



Container without Docker

Jaehong Jung | WING SENIOR | 2023.03

Reference

if(kakao) dev 2022

이게 돼요? 도커 없이 컨테이너 만들기

카카오엔터프라이즈 | 기술세션 클라우드

2022.12.09

공유하기

The image shows a YouTube video player interface. The video title is "이게 돼요? 도커 없이 컨테이너 만들기" (Is this possible? Making containers without Docker). The channel is "김삼영 (sam.0) 카카오엔터프라이즈". The video is from the event "if(kakao)dev2022". The thumbnail features a man wearing glasses and a white t-shirt with "The SEDA" written on it. The video player includes a play button, a share icon, and a "다음에서 보기: YouTube" (Watch next: YouTube) button.

Timeline

Q. Is Docker a completely new technology?

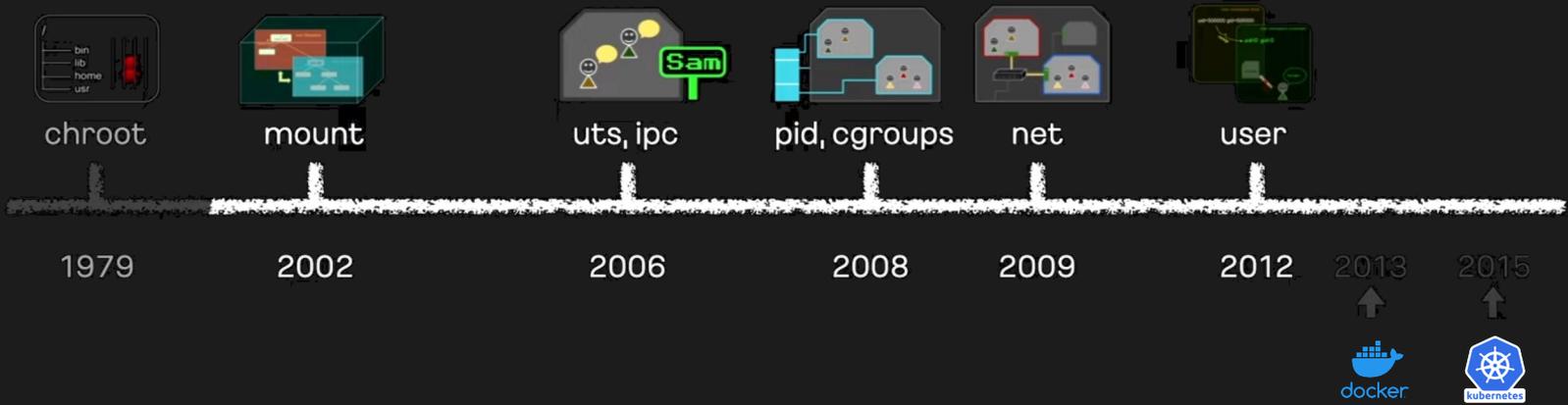


Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

[Intro](#)

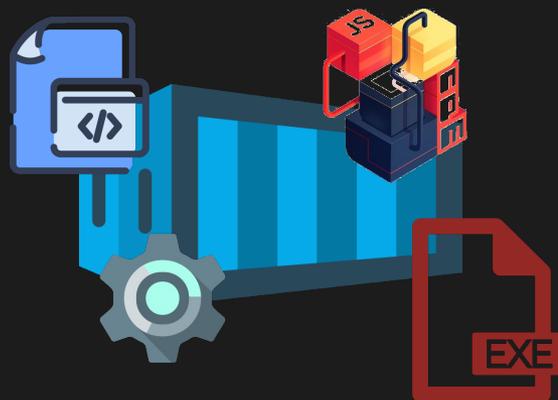
“2,000,000,000”

Containers in 2014 every week

Google is using containers to run everything in their clusters, starting **over 2 billion of them per week**

