

DOCKER – AULA 03 – RELATÓRIO

OBJETIVO

Atividade prática:

- 1 - Crie uma página index.html e utilize o **Node.js** para que rode na porta **3000**. E exiba a mensagem “Minha primeira imagem docker rodando!”.
- 2 - Usando um **Dockerfile**, crie uma imagem.
- 3 - Exponha a porta para rodar localmente com a porta **8088**.
- 4 - Suba a imagem para seu **Docker Hub**.
- 5 - Faça em forma de relatório, um documento com os comandos usados e os prints do processo.

RESOLUÇÃO

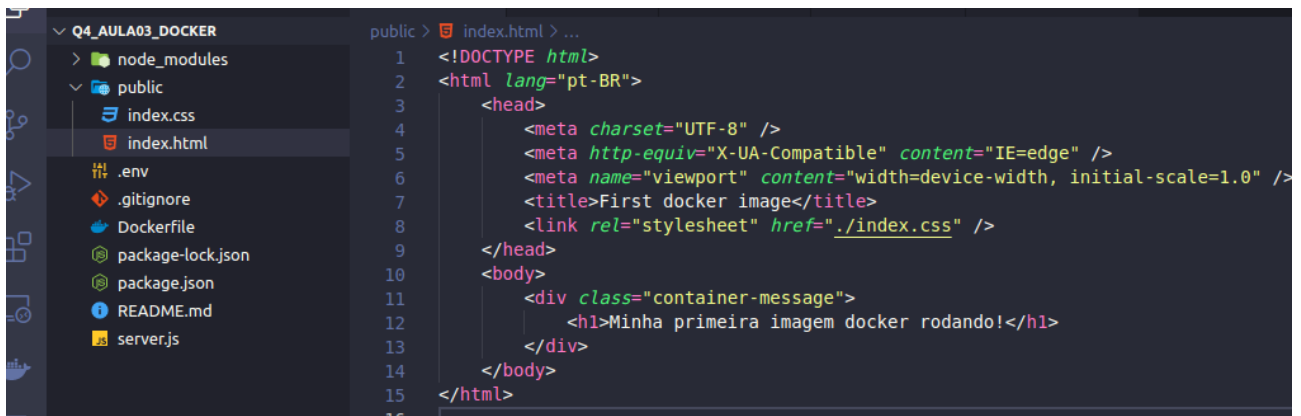
Na raiz do projeto, utilizou-se o comando “**npm init -y**”, a flag y dá a resposta padrão a algumas perguntas do npm quando desejamos iniciar um projeto. As seguintes dependências foram instaladas:

- express (desenvolver e levantar um servidor);
- dotenv (leitura das variáveis de ambiente contidas no arquivo .env).

Dentro do arquivo package.json, adicionamos um script para iniciar o servidor (“**start**”: “**node server.js**”). A imagem a seguir ilustra o arquivo package.json:

```
1  {
2    "name": "q4_aula03_docker",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "node server.js",
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "dependencies": {
14     "dotenv": "^16.0.3",
15     "express": "^4.18.2"
16   }
17 }
```

A imagem a seguir mostra como ficou a organização da nossa aplicação:



The screenshot shows a code editor with a sidebar on the left displaying the file structure of a project named 'Q4_AULA03_DOCKER'. The files listed are: node_modules, public (containing index.css and index.html), .env, .gitignore, Dockerfile, package-lock.json, package.json, README.md, and server.js. The main editor area shows the content of 'index.html' in the 'public' directory. The HTML code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>First docker image</title>
8     <link rel="stylesheet" href="./index.css" />
9   </head>
10  <body>
11    <div class="container-message">
12      <h1>Minha primeira imagem docker rodando!</h1>
13    </div>
14  </body>
15 </html>
```

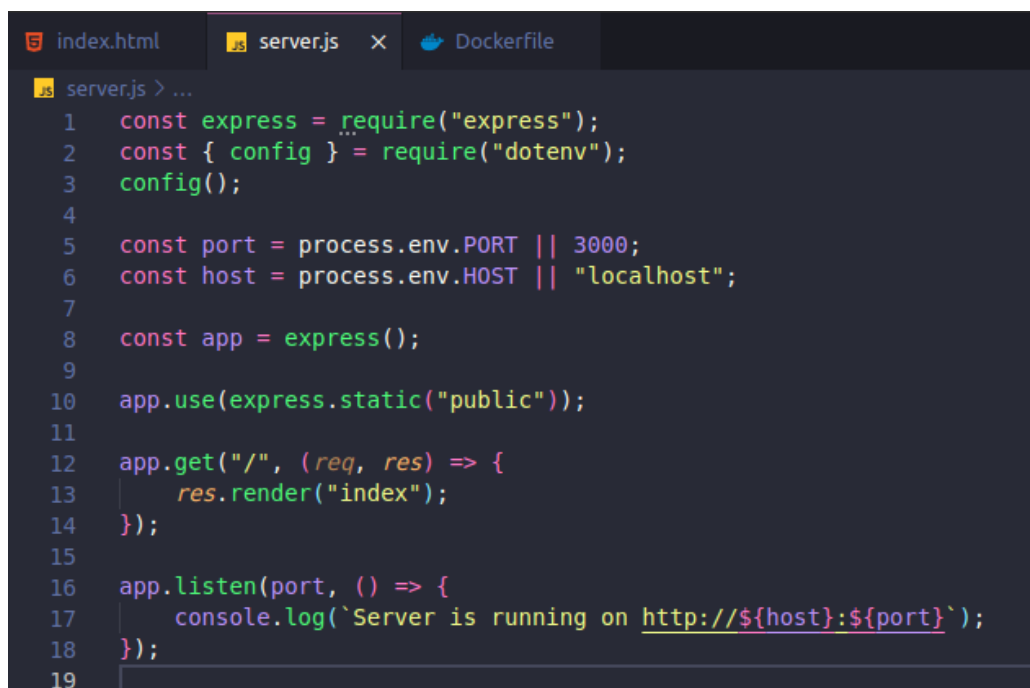
Na raiz do projeto, é possível observar uma pasta chamada **public**, que contém o arquivo **index.html** da página. Trata-se de um html simples com apenas uma mensagem centralizada.

Utilizou-se a extensão **Liveserver** para verificar a aparência durante o processo de desenvolvimento. A imagem a seguir aponta como ficou página.



The screenshot shows a web browser window with a solid blue background. In the center, the text "Minha primeira imagem docker rodando!" is displayed in a bold, black, sans-serif font.

Foi necessário criar um arquivo **server.js** na raiz do projeto, responsável por conter toda a implementação referente ao servidor **express**. Na imagem a seguir, mostra-se o conteúdo do arquivo **server.js**:



The screenshot shows a code editor with three tabs open: 'index.html', 'server.js', and 'Dockerfile'. The 'server.js' tab is active, showing the following JavaScript code:

```
1 const express = require("express");
2 const { config } = require("dotenv");
3 config();
4
5 const port = process.env.PORT || 3000;
6 const host = process.env.HOST || "localhost";
7
8 const app = express();
9
10 app.use(express.static("public"));
11
12 app.get("/", (req, res) => {
13   res.render("index");
14 });
15
16 app.listen(port, () => {
17   console.log(`Server is running on http://${host}:${port}`);
18 });
19
```

Nota-se que há duas variáveis de ambiente sendo importadas do arquivo **.env**, **port** e **host**, respectivamente. É feita a importação do **express** e do **dotenv**, mais especificamente sua funcionalidade **config**. Uma instância do **express** é atribuída a constante **app**. Para servir o conteúdo estático da pasta **public**, utilizou-se **express.static("public")**.

Criou-se uma rota **get** que renderiza o **index.html** quando o navegador acessa a raiz da aplicação. A função **listen** permite ao nosso servidor ouvir na porta indicada. Atualmente, o arquivo **.env** especifica a porta **3000**.

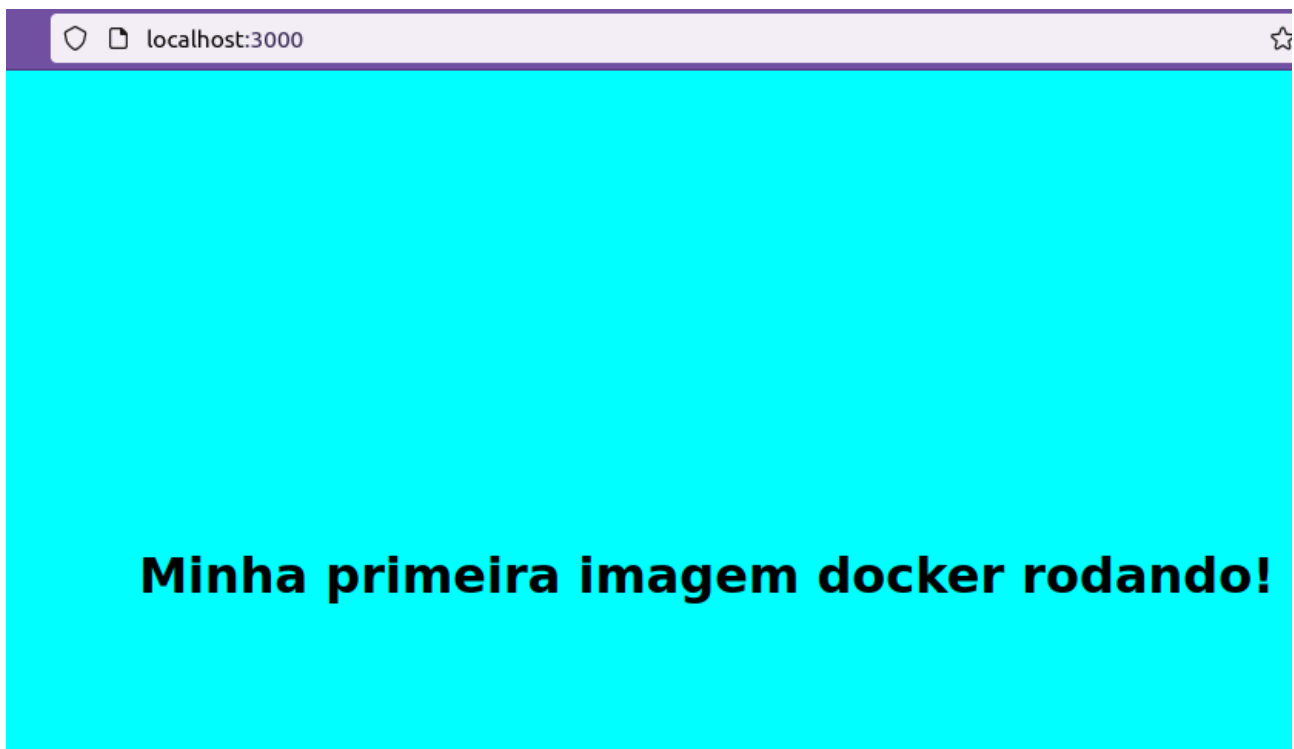
Para levantar o servidor, basta usar o comando **"npm run start"** a partir da raiz do projeto, como mostrado a seguir:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula03/Q4_aula03_docker$ npm run start

> q4_aula03_docker@1.0.0 start
> node server.js

Server is running on http://localhost:3000
```

