



Neste artigo vou apresentar os conceitos básicos relativos ao Docker sob o ponto de vista de um desenvolvedor .NET.



No [artigo anterior](#) vimos a comunicação entre dois contêineres usando a rede virtual.



A [rede virtual](#) que permitiu que os contêineres se comunicassem é um exemplo de [rede definida por software \(SDN\)](#). Como o nome sugere, as [SDNs](#) são redes criadas e gerenciadas usando software. As [SDNs](#) se comportam como redes tradicionais e usam portas de endereços IP regulares, mas não há interfaces de rede física e a infraestrutura para as redes, como serviços de nomes e roteamento, são fornecidos pelo [Docker](#).

Esta rede é na verdade uma abstração usada para facilitar o gerenciamento da comunicação de dados entre os contêineres e o ambiente externo ao docker.

Por padrão o [Docker](#) é disponibilizado com 3 redes que oferecem configurações específicas para gerenciar o tráfego de dados.

Para ver as redes digite o comando : [docker network ls](#)

```
macoratti@linux: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
macoratti@linux:~$ docker network ls  
NETWORK ID          NAME                DRIVER              SCOPE  
436f155b79ec        bridge             bridge              local  
a83847d4c15e        host               host                local  
853d3af4b37f        none               null                local  
macoratti@linux:~$
```

A rede do [host](#) é a rede do servidor [host](#) e a rede [none](#) é uma rede que não possui conectividade e que pode ser usada para isolar completamente os contêineres.

A rede [bridge](#) é a que nos interessa, pois o Docker adiciona todos os contêineres a essa rede quando os cria. Assim, cada container iniciado no docker é associado a uma rede específica. Essa é a rede padrão para qualquer container, a menos que associemos, explicitamente, outra rede a ele.

A rede confere ao container uma interface que faz [bridge](#) com a interface [docker0](#) do docker [host](#). Essa interface recebe, automaticamente, o próximo endereço disponível na rede [IP 172.17.0.0/16](#).

Todos os containers que estão nessa rede poderão se comunicar via protocolo TCP/IP. Se você souber qual endereço IP do container que deseja conectar, é possível enviar tráfego para ele, visto que estão todos na mesma rede [IP \(172.17.0.0/16\)](#). Foi o que fizemos no exemplo anterior.

Vamos inspecionar esta rede digitando o comando : [docker network inspect bridge](#)

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda

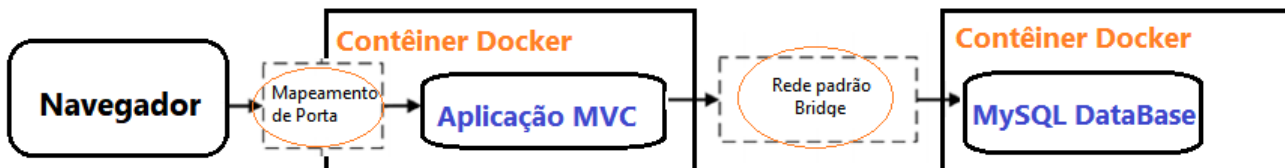
"ConfigFrom": {
  "Network": ""
},
"ConfigOnly": false,
"Containers": {
  "183f92036dd7197a11c6329fad617ce17260bdd60bad62b674e7a0d7492036c8": {
    "Name": "produtosapp",
    "EndpointID": "0293a2d53d5e3ad9d0660bb21a1d881241f776963bb1ec389ad390fce69ac7c9",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  },
  "1f5a4bb7f9d34d3a86a23a4162f0332d685d0433d49bb77c5c3ac5edfef7a505": {
    "Name": "mysql",
    "EndpointID": "458950e68fc6bda9bf79554e0030bef3717aafcecc3634325216b6bda8b6bca2",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  }
},
"Options": {
  "com.docker.network.bridge.default_bridge": "true",
  "com.docker.network.bridge.enable_icc": "true",
  "com.docker.network.bridge.enable_ip_masquerade": "true",
  "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
  "com.docker.network.bridge.name": "docker0",
  "com.docker.network.driver.mtu": "1500"
},
"Labels": {}
}
]

```

Vemos informações da rede como seu tipo, o endereço IP usado, e, na **seção "Containers"** vemos dois contêineres em execução exibindo a configuração de rede que o Docker aplicou a ambos :

- 1- contêiner **produtosapp** IP - 172.17.0.3
- 2- contêiner **mysql** IP - 172.17.0.2

Para melhor entender o efeito da rede Docker vejamos a composição da aplicação considerando a conectividade entre a aplicação **MVC(produtosapp)** e o **MySQL (mysql)** exibida na figura a seguir:



O navegador envia sua requisição HTTP para uma **porta** no sistema operacional do **host** que o Docker mapeia para uma porta no contêiner do aplicativo MVC. O aplicativo MVC solicita ao Entity Framework Core para fornecer dados, e o faz usando a **rede bridge padrão** para se comunicar com o aplicativo MySQL em execução em um contêiner separado.

O comando que usamos para criar o contêiner **mysql** é visto a seguir:

