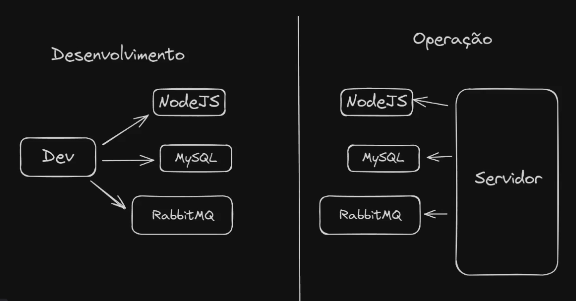
**Docker – Dia 1 – 25/08/2025**

**Introdução**

****

**O que é um container e suas vantagens**

### O que é um Container?

Um **container** é um pacote de software leve e autossuficiente que inclui tudo o que um aplicativo precisa para funcionar: o código, as bibliotecas, as dependências e as configurações. Pense nele como uma "caixa" padronizada que encapsula um aplicativo e garante que ele rode de forma idêntica, independentemente do ambiente.

Diferente de uma máquina virtual (VM), o container não precisa de um sistema operacional completo para cada aplicação. Em vez disso, ele compartilha o kernel do sistema operacional do host, o que o torna muito mais leve, rápido para iniciar e eficiente em termos de recursos.

**Vantagens:**

O uso de containers no desenvolvimento de software oferece benefícios significativos:

* **Portabilidade:** A maior vantagem. Um container criado em um computador de desenvolvedor pode ser movido e executado em qualquer outro ambiente (servidor de teste, produção, nuvem, etc.) com a garantia de que funcionará exatamente da mesma forma. Isso resolve o clássico problema de "na minha máquina funciona".
* **Eficiência e Velocidade:** Por serem leves, os containers consomem menos recursos (CPU, memória) e iniciam em segundos, em vez de minutos como as máquinas virtuais. Isso permite rodar mais aplicações em um único servidor.
* **Consistência:** Eles criam um ambiente padronizado em todas as etapas de desenvolvimento, do início ao fim. Isso elimina inconsistências e facilita a colaboração entre equipes.
* **Isolamento:** Cada container é isolado dos outros e do sistema host. Se um container falhar, ele não afeta os outros, o que melhora a segurança e a estabilidade da aplicação.
* **Escalabilidade:** É muito mais fácil e rápido criar novas instâncias de um container para lidar com o aumento de tráfego, por exemplo. Ferramentas como o Kubernetes automatizam esse processo, escalando a aplicação de forma horizontal.

**Desvantagens:**

* **Menos isolamento:** O fato de compartilharem o mesmo kernel pode gerar riscos de segurança se ele for comprometido.
* **Dependência de SO:** Todos os contêineres em um hospedeiro devem ser compatíveis com o sistema operacional desse hospedeiro.

**VM vs Containers**

### **Máquinas Virtuais (VMs)**

Uma VM é como um computador completo dentro de outro. Ela tem seu próprio sistema operacional (SO) "convidado" e aloca seus próprios recursos de hardware, como CPU, memória e armazenamento. Essa virtualização é feita por um software chamado **hypervisor**, que gerencia as VMs e os recursos do hardware hospedeiro.

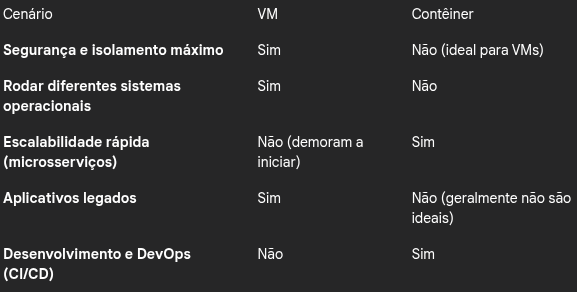
**Vantagens:**

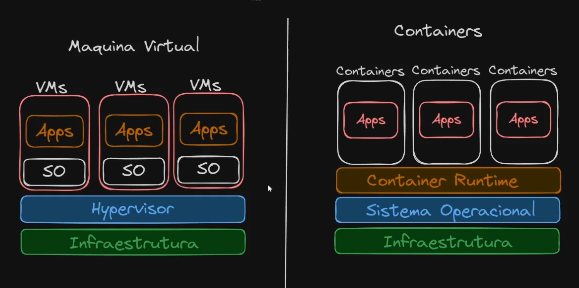
* **Isolamento total:** Cada VM é totalmente isolada, o que oferece alta segurança. Se uma VM for comprometida, as outras não são afetadas.
* **Flexibilidade:** Permite executar diferentes sistemas operacionais (Windows, Linux, macOS) na mesma máquina hospedeira.

**Desvantagens:**

* **Pesadas:** Por incluírem um SO completo, as VMs consomem mais recursos e demoram mais para inicializar.

### **Quando usar cada um?**

****

****

**Como funcionam os containers?**

Containers são como pacotes de software que reúnem tudo o que uma aplicação precisa para rodar: o código, as bibliotecas e as dependências. Pense neles como pequenas "máquinas virtuais", mas muito mais leves.