Acessando o navegador no endereço indicado, temos o resultado mostrado abaixo:



Ainda na raiz do projeto, criamos um arquivo **Dockerfile** e, dentro dele, inserimos o conteúdo abaixo:

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "server.js"]
EXPOSE 3000
```

Explicando linha a linha:

FROM indica que a imagem base que será empregada nessa aplicação será o node versão 18-alpine.

O comando **WORKDIR** define como diretório de trabalho uma pasta /app. Portanto, as próximas instruções levam em consideração esse diretório de trabalho.

O comando **COPY . .** (dois pontos espaçados) copia tudo que está no local onde o Dockerfile se encontra e manda para dentro do diretório de trabalho, app.

Nesse caso, o comando **RUN** instala as dependências node que estão especificadas no arquivo package.json quando a imagem está sendo criada.

O comando **CMD** especifica o comando padrão que deve ser executado sempre que um container é iniciado a partir dessa imagem, trata-se de um comando para iniciar a aplicação.

Por fim, o comando **EXPOSE** expõe a porta **3000** do container.

Com o arquivo Dockerfile pronto, a próxima etapa é criar uma imagem a partir dele. Com o terminal na mesma posição onde o arquivo Dockerfile está, vamos usar o seguinte comando:

docker build -t first-image .

Explicando cada elemento do comando acima:

docker build serve para criar imagens;

A **tag -t** permite atribuir uma etiqueta/marcar a imagem, assim podemos nos referenciar a ela sem que seja necessário usar o id que vai ser gerado, podemos usar o nome que foi definido após a flag.

Por fim, o ponto serve para indicar que o docker deve procurar um arquivo Dockerfile a partir do local atual do terminal.

```
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula03/Q4_aula03_docker$ docker build
-t first-image .
Sending build context to Docker daemon 2.513MB
Step 1/6 : FROM node:18-alpine
--> 6f44d13dd258
Step 2/6 : WORKDIR /app
--> Using cache
```

Os passos contidos no Dockerfile aparecem no terminal conforme são executados. Ao final do processo uma mensagem indica o sucesso da criação da imagem. Como não definimos uma versão para a imagem, fica atribuída automaticamente como latest (mais recente).

