

Curso de Docker na Prática

Alexandre Batista Vieira

Sumário

O que é Docker?	4
De onde vem o Docker?	4
Qual é a diferença entre o uso de máquinas físicas, virtuais e Containers?	5
O que são imagens do Docker?	6
Arquitetura	6
Configuração de Redes	7
Instalação do Docker no Ubuntu	8
Pré-requisitos	8
Principais comandos do Docker	9
Docker pull	9

Docker images ou Docker image ls	10
Docker run ou Docker container create	11
Exercício	11
Docker ps	11
Docker attach ou Docker exec	12
Docker commit	12
Isolamento e Mapeamento	13
Mapeamento de porta	13
Exercício	13
Mapeamento de volume	13
Exercício	13
Cópia de dados entre container e host ou vice-versa	14
Comunicação entre containers	14
Exercício	15
Exercício	15
Docker stats	16
Netdata	16
Exercício NetData	16
Dockerfile	17
Exercício Expose	18
Exercício MAINTAINER, LABEL, ARG e ENV	18
Exercício Dockerfile	19
Docker Compose	19
Cluster de serviços	20
Docker Swarm vs. Docker Stacks	21
Instalação do Cluster	21
Docker Stacks	22
Alta Disponibilidade	23
Escalar o serviço	23
Exercício - 1 serviço com 2 réplicas	24
Sticky Session	24
Exercício com Sticky Session	25
Exercício Traefik	26

O que é Docker?

Segundo documentação oficial:

“Containers Docker empacotam componentes de software em um sistema de arquivos completo, que contém tudo necessário para a execução: código, runtime, ferramentas de sistemas – qualquer coisa que possa ser instalada em um servidor”

Segundo Juracy Filho:

“é uma ferramenta que se apoia em recursos existentes no Kernel, inicialmente Linux, para isolar a execução de processos. As ferramentas que o Docker traz são basicamente uma camada de administração de containers, baseado originalmente no LXC.”

De onde vem o Docker?

A história dos containers pode ser resumida como apresentado abaixo:

- 1979: Chroot
- 2000: FreeBSD Jails
- 2001: Container Solaris
- 2005: Open VZ
- 2006: Cgroups
- 2008: LXC – Linux Container
- 2011: Warden

Em 2013 a empresa dotCloud que já usava massivamente container decidiu lançar suas ferramentas de gerenciamento de container e passou a se chamar Docker. E assim grandes empresas, como observado na figura abaixo, começaram a usar a tecnologia dando grande visibilidade a essa tecnologia.

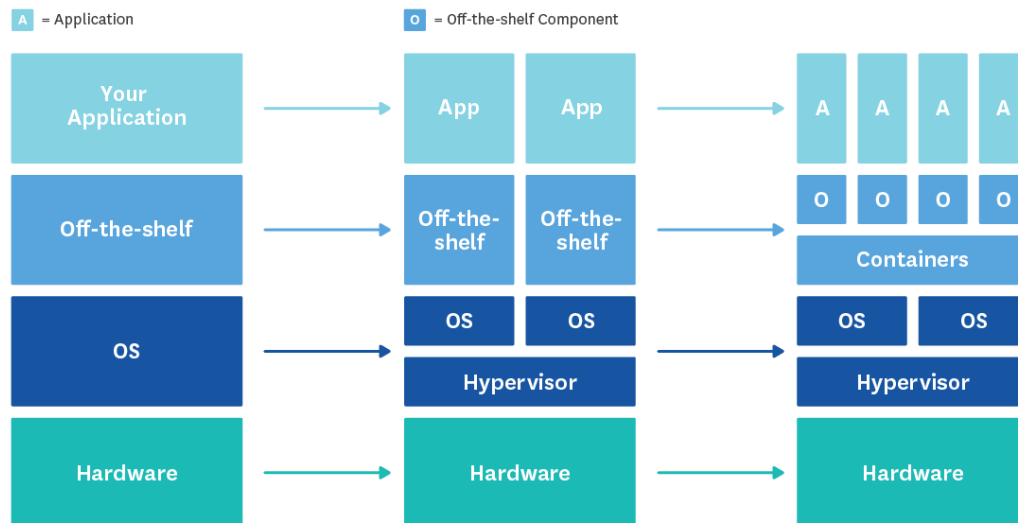
Quem usa



Empresas que usam Docker para desenvolver software

Qual é a diferença entre o uso de máquinas físicas, virtuais e Containers?

A principal diferença de usar container Docker é que este tende a utilizar menos recursos que uma máquina virtual, ou seja, ele reutiliza o kernel do host e roda o mínimo de processos necessários. Para melhor entendimento veja a imagem que ilustram os componentes de cada camada tecnológicas.



O que são imagens do Docker?

Todo container Docker deve ser instanciado a partir de uma imagem. As imagens são fábricas de containers. Através de uma imagem podemos instanciar vários containers baseado nessa imagem.