Em vez de virtualizar o sistema operacional inteiro, o container compartilha o sistema operacional do computador hospedeiro. Isso o torna **rápido de iniciar** e muito **eficiente no uso de recursos**.

Eles garantem que a aplicação funcione da mesma forma em qualquer ambiente, seja no computador de um desenvolvedor, em um servidor de testes ou em produção, resolvendo o problema clássico de "na minha máquina funciona!".

**Dia 2 - 25/08/2025**

**Antes de falarmos sobre containers, vamos falar de Linux**

O kernel Linux, o "coração" do sistema operacional, é fundamental para o conceito de containers. Ele permite o isolamento de processos e recursos através de duas tecnologias principais: **namespaces** e **cgroups**.

* **Namespaces:** Essa tecnologia cria ambientes isolados para cada container, dando a cada um sua própria visão de recursos do sistema, como processos, rede, usuários e sistema de arquivos. É como se cada container estivesse em uma caixa separada, sem saber o que as outras caixas contêm.
* **cgroups (control groups):** Já o cgroups controla a quantidade de recursos que um container pode usar. Com ele, você pode definir limites de uso de CPU, memória, I/O de disco e rede para evitar que um container consuma todos os recursos disponíveis e afete o desempenho de outros containers no mesmo sistema.

Juntos, namespaces e cgroups permitem que os containers funcionem de forma isolada, mas compartilhando o mesmo kernel do sistema host, o que os torna muito mais leves e eficientes do que máquinas virtuais.

**Container sem Docker (chroot, namespaces e cgroups)**

### **Chroot**

O **chroot** é a base. Ele altera o diretório raiz para um programa em execução. O processo não pode acessar arquivos ou diretórios fora do novo "root" (sistema de arquivos confinado), o que proporciona um isolamento básico.

**Namespaces**

Os **namespaces** criam cópias isoladas de recursos do sistema. Assim, um processo tem sua própria visão de:

* **PID (Process ID):** Cada contêiner tem seus próprios PIDs, isolados do sistema principal.
* **Rede:** Endereços IP, rotas e regras de firewall são específicos do contêiner.
* **Usuários:** IDs de usuário e grupo são mapeados para o sistema principal.
* **Montagem de Sistema de Arquivos:** Cada contêiner tem sua própria visão do sistema de arquivos.

Isso é o que dá a um contêiner a sensação de ser uma máquina virtual isolada, mas sem o overhead de virtualização.

### **Cgroups (Control Groups)**

Os **cgroups** limitam o uso de recursos de um processo ou grupo de processos. Eles controlam e isolam o uso de:

* **CPU:** Define quanto de CPU um contêiner pode usar.
* **Memória:** Limita a quantidade de RAM disponível.
* **Entrada/Saída (I/O):** Controla o acesso ao disco.

Esses três componentes juntos fornecem o isolamento de um contêiner. O Docker não reinventou a roda; ele empacotou, automatizou e simplificou o uso dessas ferramentas já existentes no Linux para criar a experiência de contêineres que conhecemos hoje.

O comando **sudo unshare --pid --fork --mount-proc /bin/bash** cria um novo ambiente de shell com isolamento de processo.

Em resumo, ele faz três coisas principais:

* **Isolamento de PID (--pid):** Cria um novo espaço de nomes de processo. Isso faz com que o novo shell e seus processos filhos vejam a si mesmos como **PID 1** e não consigam "ver" nem interagir com os processos do sistema principal.
* **Isolamento de proc (--mount-proc):** Monta um novo sistema de arquivos **/proc** que contém apenas os processos dentro desse novo espaço isolado.
* **Execução em segundo plano (--fork):** Executa o shell (**/bin/bash)** no novo espaço de nomes.

Essencialmente, o comando permite que você execute comandos num ambiente isolado, sem o risco de afetar o sistema principal. É uma forma simples de fazer "contêineres" temporários e leves.

O que **debootstrap** faz?

O comando debootstrap faz o download dos pacotes essenciais do repositório Debian e os extrai em um diretório especificado, criando uma base funcional de um sistema Debian.

A sintaxe correta é:

**sudo debootstrap [opções] <versão> <diretório> [<http://url/do/repositorio>]**

* **versão:** A versão do Debian que você quer instalar, como stable, bookworm, bullseye, etc.
* **diretório:** O caminho para o diretório onde o novo sistema será instalado.

**Exemplo:**

Para criar um sistema Debian 12 (Bookworm) no diretório /mnt/debian, o comando seria:

sudo debootstrap bookworm /mnt/debian

**Isolamento processo, filesystem e recurso:**

### **1. mkdir container e cd container**

Estes dois comandos criam e entram em um novo diretório chamado container. Este diretório servirá como a raiz do seu novo sistema Debian. É dentro dele que todos os arquivos do sistema isolado serão armazenados.

### **2. cat /etc/os-release**

Este comando exibe a versão do sistema operacional que você está usando **atualmente**, no seu sistema principal (o "host"). Ele está aqui apenas para que você possa verificar e comparar com a versão que será instalada no contêiner.

