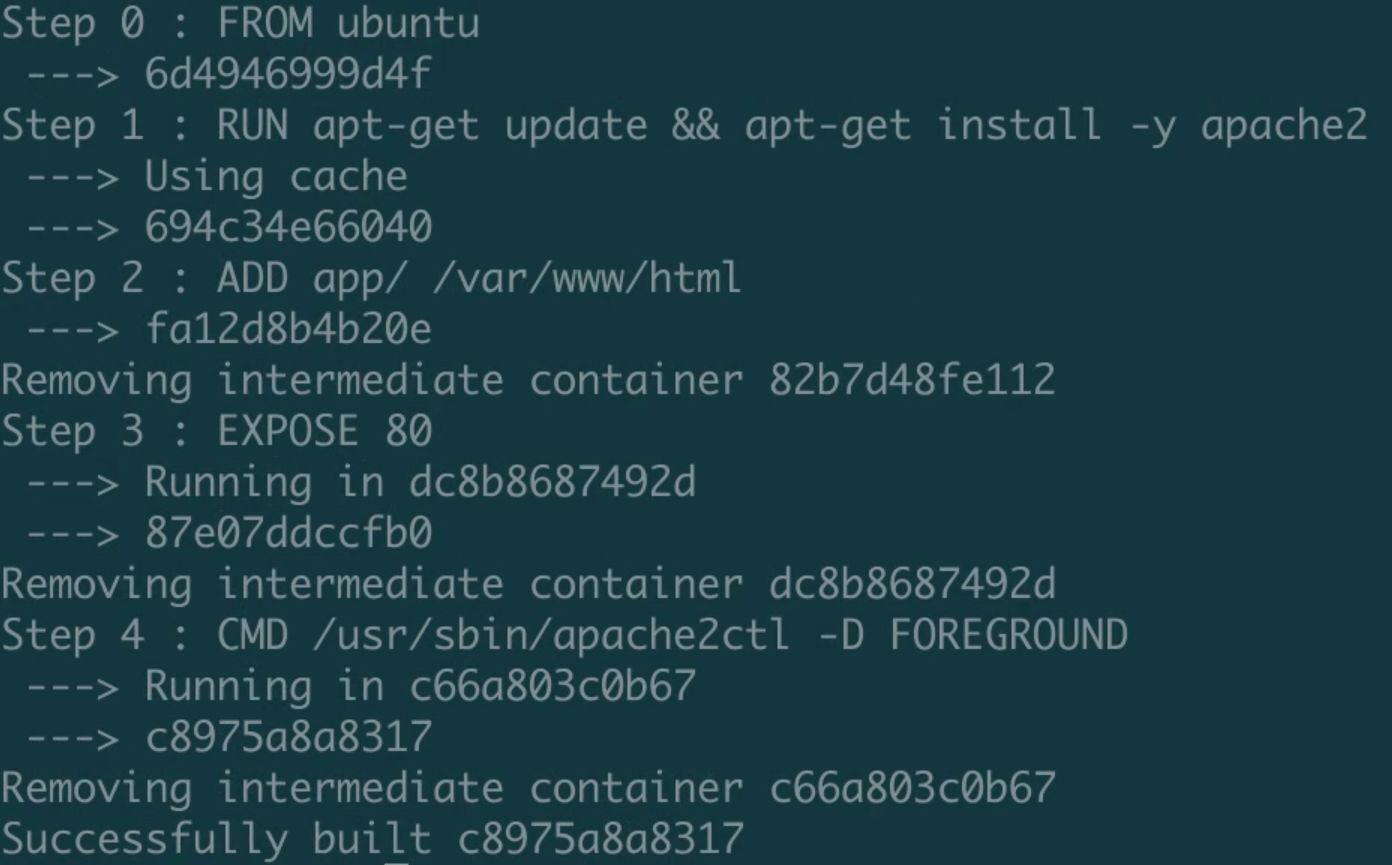
Adicionamos o diretório **app** dentro do container no caminho **/var/www/html**, que é o diretório padrão utilizado pelo Apache. Voltando para o terminal reconstruímos a nossa imagem:

docker build -t ubuntu/apache .