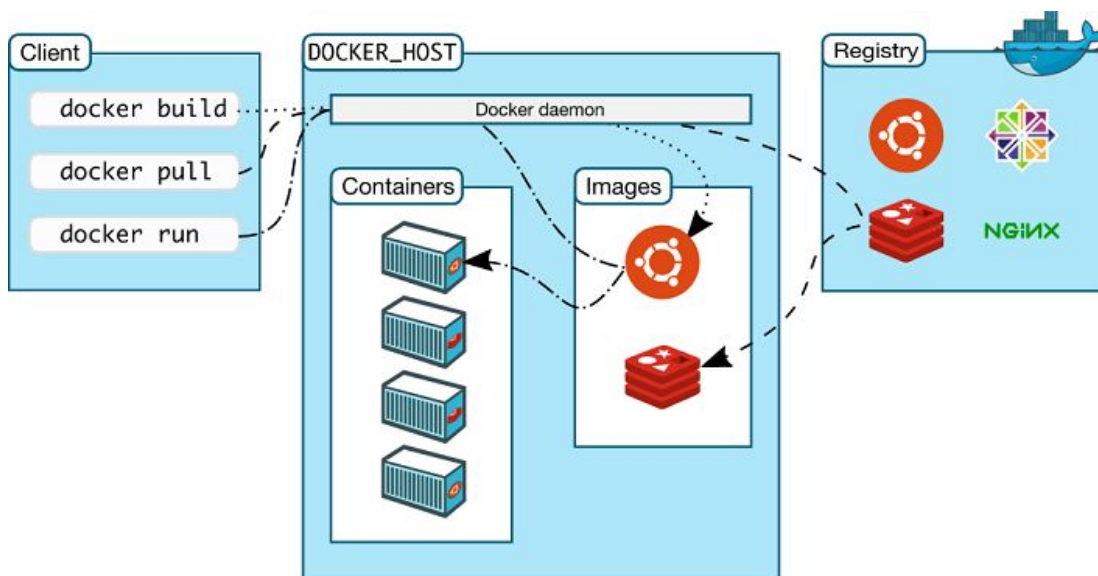
Uma das iniciativas que facilita a adoção do Docker é o Docker Hub ou Docker Store que é um repositório público de imagens do Docker. A maioria das distros do Linux podem ser encontradas lá e foram publicadas oficialmente pela empresa Docker Inc.

Além desse repositório público podemos instanciar repositórios privados para armazenar imagens criadas para nossa organização.

Arquitetura

Sendo simplista temos 3 principais componentes:

1. Ter o serviço do Docker instalado e rodando (service docker status)
2. Ter um repositório de imagens público ou privado para buscarmos a imagem
3. Acessar através Docker Client ou através do Docker API Rest para gerenciar os containers.



Configuração de Redes

Ao instalar e inicializar o Docker, a interface de rede virtual docker0 é criada. Será escolhido para esta interface um IP e uma máscara de subrede. Você pode conferir isto usando o comando ifconfig.

```
alexandre@alexandre-L40-30:/opt/cursoDocker$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:1dff:fe40:200d prefixlen 64 scopeid 0x20<link>
    ether 02:42:1d:40:20:0d txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 103 bytes 14996 (14.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Se caso você tiver problemas com o endereço escolhido é possível mudar a configuração através do arquivo /etc/docker/daemon.json na propriedade bip conforme exibido na imagem abaixo.

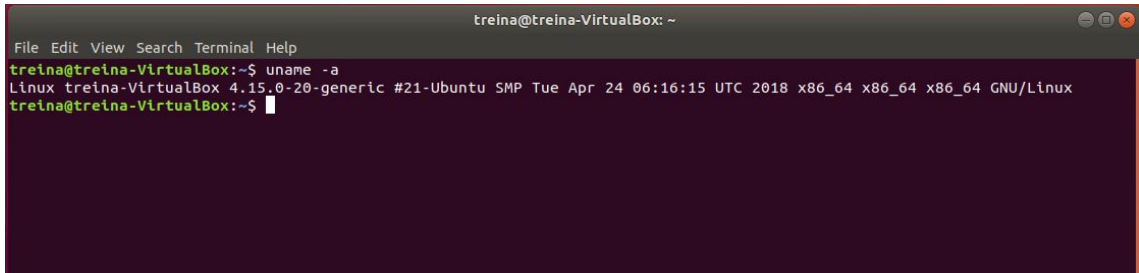
```
{
  "bip": "192.168.1.5/24",
  "fixed-cidr": "192.168.1.5/25",
  "fixed-cidr-v6": "2001:db8::/64",
  "mtu": 1500,
  "default-gateway": "10.20.1.1",
  "default-gateway-v6": "2001:db8:abcd::89",
  "dns": ["10.20.1.2", "10.20.1.3"]
}
```

Instalação do Docker no Ubuntu

Pré-requisitos

O kernel do Linux deverá ser a partir da versão 3.10. Para verificar a versão do seu Kernel digite na linha de comando:

```
uname -a
```

A screenshot of a terminal window titled 'treina@treina-VirtualBox: ~'. The terminal shows the command 'uname -a' being executed, resulting in the output: 'Linux treina-VirtualBox 4.15.0-20-generic #21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux'. The terminal has a dark background with green text.

1º Passo – Atualize os índices de pacotes do linux

```
apt-get update
```

2º Passo – Instale os pacotes abaixo

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

3º Passo – Adicione a chave GPG oficial do Docker

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

4º Passo – Adicione no repositório do apt o docker

```
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian  
$(lsb_release -cs) stable"
```

5º Passo - Atualize os índices de pacotes do Linux, novamente

```
apt-get update
```

6º Passo – Instale o Docker versão Comunidade (CE)

```
apt-get install docker-ce docker-ce-cli containerd.io
```

7º Passo – Teste a instalação

```
docker run hello-world
```

```
Activities Terminal
qui 16:23
treina@treina-VirtualBox: ~

File Edit View Search Terminal Help

Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up docker-ce-cli (5:18.09.0~3-0-ubuntu-bionic) ...
Setting up pigz (2.4-1) ...
Setting up git (1:2.17.1-1ubuntu0.3) ...
Setting up docker-ce (5:18.09.0~3-0-ubuntu-bionic) ...
update-alternatives: using /usr/bin/dockerd-ce to provide /usr/bin/dockerd (dockerd) in auto mode
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Processing triggers for ureadahead (0.100.0-20) ...
treina@treina-VirtualBox:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cab9c9fde470971e499788
Status: Downloaded newer image for hello-world:latest

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/

treina@treina-VirtualBox:~$
treina@treina-VirtualBox:~$
```

Principais comandos do Docker

Docker pull

Este comando é usado para baixar uma imagem do repositório docker público (hub.docker.com) ou privado (docker registry).

Vamos baixar a imagem do Ubuntu 18.10 com o comando abaixo:

```
docker pull ubuntu:18.10
```

```
Activities Terminal
treina@t

File Edit View Search Terminal Help

treina@treina-VirtualBox:~$ sudo docker pull ubuntu:18.10
18.10: Pulling from library/ubuntu
5940862bcfcd: Pull complete
a496d03c4a24: Pull complete
5d5e0ccd5d0c: Pull complete
ba24b170ddf1: Pull complete
Digest: sha256:20b5d52b03712e2ba8819eb53be07612c67bb87560f121cc195af27208da10e0
Status: Downloaded newer image for ubuntu:18.10
treina@treina-VirtualBox:~$
```



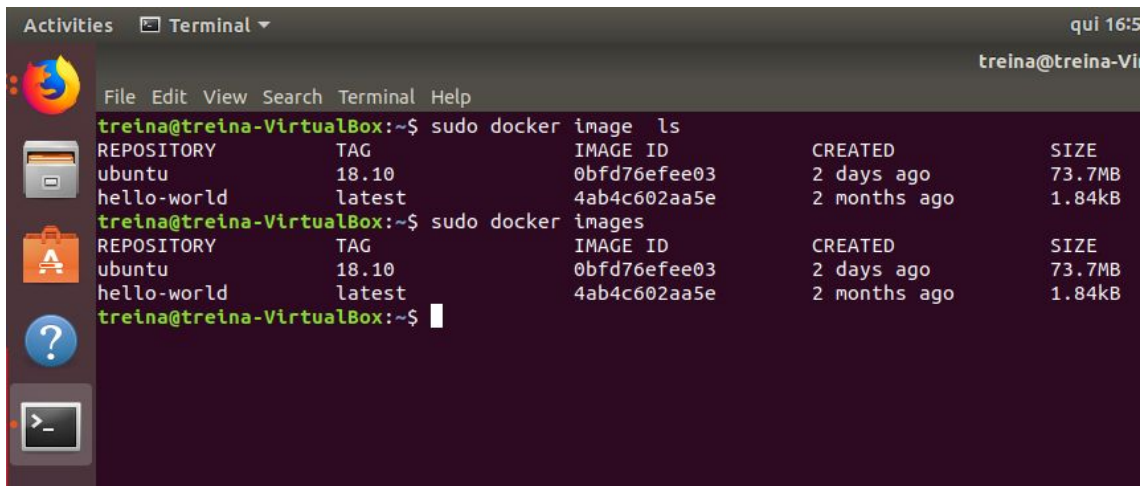
O docker File Systems armazena imagens e containers como camadas para reutilizá-las e assim otimizar o tamanho em disco.

Docker images ou Docker image ls

Agora podemos listar as imagens que obtemos no item anterior.


```
docker image ls
```

```
docker images
```



```
Activities Terminal
File Edit View Search Terminal Help
treina@treina-VirtualBox:~$ sudo docker image ls
REPOSITORY      TAG              IMAGE ID         CREATED          SIZE
ubuntu          18.10           0bfd76efee03    2 days ago      73.7MB
hello-world     latest         4ab4c602aa5e    2 months ago    1.84kB
treina@treina-VirtualBox:~$ sudo docker images
REPOSITORY      TAG              IMAGE ID         CREATED          SIZE
ubuntu          18.10           0bfd76efee03    2 days ago      73.7MB
hello-world     latest         4ab4c602aa5e    2 months ago    1.84kB
treina@treina-VirtualBox:~$
```



Perceba que hello-world:latest foi baixado quando testamos a instalação do docker.

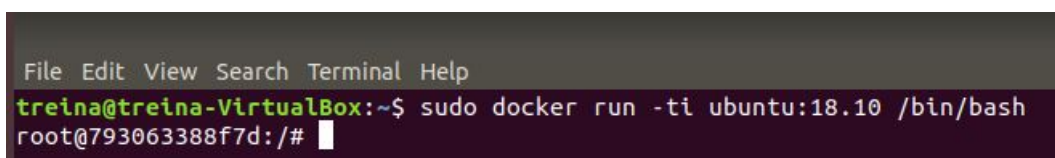
As colunas do retorno do comando são:

- Repository: o nome da imagem
- Tag: a versão da imagem. Latest é uma palavra reservada que aponta para a última versão.
- Image ID: id gerado randomicamente da imagem que serve para sabermos se estamos apontando para mesma imagem embora possam estar com Repository:Tag diferentes.
- Created: data de criação da imagem.
- Size: tamanho da imagem. Muito importante!

Docker run ou Docker container create

Iremos criar um container a partir da imagem ubuntu:18.10

```
docker run -ti ubuntu:18.10 /bin/bash
```



```
File Edit View Search Terminal Help
treina@treina-VirtualBox:~$ sudo docker run -ti ubuntu:18.10 /bin/bash
root@793063388f7d:/#
```



Perceba que o prompt mudou porque além da criação do container ele já entrou dentro do container. Para sair e parar o container use Ctrl+D ou o comando exit e, para sair sem parar o container use Ctrl+P+Q.

Pedimos para criar o container com as seguintes especificações:

Parâmetros	Significado
------------	-------------

-ti	significa tty (prompt do Linux) e interativo (queremos interagir que é o oposto do processo em background)
ubuntu:18.10	Nome da imagem e número da versão do ubuntu.
/bin/bash	Único processo que nós queremos que esteja rodando ao executar o container

Exercício

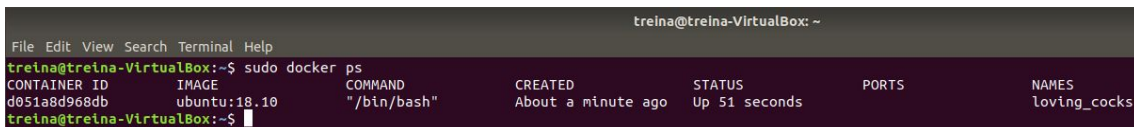
Verifique que nem sempre as versões disponíveis no sistema operacional é a mesma disponível no sistema operacional hospedado.

```
bash --version
docker run -ti debian bash --version
```

Docker ps

```
docker ps
```

O comando lista os containers ativos.



