização.



Fonte: Shutterstock.

### Deseja ouvir este material?

Áudio disponível no material digital.

### PRATICAR PARA APRENDER

Caro aluno, você já pensou de que maneira são orquestrados os serviços quando acessamos um website? Uma das principais plataformas de containerização utilizadas atualmente é o Docker, principal assunto desta seção. Aqui vamos trabalhar com a instalação do Docker no sistema operacional GNU/Linux Ubuntu e aprender alguns dos principais comandos que podem ser utilizados na plataforma Docker. Você já parou para pensar em quais comandos devem ser usados para que essa orquestração de serviços funcione adequadamente?

Quando falamos de sistemas distribuídos, é inevitável falar sobre virtualização e containerização. Na prática, essas tecnologias são frequentemente utilizadas por grandes empresas. Você concluiu seu curso e está participando de um processo seletivo para uma oportunidade como trainee na área de DevOps da maior empresa nacional de portal de notícias, cujos clientes que consomem o conteúdo disponibilizado por essa empresa são bancos, em sua maioria. Você está se saindo muito bem no processo seletivo e agora será submetido ao teste final, uma atividade prática com Docker.

O coordenador está gostando de seu desempenho nos testes e diz que, caso você consiga orquestrar o servidor web Apache em um *cluster* simples, a vaga será sua. Desta forma:

1. Crie um *cluster* com cinco réplicas do servidor web Apache utilizando o Docker Swarm;
2. Verifique em quais nós do *cluster* esse serviço está rodando;
3. Acesse a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó onde esse serviço web estiver rodando.

Para completar o desafio, nessa seção você verá, em detalhes, como se utiliza o Docker, incluindo comandos específicos a serem utilizados, tanto para configuração, quanto para constatação de que o serviço de Apache está funcionando de maneira adequada. Ficou curioso? Vamos lá!

## CONCEITO-CHAVE

O Docker é uma famosa plataforma genérica de containerização. Conforme já estudamos, o conceito de containerização é parecido com virtualização, porém é considerado “mais leve”. Contêineres são muito populares atualmente devido à facilidade e à flexibilidade que advêm de seu uso. Portanto, agora chegou a hora de colocar a mão na massa e aprender a utilizar essa famosa ferramenta.

## ■ INSTALAÇÃO DO DOCKER

Vamos fazer a instalação do Docker em uma das distribuições populares GNU/Linux, o sistema operacional Ubuntu, cuja versão utilizada foi a 14.04.5 LTS. Para isso, devemos seguir os passos a seguir. Todo o procedimento deve ser feito com um usuário com permissões de administrador. Nesse caso, utilizaremos o *root* através do comando `sudo su`.

Você também pode instalar o Docker no sistema operacional Windows, fazendo o download da versão mais atual no Portal Docker (DOCKER, [s.d.]). O processo de instalação é bem simples, é preciso somente avançar as etapas do instalador.

### ASSIMILE

Todos os procedimentos utilizados para instalação foram retirados da documentação oficial do Docker, (DOCKER, [s.d.]). As ferramentas de tecnologia atualizam-se a tal velocidade que somente acompanhando a documentação oficial é possível manter-se atualizado. Portanto, como um profissional engajado, procure sempre pelas fontes oficiais das ferramentas.

Antes de instalar a ferramenta, devemos remover versões anteriores do Docker que possam estar instaladas, usando o comando:

```
sudo apt-get remove docker docker-engine docker.io
```

Caso não tenha nenhuma versão instalada, será exibida a mensagem que foi impossível encontrar o pacote *docker-engine*.

Antes de instalar o Docker CE pela primeira vez em uma nova máquina host, você precisa configurar o repositório do Docker, atualizando os pacotes de sua máquina. Depois, você pode instalar e atualizar o Docker do repositório. Portanto, execute os comandos que vão atualizar a máquina:

```
sudo apt-get update
```

O comando acima tem o objetivo de realizar uma atualização de pacotes do Ubuntu. Na execução desse comando, o tempo pode variar de acordo com os pacotes que precisam ser atualizados, da velocidade da máquina e da conexão com a internet.

