DOCKER - ATIVIDADE 04 - RELATÓRIO

1. OBJETIVO

Utilizando a aplicação realizada no exercício 4 da última aula, crie um exemplo prático de como subir um container com cada tipo de persistência. Para isso, siga:

- https://docs.docker.com/get-started/05_persisting_data/ para persistir os dados usando Volumes.
- https://docs.docker.com/get-started/06_bind_mounts/ para persistir os dados usando Bind Mounts.
- E descubra como fazer para persistir os dados usando tmpfs.

Você deve subir um novo container para cada tipo de persistência e mostrar o passo-a-passo necessário, de maneira que seja possível verificar quais comandos você executou e qual foi o resultado de cada um deles (prints do terminal, acesso ao webserver, etc, o que prova que o comando rodou).

Em cada container, crie tarefas na aplicação web, depois reinicie o container e mostre se as tarefas criadas ainda existem ou foram apagadas.

Finamente, no caso dos contêineres com Volume e Bind Mount, mostre onde ficaram guardados (na sua máquina) os arquivos que o container criou (as tarefas criadas na aplicação web).

Faça em forma de relatório, um documento com os comandos usados e os prints e explicações do processo. Você não precisa subir os 3 contêineres ao mesmo tempo, pode fazer um de cada vez (para não esgotar a capacidade da sua máquina).

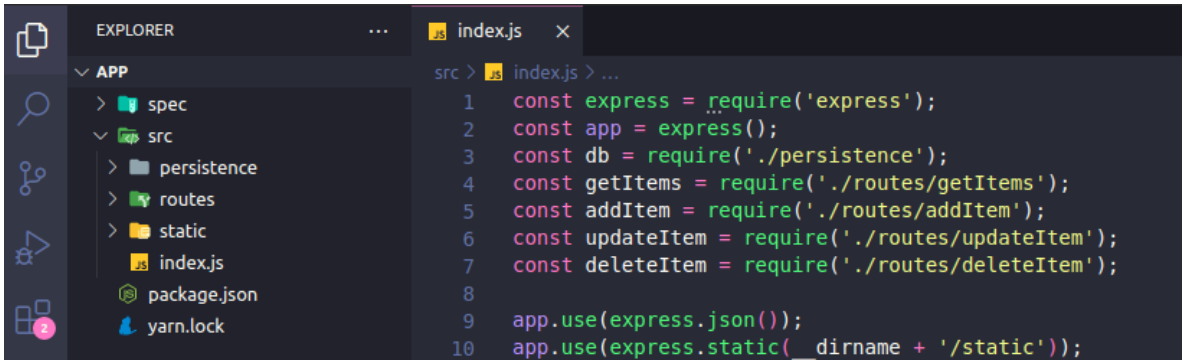
2. VOLUMES

Para fazer a persistência de dados utilizando volume, o aspirante optou por utilizar a aplicação Todo do tutorial get-started da documentação do Docker.

A aplicação do tutorial inicial do Docker está disponível via github, através desse link:

https://github.com/docker/getting-started

Para a finalidade dessa atividade, o conteúdo do subdiretório app é suficente. Portanto, esse subdiretório foi copiado para uma pasta vazia. Na imagem a seguir, nota-se a organização inicial desse conteúdo que foi baixado:



É possível observar que não há um arquivo Dockerfile, portante, faz-se necessário criá-lo e adicionar nele as instruções necessárias para a construção da imagem Docker.

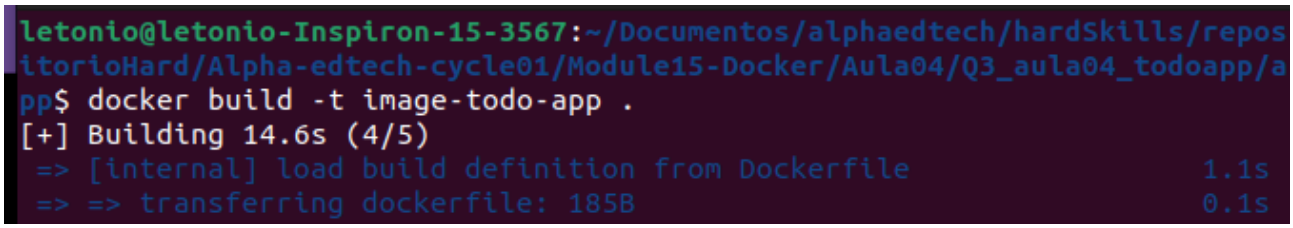
O arquivo Dockerfile foi criado na raiz do projeto e recebeu as instruções apresentadas a seguir:

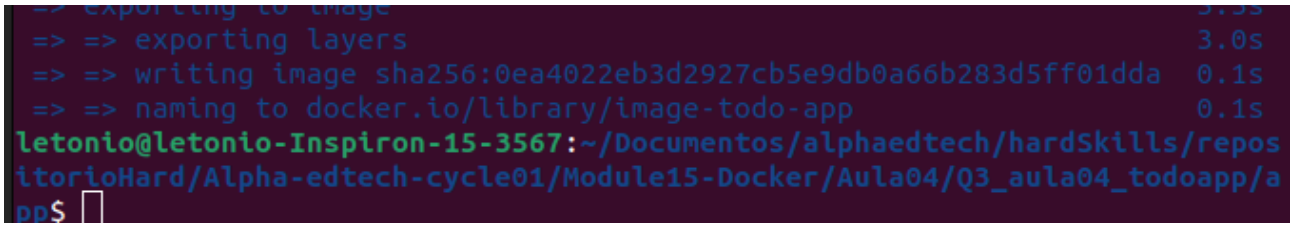
```
# syntax=docker/dockerfile:1

FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

O próximo passo é construir a imagem da aplicação, através do comando a seguir:

docker build -t image-todo-app .

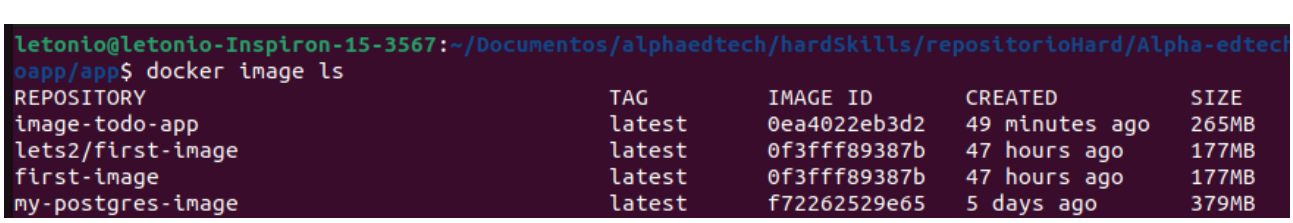
Explicação: Trata-se de um comando para criar uma imagem, a tag -t serve para rotular a imagem que será criada, permitindo ao usuário manipular a imagem usando esse nome no lugar do id. Por fim, o ponto indica que deve ser procurado um Dockerfile no diretório atual do terminal.





A imagem foi criada, o que pode ser verificado atrávés do comando a seguir:

docker image ls



A próxima etapa é criar um volume que será utilizado para fazer a persistência de dados de um database SQLite que faz parte da aplicação. Além disso, é importante recordar que essa aplicação é uma Todo List, portanto, os dados de interesse em persistir são as tarefas que o usuário adicionar. Com a persistência, reiniciar o container não acarreta na perda dos dados anteriores. A criação de um volume é realizada através do comando:

docker volume create todo-db

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills
/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula04/Q3_aula
04_todoapp/app$ docker volume create todo-db
todo-db
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills
```

A próxima etapa é criar um container que faz uso desse volume recém-criado para fazer a persistência de dados. O comando utilizado é o seguinte:

docker run -dp 3002:3000 --mount type=volume,src=todo-db,target=/etc/todos image-todo-app

Explicação: A flag combinada -dp indica que será executado em background (detached), liberando o uso do terminal, além de mapear a porta 3002 do host (máquina local) com a porta 3000 do container. Como consequência, quando o container estiver ativo, poderemos acessar o serviço que está dentro do container usando a porta 3002 para acessar a aplicação. A flag –mount configura um ponto de montagem dentro do container, de forma que o tipo de montagem é “volume”, src=todo-db indica o nome do volume que será usado. Por fim, target indica o local dentro do container onde o volume será montado.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills
/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula04/Q3_aula
04_todoapp/app$ docker run -dp 3002:3000 --mount type=volume,src=tod
o-db,target=/etc/todos image-todo-app
d5d731b186aeba034793f5a0e538f744bc6808b8440ff7168ddbb561a2956ed0
```

Conforme foi explicado nas questões anteriores, ao usar o tipo volume, o docker fica responsável por gerir dentro da máquina local. Então, podemos usar o seguinte comando para determinar o local onde os dados são armazenados quando usamos um volume:

docker volume inspect todo-db

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Al
oapp/app$ docker volume inspect todo-db
[
    {
        "CreatedAt": "2023-05-07T19:58:59-03:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/snap/docker/common/var-lib-docker/volumes/todo-db/_data",
        "Name": "todo-db",
        "Options": null,
        "Scope": "local"
    }
]
```

Na figura acima, Mountpoint corresponde ao local onde os dados são guardados de verdade. Nota-se que é um local que requer permissão de administrador.

