PÓS TECH

FULL STACK DEVELOPMENT

FASE 2 | AULA 04 -

MELHORES PRÁTICAS E SOLUÇÃO DE PROBLEMAS

SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, você aprenderá boas práticas para a criação de imagens, alguns cuidados e como reduzir a imagem base pode fazer toda a diferença em testes e desenvolvimento. Também falaremos sobre problemas comuns que o Docker e o Docker Compose podem apresentar e como solucioná-los. Por fim, veremos alguns casos de uso avançado.

HANDS ON

Nesta aula, aplicaremos as boas práticas para a criação de imagens Docker, aprenderemos a solucionar problemas comuns e apresentaremos também as camadas ou “layers” de uma imagem Docker, além de como usá-las a nosso favor para facilitar as alterações.

SAIBA MAIS

Nesta seção, vamos nos aprofundar mais sobre “layers”, que estão diretamente ligadas com as boas práticas de criação de imagens. Toda imagem criada a partir de um Dockerfile tem “layers” (ou camadas, em tradução livre). Uma camada representa uma instrução no Dockerfile, então a cada instrução adicionada no Dockerfile, será criada uma camada adicional.

A instrução “FROM” é a camada zero. As camadas têm como objetivo seguir um fluxo comum. A camada de baixo depende da camada de cima e assim sucessivamente.

Na figura, temos um exemplo de Dockerfile para visualizar as camadas na prática:

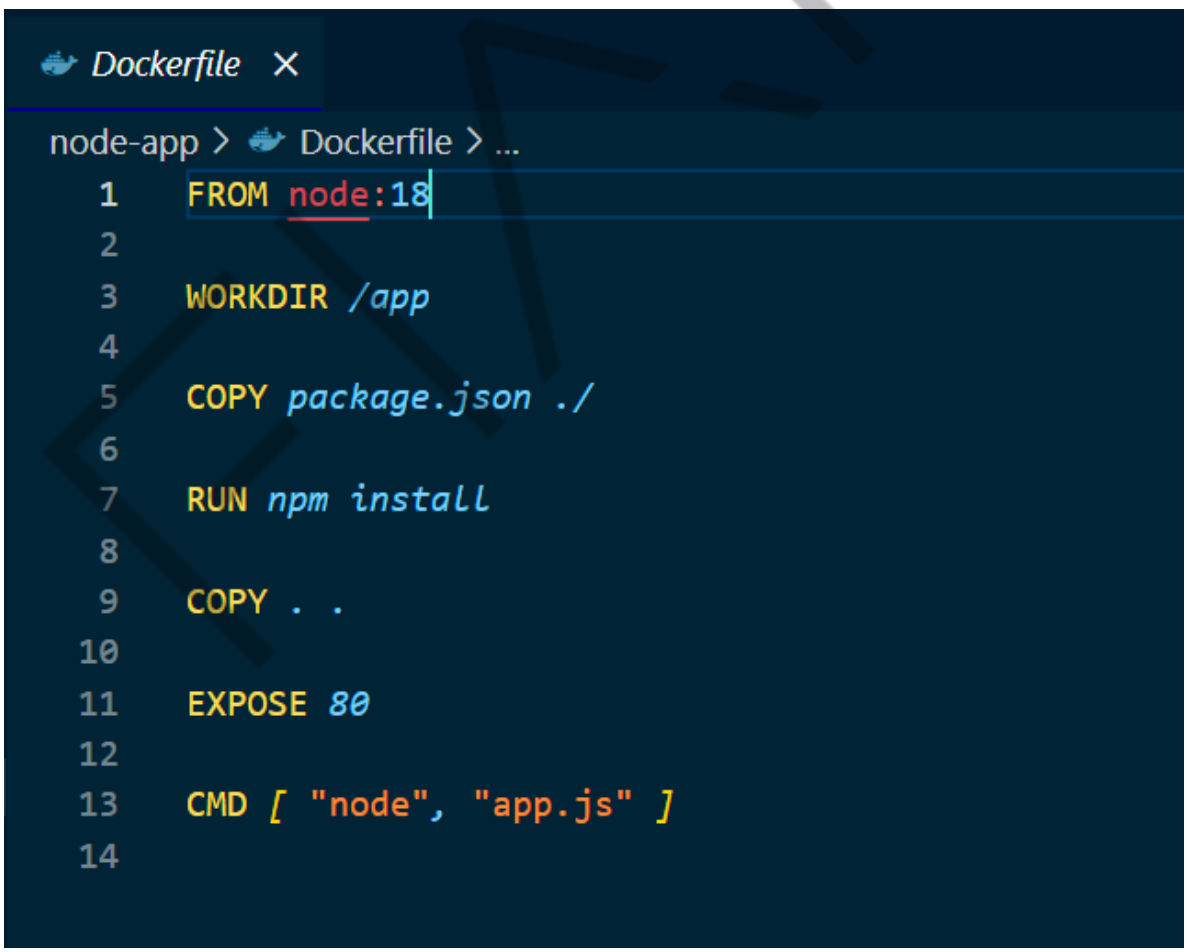

Figura 1 – Dockerfile - Layers
Fonte: Elaborado pelo autor (2023)

O arquivo mostrado tem sete (7) camadas (layers). Ao criar uma imagem, as próximas alterações no arquivo Dockerfile serão usadas como cache (dependendo em qual camada ocorreu a alteração).

Por exemplo, se adicionamos um novo arquivo na raiz do projeto, a camada “COPY”, da linha 9, sofrerá alterações. As camadas anteriores a ela, serão executadas como cache, o que agiliza muito a alteração da imagem. Após a alteração dessa camada, todas as outras camadas serão criadas novamente do zero. Um exemplo de saída com o cenário descrito:

```
Sending build context to Docker daemon  2.606MB
Step 1/7 : FROM node:18
18: Pulling from library/node
90e5e7d8b87a: Pull complete
27e1a8ca91d3: Pull complete
d3a767d1d12e: Pull complete
711be5dc5044: Pull complete
22956530cc64: Pull complete
d38ebdae17cd: Pull complete
bfacda23df76: Pull complete
4b9302a8baa0: Pull complete
Digest: sha256:a17842484dd30af97540e5416c9a62943c709583977ba41481d601ecffb7f31b
Status: Downloaded newer image for node:18
 ---> 219cd50149ac
Step 2/7 : WORKDIR /app
 ---> Running in 39b0136c06c7
Removing intermediate container 39b0136c06c7
 ---> 3328d0b69116
Step 3/7 : COPY package.json ./
 ---> 04a5cfa7df26
Step 4/7 : RUN npm install --silent
 ---> Running in e1d94b96dcf3
npm notice
npm notice New major version of npm available! 9.8.1 -> 10.2.4
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.2.4>
npm notice Run `npm install -g npm@10.2.4` to update!
npm notice
Removing intermediate container e1d94b96dcf3
 ---> 3e0b864763ad
Step 5/7 : COPY . .
 ---> 605dba4b8b8a
Step 6/7 : EXPOSE 80
 ---> Running in ceb4dda45c33
Removing intermediate container ceb4dda45c33
 ---> f1f28849f3d6
Step 7/7 : CMD [ "node", "app.js" ]
 ---> Running in f3c858a858a9
Removing intermediate container f3c858a858a9
 ---> ede35c4947ce
Successfully built ede35c4947ce
```

Figura 2 – Dockerfile – Layers Build
Fonte: Elaborado pelo autor (2023)

A imagem foi criada e, como é a primeira versão, demoramos cerca de 55 segundos para a criação. Já a figura “Dockerfile – Layers Build Cached" demorou cerca de 15 segundos, devido ao uso do cache nas camadas que não sofreram alterações:

```
Sending build context to Docker daemon  2.606MB
Step 1/7 : FROM node:18
 ---> 219cd50149ac
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 309366833734
Step 3/7 : COPY package.json ./
 ---> Using cache
 ---> a9e20c315c90
Step 4/7 : RUN npm install --silent
 ---> Using cache
 ---> 4d40d5172857
Step 5/7 : COPY . .
 ---> 28b753dbe8fd
Step 6/7 : EXPOSE 80
 ---> Running in 65afda0fed90
Removing intermediate container 65afda0fed90
 ---> 15ff51405053
Step 7/7 : CMD [ "node", "app.js" ]
 ---> Running in 28c73daa7f58
Removing intermediate container 28c73daa7f58
 ---> 2b47a65465a7
Successfully built 2b47a65465a7
```

Figura 3 – Dockerfile – Layers Build Cached
Fonte: Elaborado pelo autor (2023)

Dessa maneira, as camadas devem ser pensadas para obter essa estrutura, facilitando e agilizando os testes e o desenvolvimento locais. O uso do cache também se aplica aos repositórios on-line, como Docker Hub, AWS ECR, Azure Container Registry e Google Cloud Container Registry.

