Como contêineres funcionam? parte 1

Caio Volpato ()

5 de janeiro de 2022

O que tem pra hoje?

Existe muito ~hype~ quando o assunto é contêineres e Kubernetes.

Isso acaba gerando uma expectativa e confusão do que são e como funcionam essas tecnologias.

Nessa talk vamos falar o que de fato é um contêiner, como funciona e cria um contêiner do zero (sem docker).

O que é um contêiner?

Um contêiner é um processo rodando no kernel linux.
E esse processo é isolado dos outros processos/contêineres e do host (através de features do kernel)

Essa é a principal diferença dos contêineres para maquinas virtuais.
Todos os contêineres compartilham o mesmo kernel.

E como funciona o docker no Windows ou no Mac? Nesses sistemas, no fundo o docker (ou WSL) roda uma VM linux.

E quais são essas features do kernel que fazem tudo isso acontecer?

Namespaces

Namespaces controlam o que um processo/contêiner consegue ver.
Existem os seguintes namespaces:

- IDs dos processos (pid)
- mount points (mnt)
- Network (net)
- IDs do usuario e group (user)
- Interprocess communication (ipc)
- Control Group (cgroup)
- Unix Timesharing System (uts) -> hostname e dominios

/proc

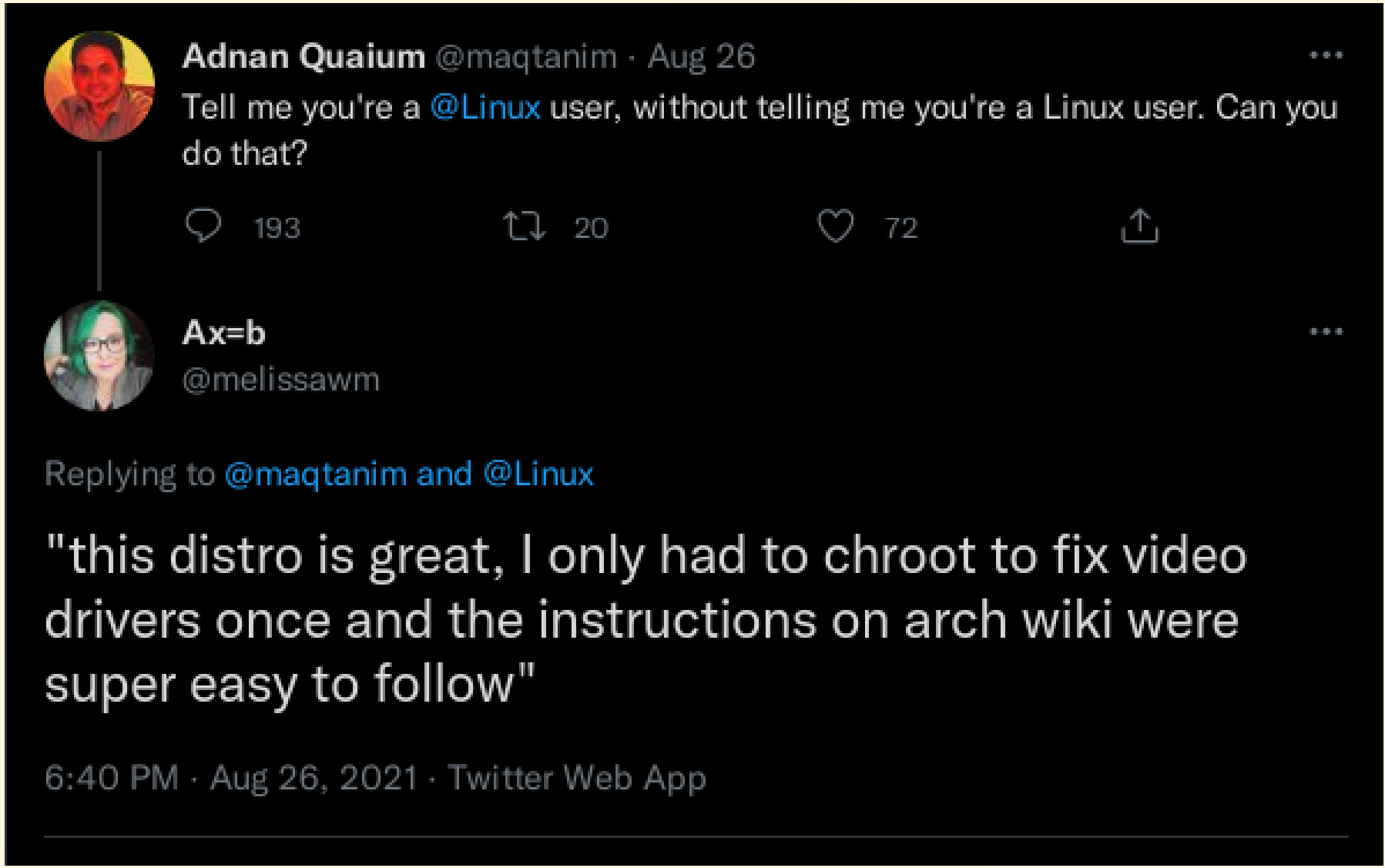
procfs (montado em /proc) é um tipo de filesystem especial.