Outra maneira de visualizar essas informações é determinar o id do container e aplicar o comando “inspect” a esse container. O comando docker ps lista todos os container em execução.

docker ps

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module
oapp/app$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS
d5d731b186ae   image-todo-app  "docker-entrypoint.s…"   12 minutes ago   Up 12 minutes   0.0.0.0:3002->3000/tcp
nes
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module
```

Podemos usar a parte inicial do id, portanto, o próximo comando a ser executado é o seguinte:

docker inspect d5d7

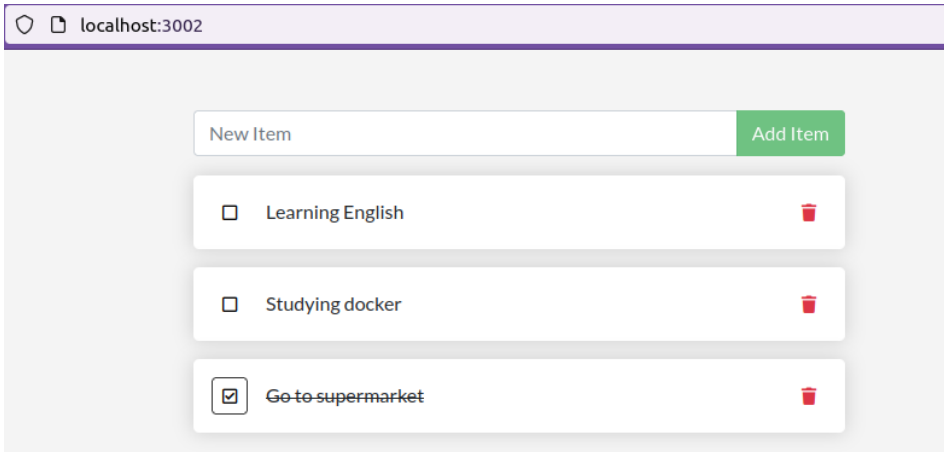
```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Al
oapp/app$ docker inspect d5d7
[
    {
        "Id": "d5d731b186aeba034793f5a0e538f744bc6808b8440ff7168ddbb561a2956ed0",
        "Created": "2023-05-17T23:08:34.710040963Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "node",
```

Esse comando retorna informações detalhadas sobre o container, distribuídas no formato JSON. Entre essa informações há diversos metadados, incluindo um campo Mounts, conforme a imagem a seguir:

```
            "Name": "overlay2"
        },
        "Mounts": [
            {
                "Type": "volume",
                "Name": "todo-db",
                "Source": "/var/snap/docker/common/var-lib-docker/volumes/todo-db/_data",
                "Destination": "/etc/todos",
                "Driver": "local",
                "Mode": "z",
                "RW": true,
                "Propagation": ""
            }
        ],
        "Config": {
```

Caso nenhuma forma de persistência de dados fosse utilizada, esse campo teria um array vazio.

Podemos acessar o serviço que está sendo fornecido pelo container através do endereço http://localhost:3002. Foram adicionadas algumas tarefas, conforme ilustrado na imagem a seguir.



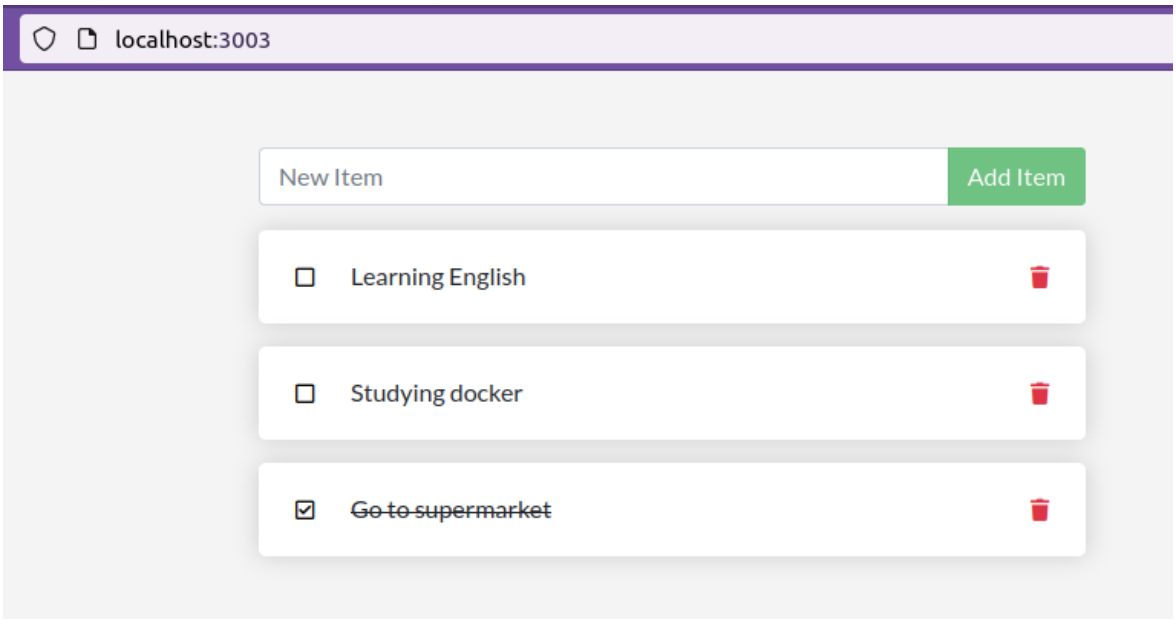
Para mostrar a persistência em ação, o container atual que está em execução será parado e excluído. Posteriormente, será empregado o mesmo comando para a criação e execução de um novo container. Os comandos a seguir foram executados em sequência:

docker rm -f d5d7
docker run -dp 3003:3000 --mount type=volume,src=todo-db,target=/etc/todos image-todo-app

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cyc
oapp/app$ docker rm -f d5d7
d5d7
```

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cyc
oapp/app$ docker run -dp 3003:3000 --mount type=volume,src=todo-db,target=/etc/todos image-todo-app
057a91eacaeafcccaf633664c8a9a8e1f77d67051d24ae345b36d1b152bcb957
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cyc
oapp/app$
```

Acessando o novo endereço http://localhost:3003, nota-se que as tarefas que foram adicionadas no container anterior permanecem as mesmas, mesmo sendo um novo container e um novo endereço.



3. BIND MOUNTS

Para o próximo tipo de persistência, utilizou-se como base os arquivos que estão dentro de /app. Conforme já foi explicado nas questões anteriores, o bind mounts permite estabelecer uma vínculo entre um diretório no container e um diretório no host (máquina local). Com a conexão estabelecida, qualquer arquivo modificado ou adicionado no diretório do host aplicará a mesma alteração/adição no container e vice-versa. Uma utilidade que vem a mente é quando estamos no processo de desenvolvimento do código e estamos fazendo mudanças. A documentação sugerida mostra a combinação desse conceito de bind mounts com o nodemon para trabalhar com a aplicação em modo de desenvolvimento e ver as modificações no servidor ativo que está dentro do container.

Não é necessário adicionar o nodemon porque ele já está presente no arquivo package.json. Caso não estivesse, seria necessário adicionar como dependência de desenvolvimento. A imagem a seguir mostra parte do arquivo package.json:

```
  },
  "devDependencies": {
    "jest": "^29.3.1",
    "nodemon": "^2.0.20",
    "prettier": "^2.7.1"
  }
}
```

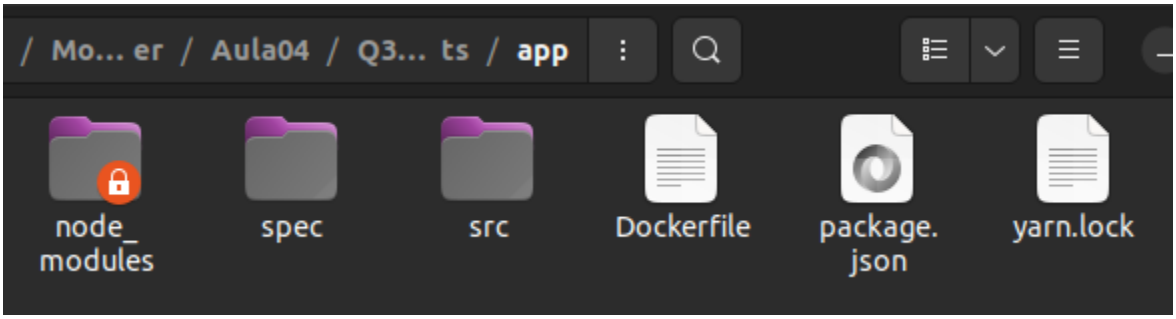
Com o terminal aberto na raiz do projeto, executou-se o seguinte comando:

docker run -dp 3004:3000 -w /app --mount type=bind,src="$(pwd)",target=/app node:18-alpine sh -c "yarn install && yarn run dev"

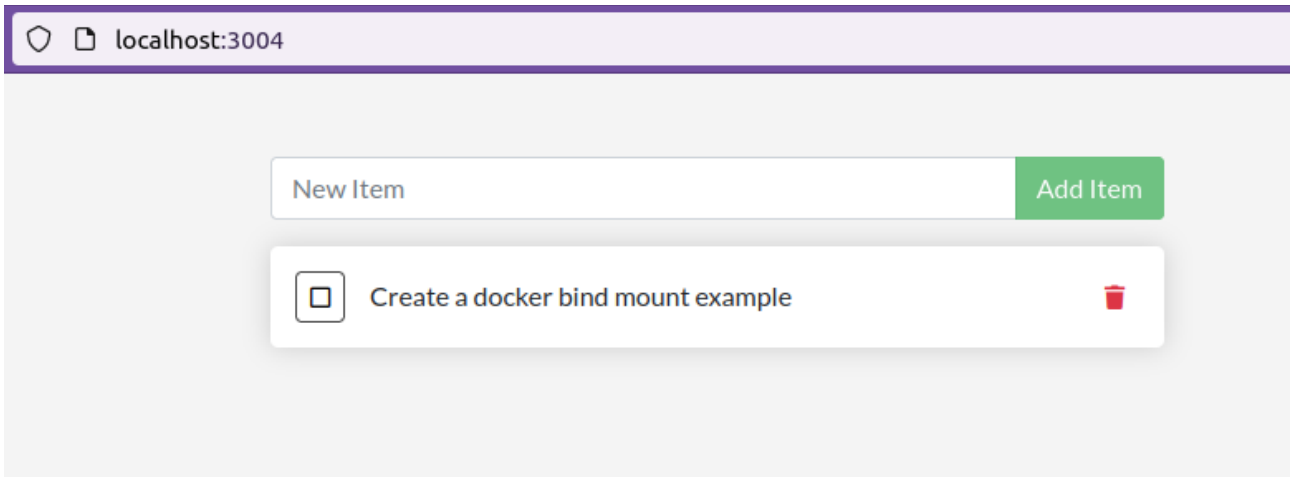
Explicação: Vai iniciar um novo container docker. A flag -dp executa o container em segundo plano e mapeia a porta 3004 do host para a 3000 do container. A flag -w define o diretório de trabalho (WORKDIR) dentro do container, portanto, comandos serão executados a partir desse diretório. O trecho --mount type=bind,src="$(pwd)",target=/app cria um ponto de montagem tipo bind mounts, estabelendo um vínculo entre o diretório atual (comando pwd) no host (máquina local) e o diretório /app dentro do container. O trecho node:18-alpine define a imagem e versão usada como base para a criação do container. Por fim, o trecho sh -c "yarn install && yarn run dev" executa dentro do container, a partir de um shell, um comando que instala as dependências de projeto (as de desenvolvimento, inclusive) e na sequência chama um script que executa o projeto em modo de desenvolvimento. No package.json esse script estabele que o nodemon executará o arquivo contido em /src/index.js dentro do container. O Nodemon é uma ferramenta que ajuda no desenvolvimento de aplicativos Node.js reiniciando automaticamente o servidor toda vez que detecta uma alteração nos arquivos do projeto. Portanto, a combinação do nodemon com o bind mounts permitirá o seguinte: sempre que adicionarmos ou alterarmos algo no diretório do host, a alteração será refletida dentro do container e, consequentemente, o nodemon que está ativo no container, notará a mudança e o servidor dentro do container será reiniciado automaticamente.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula04/Q3_aula04_todoapp_2bindmounts/a
pp$ docker run -dp 3004:3000 -w /app --mount type=bind,src="$(pwd)",target=/app
node:18-alpine sh -c "yarn install && yarn run dev"
38eca09e23591a9a9707c121bfdc47f79903ffe4d7bc9c7ddfc6f1d9b95727d2
```

O primeiro exemplo de conexão que é estabelecida entre container e host é demonstrado na imagem a seguir. Note que, apesar do comando yarn install ter sido aplicado ao container, o sistema de pastas do host também recebeu um diretório node_modules.



