

Docker – Primeiro container

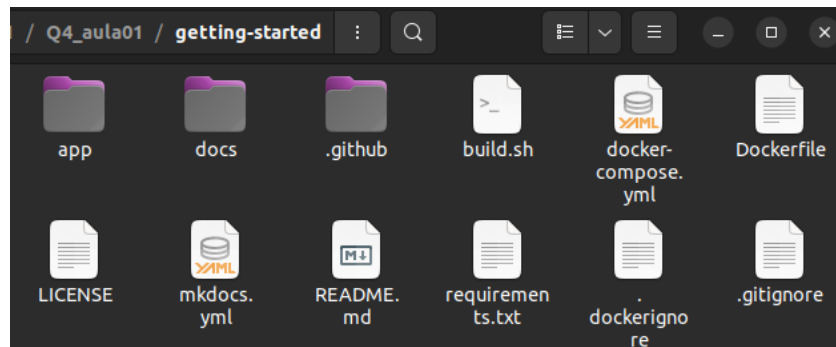
Para essa primeira atividade utilizando o docker, deve-se utilizar o passo a passo do tutorial disponível nesse link:

https://docs.docker.com/get-started/02_our_app/

Passo 1

Copiar os arquivos de uma aplicação “todo list” disponível nesse link <https://github.com/docker/getting-started/tree/master>

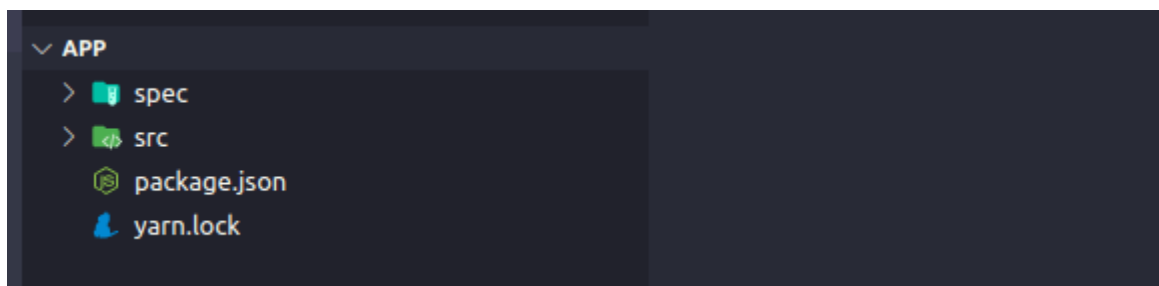
Apesar do tutorial indicar o git clone, não fiz assim porque o local onde estou trabalhando já é um repositório. Portanto, apenas copiei os arquivos da pasta “**getting-started**”:



Passo 2

Abrir o terminal a partir do diretório “app” e usar o comando “code .” para abrir o Visual Studio Code:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardS  
kills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula  
01/Q4_aula01/getting-started/app$ code .
```



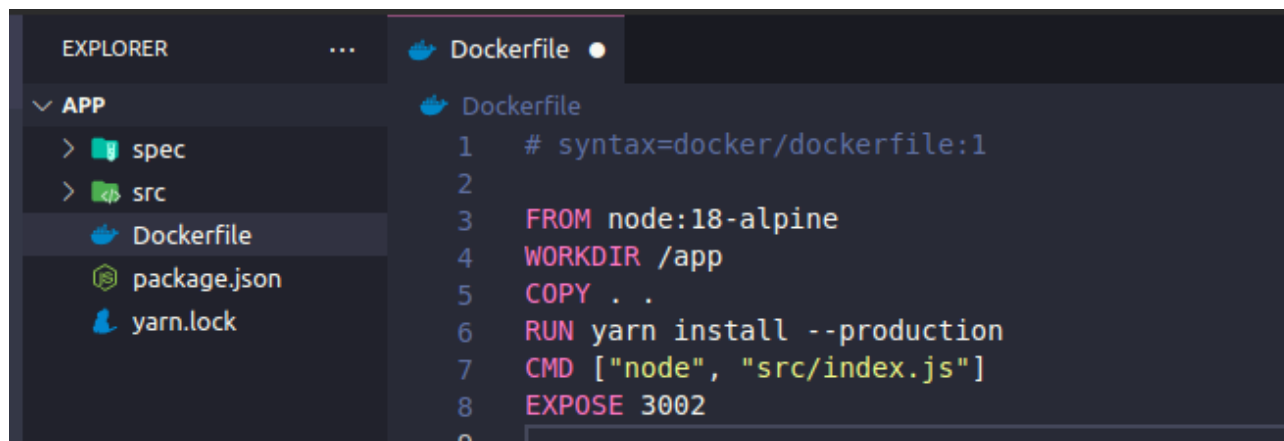
Passo 3

Construir o container image da nossa aplicação. Para isso, deve-se criar um arquivo chamado Dockerfile (sem extensão). Conforme explicado no exercício anterior, trata-se de um arquivo que conterá instruções arquivo de texto que contém uma série de instruções para criar uma imagem Docker. É usado para automatizar a construção de imagens Docker e garantir que cada imagem seja criada de forma consistente e replicável. Uma maneira de criar esse arquivo é através do comando “**touch Dockerfile**”. É importante destacar que isso deve ser feito dentro da pasta “app”

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/rep
itorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting
tarted/app$ touch Dockerfile
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/rep
```