Nele existem diversas informações do sistema como cpu, memória e principalmente os processos.

Para que os processos do host ficarem isolados do contêiner o contêiner precisa ter um /proc próprio (além de um namespace pid).

chroot

[@melissawm](https://twitter.com/melissawm)



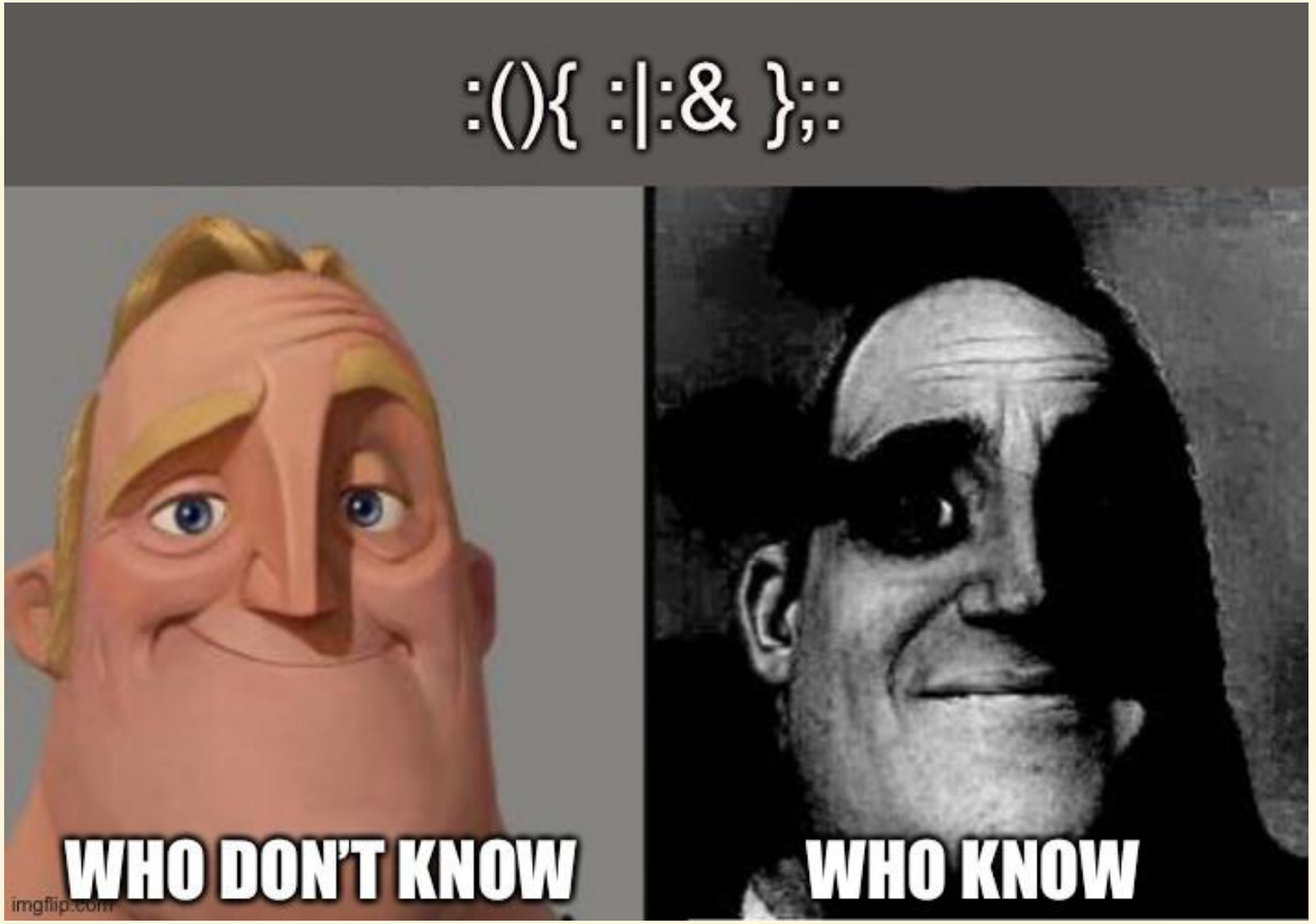
chroot

Para que o contêiner não acesse os arquivos do host,
o chroot (change root directory) faz com que o contêiner só consiga acessar um diretório do host.

Por exemplo, se fizermos um `chroot /mnt/root`
um arquivo em `/mnt/root/arq1.txt (host) -> /arq01.txt (chroot)`

Neste chroot só consegue acessar o que está em /mnt/root

control groups (cgroups)



control groups (cgroup)

```
:(){ :|:& };:
```

Provavelmente é a forkbomb mais conhecida, uma vez executada serão rapidamente criados muitos processos que que se multiplicam exponencialmente incapacitando o computador.

Precisamos de uma forma de limitar quantos recursos um determinado contêiner consegue utilizar.

Os control group (cgroup) limitam os recursos como CPU, memória, rede que cada contêiner pode utilizar.

demo: preparação

