Habilitando Aplicações Nativas de Nuvem

Introdução a Contêineres e Kubernetes

2. Aplicativos Modernos e Contêineres

Sergio Rio
www.sergiorio.tech

Refrescando nossa memória

- Na aulas passadas estudamos a arquitetura de virtualização do Linux KVM, o Windows for Linux Subsystem (WSL), origem e fundamentos dos contêineres, seus benefícios, comparação com maquinas virtuais e seus componentes e arquitetura.
- Atividades preparatórias para a aula de hoje: assistir vídeos com tutoriais de números 1 ao 7 do Docker Tutorial:Introdução (baixar e instalar)
  1. Imagens, contêineres e servidor Docker
  2. Comandos básicos
  3. Listar, criar e executar contêineres
  4. Visualizar logs, suspender e abortar execução.
  5. Remover e inspecionar contêineres.
  6. Executar um comando dentro de um contêiner em execução.

Dúvidas, questões ou comentários?

Programa: Introdução a Contêineres e Kubernetes

1. Conceitos Básicos

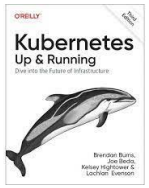
- ✓ Abstrações em Ciência da Computação
- ✓ Virtualização de Computadores
- ✓ MicroVMs e Unikernels

2. Contêineres

- Origem
- Fundamentos
- ➢ Criação e execução
- ➢ Registro e reuso
- Infraestrutura como Código
- Aplicativos Modernos

3. Kubernetes

- Origem
- Arquitetura
- Pods
- Abstrações de Recursos
- Descoberta de Serviços
- Serviços de Rede
- Instalação e administração básica
- Implantação de um caso de uso (exemplo)



Kubernetes Up & Running
Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson
Cortesia da VMware Inc.

Contêineres

Criação, execução, registro e reuso

"Os sonhos parecem reais enquanto estamos neles. Só quando acordamos é que percebemos que algo era realmente estranho."

Dom Cobb (Leonardo DiCaprio), "Inception", 2010

Os três pilares do Linux Containers

1. CGroups
2. Namespaces
3. Unionfs

Cgroups[1] – Control Groups 1/2

- Recurso do kernel Linux que isola, limita e monitora o uso de recursos (CPU, memória, E/S, rede, etc.) de um grupo de processos.
- Disponível a partir da versão 2.6.24 do kernel Linux.
- Estabelece cotas e limites para recursos como por exemplo 3 GB de memória e 70% de CPU e associa um *pid* ao *cgroup*.
- Supressão de consumo de recursos:
  - Em cenários normais uma aplicação pode exceder os limites definidos (capacidade disponível).
  - Na ausência de capacidade o consumo é suprimido conforme limites definidos.
- Utiliza uma estrutura de árvore para organizar e controlar os diferentes recursos
  - Cada processo pertence a um e somente um cgroup.
  - Os processos são organizados em tempo de execução como nós filhos dos processos que os criaram.

1 - Control Group v2

Cgroups[1] – Control Groups 2/2

Controlador	Descrição suscinta
cpu	Assegura um mínimo de compartilhamento de CPU quando sistema está ocupado mas limita o seu consumo quando houver capacidade disponível.
cpuset	Contabiliza o consumo de CPU por grupos de processos.
freezer	Suspense e restabelece todos os processos de um mesmo *cgroup*.
hugetlb	Limita do consumo de páginas de memorias virtuais muito grandes (*tlb, translation lookaside buffer*)
io	Limita o acesso e o consumo de recursos em blocos controlando (suprimindo) os nós intermediários e folha na arvore de hierarquia do recurso (disco interno por exemplo).
memory	Limita e controla (suprime) o consumo de memória de processos, memória do kernel, e virtual swap usada pelo *cgroups*.
perf_event	Permite a monitoração de desempenho de processos agrupados em um *cgroup*.
pids	Limita o número de subprocessos (e seus descendentes) que podem ser criados por uma determinado *cgroup*.
rdma	Limita o consumo de RDMA (*Remote Direct Memory Access*) de um determinado *cgroup*.

Cgroups[1] - Exemplo

- Controladores montados no sistema de arquivos (debaixo do diretório /cgroup):
  - /sys/fs/cgroup/memory
  - /sys/fs/cgroup/cpu
- Criando um *cgroup*:
  - /cgroup/memory/mytestcgroup
- Definindo limites, como por exemplo 2MB para memória e espaço para swap:
  - $ echo 2097152 > /sys/fs/cgroup/memory/mytestcgroup/memory.limit_in_bytes
  - $ echo 2097152 > /sys/fs/cgroup/memory/mytestcgroup/memory.memsw.limit_in_bytes
- Executando um processo:
  - $ cgexec -g memory:mytestcgroup ./<binary_name>