Passo 4

Adicionar as instruções dentro do Dockerfile:

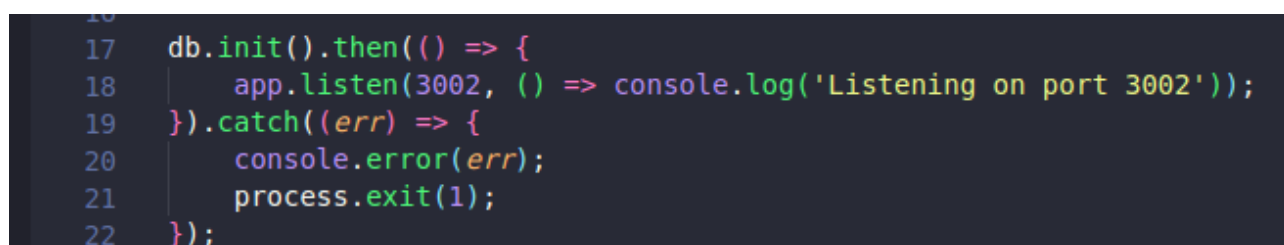


```
1 # syntax=docker/dockerfile:1
2
3 FROM node:18-alpine
4 WORKDIR /app
5 COPY . .
6 RUN yarn install --production
7 CMD ["node", "src/index.js"]
8 EXPOSE 3002
9
```

Sobre essas instruções:

- **# syntax=docker/dockerfile:1** Define a sintaxe utilizada no Dockerfile, nesse caso está sendo usado o formato de sintaxe "docker/dockerfile:1".
- **FROM node:18-alpine** Define a imagem base para a construção da nova imagem como node versão 18 com sistema operacional Alpine. Essa imagem já possui o node instalado e é uma imagem bem leve.
- **WORKDIR /app** Define o diretório de trabalho para o container. Todos os comandos subsequentes serão executados neste diretório.
- **COPY . .** Copia todo o conteúdo do diretório atual para dentro do container. O primeiro "." indica o diretório de origem (no caso, o diretório atual) e o segundo "." indica o diretório de destino (no caso, o diretório /app dentro do container).
- **RUN yarn install --production** O comando instala as dependências do projeto dentro do container e o sinalizador "--production" garante que apenas as dependências necessárias para a execução do aplicativo sejam instaladas.
- **CMD ["node", "src/index.js"]** Define o comando padrão que será executado quando o container for iniciado. Neste caso, ele executa o comando "node src/index.js", que é o comando para iniciar o aplicativo.
- **EXPOSE 3002** Informa ao Docker que o container expõe a porta 3002. No entanto, é importante ressaltar que essa instrução não publica automaticamente a porta para o host, sendo necessário definir isso no momento da execução do container.

Como tenho outras aplicações que estão usando a porta 3000, optei por usar a porta 3002. Desse modo, foi necessário efetuar algumas alterações no código para que a porta que será exposta pelo host tanto a porta exposta pelo container ficassem iguais a 3002.



```
16
17 db.init().then(() => {
18   app.listen(3002, () => console.log('Listening on port 3002'));
19 }).catch((err) => {
20   console.error(err);
21   process.exit(1);
22 });
```

Passo 5

Construir a imagem do container (build container image). Para isso, através do terminal, a partir de “app/” utiliza-se o comando:

docker build -t getting-started .

