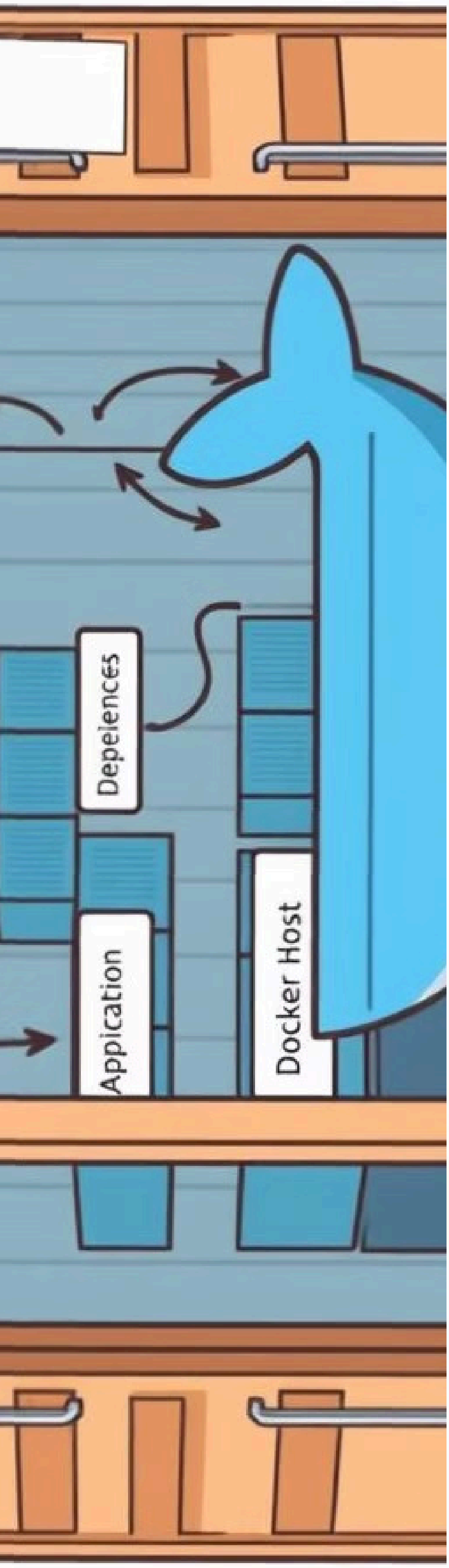

Roteiro para Aula: Docker - Volumes na Prática

Este documento apresenta um roteiro detalhado para uma aula sobre Docker, com foco em volumes e persistência de dados. A aula abrange desde conceitos básicos até práticas avançadas de gerenciamento de volumes, incluindo exemplos práticos e uma tarefa para os alunos.

por Everton Coimbra de Araújo

Introdução ao Docker e Conceitos Iniciais

Docker é uma plataforma que permite criar, distribuir e executar aplicações em containers. Containers são ambientes isolados que contêm tudo que é necessário para executar um software, proporcionando uma forma de garantir que o software rode da mesma maneira em qualquer lugar. Você pode imaginar containers como "mini-máquinas" que rodam sua aplicação com todas as dependências necessárias.

O foco da nossa aula será trabalhar com volumes. Volumes são um componente crítico no Docker para armazenar dados de forma persistente. Eles são a solução que o Docker oferece para garantir que os dados não se percam quando um container é removido ou recriado.

Docker e Persistência de Dados

Na prática, você pode imaginar volumes como "pastas" que estão fora do ciclo de vida do container, garantindo que os dados possam ser usados por outros containers e existam independente de um container específico.

Durante a execução de containers, é comum precisar armazenar dados gerados pela aplicação. Se não houver uma solução de persistência, ao remover um container, todos os dados armazenados também são perdidos. Aqui entram os volumes. Eles são áreas especiais do sistema de arquivos que são montadas dentro de containers e permitem armazenar dados que sobreviverão à recriação dos containers.

Tipos de Volumes no Docker

Volumes Anônimos

Criados automaticamente pelo Docker, geralmente sem que o usuário precise intervir diretamente.

Volumes Nomeados

Volumes que recebem um nome para facilitar seu gerenciamento.

Bind Mounts

Utilizados para mapear uma pasta específica do sistema hospedeiro para o container, oferecendo mais controle sobre onde os dados estão fisicamente armazenados.

Vamos discutir quando faz sentido utilizar cada um desses volumes e suas diferenças. Por exemplo, volumes anônimos são muito utilizados em containers temporários, enquanto bind mounts são comuns em ambientes de desenvolvimento, onde queremos alterar arquivos no hospedeiro e refletir diretamente no container.