1 - Control Group v2

Cgroups[1] – Programação de Execução

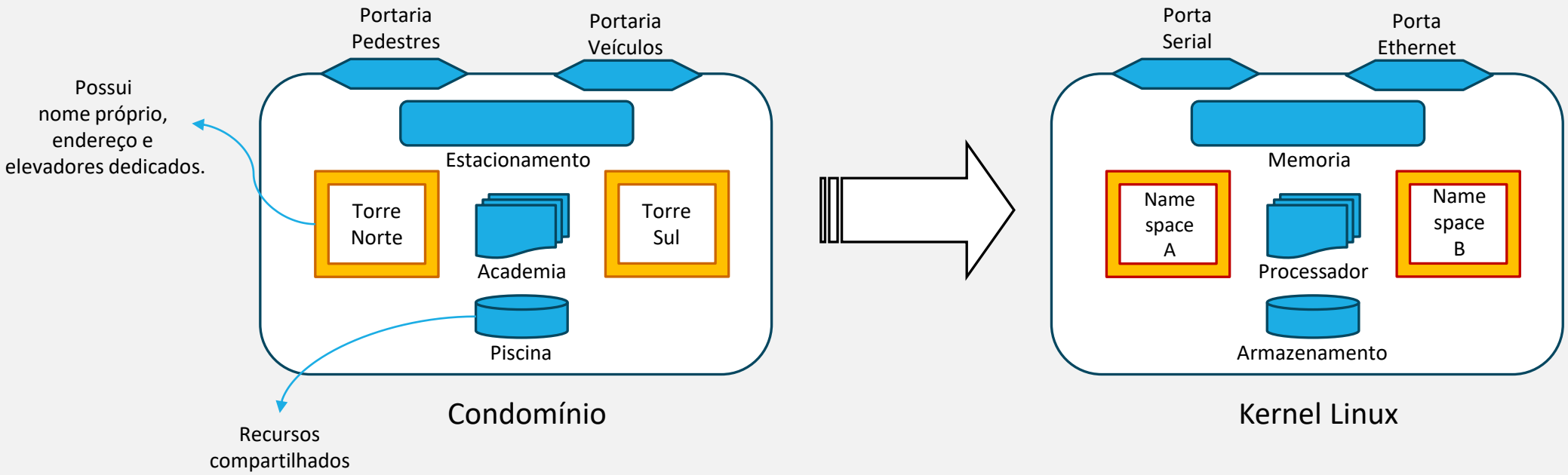
- Se considerarmos contêineres como maquina virtuais leves, parecia lógico administrar recursos computacionais como elementos discretos de mesma forma que os *hypervisors*, como por exemplo número de cores ou CPUs.
- Entretanto o kernel Linux programa a execução de processos de maneira dinâmica.
- O subsistema de CPU para acessar cada cgroup utiliza basicamente dois algoritmos:
  - *Completely Fair Scheduler*[2] (CFS), padrão no Linux e no Docker.
  - *Real Time Scheduler*[3] (RTS).
- A programação de execução usando CFS exige administrar cota de CPUs em termos de quantidades fatias de tempo (e não em unidades discretas de CPUs) definidas no arquivo cpu.shares dentro de um cgroup especifico.

1 - Control Group v2
2 - CFS Scheduler
3 - Real-Time group scheduling

Namespaces[2] 1/3

- É uma abstração do kernel Linux que permite a grupos de processos utilizar de maneira isolada recursos globais como se fossem próprios.