Outra boa prática já citada é o uso de ARGS (Argumentos) para não expor valores sensíveis em um arquivo Dockerfile. Os argumentos são passados no momento do build da imagem, que recebe esses argumentos e os insere como valor do argumento definido no arquivo Dockerfile, sendo possível utilizá-lo junto da instrução ENV. Veja um exemplo demonstrando essa boa prática:

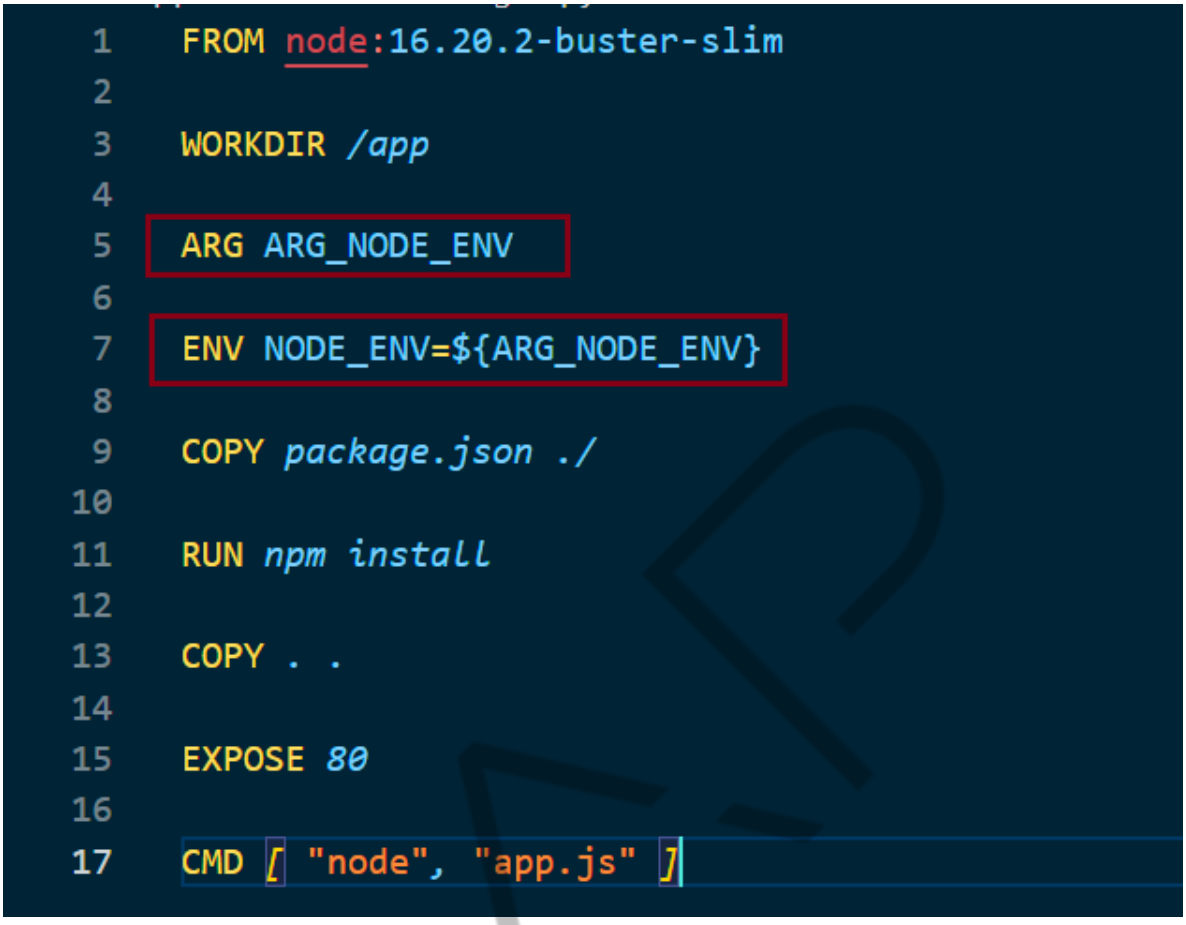

Figura 4 – Dockerfile – ARG com ENV
Fonte: Elaborado pelo autor (2023)