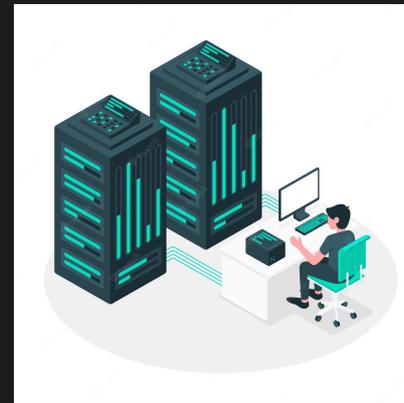
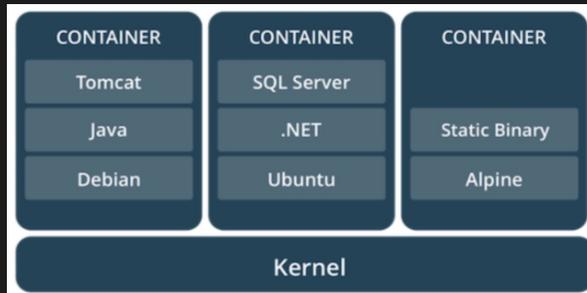
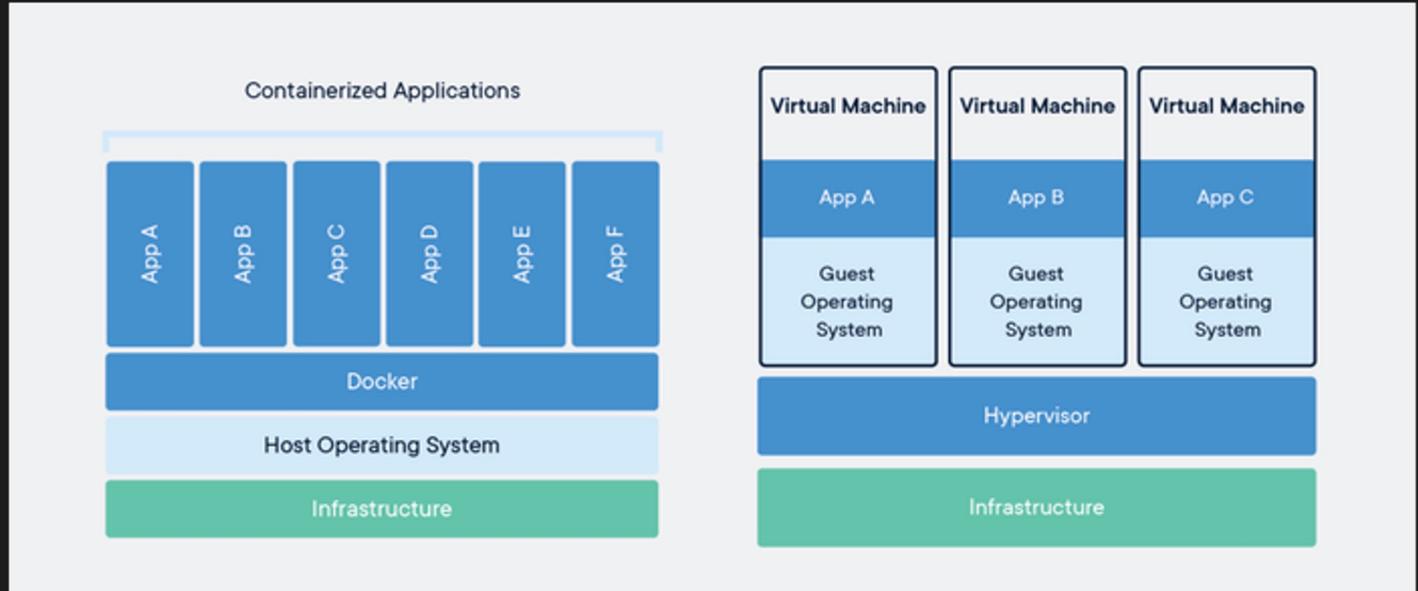
Many Application Leads to high maintenance cost

1. All-in-one **packaging**
2. **Isolation** (such as process)
3. Application **resource allocation**



Introduction

What is Docker and Why Docker?



Prerequisites

```
sudo apt-get -y install gcc make pkg-config libseccomp-dev tree jq bridge-utils

sudo apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \
    gnupg2 \
    pass

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o \
/usr/share/keyrings/docker-archive-keyring.gpg

echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

sudo -Es
```

Host vs Container

root directory

Host(up), Container(down)

```
docker run -it busybox
```

```

X root@ip-172-26-8-43: / (ssh)
root@ip-172-26-8-43:/# ls
bin    home    lib64    opt    sbin    tmp    vmlinuz.old
boot  initrd.img  lost+found  proc  snap  usr
dev    initrd.img.old  media    root  srv    var
etc    lib      mnt      run    sys    vmlinuz
root@ip-172-26-8-43:/#

root@ubuntu1804:~# docker run -it busybox
/# ls
bin    dev    etc    home    proc    root    sys    tmp    usr    var
root@ubuntu1804:~# df -h
Filesystem      Size  Used Available Use% Mounted on
/dev/xvda1      744.5M  9  744.5M  0% /sys/fs/cgroup
tmpfs            8M    0  8M      0% /dev/shm
tmpfs            61.8G  2.2G  59.6G   4% /etc/resolv.conf
tmpfs            61.8G  2.2G  59.6G   4% /etc/hostname

```

```

X root@ip-172-26-8-43: / (ssh)
/ # ls
bin    etc    lib    proc    sys    usr
dev    home  lib64  root    tmp    var
/ #