### **3. debootstrap bookworm .**

Este é o comando central. Ele usa o debootstrap para baixar e instalar o conjunto de pacotes básicos de um sistema Debian.

* **bookworm:** Especifica a versão do Debian a ser instalada.
* **.:** O ponto (.) representa o diretório atual, que é o container. Isso significa que todos os arquivos do novo sistema Debian serão instalados neste diretório.

No final desse comando, o diretório container terá uma estrutura de diretórios completa (/bin, /etc, /usr, etc.) e os pacotes essenciais para um sistema Debian 12 (Boo**kworm).**

### **4. unshare --pid --fork --mount-proc chroot . /bin/bash**

Este é o comando mais complexo e crucial para o isolamento. Ele cria um ambiente virtual temporário e isolado.

* **unshare:** É a ferramenta principal que cria os "namespaces", ou seja, os espaços de nomes isolados.
* **--pid:** Cria um novo espaço de nomes para IDs de processo. Dentro do novo ambiente, o primeiro processo iniciado terá o PID 1 e não verá os processos do sistema principal.
* **--fork:** Executa o comando em segundo plano, criando um novo processo para o ambiente isolado.
* **--mount-proc:** Monta um sistema de arquivos /proc específico e isolado para este novo ambiente, que mostrará apenas os processos dentro dele.
* **chroot . /bin/bash:** Muda a raiz do sistema (chroot) para o diretório atual (., que é o container) e executa o shell bash. Isso significa que, a partir de agora, o shell rodará dentro do novo ambiente isolado, e os comandos que você digitar afetarão apenas os arquivos e processos dentro do container.

### **5. mount -t proc proc /proc**

Após entrar no ambiente isolado, este comando é necessário para que ferramentas como ps e top funcionem corretamente. Ele monta o sistema de arquivos /proc dentro do novo ambiente, permitindo que os processos dentro do contêiner sejam inspecionados. Sem isso, o ambiente não funcionaria como esperado.

**6. echo "+cpu">> /sys/fs/cgroup/cgroup.subtree\_control e echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree\_control**

Habilitam os controladores de **CPU** e **CPUset** no cgroup atual.

* **echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree\_control:** Habilita o **controlador de CPU**. Isso permite que você limite e monitore o uso da CPU por processos nos cgroups filhos. Por exemplo, é possível definir uma porcentagem máxima de uso de CPU para um grupo de processos, garantindo que eles não monopolizem o processador.
* **echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree\_control:** Habilita o **controlador de CPUset**. Isso permite que você atribua um conjunto específico de núcleos de CPU e nós de memória a um cgroup. É útil para isolar processos em núcleos dedicados, o que pode melhorar o desempenho ao evitar a disputa por recursos de hardware entre diferentes grupos de trabalho.

Em resumo, esses comandos preparam o diretório cgroup atual para que você possa criar subgrupos e, em seguida, aplicar regras de uso de CPU e de alocação de núcleos/memória para eles. Eles são passos iniciais para configurar o gerenciamento de recursos.

Essa sequência de comandos configura e usa um **cgroup**  para isolar um processo específico em um núcleo de CPU. Essencialmente, é uma forma manual e básica de "conteinerização" para gerenciar recursos do sistema.

### **Passo a Passo da Sequência**

1. **cd /sys/fs/cgroup e ls:** Navega até o diretório raiz do **cgroup v2** (a versão mais moderna) e lista seu conteúdo. É aqui que os cgroups são gerenciados.
2. **mkdir auladocker:** Cria um novo diretório, que será nosso novo cgroup filho, chamado auladocker. Este é o grupo onde os processos serão colocados para ter seus recursos limitados.
3. **cd auladocker:** Entra no novo cgroup para configurar suas regras.
4. **echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree\_control:** Habilita o controlador de **CPU** no cgroup auladocker. Isso significa que agora é possível definir limites de uso de CPU para os processos neste grupo.
5. **echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree\_control:** Habilita o controlador de **CPUset**. Isso permite que você restrinja os processos do auladocker a um subconjunto de núcleos de CPU.
6. **htop:** Executa o monitor de processos htop para encontrar o PID de um processo que você deseja isolar. O número 139592 na próxima etapa provavelmente é o PID de algum processo que você viu no htop.
7. **echo "139592" >> /sys/fs/cgroup/auladocker/cgroup.procs:** Este comando **move o processo** com o PID 139592 para dentro do cgroup auladocker. Agora, este processo está sob o controle das regras que você vai definir.
8. **echo "0" >> /sys/fs/cgroup/auladocker/cpuset.cpus:** Esta é a etapa de isolamento. O comando diz ao controlador cpuset para restringir todos os processos dentro do cgroup auladocker a usar apenas a **CPU 0**. Isso significa que o processo 139592 agora só pode ser executado nesse núcleo específico, não podendo migrar para outros.
9. **htop:** Executa o htop novamente para visualizar o efeito. Você pode notar que o processo 139592 está agora "preso" na CPU 0, e a carga de trabalho dele não aparecerá em outros núcleos.