Acessando o endereço http://localhost:3004, podemos observar a aplicação em execução em modo de desenvolvimento e com um canal estabelecido.

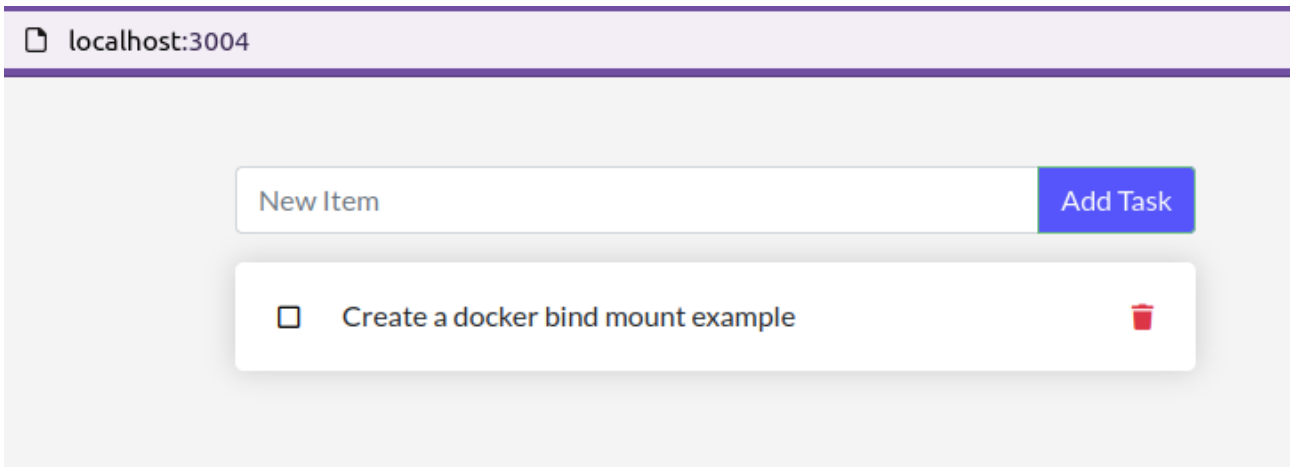


Podemos confirmar que o bind mounts está presente nesse container através do comando que dá detalhes sobre os container, a saber:

docker inspect 38ec

Pode ser difícil de enxergar, mas nota-se um campo Target (/app) e um campo Source (caminho absoluto até a posição atual do host usado como referência). Não menos importante, o “type”: “bind”, autoexplicativo.

Vamos ver o vínculo na prática, podemos modificar a cor e nome do botão pelo VS Code e, ao salvar a alteração, o servidor que está rodando dentro do container sofrerá essa mesma alteração e o servidor reiniciado. Basta usar o F5, que a página mostrará a versão atualizada:



Nota-se o botão azul e o nome modificado. O tutorial Docker recomenda a utilização de logs para verificar possíveis alterações, através do comando a seguir:

docker logs -f 38ec (id do container)

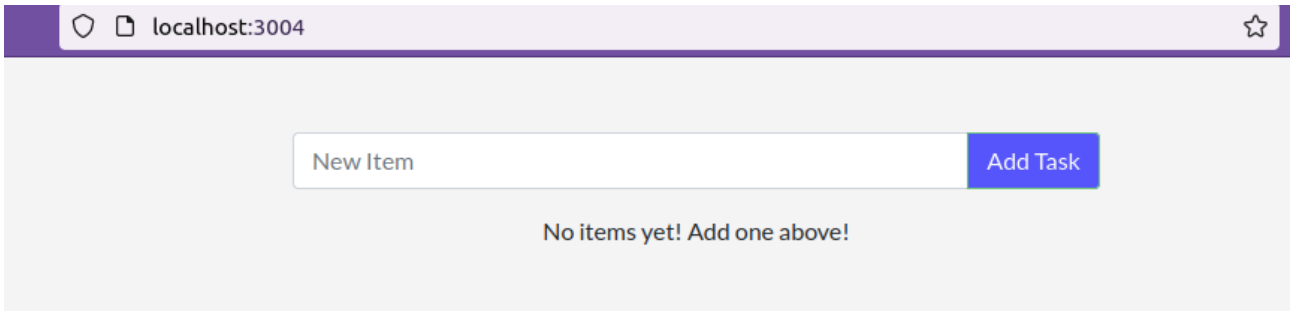
```
_brattain
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositor
oapp_2bindmounts/app$ docker logs -f 38ec
yarn install v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
Done in 38.51s.
yarn run v1.22.19
$ nodemon src/index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
Using sqlite database at /etc/todos/todo.db
Listening on port 3000
[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
Using sqlite database at /etc/todos/todo.db
Listening on port 3000
```