Quando criarmos um container essa aplicação já vai existir dentro dele. Vamos matá-lo novamente e fazer

docker run -d -p 80:80 ubuntu/apache

Se conferirmos no navegador:



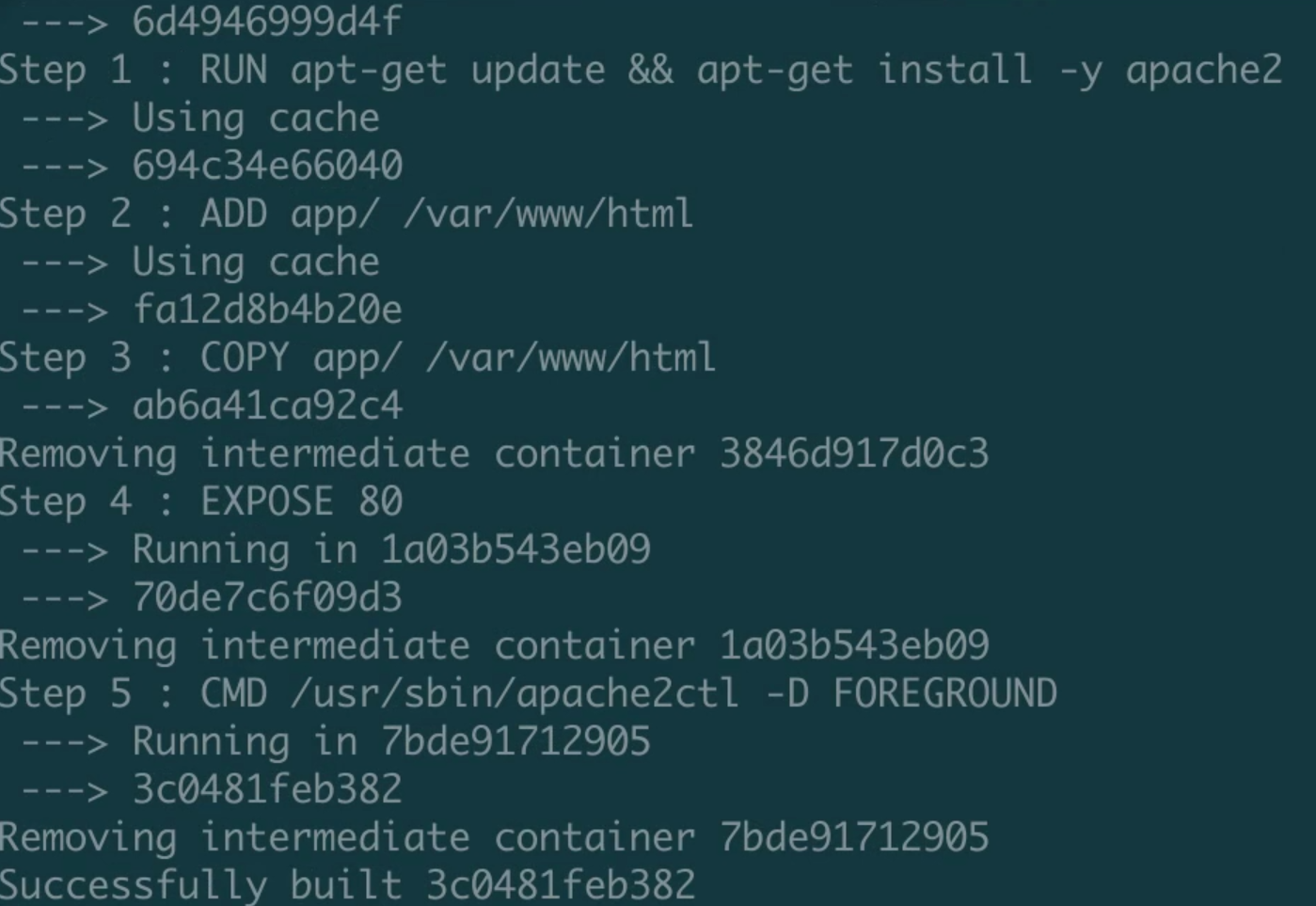
Ao inserir nossa aplicação no container, o Apache interpretou o diretório corretamente e renderizou a mensagem. Vejamos o que realmente aconteceu quando a instrução **ADD** foi executada na construção da imagem:

docker exec -it [nome ou id do container] ls /var/www/html  
database.config index.html

Os arquivos que representavam a configuração do banco de dados fictício foram inseridos diretamente na pasta onde o Web Server faz a leitura. O que o **ADD** fez de fato foi copiar os arquivos existentes no diretório "app" e enviou para "/var/www/html". Não queremos apenas os arquivos, mas sim todo o diretório, então no Dockerfile acrescentamos esta linha logo abaixo do **ADD**:

COPY app/ /var/www/html

Vamos reconstruir a imagem usando o **build** e, de fato:



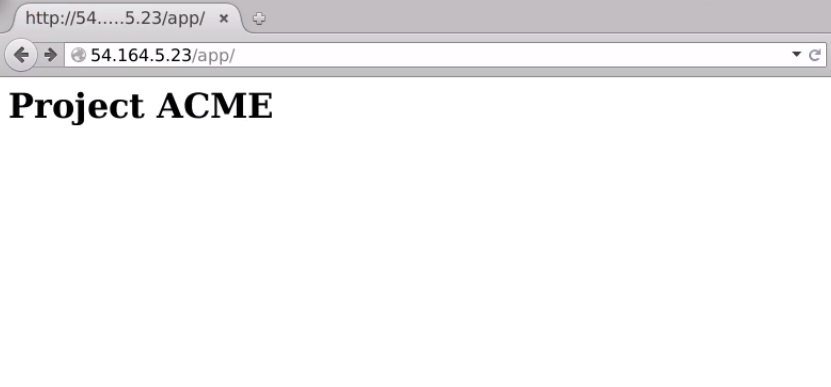
Matamos o container fazendo **docker kill [nome ou id]** e o construimos novamente:

docker run -d -p 80:80ubuntu/apache

Pelo navegador perceberemos que a Home do site ainda funciona apresentando a mensagem "Project ACME", porém se acrescentarmos "/app" depois do ip, a página não será encontrada. Isso acontece porque o **COPY** fez o mesmo trabalho do **ADD**. É necessário informar o diretório para que ele em si seja adicionado:

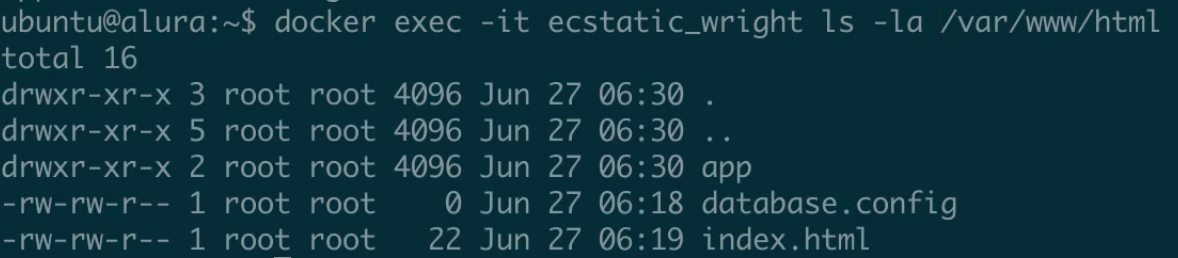
COPY app/ /var/www/html/app

Geramos novamente a imagem, matamos o container e o rodamos novamente. Agora sim:



Vamos ver tanto os arquivos, adicionados, como o diretório, copiado, para dentro do container:

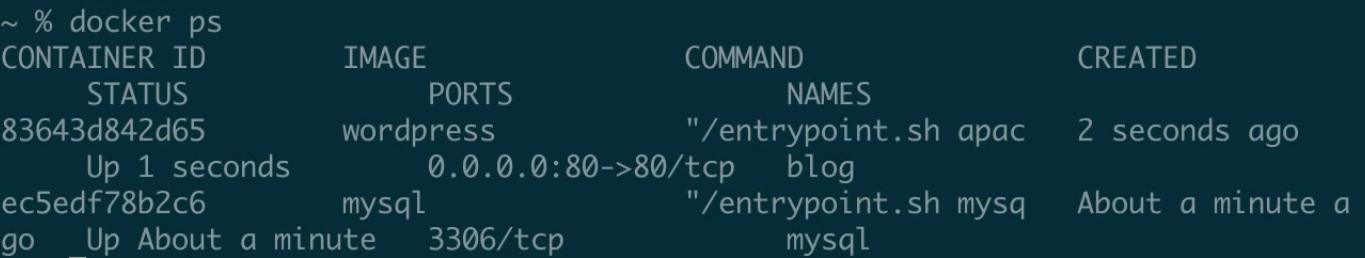
docker exec -it [nome do container] ls -la /var/www/html



# **Usando *docker compose* para controlar containers**

Na primeira aula desse curso criamos uma aplicação para o blog do Alura usando as imagens do WordPress e do MySQL. Fizemos tudo isso da seguinte forma:

docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=teste123 -d mysql  
  
docker run --name blog\_alura --link mysql:mysql -p 80:80 -d wordpress



Digitamos no navegador o ip da máquina local, o que refletiu a instalação do WordPress. Fizemos a instalação e tudo ocorreu conforme o planejado e muito rápido.

Note que na construção de ambas as imagens, utilizamos comandos iguais. Houve muita repetição e podemos resumir tais comandos criando um arquivo de Shell Script (.sh):

#!/bin/bash  
  
docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=teste123 -d mysql  
docker run --name blog -d --link mysql:mysql -p 80:80 wordpress

Salvamos com o nome de "criar\_blog.sh". No Terminal fazemos:

cat criar\_blog.sh

O que irá executar os comandos do arquivo. Lembrando sempre de pausar e remover os containers criados anterior e manualmente (**docker stop ´docker ps -qa´** e **docker rm ´docker ps -qa´**).

Criemos um diretório com o nome blog\_alura e entrar nele:

mkdir blog\_alura  
cd blog-alura/

Dentro deste diretório teremos um arquivo de manifesto chamado *docker-compose.yml*. Tal arquivo de manifesto será controlado pelo *docker compose*, o qual é uma ferramenta de complemento dentro do Docker. Comecemos a editar o arquivo:

db:  
 image: mysql  
 environment:  
 - MYSQL\_ROOT\_PASSWORD=test123  
  
blog:  
 image: wordpress  
 environment:  
 - WORDPRESS\_DB\_PASSWORD=teste123  
 links:  
 - db:mysql  
 ports:  
 - 80:80

Será criado um container de nome "db". para ele será utilizada a imagem "mysql". Além disso será feito o uso do environment passando a senha. O container "blog" tambémm fará uso de uma imagem, de uma senha, de um link com o "db:mysql" e mapeadas as portas. Salvamos e voltamos para o Terminal.