Analogia simplificada



2 - namespaces(7) — Linux manual page

Namespaces[2] 2/3

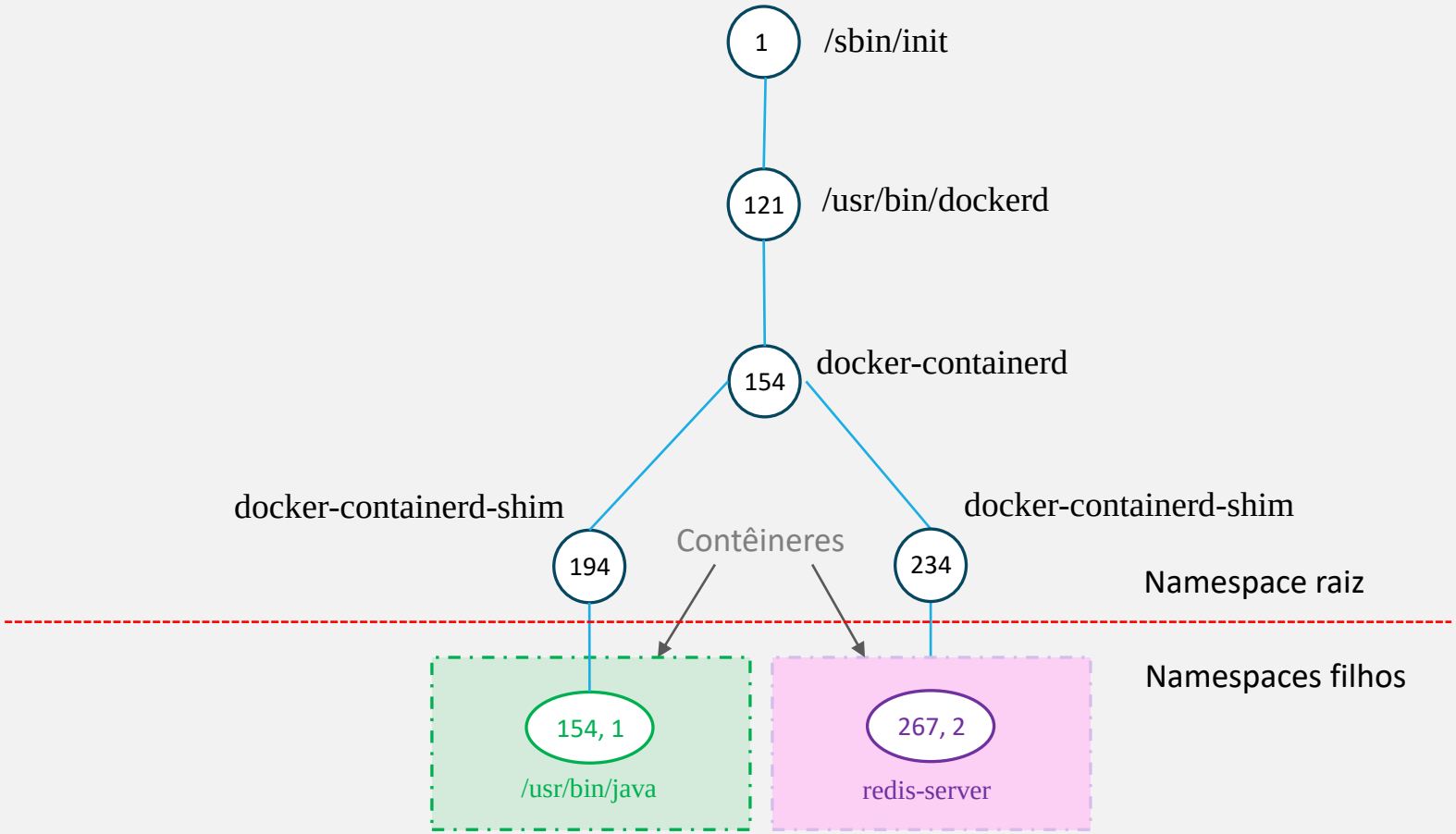
- O kernel Linux administra os processos através de uma única hierarquia com raiz em init .
- Processos com acesso privilegiado podem rastrear, suspender ou abortar a execução de processos comuns ou de usuários.
- Os namespaces permitem administrar múltiplas hierarquias de processos cada uma com sua sub-árvore.
- Processos em uma sub-árvore não acessam processos de outra sub-árvore porque “não sabem” que eles existem (isolamento).

2 - namespaces(7) — Linux manual page

Namespaces 3/3

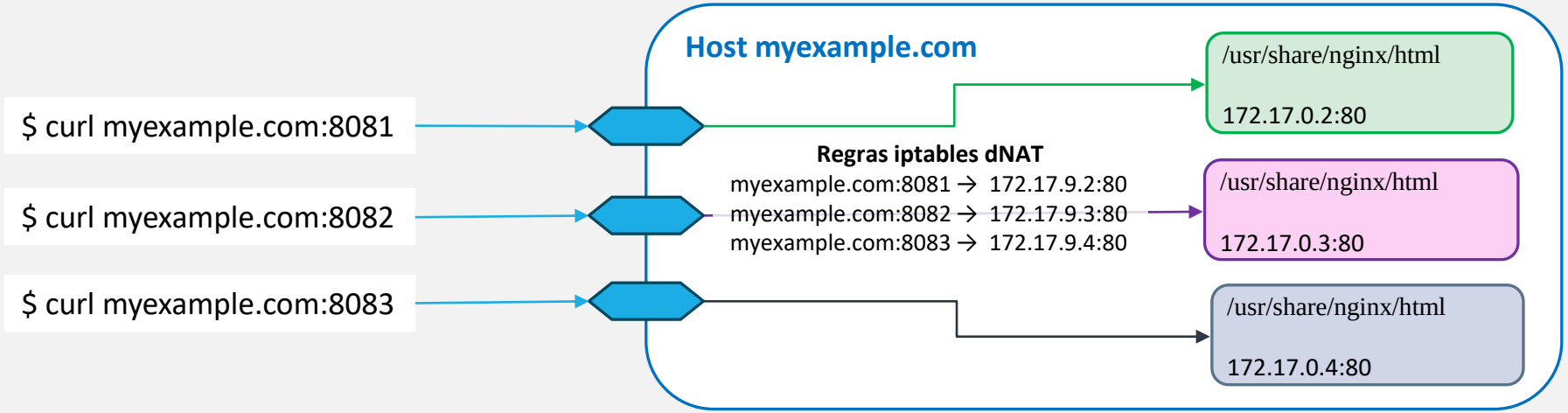
Nome	Descrição suscinta
cgroup	Diretório raiz dos *cgroups.*
IPC	Isola filas de mensagens (*System V IPC* e *POSIX*).
Network	Isola dispositivos de redes, pilhas, portas, etc. associando de maneira dedicada/exclusiva endereços IP, tabelas de roteamento, diretórios /proc/net, números de porta, etc.
Mount	Pontos de montagem no sistema de arquivos. Processos em diferentes *namespaces* podem ter diferentes vistas da hierarquia do sistema de arquivos.
PID	Isola os *pids*. Processos em diferentes namespaces **PID**s podem ter o mesmo *pid*.
Time	Relógio de boot e monotônico.
UTS	*Hostname* e *NIS domain name*. Permite a cada contêiner possuir seu próprio *hostname* e nome de domínio. Afeta as variáveis nodename e domainname retornadas pela chamada de sistema *uname()*.

