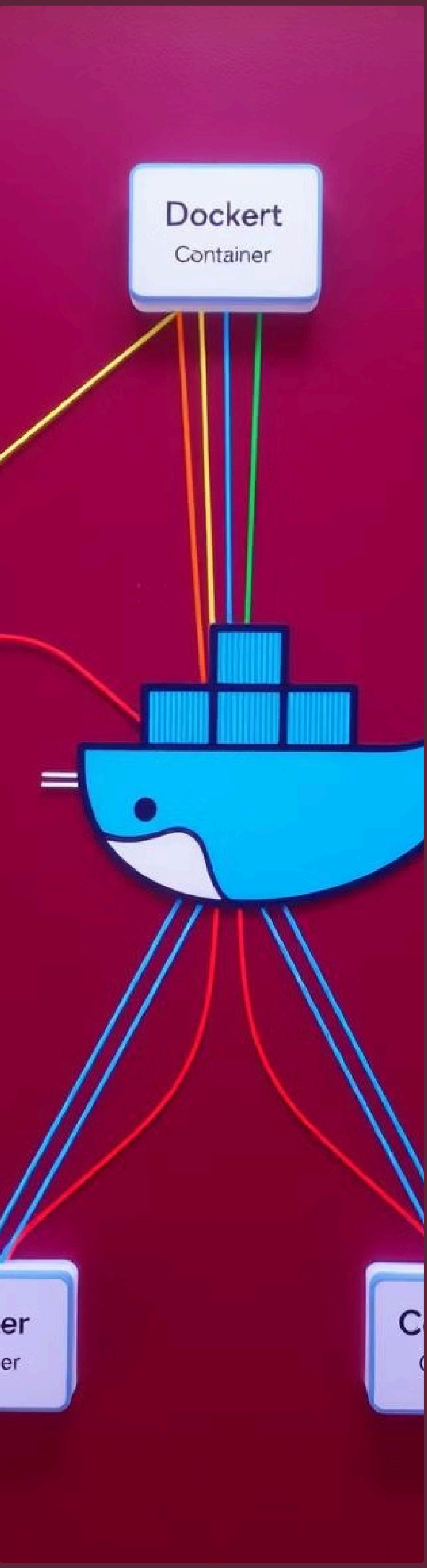

Introdução ao Uso de Networks em Docker

Este documento apresenta uma introdução abrangente ao uso de networks em Docker, cobrindo conceitos básicos, tipos de redes, comandos essenciais e exercícios práticos para entender a conectividade entre containers. O conteúdo aborda desde a definição de host no contexto do Docker até a criação e uso de redes personalizadas, fornecendo uma base sólida para o gerenciamento eficiente de redes em ambientes Docker.

por Everton Coimbra de Araújo

Introdução às Networks em Docker

Docker oferece um sistema de redes robusto e flexível que permite que containers se comuniquem entre si, com o host e com o mundo externo. Em termos simples, uma network em Docker é uma interface de comunicação que conecta containers e possibilita o tráfego de dados entre eles, garantindo que as aplicações funcionem de maneira coesa e integrada.

As networks são essenciais para a comunicação e conectividade em aplicações containerizadas, pois elas possibilitam a construção de arquiteturas de software complexas e escaláveis. Sem um sistema de redes bem definido, containers não poderiam se comunicar de forma eficiente, o que prejudicaria a funcionalidade e a performance das aplicações distribuídas.

Definição de Host no Contexto do Docker

No contexto do Docker, o termo "host" refere-se ao sistema operacional e hardware físico ou virtual em que o Docker Engine está instalado e em execução. O host é responsável por gerenciar os recursos do sistema, como CPU, memória, armazenamento e rede, e disponibilizá-los para os containers. Em outras palavras, o host é o ambiente que hospeda os containers, fornecendo a infraestrutura necessária para que eles possam ser executados.

Um exemplo prático de um host pode ser uma máquina física, como um servidor ou um computador pessoal, ou uma máquina virtual, como uma instância em uma nuvem pública (AWS, Azure, Google Cloud). O host Docker atua como um intermediário entre o sistema operacional subjacente e os containers, garantindo que cada container receba os recursos necessários de maneira isolada e segura.

Comandos Básicos de Network

Antes de explorar os diferentes tipos de redes disponíveis no Docker, é importante entender alguns comandos básicos que ajudam a gerenciar essas redes.