Hora de colocar a mão na massa!

Obs.: Os comandos foram executados no Ubuntu 20.04 (cgroup v1)

Primeiramente, vamos criar uma instalação base do Debian:

```
debootstrap bullseye ./deb11-rootfs
https://deb.debian.org/debian/
```

Depois vamos criar um cgroup:

```
cgroup_name="cg_$(shuf -i 2000-3000 -n 1)"

cgcreate -g "cpu,cpuacct,memory,pids:$cgroup_name"
```

demo: preparação

Em seguida, vamos definir limites pro nosso cgroup:

```
cgset -r cpu.shares=256 "$cgroup_name" # 0.25 cpu
cgset -r memory.limit_in_bytes=100MB "$cgroup_name" # 100MB de RAM
cgset -r pids.max=100 "$cgroup_name" # 100 processos no max
```

Finamente, vamos criar um contêiner nesse cggroup:

```
cgexec -g "cpu,cpuacct,memory,pids:$cgroup_name" \
    unshare --fork --mount --uts --ipc --pid --mount-proc \
    chroot "./deb11-rootfs" \
    /bin/sh -c "/bin/mount -t proc proc /proc && hostname contêiner && /bin/bash"
```

demo: o que está acontecendo?

Repare que o contêiner tem um hostname (contêiner), mas o host não teve o hostname alterado.

Rode `uname -a` no contêiner e host e veja que ambos compartilham o mesmo kernel.