```

Host vs Container

root directory

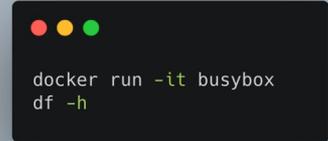
Host(up), Container(down)

```

X root@ip-172-26-8-43: / (ssh)
root@ip-172-26-8-43: /# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            229M   0 229M   0% /dev
tmpfs           48M  840K   48M   2% /run
/dev/xvda1      20G  2.1G   18G  11% /
tmpfs           240M   0 240M   0% /dev/shm
tmpfs           5.0M   0 5.0M   0% /run/lock
tmpfs           240M   0 240M   0% /sys/fs/cgroup
/dev/loop0      88M   88M   0 100% /snap/core/5328
/dev/loop1      13M   13M   0 100% /snap/amazon-ssm-agent/495
tmpfs           48M   0  48M   0% /run/user/1000
overlay         20G  2.1G   18G  11% /var/lib/docker/overlay2/bc4e4308bc68352c47350086a27c84f7e94312b1
0bc5db72132dfed30bdb25b7/merged
root@ip-172-26-8-43: /# █

X root@ip-172-26-8-43: / (ssh)
/ # df -h
Filesystem      Size  Used Available Use% Mounted on
overlay         19.3G  2.0G  17.3G  10% /
tmpfs           64.0M   0  64.0M   0% /dev
tmpfs           240.0M   0  240.0M   0% /sys/fs/cgroup
shm             64.0M   0  64.0M   0% /dev/shm
/dev/xvda1      19.3G  2.0G  17.3G  10% /etc/resolv.conf
/dev/xvda1      19.3G  2.0G  17.3G  10% /etc/hostname
/dev/xvda1      19.3G  2.0G  17.3G  10% /etc/hosts
tmpfs           240.0M   0  240.0M   0% /proc/acpi
tmpfs           64.0M   0  64.0M   0% /proc/kcore
tmpfs           64.0M   0  64.0M   0% /proc/keys
tmpfs           64.0M   0  64.0M   0% /proc/timer_list
tmpfs           64.0M   0  64.0M   0% /proc/sched_debug

```



Host vs Container process

Host(up), Container(down)

```
root@ip-172-26-8-43: / (ssh)
root      15436  0.0  1.0  23028  5032 pts/0    S   15:54   0:00 /bin/bash
root      15644  0.0  0.0     0     0 ?        I   15:56   0:00 [kworker/0
root      15654  0.0  1.4 107988  7236 ?        Ss  15:57   0:00 sshd: ubun
ubuntu    15729  0.0  0.7 107988  3620 ?        S   15:57   0:00 sshd: ubun
ubuntu    15730  0.0  1.0  23164  5124 pts/1    Ss  15:57   0:00 -bash
root      15769  0.0  0.0     0     0 ?        I   15:58   0:00 [kworker/u
root      15847  0.0  0.8  68312  4268 pts/1    S   15:58   0:00 sudo -Es
root      15848  0.0  1.0  23028  5004 pts/1    S   15:58   0:00 /bin/bash
root      15992  0.0  4.8 1186888 23608 pts/0    Sl+ 16:01   0:00 docker run

root@ip-172-26-8-43: / (ssh)
/ # ps aux
PID   USER     TIME   COMMAND
    1   root      0:00   sh
   12   root      0:00   ps aux
/ #
```

Host vs Container network

Host(up), Container(down)

```
root@ip-172-26-8-43: / (ssh)
root@ip-172-26-8-43: /# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:07:39:a7:61:80 brd ff:ff:ff:ff:ff:ff
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:d7:04:94:68 brd ff:ff:ff:ff:ff:ff
9: veth0e8e947@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT
    link/ether ae:37:40:0c:39:32 brd ff:ff:ff:ff:ff:ff link-netnsid 0
root@ip-172-26-8-43: /#

root@ip-172-26-8-43: / (ssh)
/ # ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
/ #
```

