Docker Hub e Imagens Docker: Explorando o Ecossistema Docker

Nesta aula, exploraremos o Docker Hub e o conceito de imagens Docker, entendendo como utilizar este repositório online essencial para armazenar, compartilhar e gerenciar imagens Docker. Aprenderemos a trabalhar na prática com imagens Docker, criando, gerenciando e publicando conteúdo, facilitando a colaboração entre equipes e comunidades de desenvolvimento.

por Everton Coimbra de Araújo

Introdução ao Docker Hub e Conceitos Iniciais

Nesta aula, vamos explorar o Docker Hub e o conceito de imagens Docker. O Docker Hub é um repositório online onde você pode armazenar, compartilhar e gerenciar imagens Docker. Ele faz parte essencial do ecossistema Docker e facilita a distribuição e reutilização de imagens, ajudando equipes e comunidades a colaborar de forma mais eficiente. Vamos entender como utilizá-lo e trabalhar na prática com imagens Docker, criando, gerenciando e publicando conteúdo.

Docker Hub e Seus Benefícios

O Docker Hub é mais do que apenas um repositório. Ele nos oferece uma série de funcionalidades que tornam o desenvolvimento de software mais ágil e colaborativo:

1. **Compartilhamento Fácil**
   Você pode compartilhar imagens com outras pessoas ou equipes.
2. **Acesso a Imagens Prontas**
   Disponibiliza milhares de imagens pré-configuradas, facilitando o desenvolvimento.
3. **Automatização**
   Possui integração com sistemas de CI/CD, permitindo builds automáticos.
4. **Repositórios Privados**
   Possui planos pagos que oferecem repositórios privados, garantindo maior controle e segurança.

Empresas também podem optar por configurar repositórios privados para maior segurança, seja no Docker Hub ou por meio do Docker Registry.

Imagens Docker e Sua Importância

Uma imagem Docker é um pacote imutável que inclui tudo o que uma aplicação precisa para ser executada: código, runtime, bibliotecas e dependências. Ao utilizá-las, garantimos:

Portabilidade

Aplicativos funcionam da mesma forma em qualquer lugar.

Consistência

Eliminamos problemas de "funciona na minha máquina".

Eficiência

Imagens são rápidas de construir e iniciar, facilitando o desenvolvimento e implantações.

Trabalhando com Imagens Docker

Vamos aprender a trabalhar com imagens Docker utilizando alguns comandos essenciais:

- **docker pull ubuntu**: Este comando baixa a imagem oficial do Ubuntu do Docker Hub, servindo para obter uma imagem base para criar containers ou outras imagens.
- **docker images**: Este comando lista todas as imagens Docker armazenadas localmente, permitindo que você visualize as imagens disponíveis no sistema.
- **docker rmi <image>**: Utilizado para remover uma imagem Docker específica, liberando espaço no disco.