- **docker network**: Utilizado para gerenciar redes no Docker. Com ele, você pode criar, listar, inspecionar, remover e gerenciar redes de várias outras formas.
- **docker network ls**: Lista todas as redes Docker existentes no host. É útil para verificar quais redes estão disponíveis e entender a topologia de rede atual.
- **docker network prune**: Remove todas as redes não utilizadas no Docker. Este comando é útil para limpar redes desnecessárias e liberar recursos.

Tipos de Networks Docker

Bridge Network

Rede padrão criada pelo Docker quando nenhum parâmetro específico de rede é fornecido. Ela permite que containers em um único host se comuniquem entre si. A Bridge Network é útil para cenários onde você deseja conectar vários containers no mesmo host, permitindo que eles se comuniquem por meio de endereços IP internos ou usando o nome do container. Ela cria uma rede interna isolada do host e é adequada para ambientes de desenvolvimento ou aplicações simples que não precisam se comunicar com o mundo exterior diretamente.

Host Network

Remove a camada de isolamento de rede entre o container e o host, permitindo que o container compartilhe a rede do host. A Host Network é ideal para situações em que o desempenho da rede é essencial, já que elimina a sobrecarga de virtualização da camada de rede. É frequentemente utilizada em casos onde o container precisa acessar dispositivos de rede do host diretamente, ou em aplicações que necessitam de baixa latência. No entanto, isso também implica em menor isolamento, já que o container passa a ter acesso direto à rede do host.

Overlay Network

Usada para conectar múltiplos Docker daemons em um cluster, essencial para setups de Docker Swarm. A Overlay Network é fundamental quando se trabalha com clusters Docker, especialmente com Docker Swarm ou Kubernetes. Ela permite que containers em diferentes hosts se comuniquem como se estivessem na mesma rede local, o que é crucial para a criação de serviços distribuídos que precisam de alta disponibilidade e escalabilidade. A Overlay Network é frequentemente usada para serviços que precisam se comunicar entre diferentes máquinas em um cluster.

Macvlan Network e None Network

A Macvlan Network atribui um endereço MAC a cada container, fazendo com que ele apareça como um dispositivo físico na rede. É útil quando você deseja que seus containers sejam tratados como dispositivos físicos na rede. A None Network desativa completamente a rede do container, sendo utilizada quando você deseja isolar completamente um container, impedindo qualquer tipo de comunicação externa.

Antes de prosseguirmos, vale reforçar que, embora tenhamos todos estes tipos de redes disponíveis, nosso objetivo será trabalhar com a Bridge Network, por ser mais simples e atender bem ao nosso propósito nesta aula.

Explorando a Conectividade entre Containers em Docker

Neste exercício, vamos explorar a conectividade entre containers no Docker utilizando a rede bridge. Vamos criar dois containers, inspecionar a rede para verificar suas configurações e testar a comunicação entre eles.

Objetivo

O objetivo deste exercício é:

- Criar dois containers Docker em modo interativo.
- Inspecionar a rede bridge padrão do Docker.
- Verificar a conectividade entre os containers usando endereços IP e nomes de containers.

Procedimento

1

Crie Containers Interativos

Levante dois containers Ubuntu em modo interativo:

```
$ docker run -d -it --name ubuntu1 bash
$ docker run -d -it --name ubuntu2 bash
```

2

Inspecione a Rede Bridge

Veja os detalhes da rede Bridge padrão:

```
$ docker network inspect bridge
```

Anote os IPs dos containers ubuntu1 e ubuntu2 para o próximo passo.

3

Teste a Conectividade por IP

Acesse o container ubuntu1 e pinge o IP do ubuntu2:

```
$ docker exec -it ubuntu1 bash ping <IP_DO_UBUNTU2>
```

4

Teste a Conectividade por Nome

Agora, pinge o container ubuntu2 pelo seu nome:

```
# ping ubuntu2
```

Criação e Uso de Redes Personalizadas no Docker

Além das redes padrão que o Docker cria automaticamente, podemos criar redes personalizadas para atender a necessidades específicas, garantindo maior flexibilidade e controle sobre a comunicação entre containers.

Criando uma Rede Personalizada

Podemos criar uma rede personalizada utilizando o comando docker network create. Isso nos permite especificar o driver e outras configurações de rede.

```
$ docker network create minha_rede_personalizada
```

Neste exemplo, criamos uma rede chamada **minha_rede_personalizada** usando o driver bridge.