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

Container File System

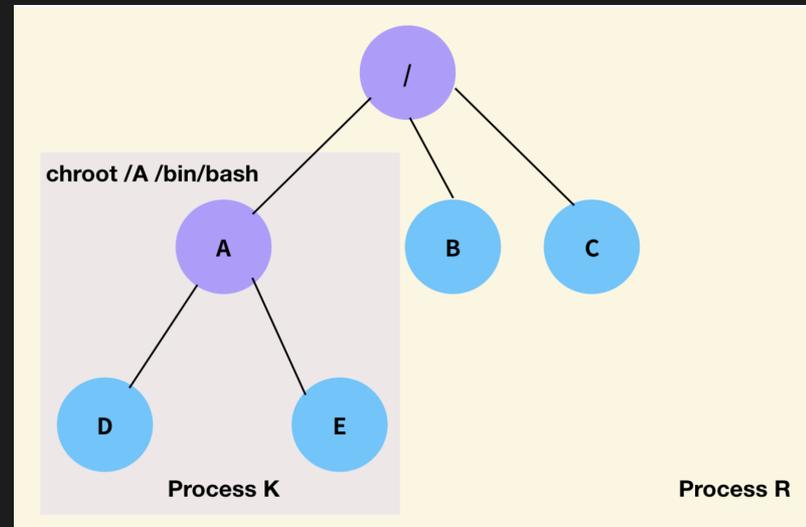
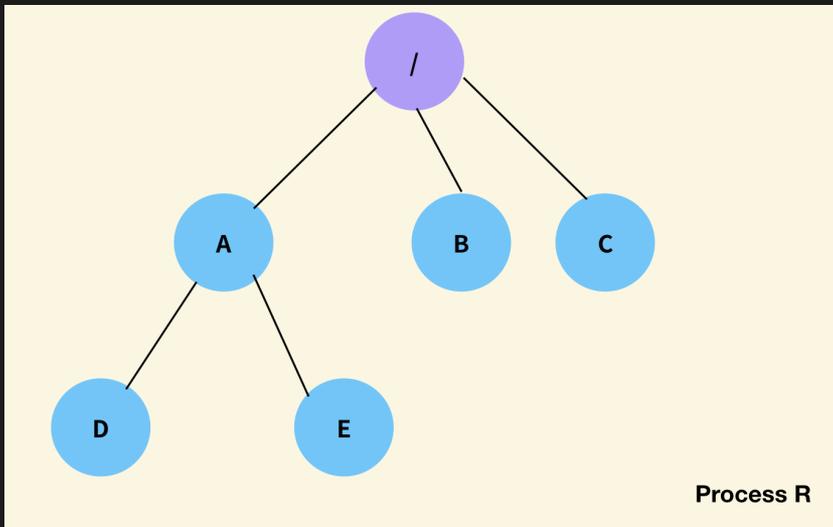
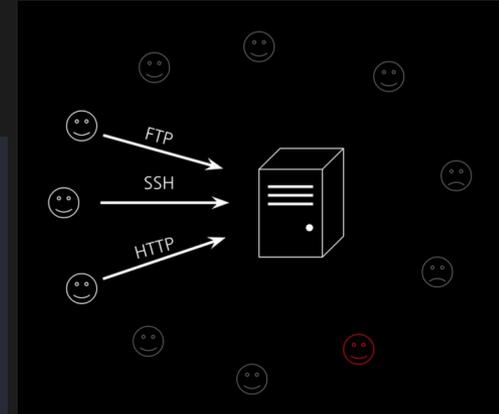
1. Isolating Process
2. Prevent breaking out chroot
3. Prevent duplicated resources

Isolating Process

chroot (1973)

```
ubuntu@ip-172-26-8-43:~$ chroot --version
chroot (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Written by Roland McGrath.



chroot



```
mkdir myroot
```

```
chroot myroot /bin/sh
```

```
# Error
```

```
# Why?
```

```
# copy sh
```

```
which sh # /bin/sh
```

```
ldd /bin/sh
```

```
mkdir -p myroot/bin
```

```
cp /bin/sh myroot/bin/
```

```
mkdir -p myroot/{lib64,lib/x86_64-linux-gnu}
```

```
cp /lib/x86_64-linux-gnu/libc.so.6 myroot/lib/x86_64-linux-gnu/
```

```
cp /lib64/ld-linux-x86-64.so.2 myroot/lib64
```

```
# copy ls
```

```
which ls
```

```
ldd /bin/ls
```

```
cp /bin/ls myroot/bin/
```

```
cp /lib/x86_64-linux-
```

```
gnu/{libselinux.so.1,libc.so.6,libpcre.so.3,libdl.so.2,libpthread.so.0} myroot/lib/x86_64-
```

```
linux-gnu/
```

```
cp /lib64/ld-linux-x86-64.so.2 myroot/lib64/
```

chroot

Cannot get out of **myroot** using `ls`

```
root@ip-172-26-8-43: /tmp (ssh)
root@ip-172-26-8-43:~# cd /tmp
root@ip-172-26-8-43:/tmp# ls
myroot
snap-private-tmp
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-resolved.service-E6CL5F
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-timesyncd.service-0Rryr7
root@ip-172-26-8-43:/tmp#
```

```
root@ip-172-26-8-43: /tmp (ssh)
root@ip-172-26-8-43:/tmp# chroot myroot /bin/sh
# ls
bin lib lib64
# cd ../../../../
# ls
bin lib lib64
#
```

chroot (nginx)

```
mkdir nginx-root;
docker export $(docker create nginx)
| tar -C nginx-root -xvf -;
```

```
root@ip-172-26-8-43:/tmp# tree -L 1 nginx-root
nginx-root
├── bin
├── boot
├── dev
├── docker-entrypoint.d
├── docker-entrypoint.sh
├── etc
├── home
├── lib
├── lib64
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var

20 directories, 1 file
```

chroot (nginx)

```
root@ip-172-26-8-43: /tmp (ssh)
root@ip-172-26-8-43:/tmp# ls /
bin  etc          initrd.img.old  lost+found  opt   run   srv   usr          vmlinuz.old
boot home        lib             media       docke proc  sbin sys   var          스트링으로 아까 만든 nginx-roc
dev  initrd.img  lib64          mnt         root  snap tmp   vmlinuz
root@ip-172-26-8-43:/tmp#