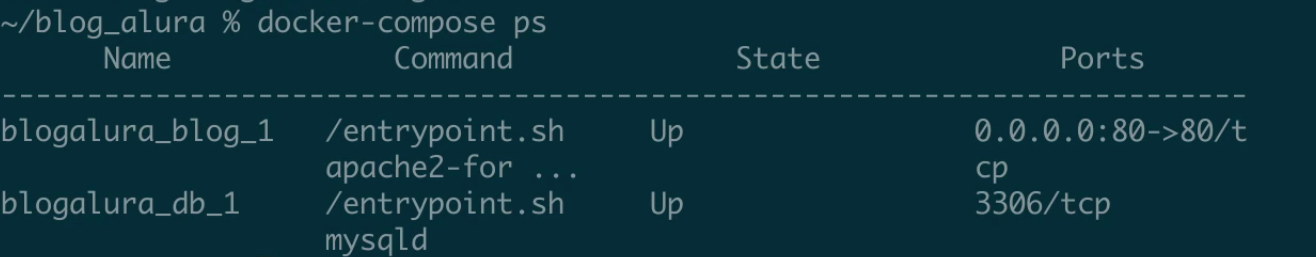
Para subir a aplicação fazemos:

docker-compose up  
  
Creating blogalura\_db\_1...  
Creating blogalura\_blog\_1...  
... [logs]

Esta é uma alternativa muito mais simples e menos confusa para rodar os containers de forma organizada. Perceba que apareceram todos os logs na criação dos containers. Podemos usar a instrução **-d** para rodar os processos em background:

docker-compose up -d

Para vermos que a operação foi realmente realizada:



# **Entendendo a comunicação entre containers no mesmo Host**

Vimos que o *docker-compose* para rodar containers de uma forma muito mais rápida. Ambos os containers criados possuem uma conexão de rede, de modo que se comuniquem usando a porta padrão. A questão agora é como isso realmente funciona por baixo dos panos. Primeiramente matamos e removemos os containers:

docker-compose kill  
...  
docker-compose rm

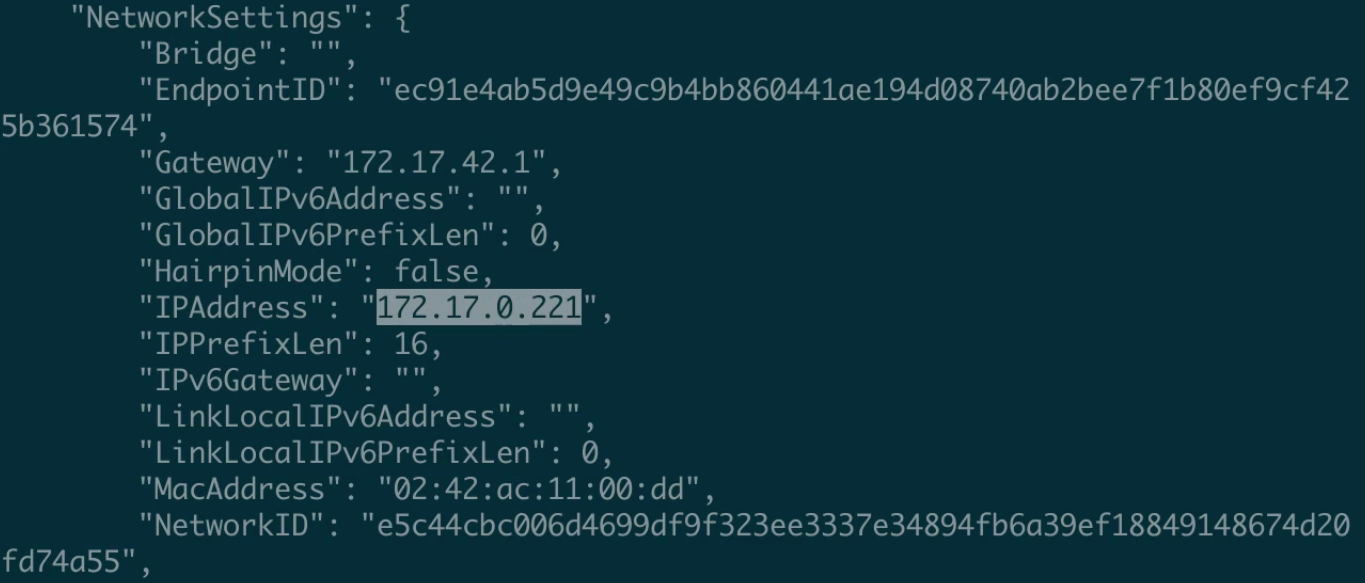
Façamos manualmente esta construção:

docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=teste123 -p 3306:3306 -d mysql