Navegue no contêiner, veja que não tem nada na home, o chroot limitou quais arquivos o contêiner consegue acessar.

Faça `ps aux` no contêiner e veja que o contêiner não consegue ver os processos do host.

demo: o que está acontecendo?

No contêiner faça `sleep 1000` edai no host faça `ps aux | grep sleep 1000`, no host conseguimos enxergar os processos do contêiner.

Faça `ls -l /proc/self/ns` no contêiner e compare com o host.

Faça `cat /proc/self/cgroup` no contêiner, e no host navegue para `/sys/fs/cgroup/` e veja os limites daquele cgroup.

Testando os limites do cgroup, abra o python no contêiner e coloque `vet = int(1e3)*[None]`, e vá aumentando o tam, quando processo morrer veja o `sudo dmesg` (out of memory killer (oom))

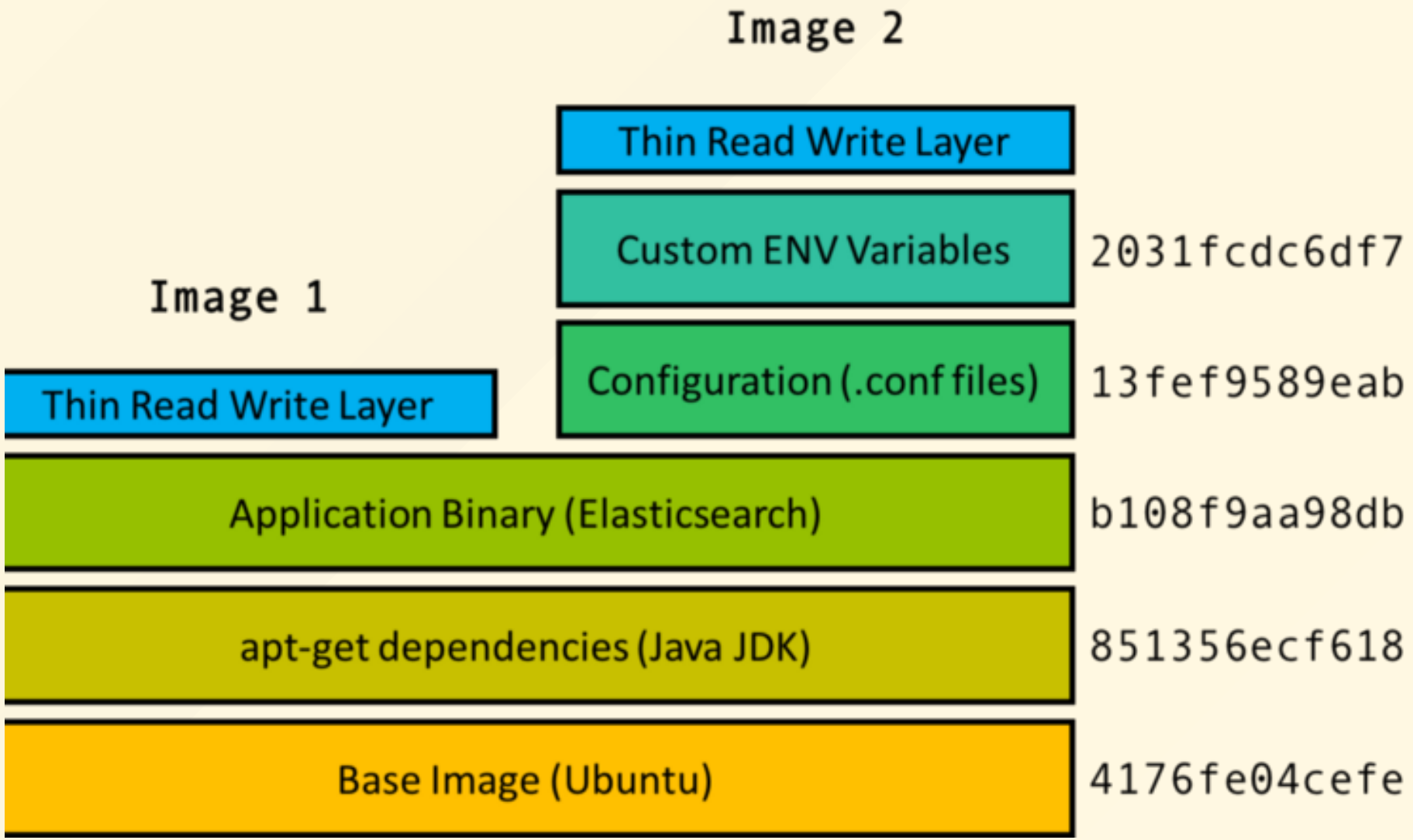
overlay filesystems

No nosso exemplo usamos uma instalação do debian como filesystem, dessa forma cada contêiner precisaria ter sua própria instalação, gastando muito espaço de disco (sem aproveitar o que é comum).