Namespaces PID - Exemplo



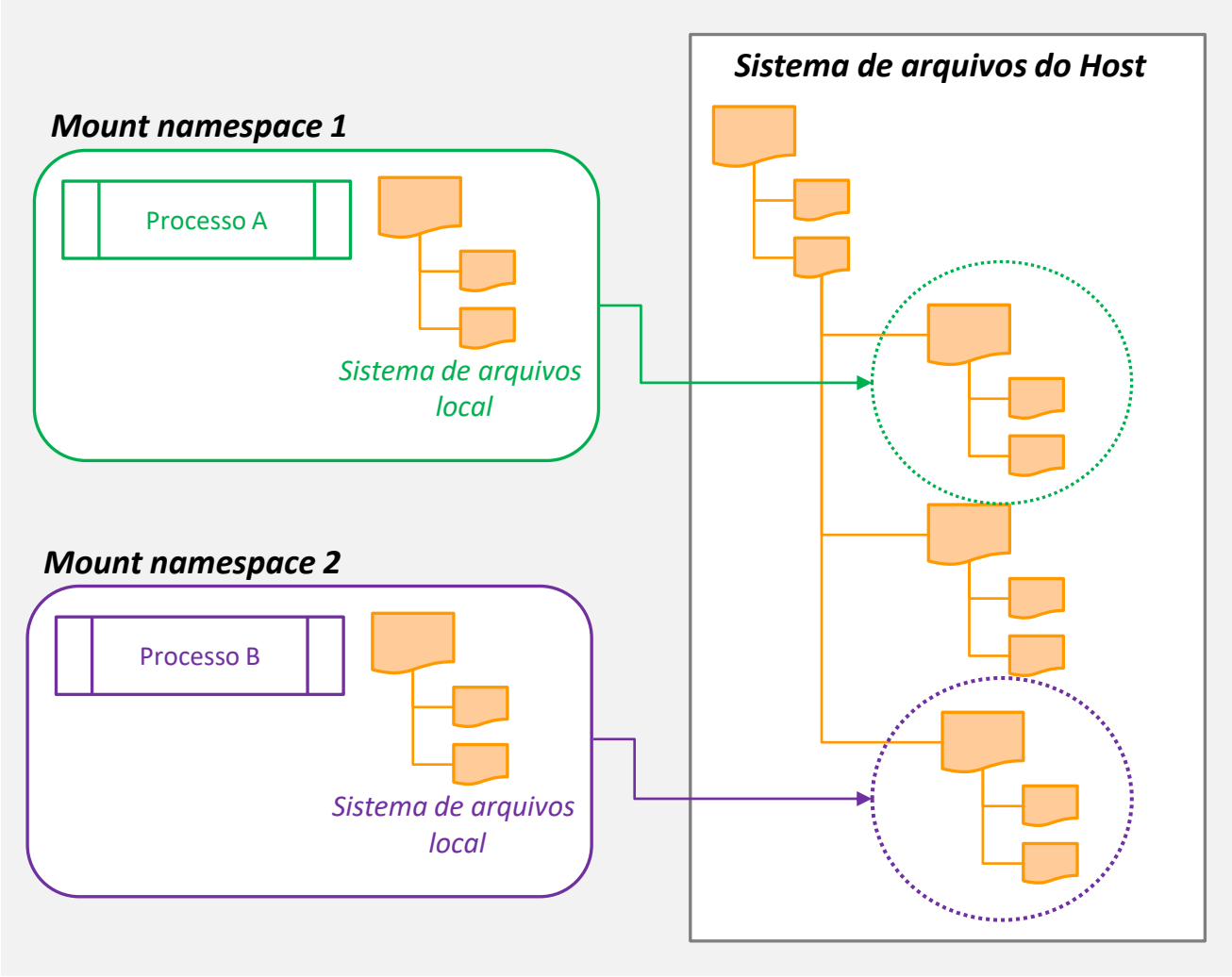
Namespace Network - Exemplo

- Permite o isolamento de todos os recursos associados a redes.
- Cada *network namespace* tem os seus próprios dispositivos, endereços IP, tabelas de roteamento IP, etc.
- Cada contêiner em execução tem o seu próprio dispositivo de rede (virtual) e seus próprios aplicativos vinculados a números de portas IP.
- Tabelas de roteamento no sistema host podem direcionar pacotes de tráfico de rede ao dispositivo de rede associado a um contêiner especifico.