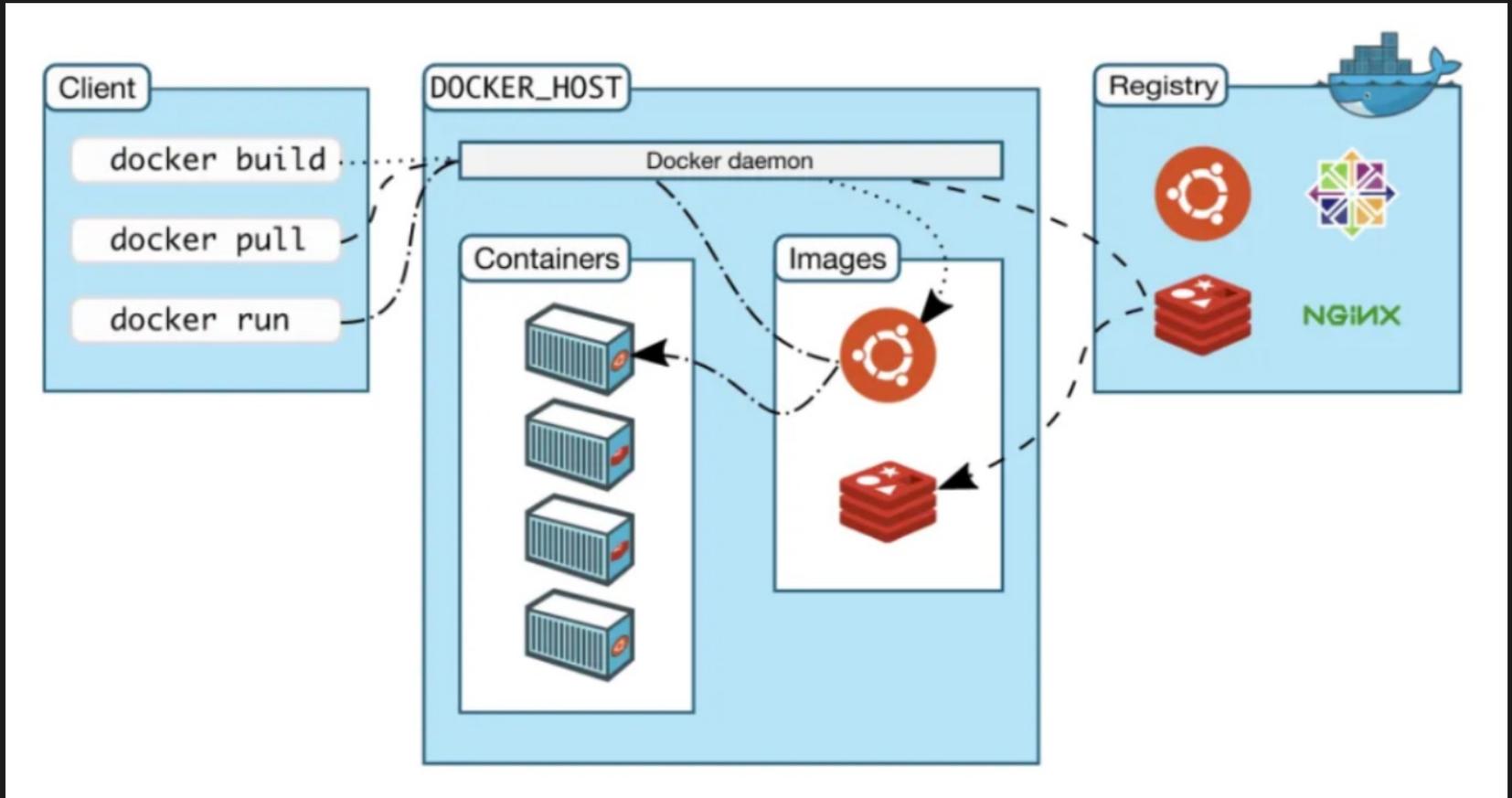
root@ip-172-26-8-43: /tmp (ssh)
├── srv
├── sys
├── tmp
├── usr
└── var

20 directories, 1 file
root@ip-172-26-8-43:/tmp# chroot nginx-root /bin/sh
# ls /
bin  docker-entrypoint.d  home  media  proc  sbin  tmp
boot docker-entrypoint.sh lib   mnt    root  srv   usr
dev  etc                 lib64 opt    run   sys   var
#
```

chroot (nginx)