```
Removing intermediate container 75717a000020
--> 0f3fff89387b
Successfully built 0f3fff89387b
Successfully tagged first-image:latest
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula03/Q4_aula03_docker$
```

Para confirmar que a imagem foi criada, podemos utilizar o comando **“docker image ls”**, conforme mostrado a seguir:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edte
ker$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
first-image	latest	0f3fff89387b	2 minutes ago	177MB
my-postgres-image	latest	f72262529e65	3 days ago	379MB
getting-started	latest	b8311f51bdbb	5 days ago	265MB

A próxima etapa é criar um container que executará dentro dele de forma isolada a aplicação que desenvolvemos a partir da imagem criada no passo anterior.

Para criar um container, utilizamos esse comando:

docker run -dp 8088:3000 --name my-first-container first-image

Explicando cada detalhe do comando acima, temos que:

- **docker run** cria e executa o container a partir da imagem fornecida
- a **flag -d** serve para executar esse comando docker em background (detached);
- a **flag -p** indica o mapeamento entre as portas, ou seja, associa uma porta da máquina local (host) com uma porta 3000 do container, desse modo é possível manipular o container quando ele estiver ativo.
- **--name** indica o nome que desejamos atribuir ao container
- **first-image** é a imagem que estamos utilizando como referência para o container que será criado.

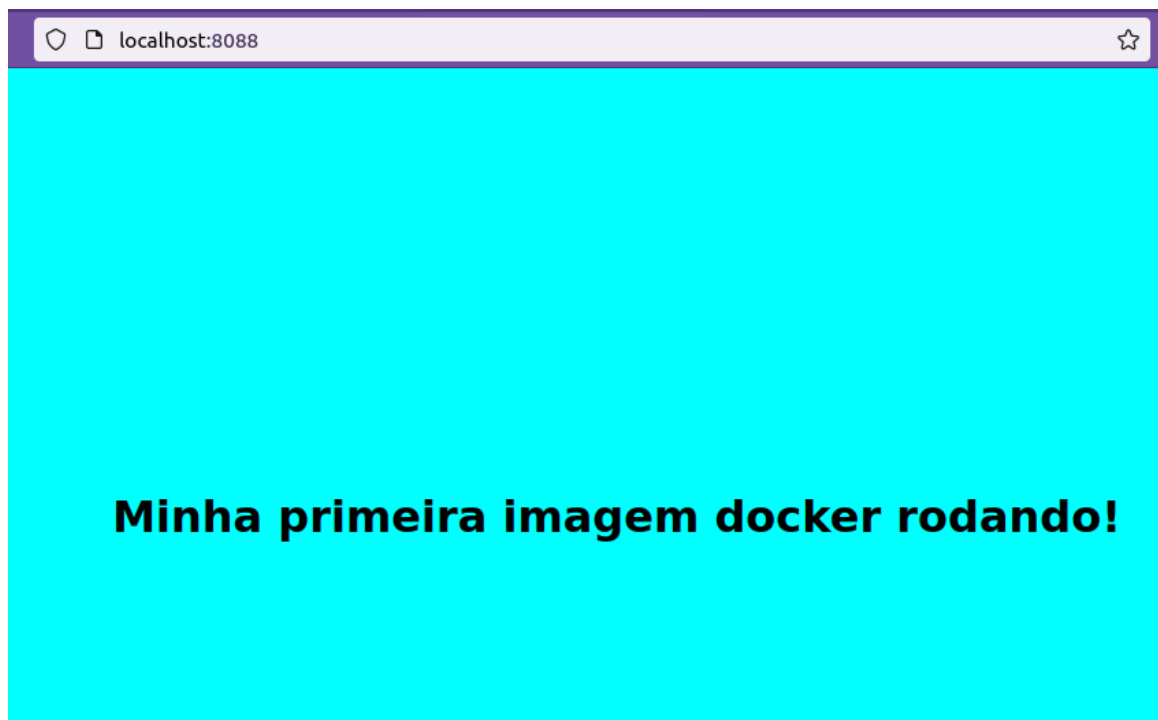
```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Modulo01$ docker run -dp 8088:3000 --name my-first-container first-image
b9acad78182af64537d413c7948416aa84a395ba6da3808c02307afe6a6a0d44
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Modulo01$
```

Para confirmarmos que o nosso container está em execução, podemos utilizar o comando “**docker ps**”.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Modulo01$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
b9acad78182a   first-image    "docker-entrypoint.s..." 43 seconds ago Up 41 seconds 0.0.0.0:8088->3000/tcp,
Container
```

Na tabela criada, temos acesso a informações importantes, como o ID do container, há quanto tempo ele está UP e o mapeamento das portas.

Vamos no navegador e acessar o endereço <http://localhost:8088>



Nota-se que o container está em execução e nossa aplicação está disponível a partir do container na porta indicada. Isso mostra o potencial do docker para desenvolvimento e implementação de aplicações.