```
docker container run -d --name mysql -v produtosdata:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=numsey -e bind-address=0.0.0.0 mysql:8.0.0
```

Observe que neste comando não incluímos o mapeamento de uma porta, o que significa que o contêiner **mysql** não está acessível através de uma porta no sistema operacional **host**.

Escalando a aplicação ASP .NET Core MVC

As redes Docker (ou *redes definidas por software*) se comportam como redes físicas e os contêineres se comportam como servidores conectados a elas.

Isso significa que escalar a aplicação MVC para que haja vários servidores ASP.NET Core é muito simples, basta criar e iniciar contêineres adicionais.

Vamos criar então outro contêiner para a aplicação MVC digitando o comando a seguir:

docker container run -d --name produtosapp2 -p 3500:80 -e DBHOST=172.17.0.2 -e aspnetcoremvc/app:2.0

```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker container run -d --name produtosapp2 -p 3500:80 -e DBHOST=172.17.0.2
-e MESSAGE="Segundo Servidor" aspnetcoremvc/app:2.0
e341b8d0724cce4680bf9642a10b7a03dbaba6ae23befd5499d20fd0ed0d8e0b
macoratti@linux:~$
```

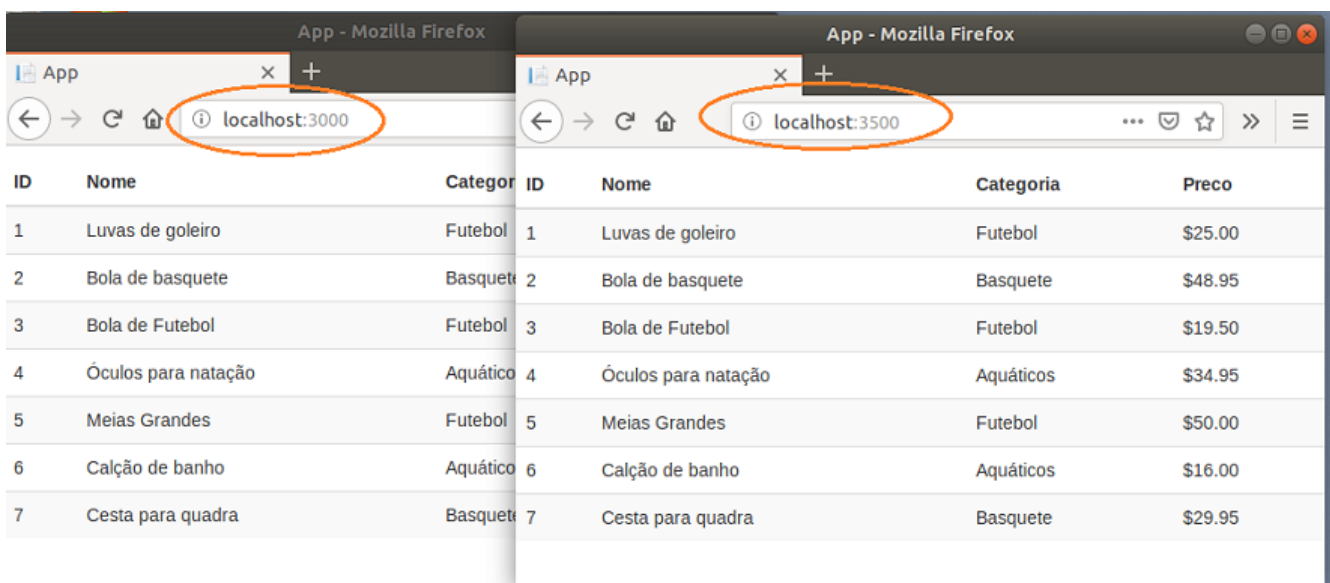
Ao criar contêineres adicionais para um aplicativo, é importante garantir que os novos contêineres tenham nomes diferentes (*produtosapp2* neste caso) e diferentes mapeamentos de porta (*porta 3500*, em vez da *porta 3000*, que é mapeada para a porta 80 para este contêiner).

O argumento **-d** informa ao Docker para iniciar o contêiner em segundo plano. O resultado é que temos um contêiner para a aplicação MVC que vai manipular requisições na porta **3000** e outro na porta **3500** do servidor host.

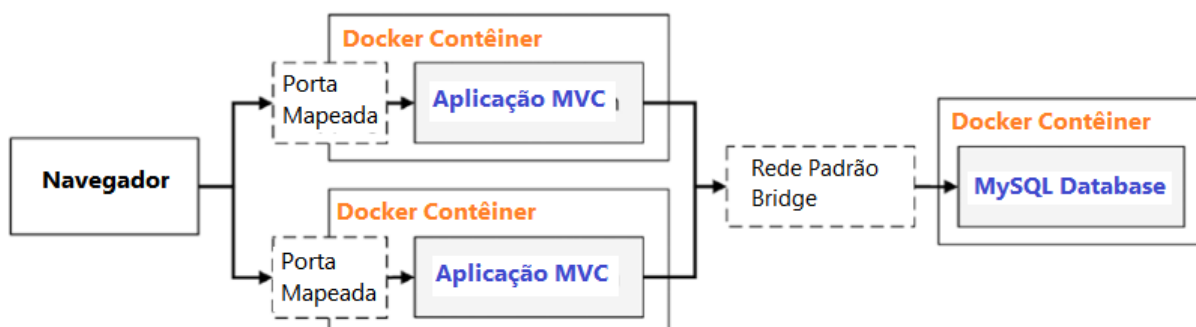
Para ver os contêineres em execução digite: **docker ps**