Através do comando abaixo atualizamos os pacotes necessários para a instalação do Docker. Para obter uma instalação bem-sucedida, é bom ter os programas utilizados nela em suas últimas versões:

```
sudo apt-get install \ apt-transport-https \ ca-certificates \ curl \ software-properties-common
```

Após atualizar os repositórios, vamos adicionar o repositório de instalação do Docker, usando o seguinte comando:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Nesse comando, estamos fazendo o apontamento do caminho de instalação oficial do Docker que o Ubuntu deve acessar quando queremos efetuar a instalação. Após o apontamento da URL indicando o local para download o Docker para Ubuntu, será possível adicionar o repositório no próximo comando:

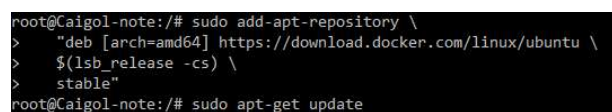
```
sudo add-apt-repository \ "deb  
[arch=amd64]https://download.docker.com/linux/ubuntu \ ${lsb_release -cs} \  
stable"
```

O comando utiliza a permissão considerada `admin` nos sistemas operacionais da família Linux, através da palavra `sudo`. Depois de usar a permissão de usuário do `sudo`, utilizamos o `add-apt-repository` para adicionar o repositório, que pode ser comparado a uma loja de aplicativos, responsável pelo download do Docker na versão Ubuntu. O restante do comando é o caminho do repositório.

Após adicionar o repositório para download do Docker, como mostra a Figura 4.17, devemos mais uma vez atualizar o `apt-get`, conforme o comando seguinte para aplicar as alterações:

```
sudo apt-get update
```

Figura 4.17 | Saída do comando que adiciona o repositório de instalação do Docker



```
root@Caigol-note:/# sudo add-apt-repository \  
> "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
> ${lsb_release -cs} \  
> stable"  
root@Caigol-note:/# sudo apt-get update
```

Fonte: captura de tela elaborada pelo autor.

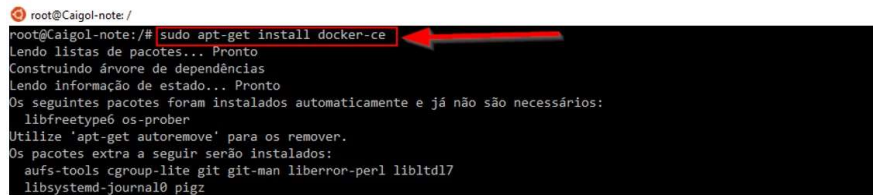
Agora vamos utilizar o comando de instalação do Docker. Lembrando que, para que esse comando funcione, devemos seguir as etapas de apontar o repositório em que o Docker está disponível e adicionar esse repositório em nossa lista. No

comando utilizamos o `sudo` para usar a permissão `admin` do sistema e o `apt-get` `install` para a fazer a instalação, por último o nome do programa que vamos instalar, no caso, o Docker (`docker-ce`).

```
sudo apt-get install docker-ce
```

Com todas as configurações feitas para atualizar os pacotes necessários e adicionar o repositório que contém o Docker, agora é possível fazer a instalação. Veja na Figura 4.18 o comando sendo aplicado no terminal e também o início da sua execução.

Figura 4.18 | Saída do comando que faz a instalação do Docker



```
root@Caigol-note: /  
root@Caigol-note: /# sudo apt-get install docker-ce  
Lendo listas de pacotes... Pronto  
Construindo árvore de dependências  
Lendo informação de estado... Pronto  
Os seguintes pacotes foram instalados automaticamente e já não são necessários:  
  libfreetype6 os-prober  
Utilize 'apt-get autoremove' para os remover.  
Os pacotes extra a seguir serão instalados:  
  aufs-tools cgroup-lite git git-man liberror-perl libltdl7  
  libsystemd-journal0 pigz
```

Fonte: captura de tela elaborada pelo autor.