Exemplo Prático: Nginx e Volumes

Vamos colocar em prática o exemplo do **nginx**. Primeiro, criaremos um container **nginx** sem volume e faremos uma alteração no arquivo **index.html** para entender a falta de persistência de dados.

Abra o terminal e execute o seguinte comando para criar um container **nginx**:

```
$ docker run -d --name nginx_container -p 8082:80 nginx
```

Com esse comando, estamos criando um container a partir da imagem **nginx** e mapeando a porta **8082** do sistema hospedeiro para a porta **80** do container.

Agora, vamos acessar o container e modificar o arquivo **index.html**. Primeiro, precisamos navegar até o diretório onde o arquivo está localizado e usar um editor de texto. No nosso caso, utilizaremos o **vim**. No entanto, o **vim** pode não estar instalado no container por padrão, então vamos instalá-lo primeiro.

Acesse o container:

```
$ docker exec -it nginx_container bash
```

Instale o **vim** (é necessário que o container tenha acesso à internet para isso):

```
# apt-get update && apt-get install -y vim
```

Após a instalação, navegue até o diretório onde está o arquivo **index.html**:

```
# cd /usr/share/nginx/html
```

Agora, edite o arquivo **index.html** com o **vim**:

```
# vim index.html
```

No **vim**, pressione **i** para entrar no modo de inserção e altere o conteúdo para::

```
<h1>Alteração no index.html</h1>
```

Para salvar e sair do **vim**, siga os seguintes passos:

1. Pressione **Esc** para sair do modo de inserção.
2. Digite **:wq** e pressione **Enter** para salvar e sair.

Podemos sair do container digitando **exit**. Abra o navegador e acesse **http://localhost:8082** para ver a mensagem "**Alteração no index.html**". Agora, vamos remover o container e recriá-lo.

```
$ docker rm -f nginx_container
$ docker run -d --name nginx_container -p 8082:80 nginx
```

Acesse novamente **http://localhost:8082** e você verá que a alteração foi perdida, pois o container foi recriado sem persistir os dados.

Aqui, uma alternativa ao uso do **vim** é utilizar o comando **echo** para alterar diretamente o conteúdo do arquivo **index.html**. Isso é útil quando você deseja uma modificação rápida e direta, sem a necessidade de um editor de texto.

Por exemplo, para alterar o arquivo **index.html**, execute o seguinte comando dentro do container:

```
# echo "Alteração rápida com echo" > /usr/share/nginx/html/index.html
```

Este comando irá substituir todo o conteúdo do arquivo **index.html** pela mensagem "**Alteração rápida com echo**". Diferente do **vim**, que permite edição parcial do conteúdo, o echo sobrescreve tudo, então deve ser usado com cuidado.

Se compararmos com o conteúdo anterior, que foi editado com **vim**, a principal diferença aqui é a simplicidade e rapidez do **echo**, mas ele não permite edições complexas ou manutenção do conteúdo existente. Se precisarmos apenas inserir uma nova linha, poderíamos usar **>>** ao invés de **>** para adicionar ao conteúdo existente:

```
# echo "Nova linha adicionada com echo" >> /usr/share/nginx/html/index.html
```

Este comando adiciona uma nova linha ao final do arquivo, sem remover o conteúdo já presente.

Mapeando Volumes Diretamente na Linha de Comando

Antes de criarmos volumes nomeados, vamos começar mapeando volumes diretamente na linha de comando. Esse método é prático e útil para situações rápidas em que desejamos garantir a persistência dos dados sem criar volumes explicitamente. Podemos fazer isso utilizando o parâmetro **-v** ao criar um container.

Por exemplo, vamos criar um container **nginx** e mapear um volume diretamente da seguinte forma:

```
$ docker run -d --name nginx_container -p 8082:80 -v $(pwd)/html:/usr/share/nginx/html nginx
```

Neste comando, estamos mapeando o diretório **html** do sistema hospedeiro (que deve existir previamente) para o diretório **/usr/share/nginx/html** dentro do container. Dessa forma, qualquer alteração que fizermos no diretório **html** do nosso computador será refletida diretamente no container.

Agora, acesse o diretório mapeado e crie um arquivo **index.html:**

```
$ echo "Conteúdo mapeado diretamente do sistema hospedeiro" > html/index.html
```

Acesse **http://localhost:8082** e veja o conteúdo renderizado diretamente do volume mapeado. Se fizermos qualquer alteração no arquivo **index.html** no sistema hospedeiro, a mudança será refletida imediatamente no container, sem a necessidade de qualquer modificação adicional no Docker.