Conectando Containers a uma Rede Personalizada

Depois de criar uma rede, podemos conectar containers a ela, permitindo que esses containers se comuniquem de forma mais eficiente.

```
$ docker run -d -it --name ubuntu1 --network minha_rede_personalizada ubuntu
$ docker run -d -it --name ubuntu2 --network minha_rede_personalizada ubuntu
```

Com isso, os containers **ubuntu1** e **ubuntu2** estarão conectados à rede minha_rede_personalizada e poderão se comunicar utilizando o nome do container.

Adicionando um Container a uma Rede Existente

Também podemos adicionar um novo container a uma rede já existente, facilitando a expansão da infraestrutura de comunicação entre containers. Por exemplo, suponha que já tenhamos uma rede personalizada minha_rede_personalizada e queremos adicionar um novo container ubuntu3 a essa rede.

1 Criar o Novo Container

Primeiro, crie o container ubuntu3 em modo desacoplado:

```
docker run -d --name ubuntu3 ubuntu
```

2 Conectar o Novo Container à Rede Existente

Conecte o container ubuntu3 à rede minha_rede_personalizada:

```
docker network connect minha_rede_personalizada ubuntu3
```

Agora, o container ubuntu3 está conectado à rede **minha_rede_personalizada** e pode se comunicar com os outros containers na mesma rede.

Conclusão e Tarefa Prática

Nesta aula, exploramos como funcionam as redes em Docker, incluindo os tipos de networks disponíveis, como criar e utilizar redes personalizadas, e como adicionar containers a redes existentes. Compreender as diferentes opções de rede no Docker é fundamental para criar ambientes de aplicação que sejam escaláveis, eficientes e fáceis de gerenciar. As redes Docker nos permitem definir claramente como os containers devem se comunicar entre si, proporcionando um controle detalhado sobre conectividade, isolamento e performance.

FAQ - "Não Existem Perguntas Idiotas"

Por que precisamos de networks no Docker?

Networks garantem que containers possam se comunicar entre si, com o host e com o mundo externo, sendo fundamentais para aplicações distribuídas.

Qual a diferença entre Bridge e Host Network?

A Bridge Network cria uma rede isolada para os containers, enquanto a Host Network compartilha a rede do host com o container.

Tarefa Prática para os Alunos

1. **Crie uma rede personalizada chamada rede_fixa.**
2. **Inicie dois containers (container1 e container2) conectados à rede_fixa.**
3. **Adicione um terceiro container (container3) à rede rede_fixa após a sua criação.**
4. **Verifique a conectividade entre os containers usando o comando ping.**
5. **Inspecione a rede para verificar os containers conectados e suas configurações de IP.**

Respostas Comentadas

Criação da Rede Personalizada

```
$ docker network create rede_fixa
```

Este comando cria uma nova rede personalizada chamada `rede_fixa` usando o driver `bridge`, permitindo que os containers conectados a ela se comuniquem de maneira mais organizada.

Iniciar os Containers e Conectar à Rede

```
$ docker run -d --name container1 --network rede_fixa ubuntu
$ docker run -d --name container2 --network rede_fixa ubuntu
```

Estes comandos iniciam dois containers chamados `container1` e `container2` e os conectam à rede `rede_fixa`. Isso garante que ambos estejam na mesma rede e possam se comunicar.

Adicionar um Terceiro Container à Rede

```
$ docker run -d --name container3 ubuntu
$ docker network connect rede_fixa container3
```

Primeiro, criamos o container `container3` sem conectá-lo a nenhuma rede. Em seguida, usamos o comando `docker network connect` para adicionar `container3` à rede `rede_fixa`. Isso permite adicionar containers a redes existentes mesmo após sua criação.

Verificar a Conectividade entre os Containers

```
$ docker exec -it container1 ping container2
$ docker exec -it container1 ping container3
```

Utilizamos o comando `ping` para testar a conectividade entre `container1` e os outros containers (`container2` e `container3`). Se a configuração da rede estiver correta, o ping deve responder sem problemas, indicando que os containers conseguem se comunicar.

Inspecionar a Rede

```
$ docker network inspect rede_fixa
```

O comando `docker network inspect` fornece detalhes sobre a rede `rede_fixa`, incluindo os containers conectados e seus endereços IP. Essa inspeção é útil para verificar a topologia da rede e garantir que todos os containers estejam corretamente conectados.