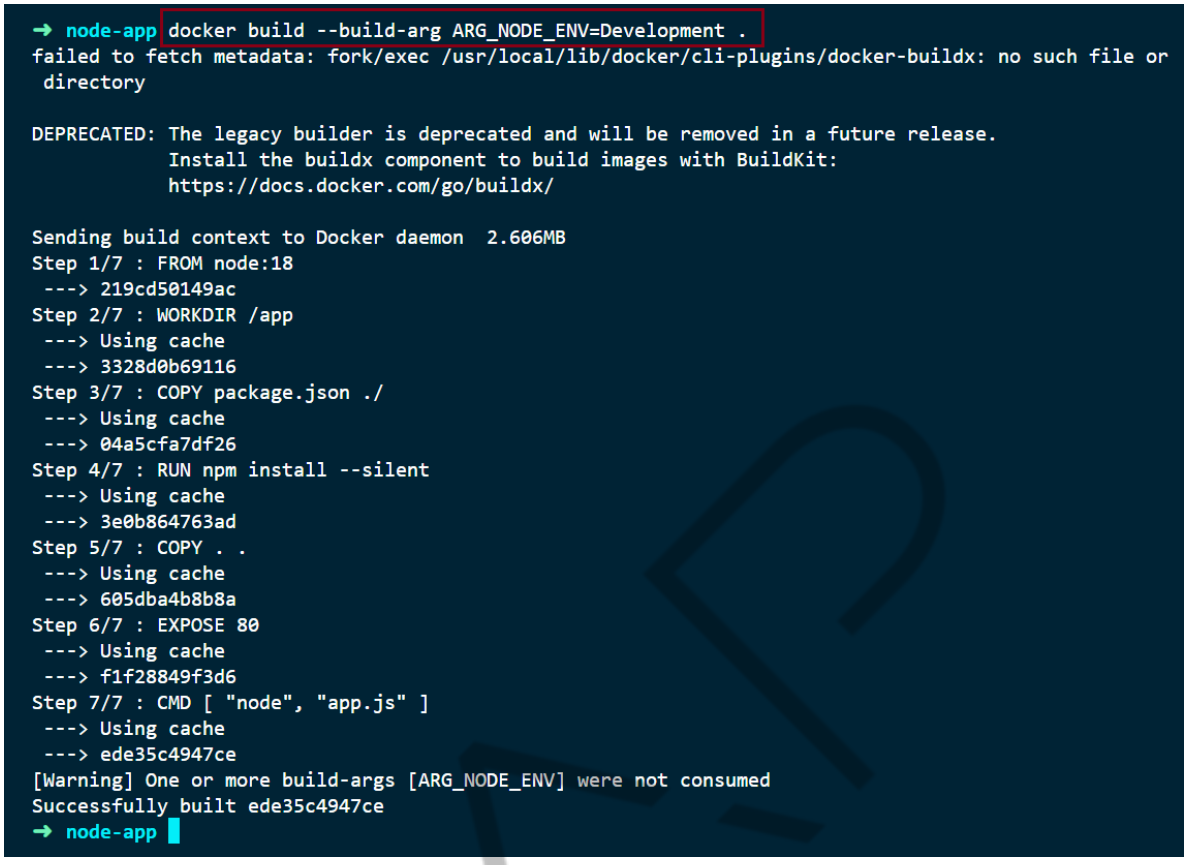

Figura 5 – Dockerfile – build-arg
Fonte: Elaborado pelo autor (2023)

Agora, por exemplo, podemos ver e entender sobre layers de imagens que estão no Docker Hub. A figura “Docker Hub – Layers” mostra os layers de uma imagem node:

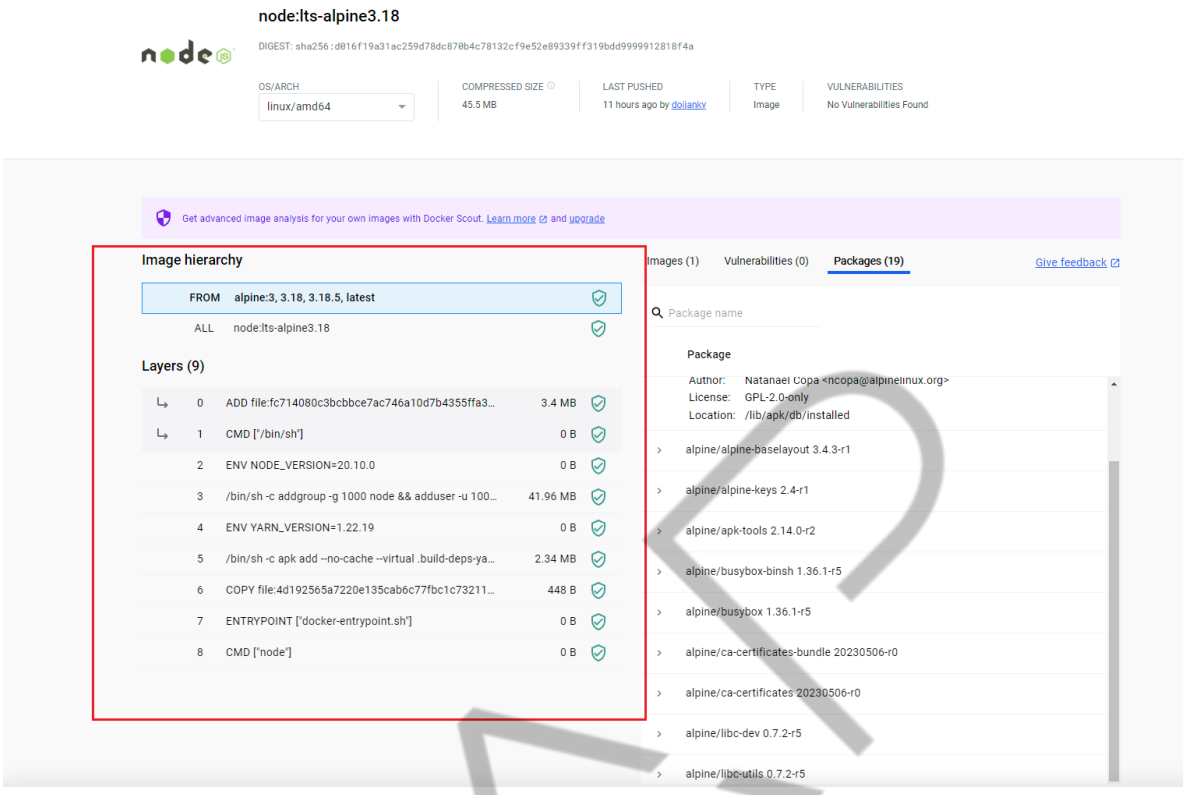

Figura 6 – Docker Hub – Layers
Fonte: hub.docker.com (2023), adaptado pelo autor (2023)

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, você viu o que são e como funcionam os layers de uma imagem docker, além da combinação das instruções ARG e ENV do Dockerfile. Esperamos que vocês tenham gostado desta aula! Em caso de dúvidas, nos procurem no Discord e não deixem de assistir às videoaulas. Até mais!

REFERÊNCIAS

BOAS práticas para criar imagens Docker eficientes, seguras e escaláveis. **Gaspar Barancelli.** 2023. Disponível em: <https://www.gasparbarancelli.com/post/boas-praticas-para-criar-imagens-docker-eficientes-seguras-e-escalaveis>. Acesso em: 09 jan. 2024.

DOCKER Hub – Node Tags and Layers. **HUB.DOCKER.COM.** [s.d]. Disponível em: <https://hub.docker.com/_/node/tags>. Acesso em: 09 jan. 2024.

PALAVRAS-CHAVE

ARG. Camadas. Dockerfile. Layers.