## INICIANDO E TESTANDO O DOCKER

Para ver se o Docker foi instalado corretamente, devemos iniciar o serviço do Docker e verificar se ele está em execução. Podemos fazer isso com os seguintes comandos:

```
sudo service docker start
```

```
service docker status
```

O comando `sudo` utiliza a permissão de usuário administrador para execução do restante da linha, e as instruções `service docker start` iniciam o serviço Docker que foi instalado anteriormente.

Após a inicialização do Docker, utilizamos o comando `service docker status` que mostra o status do serviço, ou seja, se ele está em execução ou parado.

Caso ocorra algum erro ao iniciar e testar o Docker, você deve executar os dois comandos seguintes e tentar novamente:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Esses comandos utilizam a permissão de usuário administrador através do `sudo` e, além disso, executam uma atualização de pacotes essenciais para o bom funcionamento do sistema operacional, assim como o bom funcionamento dos serviços que rodam através dele, como o Docker.

Agora que instalamos e verificamos seu funcionamento, o sistema está apto a receber as especificidades que queremos criar. Para isso, é possível usar o Docker Swarm. Essa ferramenta é nativa e permite a criação de clusters de Docker. Nesse cenário, é possível agrupar vários hosts em um mesmo pool de recursos, o que facilita o *deploy* de contêineres (DIEDRICH, 2018). Esse framework é integrado ao Docker Engine a partir da versão 1.12.0, com o chamado “swarm mode”.

**DICA**

- Você pode consultar o site, a documentação e todos os repositórios oficiais do Docker no Portal Docker, (DOCKER, [s.d.]).
- Uma alternativa à instalação do Docker é o portal *Play With Docker*, plataforma pela qual é possível testar o Docker utilizando seus comandos diretamente via navegador, mediante um cadastro (PLAY WITH DOCKER, [s.d.]).

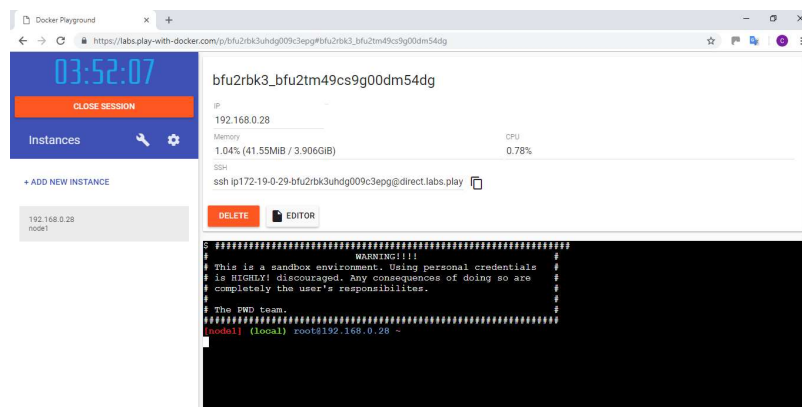
Ver anotações

**EXEMPLIFICANDO**

Após a criação do login na plataforma, teremos acesso a uma interface para criação de instâncias como *clusters* e nós.

Através do botão *Add new instance*, podemos criar nós para o nosso cluster e, então, é possível entrar com comandos na plataforma. Vamos conhecer um dos principais comandos para depois interagirmos com a plataforma. A Figura 4.19 exibe a interface após a criação de um nó:

Figura 4.19 | Plataforma *Play With Docker*



Fonte: captura de tela elaborada pelo autor.

Através dos comandos listados abaixo podemos simular algumas características de sistemas distribuídos com Docker, como criação de nós dos tipos mestre (*manager*) e escravo (*worker*) dentro do nosso *cluster*, gerenciamento de contêiner, réplica de serviços etc.