Os filesystem overlay permitem que contêineres diferentes aproveitem o mesmo espaço em disco. Isso acontece através das camadas que são reutilizadas e empilhadas para criar o filesystem final de cada contêiner.

overlay

[fonte]



capabilities, seccomp e apparmor

São features de segurança para restringir os contêiners podem fazer.

- Capabilities: São permissões especiais que permitem processos fazer determinadas ações. Por exemplo **cap_net_bind_service** permite usar portas privilegiadas (<1024).
- seccomp-bpf: Define quais chamadas de sistema (syscalls) são permitidas.
- Apparmor ou selinux: Define quais arquivos podem ser acessados e quais capabilities são permitidas.

firejail

O firejail é uma sandbox que permite enjaular aplicações Desktop linux como browsers, trazendo proteções adicionais.

Funciona de forma similar dos contêiners (namespaces e seccomp)

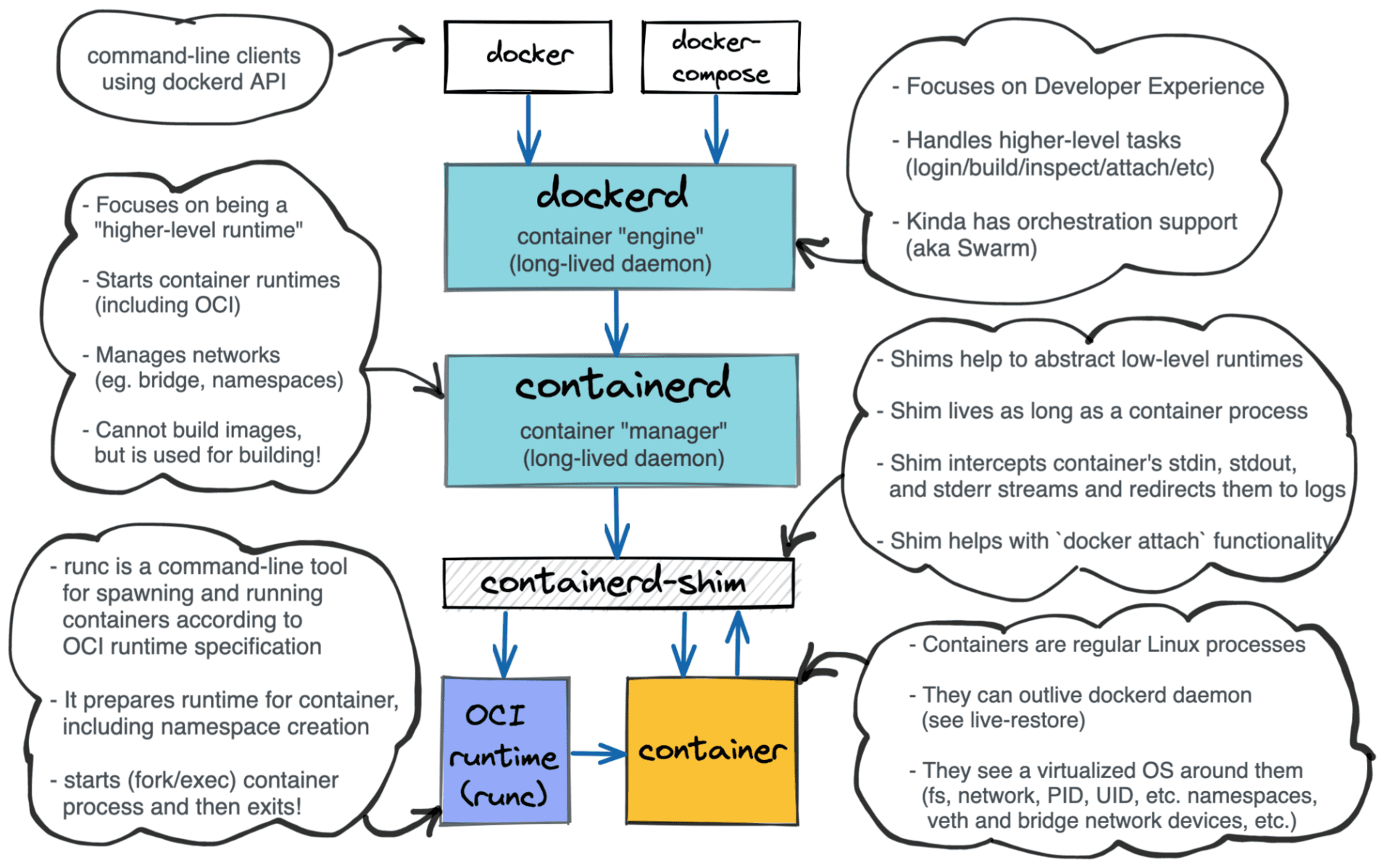
Uma feature legal é que o `/home/` do programa pode ser montado em outra pasta sem ser o home de verdade (opção `--private`).