Namespace Mount - Exemplo

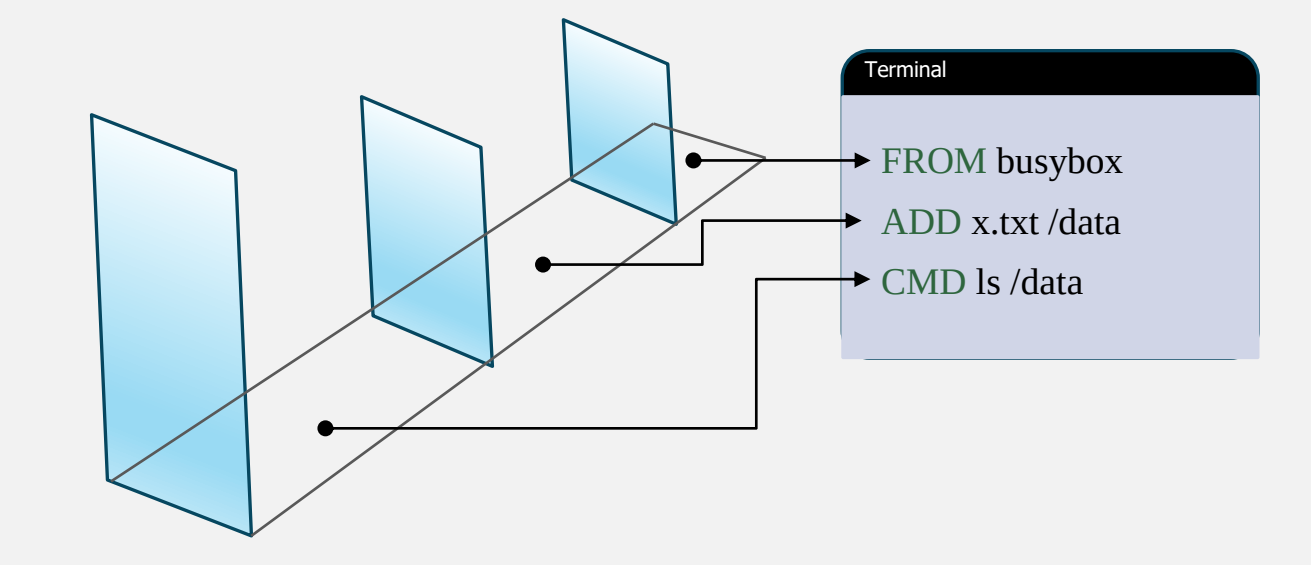
- Foi o primeiro *namespace* a ser implementado em Linux (2002).
- Isola os pontos de montagem de cada grupo de processos no sistema de arquivos.
- Mais seguro e flexível que a chamada de sistema *chroot()*.
- Processos em diferentes *namespaces* podem ter vistas diferentes da hierarquia do sistema de arquivos.



Dúvidas, questões ou comentários?

Unionfs[3] – Union File System 1/2

- Estrutura fundamental para composição de contêineres.
- Sistema de arquivos de unificação empilhável capaz de sobrepor o conteúdo de vários diretórios (ou ramos) preservando separados os seus conteúdos físicos.