Esse comando usa o Dockerfile para construir a imagem do container baseado nas instruções descritas nele. Dependendo das instruções, o docker fará o download de imagens (por exemplo, o node) e constrói camadas (layers).

```
letonio@letonio-Inspiron-15-3567:~/Documents/alphaedtech/hardSkills/repetitorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting-started/app$ docker build -t getting-started .
Sending build context to Docker daemon  4.626MB
Step 1/6 : FROM node:18-alpine
--> 6f44d13dd258
Step 2/6 : WORKDIR /app
```

Sobre o comando anterior: a tag -t serve para dar uma tag a imagem, assim sempre que eu quiser me referir aquela imagem que foi criada, posso utilizar esse nome definido como tag. Nesse exercício, foi chamado de “getting-started”. O ponto final (.) diz que docker deve procurar um arquivo chamado Dockerfile no diretório atual

Após algum tempo, temos o seguinte resultado apresentado no terminal:

```
> b8311f51bdbb
Successfully built b8311f51bdbb
Successfully tagged getting-started:latest
letonio@letonio-Inspiron-15-3567:~/Documents/alphaedtech/hardSkills/repetitorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting-started/app$
```

A imagem foi criada (build) e com sucesso, foi atribuída uma tag getting-started:latest. Como a gente não definiu uma versão (nome:versao), ele atribui como mais recente (latest).

Passo 6

Agora que temos uma imagem criada, podemos executá-la a partir de um container. Para tal, utiliza-se esse comando:

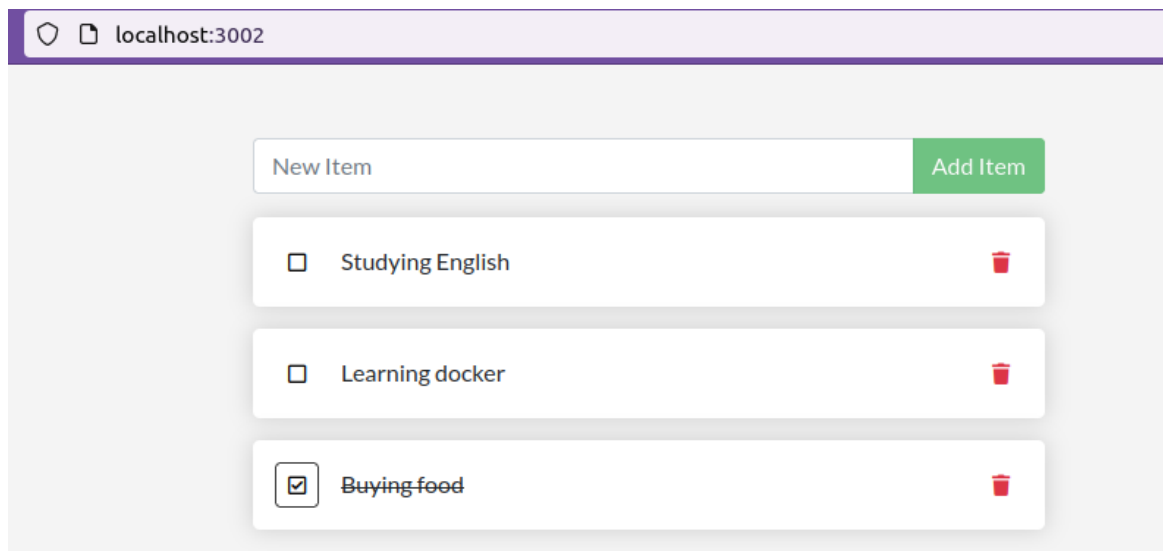
docker run -dp 3002:3002 getting-started

```
letonio@letonio-Inspiron-15-3567:~/Documents/alphaedtech/hardSkills/repetitorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting-started/app$ docker run -dp 3002:3002 getting-started
f44639058ab10987374fbd2a567a708dccf654865aa6eec567bf6b65efdc05da
letonio@letonio-Inspiron-15-3567:~/Documents/alphaedtech/hardSkills/repetitorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting-started/app$
```

Sobre esse comando: A flag -d serve para indicar que seja executado em background (detached), a tag -p indica a porta de conexão, nesse caso estamos indicando que a porta 3002 do host (meu computador) se conectará com o container a partir da porta 3002 do container. Sem essa conexão, não conseguiríamos acessar a aplicação que está dentro do container.

Passo 7

Após alguns segundos, basta abrirmos o navegador, e veremos a aplicação funcionando na porta 3002:



Nossa aplicação está funcionando porque quando criamos a imagem, um dos comandos do dockerfile era “node src/index.js”. Portanto, depois que a imagem é criada e a gente executou o container, a aplicação está funcionando.

Passo 8

Para verificar que nosso container está funcionando, podemos usar o comando “**docker ps**”:

```
letonio@letonio-Inspiron-15-3567:~/documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula01/Q4_aula01/getting-started/app$ docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
f44639058ab1	getting-started		"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	0.0.0.0:3002->3002/tcp,
:::3002->3002/tcp	inspiring_kowalevski					

Na imagem acima, nota-se que o status do container é UP há 14 minutos, o que finaliza esse tutorial.