```
treina@treina-VirtualBox: ~
File Edit View Search Terminal Help
treina@treina-VirtualBox:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS               NAMES
d051a8d968db   ubuntu:18.10  "/bin/bash"             About a minute ago    Up 51 seconds                loving_cocks
treina@treina-VirtualBox:~$
```

As colunas do retorno do comando são:

- CONTAINER ID: o hash usado para identificar esse container em outros comandos
- IMAGE: imagem e versão separados por “:”
- COMANDO: o processo que o container está em execução.
- CREATED: Quanto tempo seu container foi criado.
- STATUS: se seu container está funcionando ou está parado.
- PORTS: mapeamento de porta entre o host e o container
- NAMES: nome dado ao container usado por outros comandos no lugar do CONTAINER ID. Nome é atribuído aleatoriamente caso não haja o parâmetro - - name especificando-o.

Docker attach ou Docker exec

Comando utilizado para entrar no container em execução.

```
docker attach <container id> ou <nome da container>
```

```
docker exec -it <container id> ou <nome da container> /bin/bash
```

```
File Edit View Search Terminal Help
treina@treina-VirtualBox:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
d051a8d968db       ubuntu:18.10       "/bin/bash"        2 hours ago        Up 2 hours         22/tcp             loving_cocks
treina@treina-VirtualBox:~$ sudo docker attach d051a8d968db
root@d051a8d968db:/# read escape sequence
treina@treina-VirtualBox:~$ sudo docker exec -it d051a8d968db /bin/bash
root@d051a8d968db:/#
```

Docker commit

```
docker commit <container id> <nome da imagem>:<versão da imagem>
```

Comando gera uma imagem a partir de um container que está em execução. Então, tudo que fizemos dentro do container será gravado na imagem. Posteriormente, podemos usar o comando `docker run` para criar vários containers dessa imagem. Na demonstração abaixo commitamos o container `hello-world` como uma imagem chamada `nexus-imagens.pbh.gov.br/prodabel/hello-world` que está armazenada somente na máquina local.

```
File Edit View Search Terminal Help
treina@treina-VirtualBox:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
d051a8d968db       ubuntu:18.10       "/bin/bash"        About an hour ago   Up About an hour   22/tcp             loving_cocks
treina@treina-VirtualBox:~$ sudo docker commit d051a8d968db nexus-imagens.pbh.gov.br/prodabel/ubuntu:18.10
sha256:34e575dbaf57dc19f99ebf3f9c723fe2e09c8bb7cc0c8d4ff19c70716092214c
treina@treina-VirtualBox:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nexus-imagens.pbh.gov.br/prodabel/ubuntu   18.10              34e575dbaf57       20 seconds ago    73.7MB
ubuntu              18.10              0bfd76efee03       3 days ago        73.7MB
hello-world         latest             4ab4c602aa5e       2 months ago      1.84kB
treina@treina-VirtualBox:~$
```



Perceba que está embutido no nome da imagem a URL do repositório privado que desejamos utilizar.

Isolamento e Mapeamento

Através do docker engine o container nasce com o máximo de isolamento possível em relação ao host. E por isso, temos que fazer os mapeamentos necessários.

Mapeamento de porta

No comando abaixo o parâmetro `-p` é feito para fazer o mapeamento de porta.

```
docker run -td -p 8081:8080 --name tomcat8081 tomcat:7-jre8-slim /bin/bash
```



Considere que em todos os parâmetros que houver `“.”`, o lado esquerdo é relacionado ao host enquanto o lado direito é do container.

Exercício

Crie 5 containers da imagem do tomcat. Faça o mapeamento da porta 8080 para as portas 8081, 8082, 8083, 8084, 8085. Abra o browser e verifique se as URLs estão acessíveis. No host liste os processos que estão rodando e perceba que há 5 processos rodando tomcats sem nenhuma indicação que estão rodando dentro de containers.

-td

Mapeamento de volume

O volume é um diretório ou pasta que você quer que seja criado uma espécie de link entre o host e o container.

```
docker run -v /projeto/war:/usr/lib/tomcat/webapps ubuntu:18.10 /bin/bash
```



Considere que em todos os parâmetros que houver “:”, o lado esquerdo é relacionado ao host enquanto o lado direito é do container.

Exercício

Crie um container do postgres e mapeie o volume para o host. Crie outro container com o cliente de banco de dados pgadmin4. Manipule o banco. Como há o mapeamento, demonstre que ao excluir e criar um outro container postgres mapeado para o mesmo diretório, o banco não sofrerá qualquer problema com a substituição dos containers.

Cópia de dados entre container e host ou vice-versa

Para copiar arquivo do host para dentro do container use:

```
docker cp arquivo.war <id do container>:/usr/lib/tomcat/webapps/
```

Para copiar arquivo de dentro do container para a pasta corrente

```
docker cp <id do container>:/usr/lib/tomcat/webapps/arquivo.war .
```

Comunicação entre containers

A comunicação entre containers depende da topologia de rede do host que deve permitir a conexão entre eles e o iptables deve permitir as ligações especiais que são feitas pelo docker. Ao subir 2 containers podemos ver as regras que foram criadas no iptables e verificaremos a comunicação dos containers através da interface de redes padrão bridge0.

```
docker run -d -p 8080:80 nginx
```

```
docker run -d -p 3306:3306 mysql
```

```
alexandre@alexandre-L40-30:/opt/cursuDocker$ sudo ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:1dff:fe40:200d prefixlen 64 scopeid 0x20<link>
    ether 02:42:1d:40:20:0d txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 160 bytes 26253 (26.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
alexandre@alexandre-L40-30:/opt/cursuDocker$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy DROP)
target     prot opt source                destination
DOCKER-USER all -- 0.0.0.0/0              0.0.0.0/0
DOCKER-ISOLATION-STAGE-1 all -- 0.0.0.0/0              0.0.0.0/0
ACCEPT     all -- 0.0.0.0/0              0.0.0.0/0          ctstate RELATED,ESTABLISHED
DOCKER     all -- 0.0.0.0/0              0.0.0.0/0
ACCEPT     all -- 0.0.0.0/0              0.0.0.0/0
ACCEPT     all -- 0.0.0.0/0              0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