Apesar de apenas ter sido feito uma modificação, se adicionássemos um arquivo na pasta, esse arquivo apareceria dentro do container. O contrário também é possível, mudanças e adições no container afetam a máquina local.

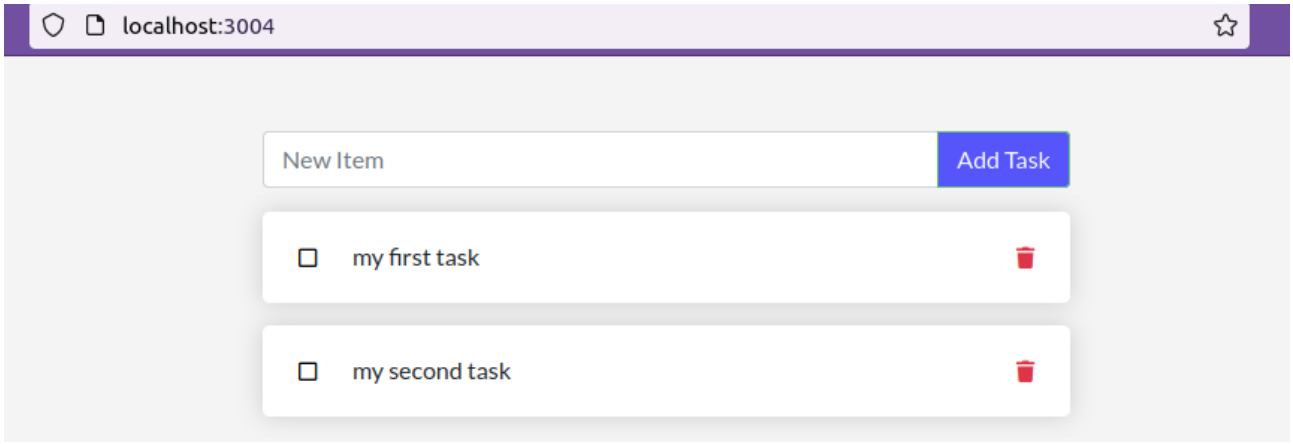
Vamos remover o container e executar o comando novamente para verificar se os dados das tarefas ficam salvos.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtec
oapp_2bindmounts/app$ docker rm -f 38ec
38ec
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtec
oapp_2bindmounts/app$ docker run -dp 3004:3000 -w /app --mount type=bind,src="$(pwd)",target=/ap
un dev"
d20201c247adff184bfa4f316f2848a2a483f641fca6c3493526d03043079034
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtec
```

Acessando mais uma vez o endereço http://localhost:3004, nota-se que a tarefa que havia sido colocada antes não foi salva:



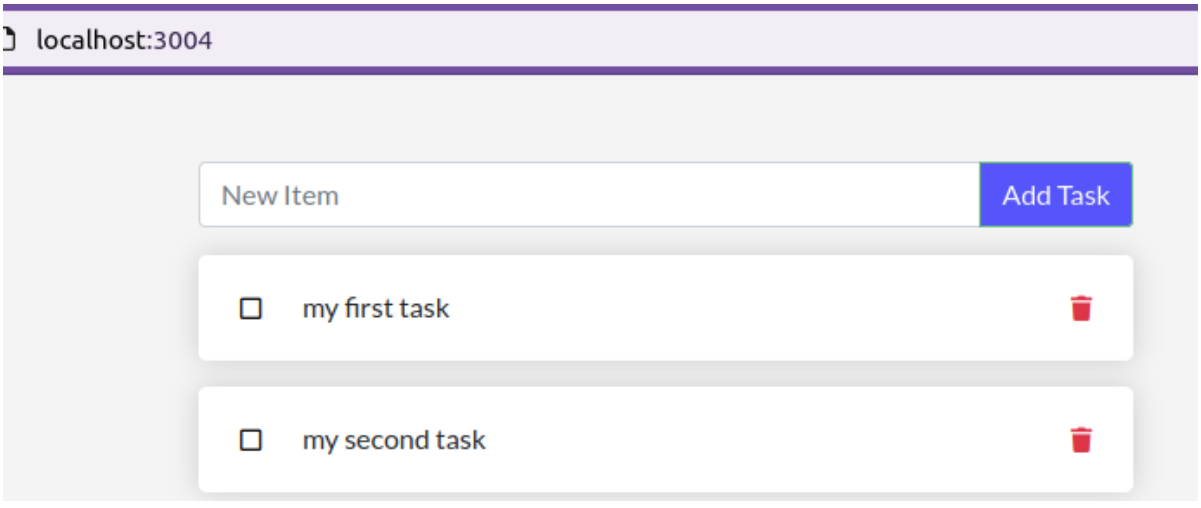
Mais um teste foi realizado. Foram adicionadas 3 tarefas, conforme a imagem a seguir.



E em seguida, o container foi parado (docker stop) e foi iniciado novamente.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
oapp_2bindmounts/app$ docker stop d202
d202
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
oapp_2bindmounts/app$ docker start d202
d202
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
```