Diferente das outras vezes, adicionamos o mapeamento de portas, para termos acesso a elas. Vamos observar esse container para sabermos o ip dele, fazendo

docker inspect mysql

Retornará diversas informações. Sendo uma delas o ip que procuramos:



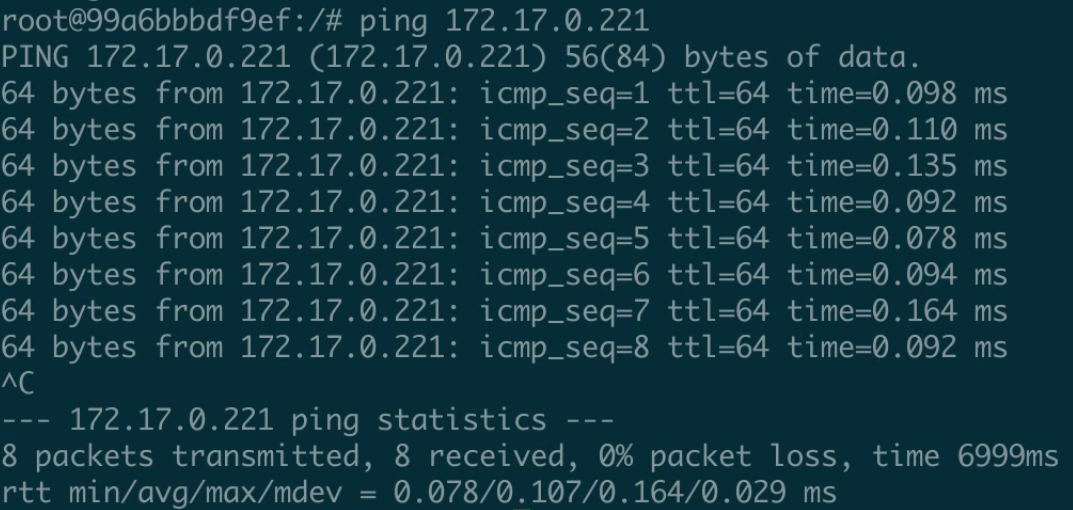
Criemos outro com a imagem do ubuntu e de forma interativa e autodestrutiva:

docker run --rm -it ubuntu bash

Depois fazemos dentro dele

ping 172.17.0.221 //o ip retornado no inspect

Para termos acesso com o ip do container do mysql. E , de fato, o ip consegue pingar:



Isso significa que, se tivermos um client de conexão com o banco de dados, podemos passar seu ip (depois de atualizar os pacotes com **apt-get update**):