**COMANDO EXECUTADO FORA DO(S) NÓ(S)**

Agora vamos ver alguns comandos que são executados fora dos nós do ambiente Docker. Um dos mais importantes é o de criação de um novo manager para o cluster. Através do comando, criamos um manager chamado de mestre:

```
docker-machine create --driver virtualbox mestext
```

O começo desse comando significa que estamos criando uma nova máquina através do Docker. Utilizaremos `docker-machine create` como driver no VirtualBox e daremos o nome de mestre para nossa máquina.

Agora, vamos fazer a criação de uma nova máquina: o comando segue o mesmo padrão do anterior, mas desta vez vamos chamar a máquina de “escravo1”. Ela será um *worker* para o cluster, conforme comando abaixo:

```
docker-machine create --driver virtualbox escravo1
```

Podemos verificar o IP de qualquer uma das máquinas de nosso cluster utilizando o comando `docker-machine ip` e, a seguir, consultamos o IP de nossa máquina chamada “mestre”:

```
docker-machine ip mestre
```

Um dos comandos Linux mais populares utilizados para acesso de máquinas/servidores é o `ssh`, que significa “Secure Shell”, em que fazemos um acesso ou uma conexão através desse protocolo de rede criptográfica, que aplica mais segurança aos serviços de redes. Na utilização do Docker o mesmo comando é usado com o objetivo de acessar os nós criados que, em nosso ambiente, possuem os nomes “mestre” e “escravo1”.

```
docker-machine ssh mestre
```

```
docker-machine ssh escravo1
```

É possível verificar os nós (nodes) criados através do comando “`ls`”. Podemos criar um escravo para o cluster com hardware diferente do padrão. Também pode-se ver no comando a seguir que a máquina chamada “escravo5” que definimos a memória com o valor de 512 MB através da instrução `virtualbox-memory “512”` e um disco rígido de 5 GB através da instrução `virtualbox-disk-size “5000”`:

```
docker-machine create --virtualbox-memory "512" --virtualbox-disk-size "5000" --driver virtualbox escravo5
```

## COMANDO EXECUTADO DENTRO DO(S) NÓ(S) MANAGER

Agora vamos ver os alguns comandos executados dentro do manager de nosso ambiente Docker. O comando a seguir pode ser utilizado para iniciar o cluster através do framework *Swarm*, de gerenciamento de contêineres.

```
docker swarm init --advertise-addr
```

Podemos consultar os nós que fazem parte do cluster utilizando o comando `node ls`:

Às vezes precisamos verificar informações sobre um nó específico. Para isso, utilizamos o comando Docker `inspect`. O exemplo a seguir contém informações sobre o nó chamado “escravo1”:

```
docker inspect escravo1
```

A seguir, é possível observar um exemplo de criação de um novo serviço (tarefa) para o cluster. Nesse caso, criamos um serviço Web através do popular servidor web `nginx`, que contém 3 réplicas:

```
docker service create --name WEB --publish 85:80 --replicas=3 nginx:1.12.1
```

No parâmetro `name` definimos um nome para a execução do serviço, em nosso caso, `WEB`. Através do parâmetro `publish` definimos uma porta de execução para o Docker e uma para o servidor web `Nginx`, no caso, "85" e "80" e, por último, definimos que o serviço terá 3 réplicas e utilizará o `Nginx` na versão 1.12.1.

Agora que criamos nosso serviço de internet chamado `WEB`, podemos utilizar o comando `ps` seguido do nome do serviço para verificar suas informações. Portanto, o comando a seguir mostra informações sobre um serviço específico aqui denominado como `WEB`:

```
docker service ps WEB
```

É possível usar os comandos a seguir para alterar a versão das instâncias do `Nginx`. No primeiro comando atualizamos, através do comando `update`, a versão do `Nginx` para 1.13.5 (anteriormente estávamos com a versão 1.12.1). Após isso, utilizamos o comando `ps` com o parâmetro `-f` que significa *filter* ou *find*, que pode ser considerado um filtro para a busca, em que usamos a palavra-chave `Running` para descobrir qual serviço está em execução e se sua versão foi mesmo atualizada. O resultado dos comandos está ilustrado na Figura 4.20.

```
docker service update --image nginx:1.13.5 WEB
```

```
docker service ps -f "desired-state=Running" WEB
```