```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS                    NAMES
e341b8d0724c   aspnetcoremvc/app:2.0 "dotnet App1.dll"       2 hours ago   Up 2 hours   0.0.0.0:3500->80/tcp     produto
183f92036dd7   aspnetcoremvc/app:2.0 "dotnet App1.dll"       3 hours ago   Up 3 hours   0.0.0.0:3000->80/tcp     produto
1f5a4bb7f9d3   mysql:8.0.0         "docker-entrypoint.s..." 3 hours ago   Up 3 hours   3306/tcp                 mysql
macoratti@linux:~$
```

Para testar os contêineres, vamos abrir duas guias do navegador e digitar <http://localhost:3000> e <http://localhost:3500>, que produzirá o resultado mostrado na figura abaixo:



A figura a seguir mostra a composição da aplicação usando o segundo contêiner da aplicação MVC:



Assim, o Docker atribui a cada contêiner o seu próprio endereço IP, e, os contêineres podem se comunicar uns com os outros facilmente.

Criando redes customizadas

Podemos criar redes customizadas usando o comando **docker network create** seguido pelo nome da nova rede.

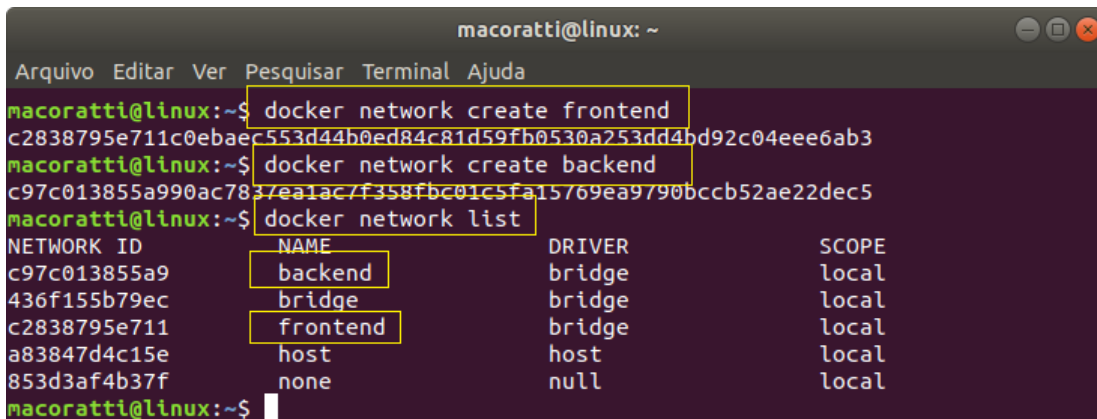
Ex: **docker network create <nome>**

Nota: Antes de prosseguir vamos remover todos os contêineres criados usando o comando :

docker container rm -f \$(docker ps -aq)

Vamos criar duas novas redes chamadas **frontend** e **backend** e a seguir exibir as redes:

```
docker network create frontend
docker network create backend
docker network list
```



```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker network create frontend
c2838795e711c0ebae553d44b0ed84c81d59fb0530a253dd4bd92c04eee6ab3
macoratti@linux:~$ docker network create backend
c97c013855a990ac7837eaa1ac7f358fbc01c5fa15769ea9790bccb52ae22dec5
macoratti@linux:~$ docker network list
NETWORK ID          NAME       DRIVER      SCOPE
c97c013855a9        backend    bridge      local
436f155b79ec        bridge     bridge      local
c2838795e711        frontend   bridge      local
a83847d4c15e        host      host        local
853d3af4b37f        none      null        local
macoratti@linux:~$
```

A rede **frontend** será usada para receber requisições HTTP pelos contêineres MVC.

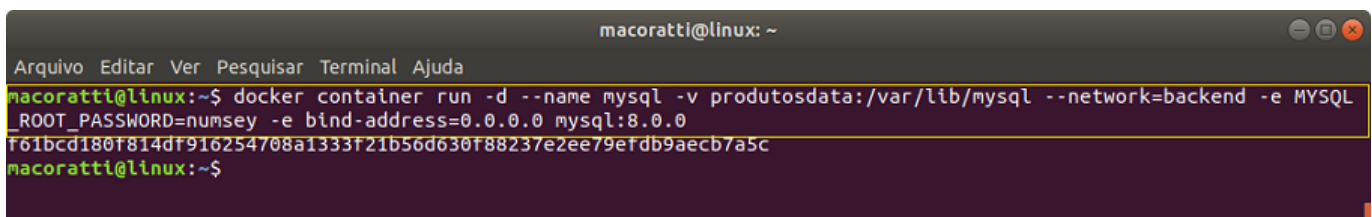
A rede **backend** será usada para consultas SQL entre os contêineres MVC e o contêiner MySQL.

Conectando um contêiner com a rede customizada

Depois de criar as redes customizadas podemos conectar os contêineres a elas usando o argumento **--network**, com os comandos **docker create** e **docker run**.

Vamos então criar um novo contêiner de banco de dados conectado à rede **backend** digitando o comando:

```
docker container run -d --name mysql -v produtosdata:/var/lib/mysql --network=backend -e
MYSQL_ROOT_PASSWORD=numsey -e bind-address=0.0.0.0 mysql:8.0.0
```



```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker container run -d --name mysql -v produtosdata:/var/lib/mysql --network=backend -e MYSQL
_ROOT_PASSWORD=numsey -e bind-address=0.0.0.0 mysql:8.0.0
f61bcd180f814df916254708a1333f21b56d630f88237e2ee79efdb9aecb7a5c
macoratti@linux:~$
```

Neste comando estamos usando o argumento **--network** que esta atribuindo o contêiner **mysql** à rede **backend**.

Para confirmar isso digite o comando : **docker network inspect backend**

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
  "Network": ""
},
"ConfigOnly": false,
"Containers": {
  "f61bcd180f814df916254708a1333f21b56d630f88237e2ee79efdb9aecb7a5c": {
    "Name": "mysql",
    "EndpointID": "af3b6ed81f30abfc2462d426b230cbb022d4471db2ca16f87748bc858020e21b",
    "MacAddress": "02:42:ac:13:00:02",
    "IPv4Address": "172.19.0.2/16",
    "IPv6Address": ""
  }
},
"Options": {},
"Labels": {}
}
]
macoratti@linux:~$

```

Observe que não há mapeamentos de porta nesse comando, o que significa que o banco de dados não pode ser alcançado através do sistema operacional do **host**. Em vez disso, ele só poderá receber conexões por meio da rede **backend** definida por software.