Utilizando Volumes para Persistir Dados

Agora, vamos utilizar volumes nomeados para garantir a persistência da alteração no arquivo **index.html**. Diferente do mapeamento direto que vimos anteriormente, volumes nomeados oferecem uma solução mais organizada e replicável, sendo especialmente úteis em ambientes de produção, onde a consistência dos dados é essencial.

Crie um volume nomeado:

```
$ docker volume create nginx_volume
```

Em seguida, crie um container **nginx** associando o volume criado:

```
$ docker run -d --name nginx_container -p 8080:80 -v nginx_volume:/usr/share/nginx/html nginx
```

Acesse o container e faça a mesma modificação no arquivo index.html:

```
$ docker exec -it nginx_container bash
# apt-get update && apt-get install -y vim
# cd /usr/share/nginx/html # vim index.html
```

No **vim**, pressione **i** para entrar no modo de inserção e altere o conteúdo para:

> <h1>Alteração Persistente no index.html</h1>

Para salvar e sair do **vim**, pressione **Esc** e depois digite **:wq** seguido de **Enter**.

Bind Mounts na Prática

Agora vamos trabalhar com bind mounts, uma abordagem que permite mapear uma pasta do sistema hospedeiro diretamente para um container. Diferente dos volumes nomeados, que são gerenciados pelo Docker e mais indicados para ambientes de produção, os bind mounts proporcionam um controle mais direto, pois vinculam uma pasta específica do hospedeiro ao container. Essa abordagem é muito útil durante o desenvolvimento, onde as alterações feitas no hospedeiro precisam ser imediatamente refletidas no container, proporcionando agilidade e eficiência no teste e ajuste do código.

Há duas formas principais de utilizar bind mounts: com a opção **-v** ou com a opção **--mount**. A seguir, vamos explorar ambas.

1

Usando a opção -v

Este método é mais compacto e geralmente é usado em situações mais simples ou rápidas. Por exemplo:

```
$ docker run -d --name nginx_container -p 8082:80 -v $(pwd)/html:/usr/share/nginx/html nginx
```

Neste comando, estamos mapeando o diretório html do sistema hospedeiro (que deve existir previamente) para o diretório **/usr/share/nginx/html** dentro do container. Dessa forma, qualquer alteração que fizermos no diretório html do nosso computador será refletida diretamente no container.

2

Usando a opção --mount

Esta abordagem oferece uma sintaxe mais explícita e flexível, sendo útil quando há necessidade de múltiplas opções ou quando se quer mais clareza. Por exemplo:

```
$ docker run -d --name nginx_container -p 8082:80 --mount type=bind,source=$(pwd)/html,target=/usr/share/nginx/html nginx
```

Aqui, usamos `type=bind` para especificar que estamos utilizando um bind mount, seguido de `source=`, que indica o caminho do sistema hospedeiro, e `target=`, que define o destino no container. Essa forma pode ser mais fácil de entender e gerenciar quando se lida com configurações mais complexas.

Exemplos Comparativos

Em termos de escolha, a recomendação é utilizar **-v** para situações de desenvolvimento ou experimentação rápida, pois a sintaxe é mais curta e direta. Já o **--mount** deve ser preferido em ambientes onde a clareza e a precisão são fundamentais, como em ambientes de produção ou quando se trabalha com equipes, pois facilita o entendimento e o gerenciamento das configurações.

- A opção **-v** é mais concisa, o que a torna ideal para comandos rápidos e simples durante o desenvolvimento.
- A opção **--mount** é mais detalhada e explícita, o que é vantajoso quando precisamos configurar múltiplas propriedades, como permissões ou pontos de montagem mais específicos.

Podemos refletir sobre as diferenças e quando usar volumes ou bind mounts. Volumes são mais flexíveis e mais fáceis de gerenciar, sendo ideais para ambientes de produção, enquanto bind mounts são úteis em ambientes de desenvolvimento, onde você quer ter acesso direto aos arquivos no sistema hospedeiro.

Gerenciando Volumes e Containers

Para finalizar, é importante aprender a gerenciar volumes e containers, incluindo listar, obter informações e remover volumes.

Para listar todos os volumes existentes no Docker, podemos utilizar o comando:

```
$ docker volume ls
```

Este comando mostrará uma lista de todos os volumes criados, incluindo volumes anônimos e nomeados.

Para obter mais detalhes sobre um volume específico, como o caminho onde ele está armazenado no sistema hospedeiro, use o comando:

```
$ docker volume inspect meu_volume
```

O comando **inspect** retorna informações detalhadas, como o ponto de montagem, o driver utilizado e os containers que estão usando o volume.