Figura 4.20 | Plataforma *Play With Docker*

```
[manager1] (local) root@192.168.0.55 ~
$ docker service update --image nginx:1.13.5 WEB
WEB
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
[manager1] (local) root@192.168.0.55 ~
$ docker service ps -f "desired-state=Running" WEB
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURR
c9zcl5td90at	WEB.1	nginx:1.13.5	manager1	Running	Runn
ing 18 seconds ago					
qavj9bm3n70r	WEB.2	nginx:1.13.5	manager3	Running	Runn
ing 25 seconds ago					
k3tmjqzwyvsg	WEB.3	nginx:1.13.5	worker2	Running	Runn
ing 11 seconds ago					

Fonte: captura de tela elaborada pelo autor.

Caso quiséssemos desfazer nossa atualização de versão das instâncias do `Nginx`, podemos utilizar o comando `update` como parâmetro `rollback` aplicado sobre o serviço `WEB`. Com isso, teremos a volta da versão das instâncias do `Nginx` para a versão anterior. Que pode ser consultado através do mesmo comando `ps` utilizado no exemplo anterior, conforme a seguir:

```
docker service update --rollback WEB
```

```
docker service ps -f "desired-state=Running" WEB
```

Caso seja necessário parar algum nó e seus serviços, podemos utilizar o comando `update` seguido do parâmetro `availability drain` e o nome do nó que desejamos parar, como em nosso exemplo, o nó denominado como `escravo2`:

```
docker node update --availability drain escravo2
```

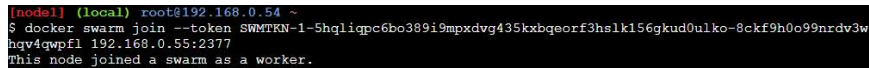
Quando executamos esse comando, tanto o nó quantos os serviços que estão em execução serão parados.

## COMANDO EXECUTADO DENTRO DO(S) NÓ(S) *WORKER*

Agora vamos ver um dos comandos que são executados dentro dos nós escravos (*worker*) do ambiente Docker. O comando a seguir pode ser utilizado para adicionar um *worker* ao nosso cluster (Figura 4.21). Para isso, devemos acessar o nó escravo que queremos adicionar a um cluster e executar o comando a seguir, passando o *token* de segurança e o IP do nó mestre (*manager*) seguido por porta, conforme a sintaxe representada no comando:

```
docker swarm join --token
```

Figura 4.21 | Plataforma *Play With Docker*



```
[node1] (local) root@192.168.0.54 ~  
$ docker swarm join --token SWMTKN-1-5hqliqpc6bo389i9mpxdvg435kxbgeorf3hslk156gkud0ulko-8ckf9h0o99nrdv3w  
hqv4qwpfl 192.168.0.55:2377  
This node joined a swarm as a worker.
```

Fonte: captura de tela elaborada pelo autor.

## COMANDO EXECUTADO DENTRO DE CADA UM DOS NÓS (*MANAGERS* E *WORKERS*)

Agora vamos ver um dos comandos executados dentro dos nós tanto *worker*, quanto *managers* de nosso ambiente Docker. Quando executamos o comando Docker system prune com o parâmetro *a11* dentro de um nó, ele será responsável por apagar/deletar tudo o que foi feito dentro do mesmo. Observamos a seguir sua utilização:

```
docker system prune --a11
```

Agora que já conhecemos alguns comandos, vamos através da plataforma *Play With Docker* orquestrar um servidor web Apache em um cluster simples. Primeiramente, você deve estar logado na plataforma de *playground* do Docker, conforme orientações já apresentadas anteriormente. Uma vez que obteve acesso à plataforma, você deverá:

1. Criar um cluster com 3 nós, que serão suficientes para analisarmos nosso cluster sem comprometer a usabilidade da plataforma de testes do Docker. Sendo assim, você deve adicionar 3 nós, que farão parte do cluster, através do botão *Add new instance*.
2. No nó que você deseja que seja o mestre, digite o seguinte comando:

```
docker swarm init --advertise-addr
```

Este comando define o nó como manager do cluster. Repare que, ao executá-lo, é apresentada uma saída com a mensagem: “Para adicionar um *worker* ao *swarm*, execute o seguinte comando”, conforme pode ser visto (em inglês) na Figura 4.22. Sendo assim, é necessário copiar a saída apresentada a você (que vai ser diferente da destacada em amarelo na Figura 4.22), pois esse comando deverá ser executado em cada um dos demais nós do cluster, adicionando-os como *workers* desse cluster.



Figura 4.22 | Saída do comando docker swarm init

```
[node1] (local) root@192.168.0.18 ~
$ docker swarm init --advertise-addr 192.168.0.18
Swarm initialized: current node (1v3ezy8z8ydd1jd1j8kcdjk8v) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1eu6jq2a7j5w76ulzfevphb207kuc13nymzkgm317n4a3ohvqc-eexpp319erwvr
    vjsdvk6tq52 192.168.0.18:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Fonte: captura de tela elaborada pelo autor.

3. Agora que os nós estão criados e seus papéis definidos, para criar o serviço que estará rodando (de maneira distribuída, replicada) do servidor web Apache, digite o seguinte comando no nó mestre:

```
docker service create --name WEB --publish 80:80 --replicas=5 httpd
```

Esse comando cria 5 instâncias de um servidor web Apache, que responderá na porta mapeada (80, nesse caso) e, para facilitar sua monitoração, demos um nome “amigável” a este serviço: WEB.

4. Para saber em quais nós as 5 réplicas desse serviço estão sendo executadas, digite o seguinte comando:

```
docker service ps WEB
```

No caso da Figura 4.23 abaixo, podemos ver que 2 instâncias estão rodando no nó 1, e as outras 3 estão rodando no nó 2.

Figura 4.23 | Porta mapeada aparece como um hyperlink

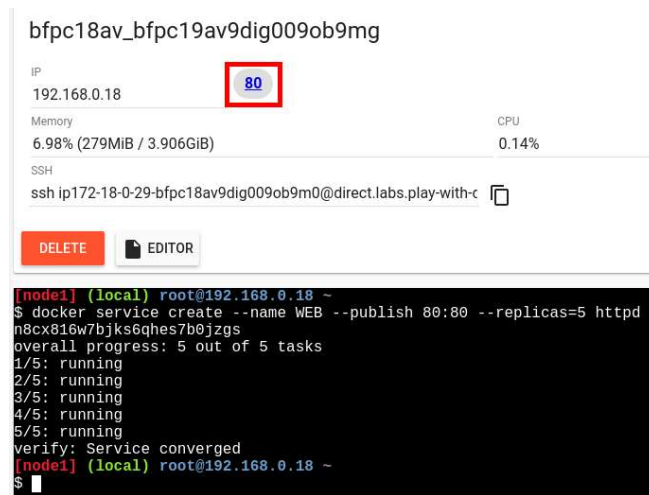
```
[node1] (local) root@192.168.0.18 ~
$ docker service ps WEB
```

ID	PORTS	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
hy4vt3bspf9t	80:80	WEB.1	httpd:latest	node1	Running	Running 7 minutes ago	
34f60ulp06x5	80:80	WEB.2	httpd:latest	node2	Running	Running 7 minutes ago	
umf1me3zqdsx	80:80	WEB.3	httpd:latest	node1	Running	Running 7 minutes ago	
o1xz02bp40nh	80:80	WEB.4	httpd:latest	node2	Running	Running 7 minutes ago	
w49ap6ndeosb	80:80	WEB.5	httpd:latest	node2	Running	Running 7 minutes ago	

Fonte: captura de tela elaborada pelo autor.