Criando os contêineres MVC

O recurso do DNS incorporado e o fato de que os mapeamentos de portas não são necessários, tornam mais simples a criação vários contêineres para a aplicação MVC, porque somente os **nomes de contêineres devem ser exclusivos**.

Vamos então criar 3 contêineres para a aplicação ASP .NET Core MVC digitando cada um dos comandos a seguir:

docker create --name produtosapp1 -e DBHOST=mysql --network backend aspnetcoremvc/app:2.0

docker create --name produtosapp2 -e DBHOST=mysql --network backend aspnetcoremvc/app:2.0

docker create --name produtosapp3 -e DBHOST=mysql --network backend aspnetcoremvc/app:2.0

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker create --name produtosapp1 -e DBHOST=mysql --network backend aspnetcore
mvc/app:2.0
8a3940c0151f67d766076b1685db32473bf0da0bb583f52696629b153f12bc37
macoratti@linux:~$ docker create --name produtosapp2 -e DBHOST=mysql --network backend aspnetcore
mvc/app:2.0
bfe85a50cd53a5261c581a4d8c5064807c63a652700a81935e3e99886a2cf7c2
macoratti@linux:~$ docker create --name produtosapp3 -e DBHOST=mysql --network backend aspnetcore
mvc/app:2.0
cae55df9a3d89985001703df10fe2d34c85368c119d97a76c700da236ae2b59f
macoratti@linux:~$

```

Nota : Os comandos **docker run** e **docker create** podem conectar um contêiner apenas a uma única rede.

Para ver os contêineres criados digite: **docker ps -a**

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
NAMES
cae55df9a3d8        aspnetcoremvc/app:2.0  "dotnet App1.dll"   4 minutes ago       Created
produtosapp3
bfe85a50cd53        aspnetcoremvc/app:2.0  "dotnet App1.dll"   4 minutes ago       Created
produtosapp2
8a3940c0151f        aspnetcoremvc/app:2.0  "dotnet App1.dll"   5 minutes ago       Created
produtosapp1
f61bcd180f81        mysql:8.0.0          "docker-entrypoint.s..." 21 minutes ago      Up 21 minutes
mysql
macoratti@linux:~$

```

Após criar cada um dos contêineres vamos executar os comandos para conectar cada contêiner da aplicação MVC à rede **frontend**, digitando os comandos:

`docker network connect frontend produtosapp1`

`docker network connect frontend produtosapp2`

`docker network connect frontend produtosapp3`

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker network connect frontend produtosapp1
macoratti@linux:~$ docker network connect frontend produtosapp2
macoratti@linux:~$ docker network connect frontend produtosapp3
macoratti@linux:~$

```

entender mais um pouco sobre como o Docker realiza a comunicação entre contêineres.

O comando `docker network connect` conecta os contêineres existentes às redes definidas por software. No exemplo estamos conectando os contêineres à rede `frontend`.

Agora que os contêineres da aplicação MVC foram conectados a ambas as redes do Docker, vamos executar o comando abaixo para iniciar os contêineres.

`docker start produtosapp1 produtosapp2 produtosapp3`

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker start produtosapp1 produtosapp2 produtosapp3
produtosapp1
produtosapp2
produtosapp3
macoratti@linux:~$

