# **Capítulo 5 - Usando *Dockerfile* para construir imagens**

Na aula passada criamos a imagem *ubuntu/apache* através da instrução **docker commit**. Porém, antes disso, tivemos que criar um container com **docker run** e, dentro dele, inserimos a instruções de **apt-get** para atualização dos pacotes e instalação do Apache 2.

Essa tarefa acaba se tornando muito repetitiva. Seria interessante se tivéssemos como automatizá-la. Fazemos isso através de um arquivo de texto que o Docker irá utilizar, com o nome de **Dockerfile**, para interpretar instruções de automação. Comecemos, então, a escrever no arquivo:

FROM ubuntu //imagem base  
  
RUN apt-get update && apt-get install -y apache2 //executa a instrução de instalação do Apache

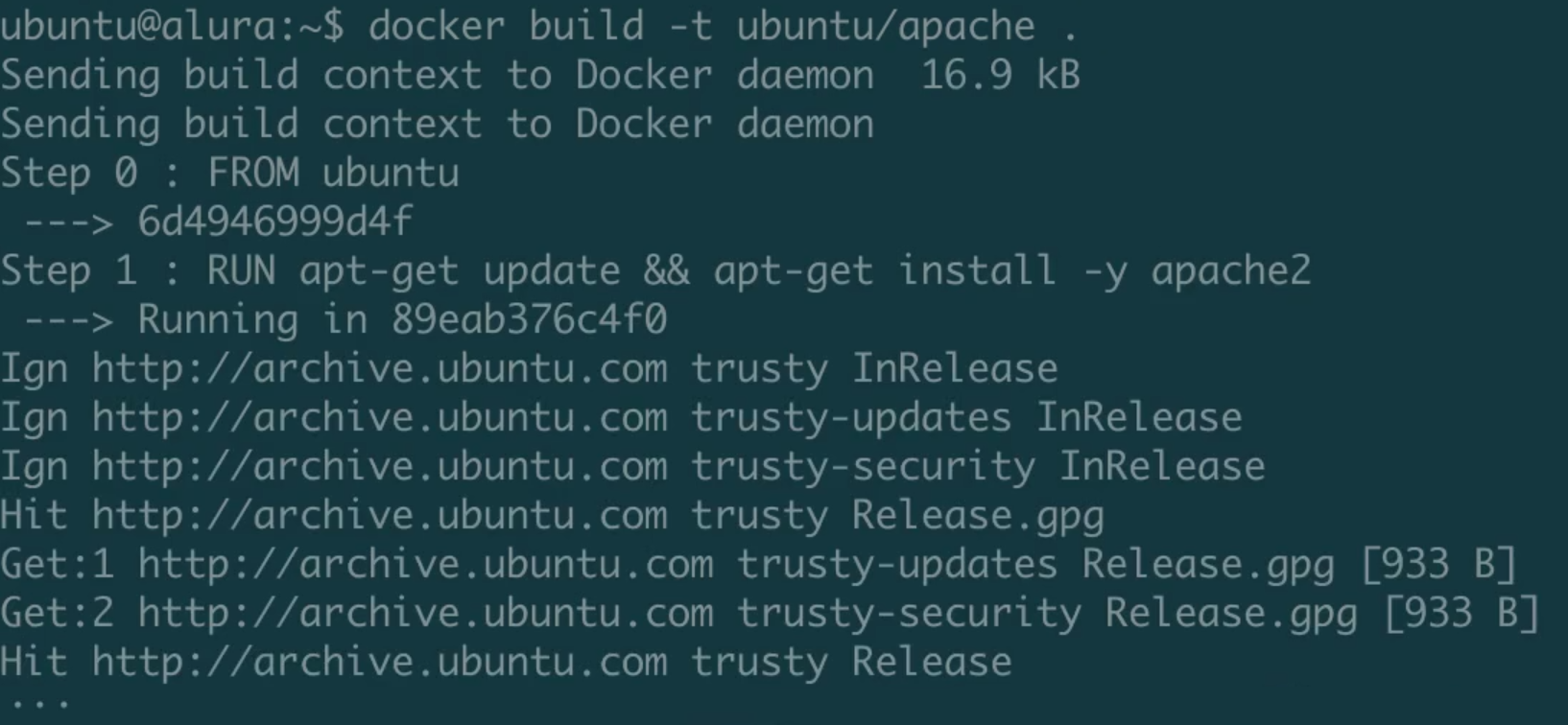
Antes de rodarmos o arquivo pelo Terminal, vamos destruir essa imagem do Ubuntu que já existe:

docker rmi ubuntu/apache

Agora nós podemos criar com as intruções de automação no nosso arquivo. Para isso o Docker possui uma feature chamada **build**, para a qual indicamos o nome da imagem com a instrução de tag **-t**. Precisamos ainda referenciar onde está o *Dockerfile*. Como aqui estamos no mesmo diretório só será necessário passar um ponto final:

docker build -t ubuntu/apache .