Chain DOCKER (1 references)
target     prot opt source                destination
ACCEPT     tcp -- 0.0.0.0/0              172.17.0.2          tcp dpt:8080
ACCEPT     tcp -- 0.0.0.0/0              172.17.0.3          tcp dpt:3306

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
target     prot opt source                destination
DOCKER-ISOLATION-STAGE-2 all -- 0.0.0.0/0              0.0.0.0/0
RETURN     all -- 0.0.0.0/0              0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
target     prot opt source                destination
DROP       all -- 0.0.0.0/0              0.0.0.0/0
RETURN     all -- 0.0.0.0/0              0.0.0.0/0

Chain DOCKER-USER (1 references)
target     prot opt source                destination
RETURN     all -- 0.0.0.0/0              0.0.0.0/0
```

apt-get install bridge-utils

```
alexandre@alexandre-L40-30:/opt/cursuDocker$ sudo brctl show
bridge name      bridge id        STP enabled      interfaces
docker0          8000.02421d40200d no                veth89f3602
                 vethd711fc7
```

Exercício

Sabemos que containers devem ter nomes únicos e não é raro esbarrarmos com esse erro. Então, crie dois containers e acostume com esta mensagem de erro.

```
docker run --name meucontainer -it debian bash
Ctrl + D
docker run --name meucontainer -it debian bash
```

Ao pressionar Ctrl+D o container fica em stop. Para listar o container além dos ativos use “docker ps -a”.

Exercício

Gere um container a partir da imagem ubuntu:18.10. Pela linha de comando do container instale o apache2. Committe a imagem com o nome nexus-imagens.pbh.gov.br/prodabel/ubuntu_apache:18.10. Depois disso, inicie um container desta imagem criada e mapeie a porta 80 do container para a porta 80 do host. Depois acesse o navegador e veja a página inicial do apache2.

Docker stats

Este comando monitora CPU, memória e redes.

```
docker stats <id do container>
```

File Edit View Search Terminal Help

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
9a6bba1e5d5b	ubuntu	0.00%	4.727MiB / 1.946GiB	0.24%	14.4MB / 278kB	46.8MB / 14.8MB	1

Se preferir usar Docker Rest API digite na linha de comando

```
curl --unix-socket /var/run/docker.sock http://docker/containers/<id do container>/stats
```

Se quiser aprofundar mais no uso do Docker Rest API leia o link abaixo:

<https://docs.docker.com/engine/api/v1.37/>

Netdata

É uma ferramenta para monitorar a saúde e performance de aplicações e sistemas em tempo real. Atualmente ele funciona em Linux, FreeBSD e Mac. Coleta métricas de servidores web, banco de dados e containers. Possui sistema de notificação e alarmes. Possui dashboard para exibir as métricas e se houver algum incidente de desempenho ele mata o console automaticamente.

Exercício NetData

Instale o NetData conforme URL <https://github.com/netdata/netdata#quick-start>.

Depois disso, abra seu browser e acesse <http://<ip>:19999> e veja o dashboard. O Netdata para enviar e-mails usa o sendmail. Configure o Sendmail, EXIM4 e o Netdata para enviar e-mails.

Use a URL <http://bernaerts.dyndns.org/linux/75-debian/278-debian-sendmail-gmail-account> para auxiliar a configurar o envio de e-mail no Debian e lembre-se que o servidor smtp da Prodabel é **envia.aplicacao-hmcor.pbh:587**.

O próximo passo é testar o envio de e-mail pelo send mail conforme linha abaixo:
echo "Teste envio de e-mail" | sendmail -v alexandre@pbh.gov.br

<https://docs.netdata.cloud/health/notifications/>

Comando para editar o arquivo de configuração de notificação do netdata
/etc/netdata/edit-config health_alarm_notify.conf
Ao abrir o editor procure e alterar a linha DEFAULT_RECIPIENT_EMAIL
para incluir seu e-mail.
Execute a linha abaixo para testar envio de e-mail pelo NetData
/usr/libexec/netdata/plugins.d/alarm-notify.sh test

Agora, você está apto para passar para o próximo exercício.

Exercício Stress

Faça um teste de stress em sua VM para você receber o e-mail de alerta do Netdata.
Sugestão: use um container para fazer isso.

```
docker run --rm -it progrium/stress --cpu 2 --io 1 --vm 2 --vm-bytes 128M  
--timeout 10s
```

```
docker run --rm -it progrium/stress --cpu 40 --io 1 --vm 10 --vm-bytes  
1024M --timeout 10s
```

Dockerfile

Este arquivo é um script que permite criar uma imagem. É um arquivo de texto com todos os passos que seria executado manualmente para preparar o ambiente de execução. É como se fosse uma receita de bolo onde cada comando é registrado como uma camada de reutilização.

Principais comandos:

FROM: a partir de qual imagem será gerada a nova imagem.

MAINTAINER: Campo opcional de documentação que informa quem é o autor da imagem.

RUN: realiza a execução de comandos como se fosse no terminal do linux.

ENTRYPOINT ou CMD: execução de um script na inicialização de um container.

LABEL: adiciona um metadado no container

EXPOSE: faz a documentação das portas que poderão ser expostas pelo container.

VOLUME: mapeia um diretório do host a ser acessível pelo container

ENV: define a variável de ambiente do linux

USER: ativa o usuário que executará as instruções abaixo desta definição.

WORKDIR: define o diretório base de trabalho. Ao entrar no container já cairá neste diretório.

ONBUILD: em uma herança de imagens, este comando deixa atividades a serem feitas no momento do build.

Exercício Expose

Obtenha a imagem do nginx. Crie um Dockerfile a partir desta imagem. Exponha a porta 80. Builde a imagem com o nome nginx. Crie container com o comando abaixo e observe através do comando “docker ps” em qual a porta foi mapeada para a porta 80.