```

Verificando cada contêiner em execução : `docker ps`

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
cae55df9a3d8   produtosapp3                        "dotnet App1.dll"       12 minutes ago Up About a minute 80/tcp
bfe85a59cd53   produtosapp2                        "dotnet App1.dll"       12 minutes ago Up About a minute 80/tcp
8a3940c0151f   produtosapp1                        "dotnet App1.dll"       13 minutes ago Up About a minute 80/tcp
f61bcd180f81   mysql:8.0.0                         "docker-entrypoint.s..." 29 minutes ago Up 29 minutes    3306/tcp

```

Usando um balanceador de carga : HAProxy

Dessa forma criamos os contêineres MVC sem mapeamentos de portas, o que significa que eles são acessíveis somente por meio de redes definidas pelo software do Docker e não podem ser acessadas pelo sistema operacional do `host`.

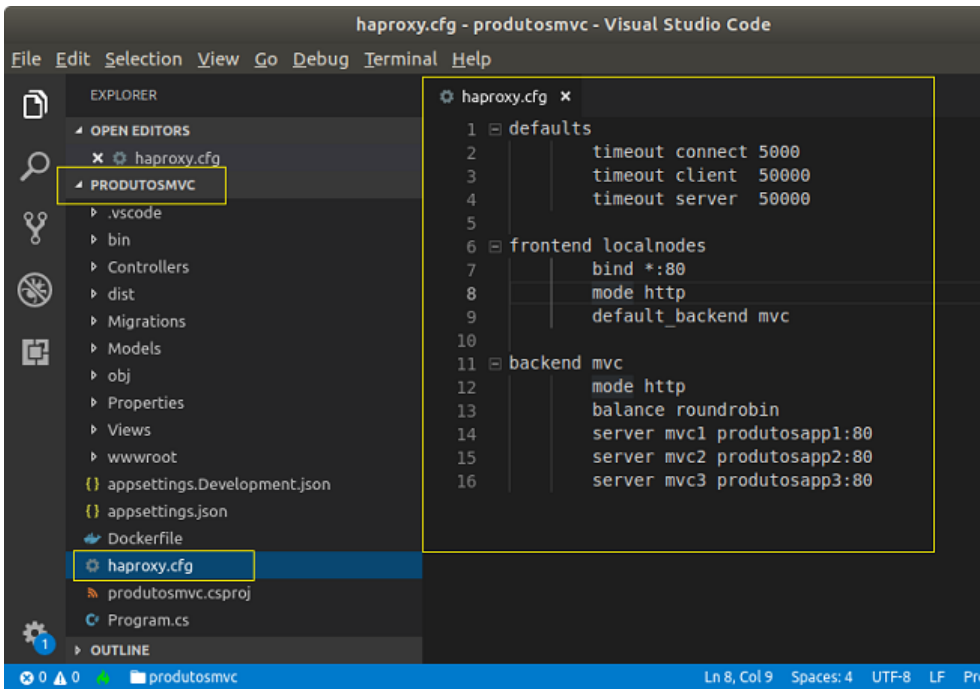
A peça que falta no quebra-cabeça é um `balanceador de carga`, que receberá requisições HTTP em uma única porta mapeada para o sistema operacional do `host` e vai distribuí-las para os contêineres do aplicativo MVC sobre a rede `frontend` do Docker.

Há muitas opções quando se trata de balanceadores de carga, e a escolha geralmente está ligada ao plataforma de infraestrutura para a qual o aplicativo foi implantado.

Para este exemplo, vamos usar o `HAProxy`, que é um excelente `balanceador de carga` HTTP que foi adotado pelo Docker para uso em seu ambiente de nuvem.

Nota: Não precisamos instalar o balanceador pois vamos usá-lo em um contêiner, mas, para instalar o `HAProxy` digite o comando no terminal : `sudo apt-get install haproxy`

Para configurar o `HAProxy` inclua um arquivo `haproxy.cfg` na pasta `produtosmvc` da aplicação MVC com o seguinte conteúdo: (este arquivo usa o `formato yaml`)



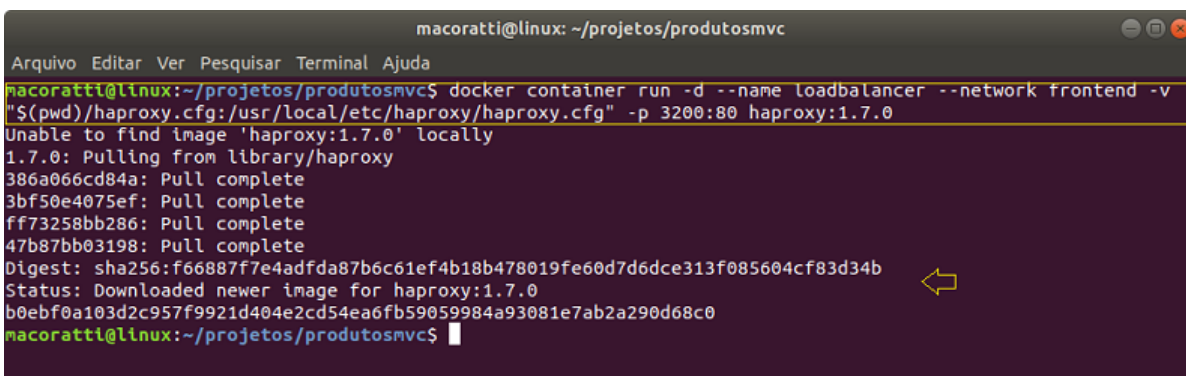
Esta configuração diz ao [HAProxy](#) para receber conexões na porta 80, que será exposta através um mapeamento no sistema operacional do [host](#) e vai distribuí-los aos três contêineres MVC, que são identificados usando seus nomes de contêiner, para aproveitar o recurso Docker DNS.