Com isso o Docker lerá as instruções no arquivo, irá gerar a imagem e executar os comandos de *apt-get*:



Para usar outras instruções direto no Terminal faríamos:

docker run -d -p 80:80 ubuntu/apache /usr/sbin/apache2ctl -D FOREGROUND

Dentro de um container do Docker, todo processo será em foreground. O Docker não trata processos em segundo plano, pois ele não possui um serviço de gerenciamento de processos.

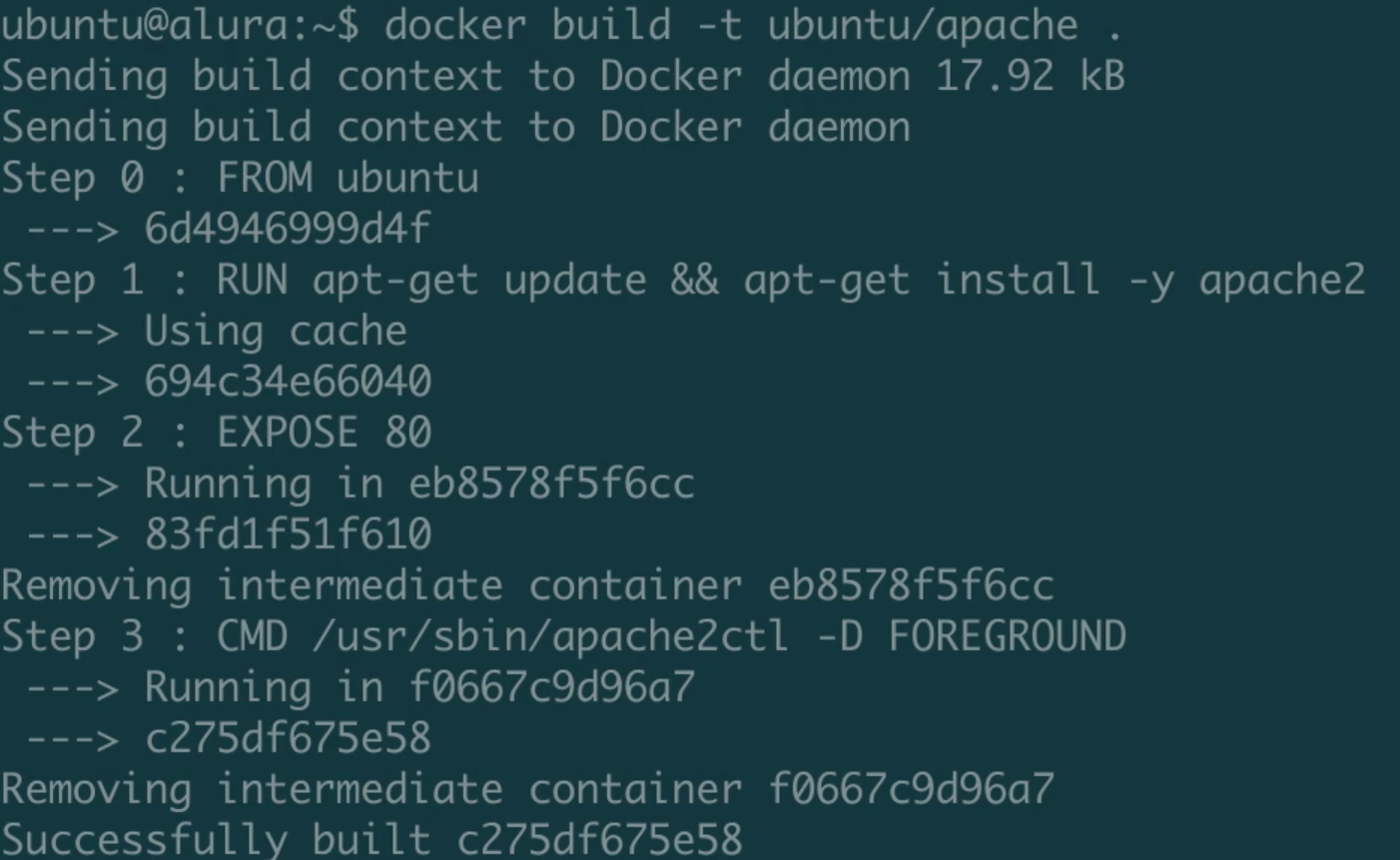
Voltando para o Dockerfile, podemos escrever a linha acima dessa maneira:

EXPOSE 80

Para liberar uma porta usamos a instrução **EXPOSE**. Além disso fazemos

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

Essa instrução executa o Apache. Então quando criarmos um container com essa imagem, não mais precisaremos inserir aquela instrução enorme. O **CMD** lê grupos de strings, por isso se faz necessária a separação com aspas. Salvamos e voltamos para o Terminal e fazemos novamente o **build**:



Perceba que os passos (steps) são mostrados para nós. No passo 1, na reconstrução da imagem, se faz uso do *cache*, e é isso que permite que ela seja bem mais rápida. Nós já havíamos passado por essa etapa anteriormente, então o Docker é inteligente o suficiente para usar um cache e não instalar tudo novamente.

Já que agora tudo está automatizado dentro da imagem, podemos construir o container:

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

De fato, tudo ocorreu como previsto:



Na próxima aula veremos mais automatizações usando o Dockerfile.