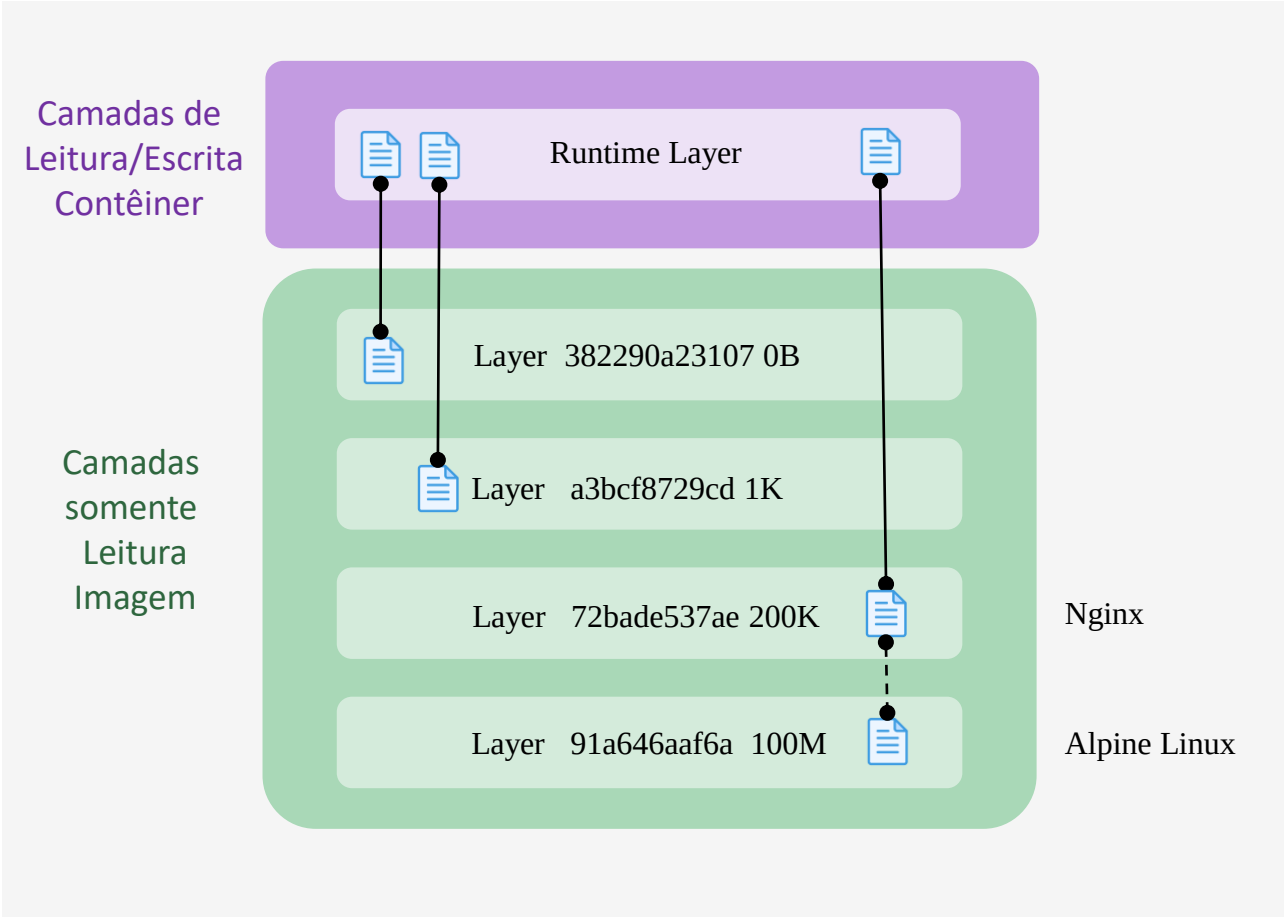


- Consolida em único ponto de montagem todos os diretórios sobrepostos:
  - Os conteúdos com o mesmo caminho dentro dos ramos sobrepostos serão acessados como um único diretório dentro do novo sistema de arquivos virtual.

3 - unionfs(8) - Linux man page

Unionfs – Union File System 2/2

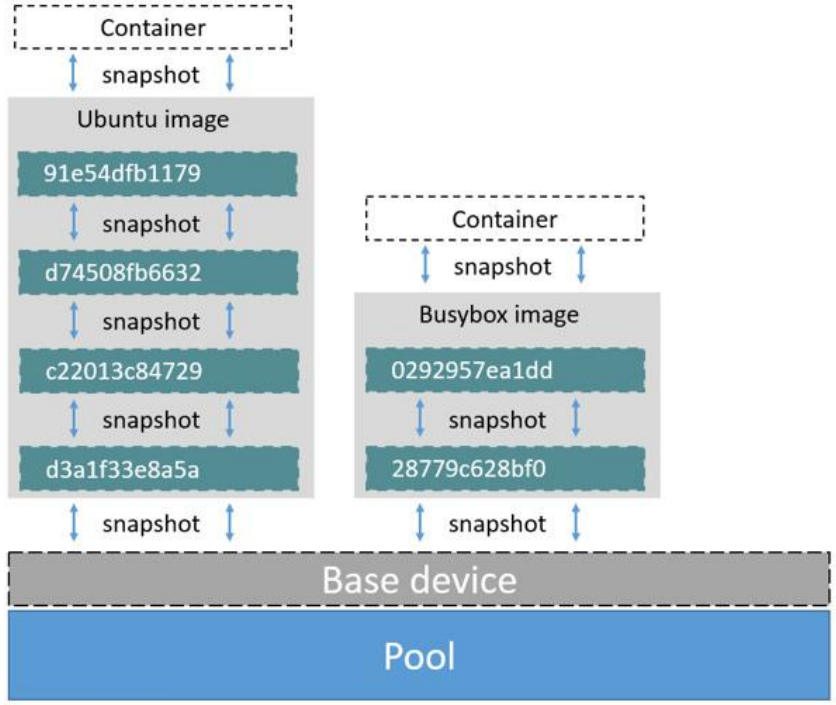
- Com a montagem unificada, os diretórios das camadas superiores são mesclados com os diretórios das camadas inferiores.
- O acesso a arquivos é realizado por camadas, da superior às inferiores.
- A modificação de arquivos nas camadas inferiores (somente leitura) ocorre com sua cópia e modificação na camada superior (escrita/leitura): *copy-on-write (COW)*.
- A localização de arquivos e diretórios é transparente para a aplicativo em execução no contêiner.