As requisições HTTP serão distribuídas para cada contêiner MVC, o que torna mais fácil testar o balanceador de carga e mostrar que os contêineres MVC estão sendo usados.

Para disponibilizar o arquivo de configuração para o balanceador de carga, vamos usar o recurso de [volumes do Docker](#) que permite que arquivos e pastas do sistema operacional host forneçam o conteúdo de um diretório em um sistema de arquivos de um contêiner, em vez de exigir um volume dedicado do Docker.

Agora basta digitar o comando abaixo a partir da pasta [produtosmvc](#) para criar o contêiner e fazer a conexão com a rede [frontend](#):

`docker container run -d --name loadbalancer --network frontend -v`
`"$(pwd)/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg" -p 3000:80 haproxy:1.7.0`



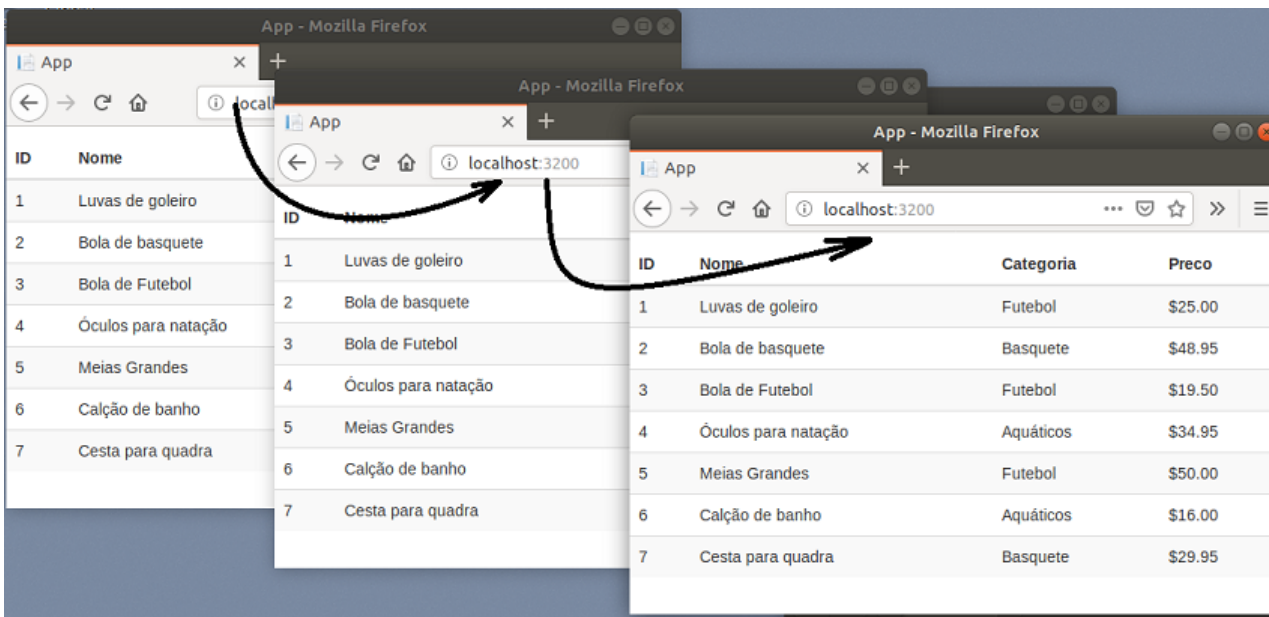
Vamos entender o comando:

1 - O argumento **`-v`** informa ao Docker para montar o arquivo [haproxy.cfg](#) que está na pasta [produtosmvc](#) no contêiner para que apareça no diretório [/usr/local/etc/](#) do [haproxy](#). (Esse é um recurso útil para criar contêineres que exigem arquivos de configuração.)

2- O argumento **`--network`** conecta o contêiner do balanceador de carga à rede [frontend](#) para que ela possa se comunicar com os contêineres MVC.

3- O argumento **`-p`** mapeia a [porta 3200](#) no sistema operacional do [host](#) para [porta 80](#) no contêiner para que o balanceador de carga possa receber solicitações do mundo externo.

Depois que o balanceador de carga for iniciado, abra uma guia do navegador e digite o endereço URL <http://localhost:3200>. Recarregue a página e você verá que o balanceador de carga distribui a solicitação entre os contêineres da aplicação MVC



O resultado final é um aplicativo que contém cinco contêineres, duas redes definidas por software e mapeamento de porta única.

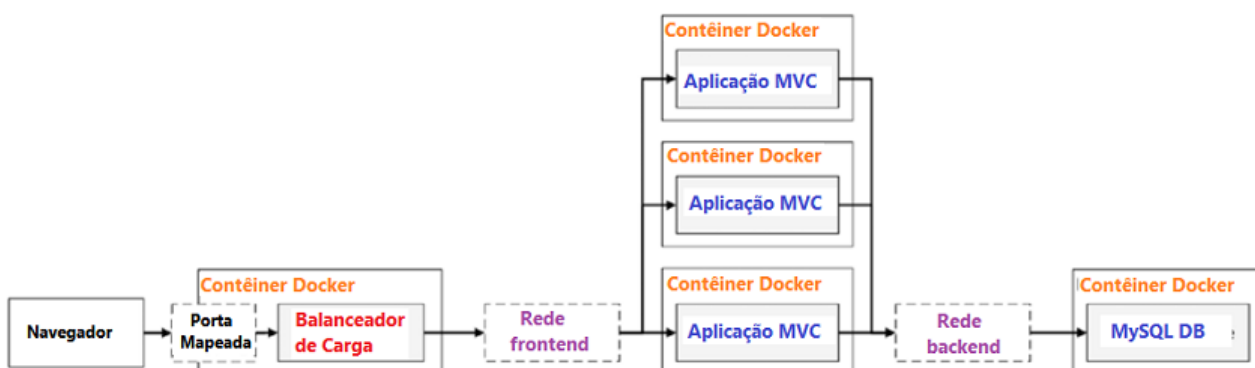
Exibindo todos os contêineres criados : [docker ps](#)