5. Por fim, precisamos acessar a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó em que esse serviço web estiver rodando. Reparou que, ao criar o serviço, a porta que você mapeou aparece na parte superior, como um hyperlink? Caso não tenha percebido, veja a porta 80, destacada em vermelho na Figura 4.24. Para acessar a página de boas-vindas, basta clicar nessa porta, em cada um dos nós onde esse serviço está rodando (no caso, nós 1 e 2 do cluster), para vermos a famosa mensagem “It works!” do Apache.

Figura 4.24 | Porta mapeada aparece como um hyperlink



Fonte: captura de tela elaborada pelo autor.

Esta sequência de comandos que utilizamos para orquestrar um servidor web Apache em 3 nós é a configuração utilizada em sistemas distribuídos, para que os acessos a um website sejam balanceados e, caso ocorra algum problema em um dos nós que mantêm a aplicação, o outro nó assume a execução.

**REFLITA**

Parou para pensar em quantas aplicações esses conceitos podem ser aplicados? E quantas melhorias as implantações desses conceitos podem trazer a um sistema web?

Nesta seção você conheceu alguns comandos Docker e aprendeu a fazer uma configuração de cluster e de nós para servidor web.

Bons estudos!

**FAÇA VALER A PENA****Questão 1**

Com o Docker existe uma série de comandos que devemos utilizar para realizar determinadas tarefas dentro do cluster, por exemplo, criação de nós e definição de papéis de nós, como o nó mestre (manager) e o nó escravo (*worker*). Com isso, devemos estar atentos à sintaxe na utilização correta dos comandos.

De acordo com o texto, sabemos que há um comando para criação de nós em um cluster. Assinale a alternativa que corresponde à sintaxe correta deste comando para a criação de um nó chamado “mestre”.

a. `docker-cluster create --driver virtualbox mestre`

b. `docker-machine create --driver virtualbox mestre`

c. `docker-machine create node --driver virtualbox mestre`

d. `docker-machine new --driver virtualbox mestre`

e. `docker cluster create --node mestre`

**Questão 2**

Com o Docker existe uma série de comandos que devemos utilizar para realizar determinadas tarefas dentro do cluster, por exemplo: acessar nós via nome. Com isso, devemos estar atentos à sintaxe na utilização correta dos comandos.

De acordo com o texto sabemos, que há um comando para acesso de nó via nome. Assinale a alternativa que corresponde à sintaxe correta deste comando para acessar um nó chamado “mestre”.

a. `docker ssh mestre`

b. `docker-machine acess mestre`

c. `docker-machine into mestre`

d. `docker-machine ssh mestre`

e. `docker-machine create mestre`

### Questão 3

A instalação do Docker em máquinas com o sistema operacional Ubuntu da distribuição GNU/Linux segue uma lista de procedimentos de acordo com a documentação oficial do Docker. Cada etapa é importante para que a instalação seja encerrada com sucesso.

Assinale a alternativa que traz a ordem correta de procedimentos que devem ser realizados para a instalação do Docker no sistema operacional Ubuntu.

a. Remover as versões anteriores do Docker, atualizar os pacotes e os repositórios, adicionar o repositório de instalação do Docker, fazer a instalação do Docker e verificar se foi instalado corretamente.

b. Adicionar o repositório de instalação do Docker, fazer a instalação do Docker e verificar se foi instalado corretamente.

c. Atualizar os pacotes e os repositórios e fazer a instalação do Docker.

d. Remover as versões anteriores do Docker, atualizar os pacotes e os repositórios, fazer a instalação do Docker e verificar se foi instalado corretamente.

e. Atualizar os pacotes e o repositório, pois o Docker já é nativo no sistema operacional Ubuntu.

### REFERÊNCIAS

DIEDRICH, C. **Docker Swarm**. Disponível em: <https://bit.ly/3ju0xIU>. Acesso em: 5 dez. 2018.

DOCKER (Portal) **Documentação**. Disponível em <https://dockr.ly/3cVGWQy>. Acesso em: 23 nov. 2018.

PLAY WITH DOCKER. **A simple, interactive and fun playground to learn Docker**. [s.d.] Disponível em: <https://bit.ly/2OdpX1L>. Acesso em: 8 fev. 2019.