Construindo Imagens Personalizadas com Dockerfile

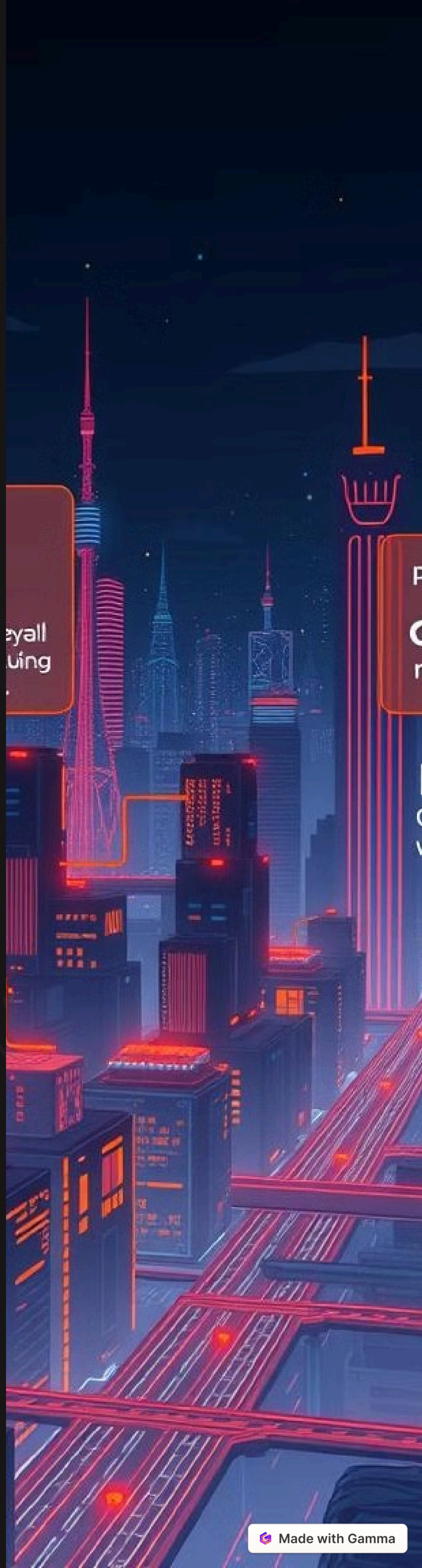
Uma imagem personalizada é importante quando precisamos adaptar um ambiente às necessidades específicas de uma aplicação. Ao criar uma imagem personalizada, podemos incluir apenas as dependências necessárias, otimizando o uso de recursos e garantindo que todas as ferramentas e configurações estejam presentes, proporcionando um ambiente controlado e consistente. Para criar imagens personalizadas, usamos um **Dockerfile**, um arquivo de texto que define cada passo para a construção de uma imagem. Vamos criar um exemplo prático onde instalamos o **Vim** em uma imagem baseada no **Nginx**:

```
# Use uma imagem base do Nginx
FROM nginx:latest

# Instale o Vim
RUN apt-get update && apt-get install -y vim

# Comando padrão para iniciar o Nginx
CMD ["nginx", "-g", "daemon off;"]
```

O comando **CMD** define o que será executado quando o container for iniciado. Neste caso, o Nginx é executado em primeiro plano para que o Docker possa gerenciar o ciclo de vida do container.



Criando e Executando uma Imagem Personalizada

Para construir a imagem personalizada a partir do Dockerfile, utilizamos o comando:

```
$ docker build -t evertoncoimbradearaujo/nginx-com-vim:latest .
```

Neste comando:

- **-t**: Nomeia e marca a imagem, facilitando o gerenciamento e a publicação.
- **evertoncoimbradearaujo/nginx-com-vim:latest**: Nome e tag atribuídos à imagem.
- **.**: Indica o diretório atual como o contexto de construção.

Depois de construir a imagem, podemos executar um container a partir dela usando o comando:

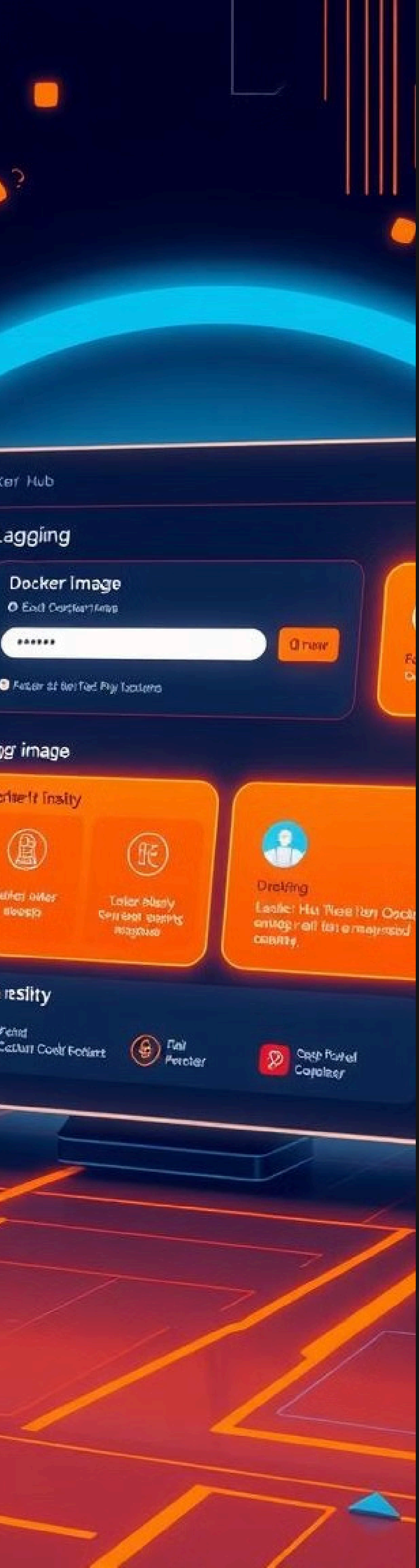
```
$ docker run -it -d -p 8082:80 --name nginx-com-vim evertoncoimbradearaujo/nginx-com-vim
```