Acessando o endereço http://localhost:3004, nota-se que as tarefas permanecem lá, mas isso não está relacionado ao tipo de persistência, mas sim ao próprio container e o armazenamento do SQLite dessa aplicação.



4. TMPFS

Para finalizar, o objetivo é utilizar o tipo de persistência tmpfs. Nesse caso, a persistência é feita com arquivos temporário na memória ram, em vez de usar o sistema de arquivos do host ou volumes persistentes. É útil para casos em que desejamos armazenamento rápido, por exemplo, para operações temporárias ou para melhorar o desempenho de operações intensivas de leitura e gravação.

Considerando a imagem image-todo-app que foi criada no início. Vamos criar e iniciar um container passando o tipo de persistência tmpfs:

docker run -dp 3006:3000 --mount type=tmpfs,destination=/app image-todo-app

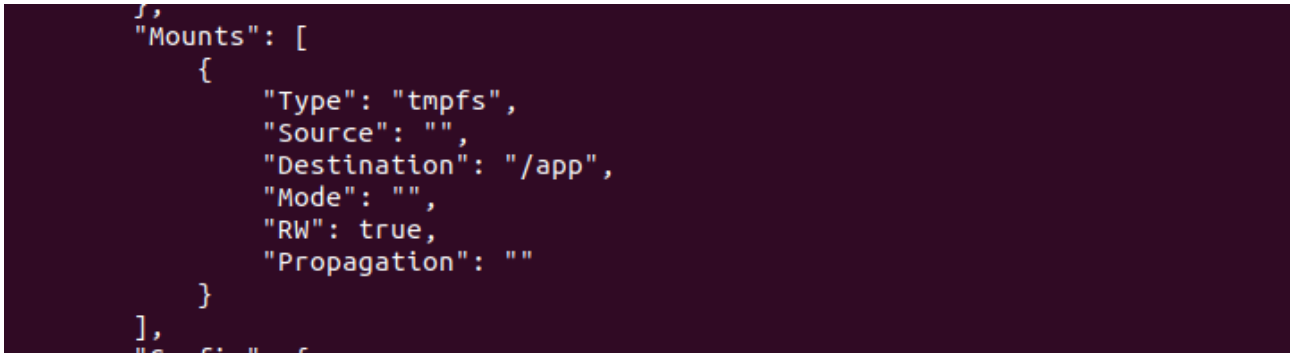
Opcionalmente, podemos empregar a sintaxe:

docker run -dp 3006:3000 --tmpfs /app image-todo-app

Explicação: Executa um container mapeando a porta 3006 do host com a 3000 do container. No caso do tipo tmpfs, temos apenas uma destino (/app), não existe fonte.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula04/Q3_aula04_todoapp_3tmpfs/app$ d
ocker run -dp 3006:3000 --mount type=tmpfs,destination=/app image-todo-app
b0290dad26c76ec56f2b15f58474eab7b23b577b550e87be945ade5904812a0b
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module15-Docker/Aula04/Q3_aula04_todoapp_3tmpfs/app$
```

Para confirmar, podemos usar o comando “docker inspect b029”, onde b029 é o início do id do container que está em execução.



O que finaliza essa atividade.