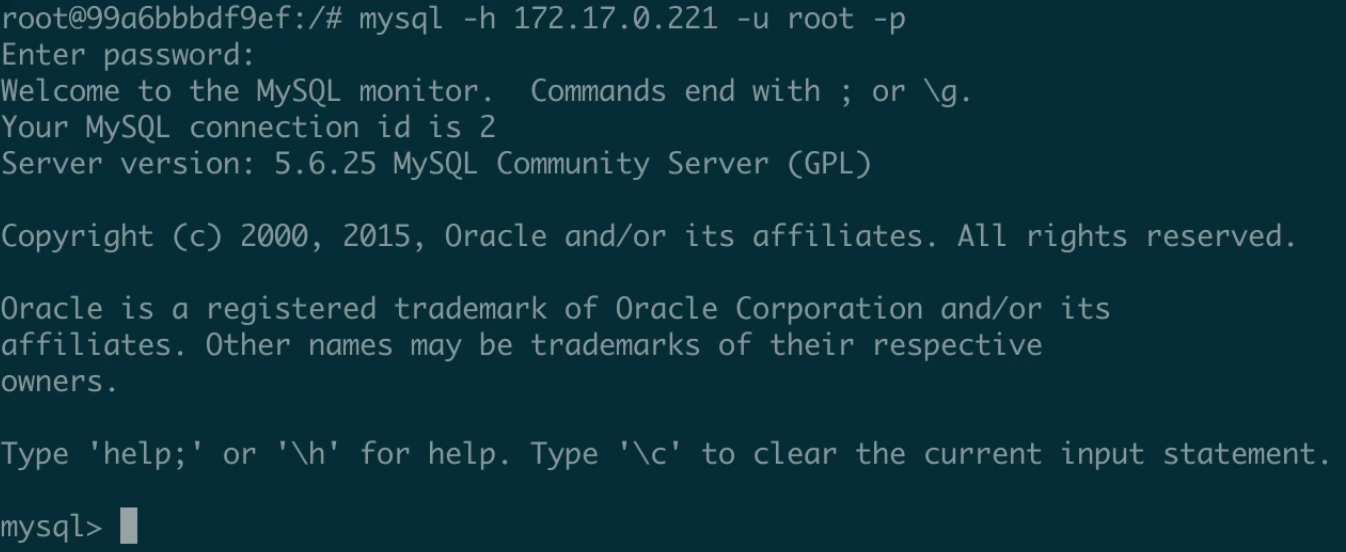
apt-get install -y mysql-client

Após isso, façamos um teste

mysql -h 172.17.0.221 -u root -p

* **-h** informa o ip,
* **-u** informa o usuário (root),
* **-p** informa a senha

Será pedida a senha (teste123) e será estabelecida a conexão com o MySQL, o qual está em outro container. Foi feita uma interatividade de rede via ip:

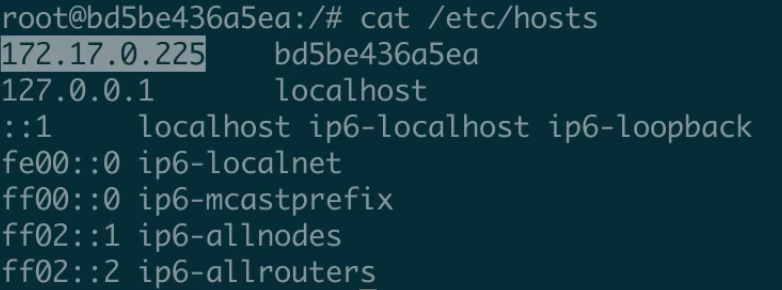


Porém, observe que se pausarmos o container

docker stop mysql  
  
docker start mysql

e restartarmos ele novamente, seu ip será outro! Vamos coletá-lo de uma maneira diferente e mais direta:

docker exec -it mysql bash //entra no prompt do container  
  
cat /etc/hosts //mostra o ip



Perceba que de fato o ip é diferente do anterior. Isso significa que, se a aplicação estivesse configurada para trabalhar com o primeiro, falharia assim que restartássemos o container. Vejamos a solução para esse problema.

A instrução **--link** informa o nome do container que queremos linkar e seu *alias*, ou apelido. Aplicamos isso tanto no Shell Script quanto no docker-compose. Fazendo isso manualmente criando um novo container:

docker run --rm --link mysql:db -it ubuntu bash

Se pedirmos para o "db" pingar, será pingado o mesmo ip (172.17.0.225). Isso significa que a aplicação vai poder utilizar sempre tal apelido (db). Este será sempre atribuído ao ip, independente de o ip mudar ou não.

Toda essa conectividade que acabamos de ver se chama \*\*Comunicação de rede entre containers no mesmo Host. Afinal, todos estes containers que estamos criando estão no mesmo host local. Ou seja, quando solicitamos que o compose crie o db e o blog, eles estão sendo criados na nossa máquina local estabelecendo comunicação no mesmo host.