```
docker run -td -P --name nginx nginx
```

O parâmetro -P faz o mapeamento para uma porta das portas expostas na imagem.

Exercício MAINTAINER, LABEL, ARG e ENV

Crie um Dockerfile a partir da imagem do Debian. Faça documentação do mantenedor colocando seu nome e e-mail. Receba o argumento como valor default “fulano” e coloque o conteúdo deste argumento disponível como variável de ambiente. Logo depois faça a impressão no console “Hello World, fulano!”.

```
FROM debian
MAINTAINER 'Alexandre alexandre@pbh.gov.br'
ENTRYPOINT /entrypoint.sh
ARG NOME=fulano
ENV NOME=$NOME
RUN echo "echo \"Hello World, \$NOME\"" > /entrypoint.sh && \
    chmod a+x /entrypoint.sh
```

Para executar passando o parâmetro, execute o comando abaixo:

```
docker build --build-arg NOME="Seu Nome" -t nexus-imagens.pbh.gov.br/prodabel/hello .
docker run -ti --name hello nexus-imagens.pbh.gov.br/prodabel/h-pello
```

Exercício Dockerfile

Obtenha a imagem wildfly do hub.docker.com. Adicione um usuário e senha administrativo para o Wildfly no modo silencioso. Coloque o usuário root para executar o build. Depois

sobrescreva a execução do wildfly setando o ip tanto do servidor quanto da área de gerenciamento do servidor para receber requisição de qualquer host. Construa a imagem através do comando `build docker build -t nexus-imagens.pbh.gov.br/prodabel/wildfly`. Suba o container mapeando as portas 8080 e 9990.

```
FROM jboss/wildfly
USER root
MAINTAINER alexandre@pbh.gov.br
RUN $JBOSS_HOME/bin/add-user.sh --silent=true admin admin
ENTRYPOINT $JBOSS_HOME/bin/standalone.sh -b 0.0.0.0 -bmanagement 0.0.0.0
```

Docker Compose

Como explicado anteriormente um container é para colocar um serviço contendo um único processo principal. Entretanto, sabemos que as aplicações dependem de vários serviços, como por exemplo:

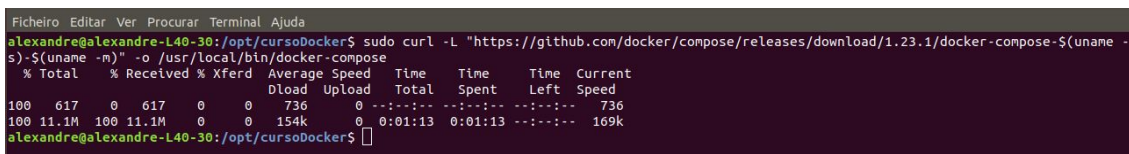
- Banco de dados
- servidor de aplicação para o backend
- servidor web para o frontend

Cada um desses serviços rodam em um container e para gerenciá-los podem possuir sequência de inicialização, dependência de portas entre os containers, entre outras características.

Para resolver este problema pode ser criado um `docker-compose.yml` que irá tratar os microserviços. Para simplificar usaremos no exemplo um cliente e servidor de banco de dados.

Instale o docker-compose com o seguinte comando abaixo:

```
curl -L
"https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
```



```
Ficheiro Editar Ver Procurar Terminal Ajuda
alexandre@alexandre-L40-30:/opt/cursosDocker$ sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -
s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 617 0 617 0 0 736 0 --:--:-- --:--:-- --:--:-- 736
100 11.1M 100 11.1M 0 0 154k 0 0:01:13 0:01:13 --:--:-- 169k
alexandre@alexandre-L40-30:/opt/cursosDocker$
```

Dê permissão de execução ao docker-compose

```
chmod 755 /usr/local/bin/docker-compose
```

`docker-compose.yml`

```
version: '3.1'

services:

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: postgres
    ports:
      - 5432:5432

  admin:
    image: chorss/docker-pgadmin4
    restart: always
    ports:
      - 5050:5050
```

- YAML File: Segundo Wikipédia “é um formato de serialização de dados legíveis por seres humanos inspirado no XML e em outras linguagens.”
- Vejam no link <https://docs.docker.com/compose/compose-file/compose-versioning/> as versões dos docker-compose.yml.

Para inicializar:

```
docker-compose up -d
```

Para listar

```
docker-compose ps
```

Para desligar

```
docker-compose down
```

Cluster de serviços

Além de gerenciar os microsserviços temos que nos preocuparmos com a escalabilidade, alta disponibilidade, balanceamento de carga e tolerância a falhas e por isso iremos avançar para os próximos conceitos abaixo.

Docker Swarm vs. Docker Stacks

Há uma separação desses dois conceitos que enquanto o primeiro está relacionado com manutenção da infraestrutura, o segundo está ligado aos serviços que serão disponibilizados sobre a infraestrutura. Se seguirmos as boas práticas podemos dar manutenção na

infraestrutura sem nos preocuparmos com a publicação de serviços e os serviços poderão ser instalados sem se preocupar com a infraestrutura que está contida no Cluster.

Instalação do Cluster

Inicialize o cluster em seu host.

```
docker swarm init
```

```
alexandre@alexandre-L40-30:/opt/cursosDocker$ sudo docker swarm init
Swarm initialized: current node (yob26nwkasin6z7o3e0xrgvn1) is now a manager.