```
root@ip-172-26-8-43: /tmp (ssh)
dev  initrd.img  lib64  mnt  root  snap  tmp  vmlinuz
root@ip-172-26-8-43:/tmp# curl localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
root@ip-172-26-8-43: /tmp (ssh)
├── sys
├── tmp
├── usr
└── var
20 directories, 1 file
root@ip-172-26-8-43:/tmp# chroot nginx-root /bin/sh
# ls /
bin  docker-entrypoint.d  home  media  proc  sbin  tmp
boot  docker-entrypoint.sh  lib  mnt  root  srv  usr
dev  etc  lib64  opt  run  sys  var
# nginx -g "daemon off;"
[
```

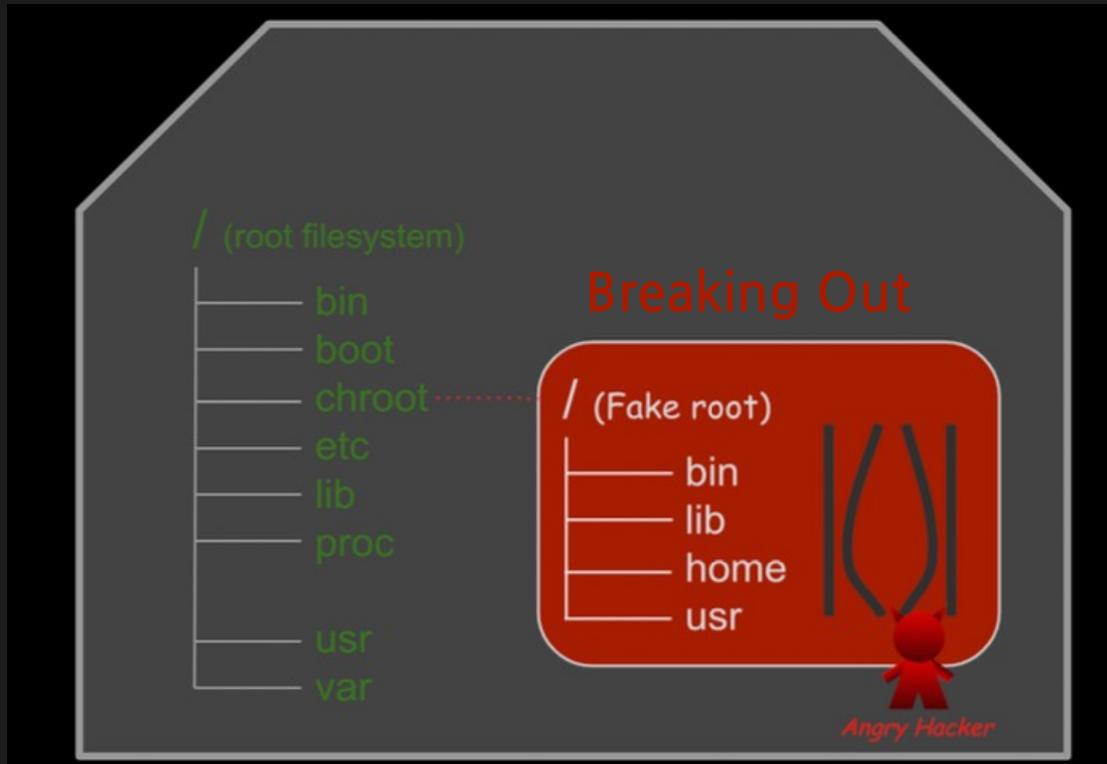
chroot



chroot

1. Packaging
2. Isolation

Breaking out from chroot



Breaking out from chroot

```
#include <sys/stat.h>
#include <unistd.h>

int main(void) {
    mkdir(".out", 0755);
    chroot(".out");
    chdir("../..../..../..../");
    chroot(".");
    return execl("/bin/sh", "-i", NULL);
}
```

Breaking out from chroot