Este comando cria um container interativo em segundo plano, mapeando a porta 80 do container para a porta 8082 do sistema hospedeiro. Assim, podemos verificar se o Nginx e o Vim estão funcionando conforme o esperado.

Para acessar o terminal de um container em execução, utilizamos:

```
$ docker exec -it nginx-com-vim /bin/bash
```

Este comando permite que você entre no container e verifique ou modifique arquivos manualmente.



Publicando uma Imagem no Docker Hub

Uma vez que criamos uma imagem, podemos publicá-la no Docker Hub para compartilhá-la com outras pessoas ou com nossa equipe:

1

Fazer login no Docker Hub

```
$ docker login
```

2

Marcar a imagem

```
$ docker tag evertoncoimbradearaujo/nginx-com-vim:latest evertoncoimbradearaujo/nginx-com-vim:latest
```

3

Enviar a imagem

```
$ docker push evertoncoimbradearaujo/nginx-com-vim:latest
```

Conclusão

Encerrando nossa aula, vimos que o Docker Hub é uma ferramenta essencial para armazenar e compartilhar imagens Docker, e que as imagens personalizadas permitem criar ambientes ajustados às necessidades específicas de nossas aplicações. Usar essas ferramentas de forma eficiente melhora a colaboração e garante consistência entre os ambientes de desenvolvimento, teste e produção.

FAQ - "Não Existem Perguntas Idiotas"

Por que precisamos do Docker Hub?

O Docker Hub permite compartilhar imagens, reutilizar soluções e colaborar de forma eficiente com outros desenvolvedores.

Qual a diferença entre uma imagem e um container?

A imagem é um modelo para criar containers. O container é uma instância em execução de uma imagem.

Quando usar repositórios privados?

Repositórios privados são recomendados para imagens que contêm informações sensíveis ou que não devem ser compartilhadas publicamente.

Tarefa Prática

Para consolidar o que aprendemos, vamos realizar uma tarefa prática:

1

Crie sua própria imagem Docker

Construa uma imagem personalizada baseada no Ubuntu, instalando o editor nano.

2

Construa e execute sua imagem

Construa a imagem usando o comando docker build e execute um container a partir dela, mapeando a porta 8083 do sistema hospedeiro para a porta 80 do container.

3

Verifique a instalação do nano

Acesse o bash do container e confirme que o nano foi instalado corretamente.

4

Publique sua imagem no Docker Hub

Publique sua imagem personalizada no Docker Hub utilizando seu nome de usuário.

5

Limpe suas imagens locais

Liste as imagens locais e remova aquelas que não são mais necessárias.

Respostas Comentadas

Para criar o Dockerfile com nano instalado, você deve usar:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y nano
CMD ["/bin/bash"]
```

Para construir a imagem:

```
$ docker build -t meuusuario/ubuntu-com-nano:latest .
```

Para executar o container:

```
$ docker run -it -d -p 8083:80 --name ubuntu-com-nano meuusuario/ubuntu-com-nano
```

Para acessar o bash do container e verificar o nano:

```
$ docker exec -it ubuntu-com-nano /bin/bash
```

Com isso, reforçamos os conceitos trabalhados e praticamos a criação, execução e publicação de imagens Docker.