Para remover um volume, podemos utilizar o comando:

```
$ docker volume rm meu_volume
```

É sempre bom lembrar que, ao remover um volume, todos os dados contidos nele serão apagados, por isso, é essencial garantir que não existem dados importantes antes de executar esse comando.

Conclusão

Encerrando nossa aula, podemos perceber que os volumes são fundamentais para manter a persistência dos dados nos containers Docker. Eles oferecem uma forma segura e eficiente de trabalhar com dados, principalmente em aplicações que precisam ser escaláveis e resilientes. O uso de bind mounts em desenvolvimento também proporciona agilidade na modificação de arquivos sem necessidade de rebuild constante.

FAQ - "Não Existem Perguntas Idiotas"

Por que precisamos de volumes no Docker?

Volumes garantem que os dados permaneçam mesmo após o container ser removido ou recriado, o que é essencial para aplicações que geram ou manipulam informações que não podem ser perdidas.

Quando utilizar bind mounts ao invés de volumes?

Bind mounts são ideais quando você precisa de acesso direto aos arquivos do sistema hospedeiro, principalmente durante o desenvolvimento.

É seguro usar volumes em produção?

Sim, volumes são seguros e recomendados para produção, especialmente porque o Docker os gerencia de forma eficiente, facilitando backups e migrações.

Tarefa Prática

Para consolidar tudo o que aprendemos sobre Docker e volumes, vamos realizar uma tarefa prática onde você deve aplicar os conceitos trabalhados:

Crie o Diretório

Comece criando um diretório chamado **docker_pratica** no seu sistema.

Execute o Nginx

Use o comando **docker run** para criar um container Nginx e mapeie o diretório **html** para dentro do container.

1

2

3

Adicione o HTML

Dentro desse diretório, crie um subdiretório **html** e um arquivo **index.html** com o conteúdo inicial.

1

Edite o Arquivo

Acesse o container e atualize o arquivo **index.html** adicionando uma nova linha.

2

Crie um Volume

Remova o container e recrie-o usando um volume nomeado **nginx_data**.

3

Verifique a Persistência

Verifique se a alteração feita anteriormente foi preservada.

Explore os Volumes	Limpe os Volumes
Liste todos os volumes disponíveis e inspecione o **nginx_data**.	Para remover um volume, use o comando apropriado.

Respostas Comentadas

1. Para criar o diretório e o subdiretório, utilize os seguintes comandos:

```
$ mkdir -p docker_pratica/html
$ echo "<h1>Este é o conteúdo inicial</h1>" > docker_pratica/html/index.html
```

Esse passo prepara o ambiente local que será utilizado no container, garantindo que tenhamos um arquivo inicial.

2. Para criar o container nginx utilizando bind mounts, utilize o comando

```
$ docker run -d --name nginx_container -p 8082:80 -v $(pwd)/docker_pratica/html:/usr/share/nginx/html nginx
```

Esse comando cria um container nginx mapeando o diretório **html** do sistema hospedeiro. Assim, qualquer alteração feita localmente será refletida no container.

3. Acesse o container e edite o arquivo index.html:

```
$ docker exec -it nginx_container bash
# echo "<p>Alteração feita no container</p>" >> /usr/share/nginx/html/index.html
```

Aqui, usamos o **echo** para adicionar uma nova linha ao arquivo **index.html** dentro do container. Essa alteração será temporária, pois estamos utilizando bind mounts.

4. Remova o container e recrie-o utilizando um volume nomeado:

```
$ docker rm -f nginx_container $ docker volume create nginx_data
$ docker run -d --name nginx_container -p 8082:80 -v nginx_data:/usr/share/nginx/html nginx
```

Ao remover o container e recriá-lo utilizando um volume nomeado, garantimos que os dados persistam, mesmo após a remoção do container.

5. Acesse o container novamente para verificar o conteúdo do index.html:

```
$ docker exec -it nginx_container bash
# cat /usr/share/nginx/html/index.html
```

Você perceberá que o conteúdo anterior não foi preservado, pois o volume nomeado não estava associado inicialmente ao container onde fizemos a alteração. Agora podemos trabalhar com persistência de dados de maneira mais organizada.

6. Liste todos os volumes disponíveis e inspecione o volume nginx_data:

```
$ docker volume ls
$ docker volume inspect nginx_data
```

Esses comandos permitem verificar todos os volumes criados e inspecionar detalhes como o ponto de montagem e quais containers estão utilizando o volume. É fundamental entender o gerenciamento de volumes para um uso eficiente do Docker.