```
root@ip-172-26-8-43:/tmp# chroot myroot /bin/sh
# ls /
bin  escape_chroot  lib  lib64  proc  usr
# cd ../../
# ls
bin  escape_chroot  lib  lib64  proc  usr
# ./escape_chroot
# ls /
bin      etc      initrd.img.old  lost+found  opt  run  srv  usr      vmlinuz.old
boot    home    lib             media       proc  sbin  sys  var
dev     initrd.img  lib64          mnt         root  snap  tmp  vmlinuz
# █
```

아직 탈출 안됨

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

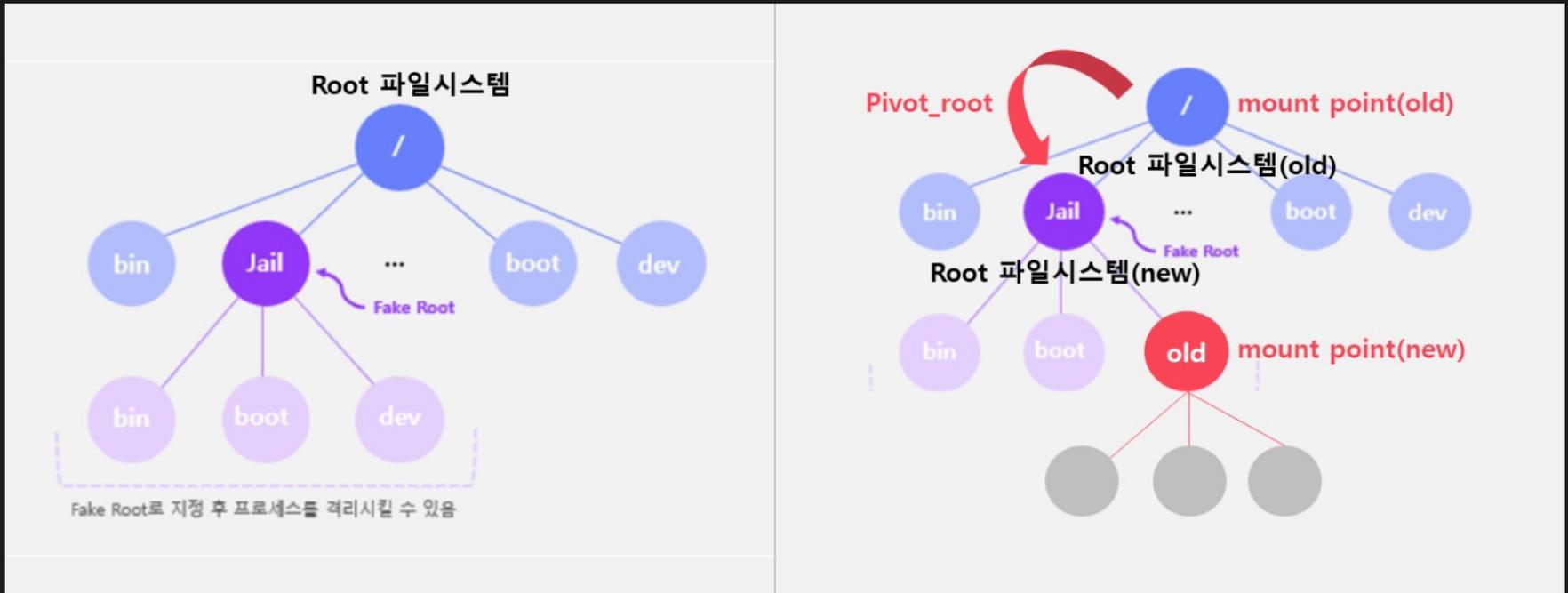
5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

pivot_root



Reference: https://velog.io/@_gyullbb/1-2.-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88-%EA%B2%A9%EB%A6%ACpivotroot

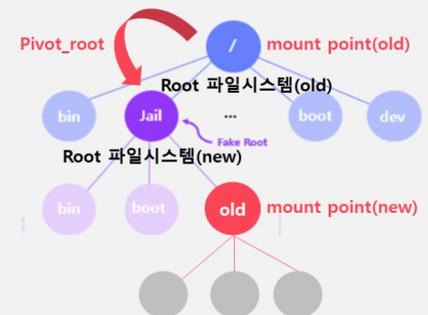
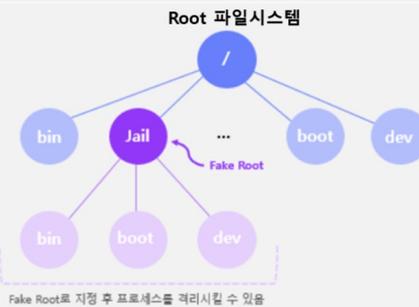
pivot_root

```

mkdir jail
docker export $(docker create nginx:latest) | tar -C jail -xvf -

chroot jail /bin/sh
./escape_chroot

```



pivot_root

```
mkdir new_jail
mount -n -t tmpfs -o size=800M none ./new_jail/
df -h /new_jail
---
```

Filesystem	Size	Used	Avail	Use%	Mounted on
none	800M	0	800M	0%	/new_jail

```
---
```

```
mount | grep new_jail
---
```

none	on	/new_jail	type	tmpfs	(rw,relatime,size=819200k)
------	----	-----------	------	-------	----------------------------