Com isso da pra usar o app do teams no ubuntu em 2 contas :P (eldorado e cliente)

Quem faz acontecer?

[fonte]

[Talk]



Container runtime

Quem usa as features do kernel linux que falamos (namespaces, cgroups, etc ...) para criar contêineres é o container runtime.

Ao longo dos anos foram sendo criadas interfaces padronizadas, primeiramente a OCI (open containers iniciative) em jun/2015 e nasceu o runc.

Depois foi criada a CRI (container runtime interface) em Dez/2016 e nasceu o containerd.

rootless

Por padrão o docker daemon (e o containerd) roda como root no host, caso um contêiner consiga escapar o isolamento isso vai comprometer totalmente o host.

Uma vulnerabilidade grave no runc foi a CVE-2019-5736 que permitia um container escapar e ter acesso root no host.

Rootless roda o docker como um usuário não root, trazendo um avanço significativo de segurança.

Até então o modo rootless no docker era experimental, mas na versão 20.10 (lançado em dez/2020) virou estável.

Resumo

contêiners são processos que estão isolados através de features do linux kernel (namespaces, cgroups, chroot, etc ...).

Todos os contêiners compartilham o mesmo kernel, ao contrario de maquinas virtuais.

Os contêiners são isolados entre si e do host, mas o host consegue ver e acessar tudo (em uma VM não).

Sempre mantenha o host atualizado!

Referências: Pra quem ta começando

- docker-curriculum.com: Tutorial bem pratico e completo.
- Descomplicando Docker: Curso do Linuxtips (pt-br).
- Docker Mastery: Curso udemy super completo (contempla docker swarm e kubernetes).

Referências

livro: contêiner Security: Fundamental Technology Concepts that Protect contêinerized Applications by Liz Rice

talk: Building a contêiner from scratch in Go - Liz Rice -> código

Obrigado! Perguntas?

turnoff.us