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

    docker swarm join --token SWMTKN-1-43t8lpxvrgpks3lrxkw7wbas0s3tkqjnv0jazjdbb5ua0skfxe-dl7c9ckeclty0gsbo4gm6totw 192.168.0.141:2377

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

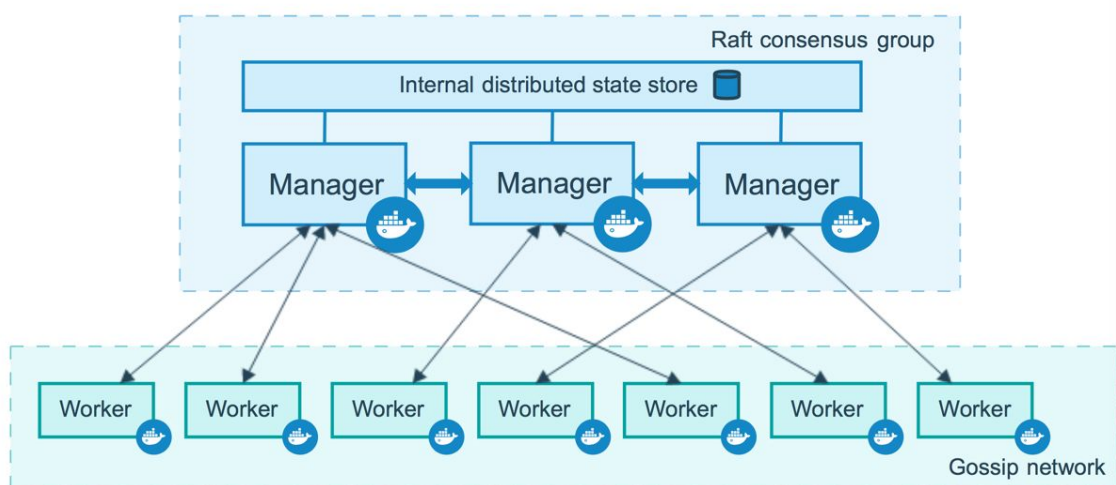
alexandre@alexandre-L40-30:/opt/cursosDocker$ sudo docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-43t8lpxvrgpks3lrxkw7wbas0s3tkqjnv0jazjdbb5ua0skfxe-bmd8ry9bajidtxtvgawbybpvx 192.168.0.141:2377

alexandre@alexandre-L40-30:/opt/cursosDocker$
```

Ao inicializar o cluster é dado a opção para adicionar nós Manager ou nós Worker. Por padrão o Manager faz o papel do Worker e mais algumas coisas como gerenciar o estado do cluster, agendar serviços e servir de endpoint para a API do Swarm, ou seja, você só poderá administrar seu cluster através dos Managers enquanto que os Workers apenas executam os containers dos serviços.

Neste curso, o docker swarm será composto por apenas 1 nó Manager mas caso você quera construir um cluster com vários nós, execute os comandos de join para Manager ou Worker como é exibido na figura acima. Podemos crescer ou diminuir o cluster a qualquer momento.



Em um ambiente de produção, a boa prática indica o algoritmo de Ralf e estabelece números ímpares de nós Manager sendo 3 ou 5 nós.³

³<https://success.docker.com/article/docker-ee-best-practices>

Docker Stacks

Um arquivo stack é um arquivo no formato YAML, similar ao docker-compose.yml, que é usado para definir um ou mais serviços para ser publicado sobre o cluster Docker Swarm.

Veja abaixo o conteúdo do arquivo stack.yml

```
version: "3.1"
services:
  webapp:
    image: nexus-imagens.pbh.gov.br/prodabel/jsecurity6swarm:latest
    ports:
      - 80:80
    deploy:
      restart_policy:
        condition: any
      mode: replicated
      replicas: 3
      update_config:
        delay: 1s
```

Faça o deploy do arquivo stack.yml com o comando abaixo:

```
docker stack deploy -c stack.yml jsecurity
```

E para remover a stack o comando é o seguinte:

```
docker stack rm jsecurity
```

Alta Disponibilidade

Mesmo ao definir 1 réplica na stack anterior, dependendo do cluster (Docker Swarm) que foi montado, vamos usufruir de alta disponibilidade do serviço. A partir de 2 nós, sendo eles Manager, já teríamos alta disponibilidade pois se um nó falhar a transferência do container da stack é transparente.

Escalar o serviço

“O docker usa o “routing mesh”, ou, roteamento de malha. Redes de malha e seus algoritmos de roteamento não são novidade, a equipe de engenharia do Docker tem usado essa abordagem para simplificar a entrega de mudanças de software e descoberta de serviços em arquiteturas de microsserviços . A “malha” é simplesmente a nova maneira de rotear e equilibrar o tráfego para os contêineres no Docker. A nova estratégia de roteamento permite que um serviço seja alcançado pela mesma porta em todos os nós no swarm, mesmo se o nó não tiver o serviço implementado nele. A malha de roteamento também equilibra de forma transparente as solicitações em todos os serviços disponíveis no swarm, detectando falhas nos nós.

Essa nova abordagem torna muito fácil configurar o balanceamento de carga dos serviços: imagine um swarm de três nós com sete serviços diferentes espalhados pelo swarm. Do lado de fora, podemos enviar o pedido para qualquer um dos nós, e ele será roteado para um serviço aleatório automaticamente. Ou podemos sempre enviar a solicitação para um único nó e o Docker carregará internamente o equilíbrio entre os serviços. Portanto, obtemos balanceamento de carga de serviços nativamente.

A malha de roteamento nos permite tratar os contêineres de forma verdadeiramente transparente, no sentido de que não nos importamos com quantos contêineres estão atendendo ao nosso aplicativo, o cluster lida com todas as redes e balanceamento de carga para nós. Onde antes tínhamos que colocar um proxy reverso na frente dos serviços para agir como um balanceador de carga, agora podemos relaxar e deixar o docker fazer o balanceamento de carga para nós. Como o Docker lida com todo o trabalho sujo para nós, a diferença entre ter um contêiner e ter 100 contêineres agora é trivialmente pequena; é apenas uma questão de ter recursos de computação suficientes para escalar e executar um comando para aumentar a escala. Não precisamos mais pensar em considerações de arquitetura antes de dimensionar, já que o Docker cuida de tudo. Quando se trata de dimensionamento, podemos relaxar sabendo que o Docker vai lidar com isso de forma transparente, um contêiner é o mesmo que 100.” ⁴

⁴<https://medium.com/devopsion/solving-the-routing-mess-for-services-in-docker-73492c37b335>

Para serviços stateless é ótimo este tipo de roteamento. Para aplicativos monolíticos que lidam, por exemplo, com sessão de usuário teremos que tratar de forma diferente.

Exercício - 1 serviço com 2 réplicas

Modifique a stack.yml para 2 réplicas e faça o deploy. Depois logue na aplicação e abra o log de cada uma das réplicas e verifique se as requisições sempre fica no mesmo container. Se não, qual a sua sugestão para corrigir este problema?

Sticky Session

<https://www.incapsula.com/load-balancing/sticky-session-persistence-and-cookies.html>

“A sessão pegajosa, também chamada persistência de sessão, é um processo no qual um balanceador de carga cria uma afinidade entre um cliente e um servidor de rede específico durante a duração de uma sessão (ou seja, o tempo que um IP específico gasta em um site). O uso de sessões persistentes pode ajudar a melhorar a experiência do usuário e otimizar o uso de recursos da rede.

Com sessões fixas, um balanceador de carga atribui um atributo de identificação a um usuário, geralmente emitindo um cookie ou rastreando seus detalhes de IP. Em seguida, de acordo com o ID de rastreamento, um balanceador de carga pode iniciar o roteamento de todas as solicitações desse usuário para um servidor específico durante a duração da sessão.

Isso pode ser muito útil, pois o HTTP / S é um protocolo sem estado que não foi desenvolvido com a persistência da sessão em mente. No entanto, muitos aplicativos da Web têm a necessidade de fornecer dados de usuário personalizados (por exemplo, manter registros de itens em um carrinho de compras ou conversas de bate-papo) ao longo de uma sessão.

Sem a persistência da sessão, o aplicativo da Web precisaria manter essas informações em vários servidores, o que pode ser ineficiente, especialmente em redes grandes.

ADESÃO À SESSÃO: VANTAGENS E DESVANTAGENS

A fixação da sessão oferece vários benefícios que podem melhorar o desempenho do seu aplicativo da Web, incluindo:

- *Troca de dados minimizada - Ao usar sessões persistentes, os servidores em sua rede não precisam trocar dados da sessão, um processo caro quando feito em escala.*
- *Utilização do cache de RAM - Sessões aderentes permitem uma utilização mais eficiente do cache de RAM do aplicativo, resultando em melhor capacidade de resposta.*

Dito isto, as sessões complicadas também tornam mais difícil manter os servidores em equilíbrio. Um servidor pode ficar sobrecarregado se acumular muitas sessões ou se sessões específicas exigirem um alto número de recursos. Isso pode fazer com que seu balanceador de carga tenha que transferir um cliente para outro servidor no meio da sessão, resultando em perda de dados.

PERSISTÊNCIA USANDO COOKIES DE SESSÃO

Existem dois tipos de persistência de sessão baseada em cookie: baseada em duração e controlada por aplicativo.

- *Persistência de sessão baseada em duração*

Seu balanceador de carga emite um cookie que define um prazo específico para a viscosidade da sessão. Cada vez que o balanceador de carga recebe uma solicitação do cliente, ele verifica se esse cookie está presente.

Após a duração especificada e o cookie expirar, a sessão não ficará mais aderente.

- *Persistência de sessão controlada por aplicativo*

Seu aplicativo gera um cookie que determina a duração da viscosidade da sessão. O balanceador de carga ainda emite seu próprio cookie de sessão sobre ele, mas agora segue o tempo de vida do cookie do aplicativo.

Isso torna as sessões fixas mais eficientes, garantindo que os usuários nunca sejam roteados para um servidor após o cookie de sessão local já ter expirado. No entanto, é mais complexo implementar porque requer integração adicional entre o balanceador de carga e o aplicativo.

GERENCIAMENTO DE SESSÕES PEGAJOSAS COM BALANCEAMENTO DE CARGA IMPERVA
A rigidez da sessão fornece uma maneira eficiente e precisa de manter as informações da sessão entre um visitante e um servidor em uma configuração de balanceamento de carga e pode ajudar a reduzir a carga de trabalho da rede.”

Exercício com Sticky Session

Para facilitar o gerenciamento dos containers Docker, vamos instalar o aplicativo Portainer no cluster. Para isto rode os comandos a seguir:

```
curl -L https://portainer.io/download/portainer-agent-stack.yml -o
portainer-agent-stack.yml
docker stack deploy --compose-file=portainer-agent-stack.yml portainer
```

Acesse o aplicativo no endereço 127.0.0.1:9000, defina a senha de administrador e acesse a tela principal.

Exercício Traefik

Use o servidor Traefik para implementar Stick Session no Docker Swarm. Publique o arquivo abaixo que irá implantar o Traefik.

```
version: "3"

services:

  loadbalancer:
    image: traefik
    command: --docker \
      --docker.swarmmode \
      --docker.watch \
      --web \
      --loglevel=DEBUG
    ports:
      - 80:80
      - 9090:8080
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

```
deploy:
  restart_policy:
    condition: any
  mode: replicated
  replicas: 1
  placement:
    constraints: [node.role == manager]
  update_config:
    delay: 2s
  networks:
    - net

networks:
  net:
```

Publique uma nova stack com o jsecurity que usará o traefik para fazer a Stick Session.

```
version: "3"

services:

  backend:
    image: nexus-imagens.pbh.gov.br/prodabel/jsecurity6swarm:latest
    networks:
      - traefik_net
    ports:
      - "80"
    deploy:
      restart_policy:
        condition: any
      mode: replicated
      replicas: 2
      update_config:
        delay: 2s
      labels:
        - "traefik.docker.network=traefik_net"
        - "traefik.port=80"
        - "traefik.frontend.rule=Host:jsecurity-treinamento.qa.pbh;"
        - "traefik.backend.loadbalancer.sticky=true"
    networks:
      traefik_net:
        external:
          name: traefik_net
```

A estratégia que adotamos de implantar 2 stacks independentes é para que o traefik seja reutilizado para todas as aplicações que precisarem de stick session. Através de labels colocados no serviço wildfly_backend é informado para o traefik as configurações que este serviço se propõe a receber as requisições.

Configuramos o traefik para responder na porta 80 e o link entre o traefik e os demais serviços, como o Wildfly, é feito através da sua capacidade dinâmica.