4- About storage drivers

Motor de grafos de camadas[4,5]

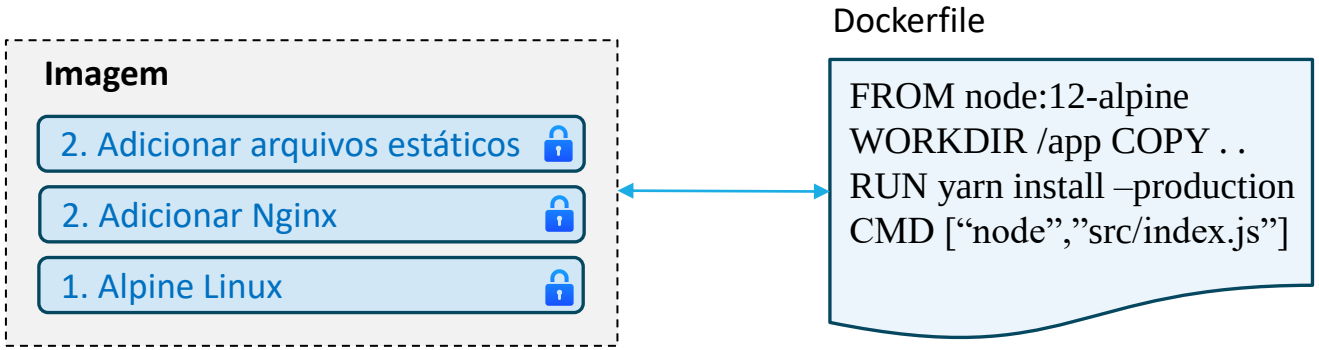
- O *runtime* local de contêineres administra um *cache* ou *pool* local com camadas de imagens.
- O cache é criado e/ou modificado com as instrucoes docker pull e docker build.
- O driver utilizado para administração das camadas se chama *graph driver*.
- Opções: vfs, aufs, overlay, overlay2, btrfs, zfs, devicemapper e windows.
- vfs é uma implementação ingênua (naïve) porque não utiliza unionfs ou técnica *COW*.
- Overlay, overlay2 e aufs: unionfs em cima de um sistema de arquivos real como por exemplo ext4 ou xfs.
- btrfs, zfs, devicemapper, windows: o sistema de arquivos real se encarrega da unificação (mescla e sobreposição de diretórios e arquivos).



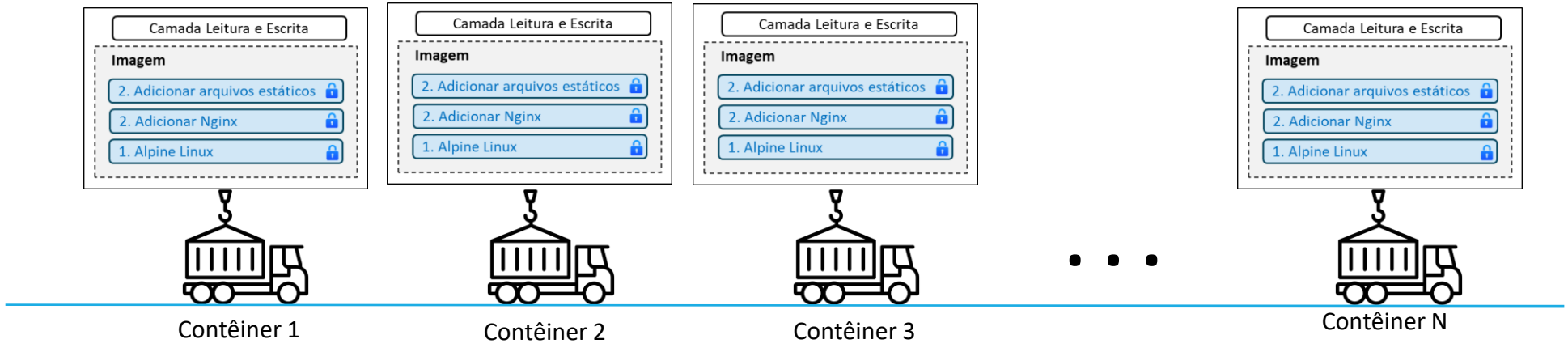
4 - Storage Drivers in Docker: A Deep Dive
5 - Docker overview

Criação e Execução de imagens de Contêineres 1/2

- A imagem de um contêiner é construída com uma pilha de camadas imutáveis.