```

macoratti@linux: ~/projetos/produtosmvc
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/produtosmvc$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS        PORTS
b0ebf0a103d2   haproxy:1.7.0         "/docker-entrypoint..." 3 hours ago    Up 3 hours    0.0.0.0:3200->80/tcp
caebf0a103d2   loadbalancer
cae55df9a3d8   aspnetcoremvc/app:2.0  "dotnet App1.dll"        6 hours ago    Up 6 hours    80/tcp
produtosapp3
bfe85a50cd53   aspnetcoremvc/app:2.0  "dotnet App1.dll"        6 hours ago    Up 6 hours    80/tcp
produtosapp2
8a3940c0151f   aspnetcoremvc/app:2.0  "dotnet App1.dll"        6 hours ago    Up 6 hours    80/tcp
produtosapp1
f01bd180f81   mysql:8.0.0           "docker-entrypoint.s..." 7 hours ago    Up 7 hours    3306/tcp
mysql
macoratti@linux:~/projetos/produtosmvc$

```

A composição da aplicação é mostrada na figura abaixo, sendo muito parecida com as composições convencionais, embora todos os componentes estejam sendo executados em um [único servidor host](#) e são isolados por contêineres.



Observe que o contêiner do banco de dados está conectado apenas à rede [backend](#). Os contêineres da aplicação MVC estão conectados a ambas as redes front-end e backend; eles recebem requisições HTTP por meio da rede frontend e fazem consultas SQL pela rede [backend](#).

O contêiner do balanceador de carga está conectado apenas à rede frontend e possui um mapeamento de portas no sistema operacional [host](#); recebe solicitações HTTP via mapeamento de porta e os distribui para os contêineres da aplicação MVC.

Vimos assim como usar os recursos de rede para usar em aplicações mais complexas pela combinação de contêineres.

Na [próxima aula](#) vamos tratar do [Docker Compose](#).

