

Objetivos:

- I. Bind mounts;
- II. Espelhamento usando bind mounts.

Atenção: Para reproduzir os exemplos e exercícios, utilize o seguinte repositório:
<https://github.com/arleysouza/bind-mount>.

Veja o vídeo se tiver dúvidas - <https://youtu.be/WRLRfqs6-BU>

I. Bind mounts

No Docker, *volumes* e *bind mounts* são dois mecanismos para persistência e compartilhamento de dados entre o host e os containers. Embora ambos sirvam a esse propósito, eles se diferenciam quanto à forma de gerenciamento e aplicação.

Bind mount é uma técnica que permite vincular diretamente um diretório (ou arquivo) existente no sistema de arquivos do host a um caminho interno do container. Em termos práticos, trata-se de um espelhamento em tempo real: qualquer alteração feita no diretório local é automaticamente refletida dentro do container, sem necessidade de reconstrução da imagem.

Exemplo de funcionamento

Considere uma aplicação Node.js localizada em `./server`. Ao executar um container com essa pasta montada via bind mount, as alterações no código fonte local serão imediatamente refletidas dentro do container. Isso é extremamente útil durante o desenvolvimento, pois agiliza o ciclo de testes e validações.

Diferenças entre *bind mount* e volume

Característica	Bind mount	Volume
Gerenciado pelo Docker	Não (depende do caminho no host)	Sim (via <code>docker volume</code> ou <code>docker compose</code>)
Portabilidade	Baixa (depende da estrutura do host)	Alta (dados ficam isolados do host)
Visibilidade fora do Docker	Total (são pastas locais)	Limita (geralmente em <code>/var/lib/docker/volumes</code>)
Casos de uso recomendados	Desenvolvimento, hot reload	Produção, bancos de dados, logs persistentes

Atenção: Por conceder acesso direto ao sistema de arquivos do host, os *bind mounts* devem ser utilizados com cautela, especialmente em ambientes de produção. Um erro de configuração pode expor ou sobrescrever arquivos críticos.

II. Espelhamento usando bind mounts

Como exemplo será usado o mesmo projeto da aula anterior. A estrutura de diretórios utilizada é a seguinte:

```
app/
├── db/
│   └── init.sql
├── front-um/
│   ├── Dockerfile
│   └── index.html
├── front-dois/
│   ├── public/
│   ├── src/
│   ├── Dockerfile
│   ├── package-lock.json
│   ├── package.json
│   ├── tsconfig.json
│   └── vite.config.ts
├── server/
│   ├── dist/
│   ├── src/
│   ├── Dockerfile
│   ├── nodemon.json
│   ├── package-lock.json
│   ├── package.json
│   └── tsconfig.json
└── compose.yml
```

O espelhamento ocorre por meio da diretiva `volumes` nos serviços do `compose.yml`. A seguir, apresentam-se os mapeamentos realizados:

Bind mounts no compose.yml:

Serviço frontum:

`volumes:`

- `./front-um:/usr/share/nginx/html`

Serviço frontdois:

`volumes:`

- `./front-dois:/app`
- `/app/node_modules`

```
# Serviço server:
volumes:
  - ./server/dist:/app/dist
```

Cada mapeamento conecta uma pasta do host (em amarelo) a um caminho no container (em verde), com finalidades distintas.

Estratégias de espelhamento aplicadas

1. Espelhamento de arquivos estáticos

O projeto localizado em `front-um/` consiste em arquivos estáticos (HTML, CSS, JS), que não exigem compilação.

- Diretiva: `./front-um:/usr/share/nginx/html`
- Justificativa: O caminho `/usr/share/nginx/html` é o diretório padrão de publicação do Nginx, conforme configurado no `Dockerfile`.

Assim, alterações feitas localmente em `front-um/` são refletidas diretamente no container (`um-container`) e podem ser observadas no navegador, sem necessidade de reconstrução da imagem.

2. Compilação externa ao container (pré-compilação no host)

Para o projeto localizado em `server/`, a estratégia adotada é compilar o TypeScript no host e compartilhar somente a pasta `dist/` com o container.

- Diretiva: `./server/dist:/app/dist`
- Após editar o código em `server/src/`, deve-se executar:

```
npm run build
```

Isso atualizará a pasta `dist/` no host, a qual está espelhada no container.

- O `Dockerfile` utilizado é o seguinte:

```
FROM node:20-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install
RUN npm install -g nodemon

COPY dist ./dist
COPY nodemon.json .

CMD ["nodemon"]
```

- A arquivo `nodemon.json`:

```
{
  "watch": ["dist"],
  "ext": "js",
```

```
"exec": "node dist/index.js",
"legacyWatch": true,
"verbose": true
}
```

O Nodemon monitora a pasta `dist/`, garantindo o reinício automático da aplicação sempre que o build for recompilado no host.

3. Compilação interna ao container (modo desenvolvimento com hot reload)

O projeto `front-dois/` (React + TypeScript com Vite) se beneficia do espelhamento em tempo real e da compilação dentro do container. Essa abordagem é ideal para projetos que exigem hot reload durante o desenvolvimento.

- Configuração do `vite.config.ts`:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    host: '0.0.0.0',
    port: 5173,
    watch: {
      usePolling: true,
    }
  }
})
```

- Dockerfile:

```
FROM node:20-alpine

WORKDIR /app # Define diretório de trabalho

COPY package*.json ./

RUN npm install

EXPOSE 5173 # Porta configurada no vite.config.ts

CMD ["npm", "run", "dev"] # Modo de desenvolvimento
```

- Trecho do `compose.yml`:

```
frontdois:
  build:
    context: ./front-dois
    dockerfile: Dockerfile
  container_name: dois-container
  ports:
```

```
- "3003:5173"
networks:
- minha-rede
depends_on:
- server
volumes:
- ./front-dois:/app
- /app/node_modules
```

Observações:

- `./front-dois:/app`: espelha a pasta do projeto do host no container, permitindo que alterações feitas em tempo real sejam refletidas imediatamente;
- `/app/node_modules`: impede que a pasta `node_modules` do host (geralmente incompatível com Alpine Linux) sobrescreva a do container.

Conclusão

As estratégias de *bind mount* demonstradas permitem acelerar o ciclo de desenvolvimento e teste, adaptando-se a diferentes necessidades:

- Projetos estáticos: mapeamento direto;
- Projetos compilados: build externo com sincronização de pasta compilada;
- Hot reload: compilação no container com espelhamento total do projeto.

O uso consciente e adequado dos *bind mounts* garante agilidade sem comprometer a segurança e a organização do ambiente de desenvolvimento.