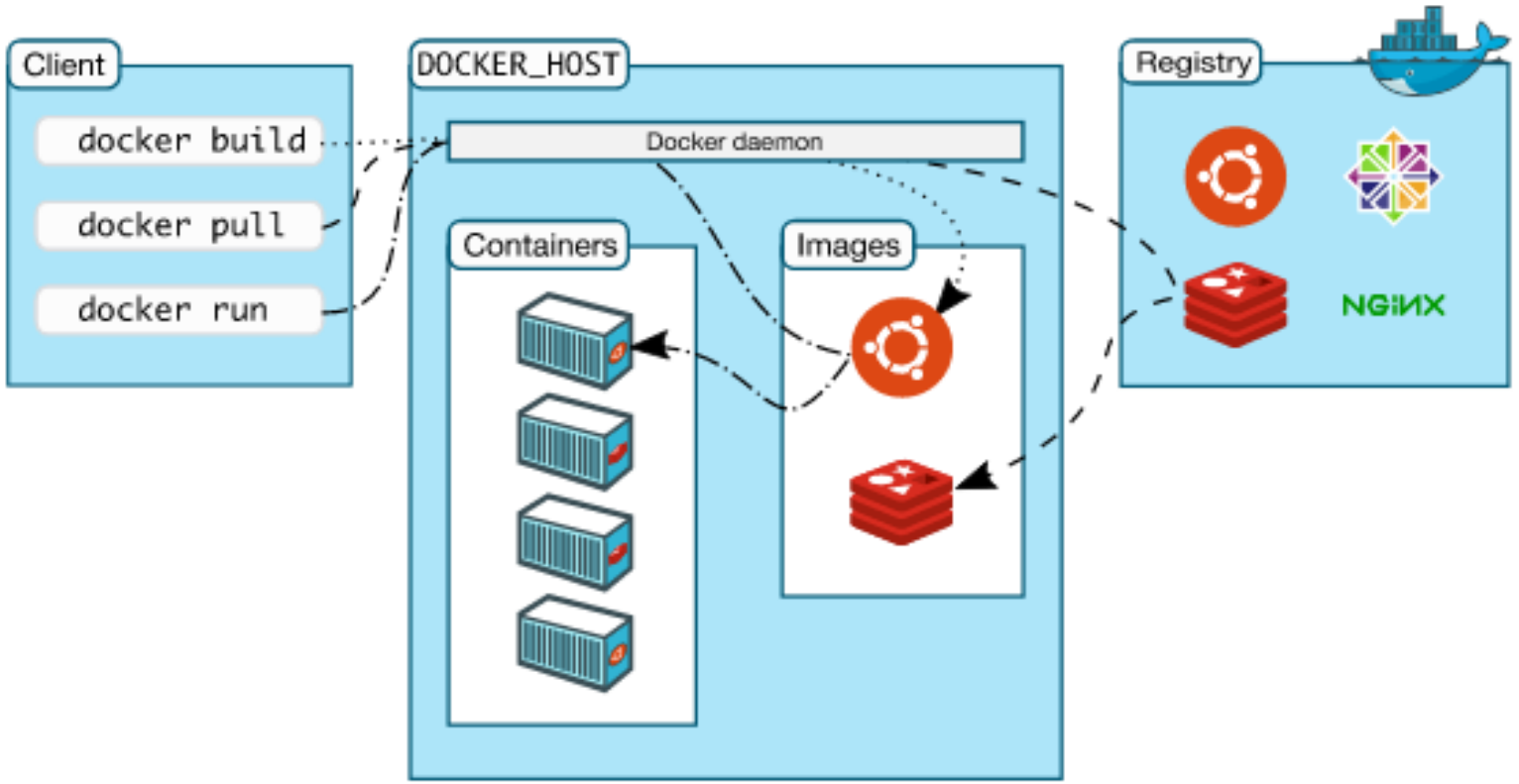


- Em tempo de execução o *runtime* adiciona uma camada de leitura/escrita sobre a pilha imutável.



3 - unionfs(8) - Linux man page

Criação e Execução de imagens de Contêineres[5] 2/2



5 - Working with Docker images

Dúvidas, questões ou comentários?

Atividades para a próxima aula

- Ler Storage Drivers in Docker: A Deep Dive e Docker overview
- Assistir vídeos com tutoriais de números 8 ao 13 do Docker Tutorial:

8. Criar uma imagem customizada
9. Criar uma imagem a partir de um Dockerfile
10. COPY e ADD
11. Criar um projeto realístico
12. Depurar e executar um projeto
13. Rebuilds desnecessários e o arquivo dockerignore