A próxima etapa do exercício é disponibilizar a nossa aplicação no Docker Hub. É necessário ter conta no Docker Hub, nesse caso, o nome de usuário da conta é **lets2**. Isso é importante porque disponibilizar imagens no docker hub segue o padrão **nome-de-usuario/nome-da-imagem**.

Primeira etapa é fazer o login via terminal, usando o comando “**docker login**”:

```
ker$ docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/letonio/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

A próxima etapa é modificar a tag que atribuímos a nossa imagem, para que ela siga o padrão esperado. Portanto, utiliza-se o comando:

docker tag first-image lets2/first-image

Usamos o comando **docker image ls** para verificar se a mudança foi um sucesso:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-c
ker$ docker tag first-image lets2/first-image
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-c
ker$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
lets2/first-image	latest	0f3fff89387b	15 minutes ago	177MB
first-image	latest	0f3fff89387b	15 minutes ago	177MB
my-postgres-image	latest	f72262529e65	3 days ago	379MB

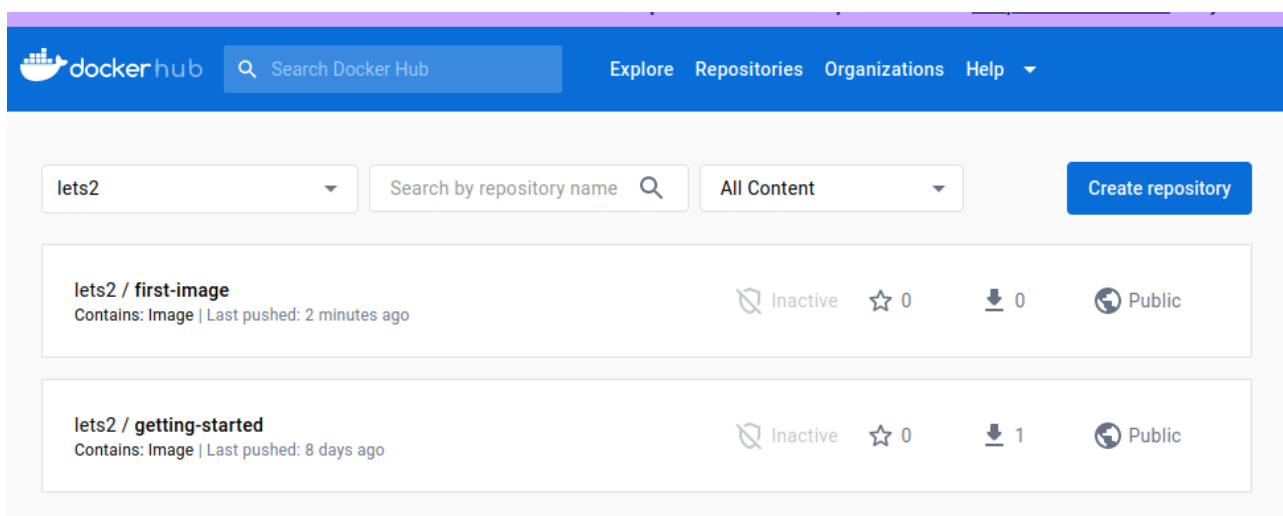
Para finalizar, vamos fazer o push para o docker hub usando o comando:

docker push lets2/first-image

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/A
ker$ docker push lets2/first-image
Using default tag: latest
The push refers to repository [docker.io/lets2/first-image]
5e225085a756: Pushing [=====] 414.2kB
00e8cb6e8da9: Pushing [=====] 2.513MB
e76f30b96e2b: Pushing 1.536kB
885a5d40fc11: Preparing
```

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-c
ker$ docker push lets2/first-image
Using default tag: latest
The push refers to repository [docker.io/lets2/first-image]
5e225085a756: Pushed
00e8cb6e8da9: Pushed
e76f30b96e2b: Pushed
885a5d40fc11: Mounted from lets2/getting-started
1b6c3782871e: Mounted from lets2/getting-started
b0e46d71a47b: Mounted from lets2/getting-started
f1417ff83b31: Mounted from lets2/getting-started
latest: digest: sha256:46297114bad249affad6e1a882d6c03c1e3acde784e693bb6f009508d6d83803 size: 1783
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-c
ker$
```

Por fim, acessando o Docker Hub é possível observar a imagem disponível:



Caso alguém deseje fazer uso dessa imagem, basta usar o comando:

docker pull lets2/first-image

Fique à vontade para utilizar a imagem que foi criada.