```
---
```

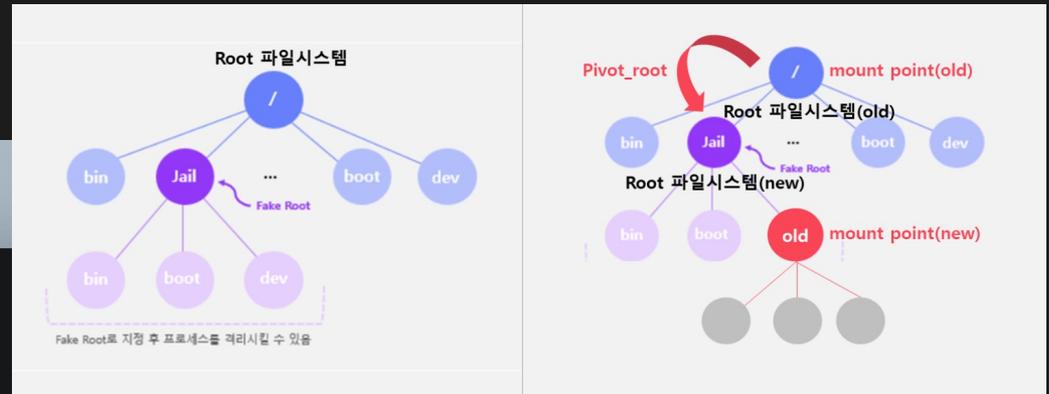
pivot_root



```
cp -r jail/* ./new_jail/
mkdir ./new_jail/old_jail
```

```
tree -L 1 ./new_jail/
```

```
---
./new_jail/
├── bin
├── ...
├── escape_chroot
├── ...
├── old_jail
├── ...
└── var
```



`pivot_root`

Q. If the root file system is pivoted, will it have an impact on the host?

A. Yes, it will definitely have an impact.

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

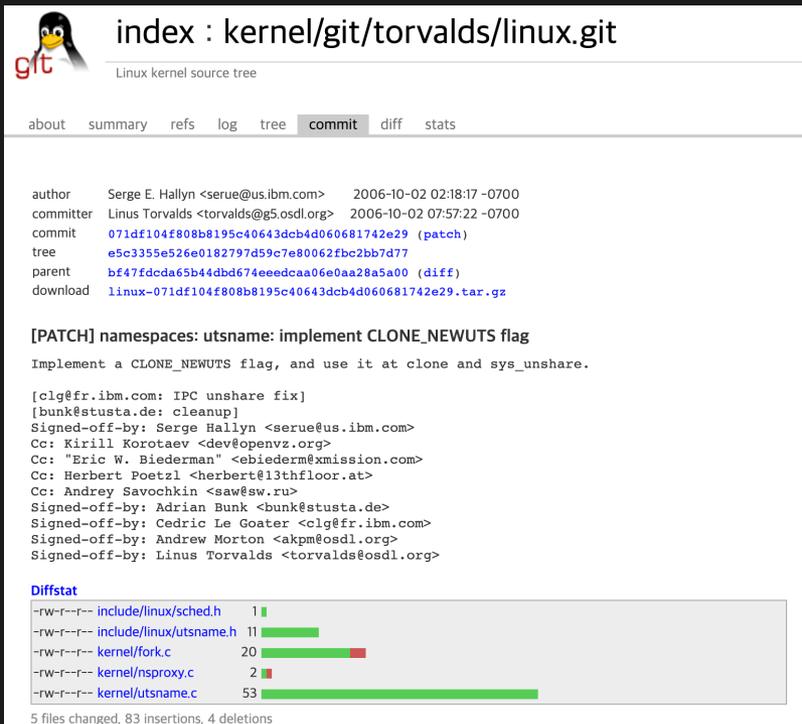
5-2 Container layout

6 Making own container

6-1 Let's do it!

Namespace (2002)

Feature of the Linux kernel that allow for creating isolated environments, for different system resources, such as process IDs, network interfaces, file systems, and more



index : kernel/git/torvalds/linux.git

Linux kernel source tree

about summary refs log tree **commit** diff stats

author Serge E. Hallyn <serue@us.ibm.com> 2006-10-02 02:18:17 -0700
 committer Linus Torvalds <torvalds@g5.osdl.org> 2006-10-02 07:57:22 -0700
 commit 071df104f808b8195c40643dcb4d060681742e29 (patch)
 tree e5c3355e526e0182797d59c7e80062fbc2bb7d77
 parent bf47fdca65b44dbd674eedcaa06e0aa28a5a00 (diff)
 download linux-071df104f808b8195c40643dcb4d060681742e29.tar.gz

[PATCH] namespaces: utsname: implement CLONE_NEWUTS flag
 Implement a CLONE_NEWUTS flag, and use it at clone and sys_unshare.

[clg@fr.ibm.com: IPC unshare fix]
 [bunk@stusta.de: cleanup]
 Signed-off-by: Serge Hallyn <serue@us.ibm.com>
 Cc: Kirill Korotaev <dev@openvz.org>
 Cc: "Eric W. Biederman" <ebiederm@xmission.com>
 Cc: Herbert Poetzl <herbert@13thfloor.at>
 Cc: Andrey Savochkin <saw@sw.ru>
 Signed-off-by: Adrian Bunk <bunk@stusta.de>
 Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
 Signed-off-by: Andrew Morton <akpm@osdl.org>
 Signed-off-by: Linus Torvalds <torvalds@osdl.org>

Diffstat

-rw-r--r--	include/linux/sched.h	1
-rw-r--r--	include/linux/utsname.h	11
-rw-r--r--	kernel/fork.c	20
-rw-r--r--	kernel/nsproxy.c	2
-rw-r--r--	kernel/utsname.c	53

5 files changed, 83 insertions, 4 deletions

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

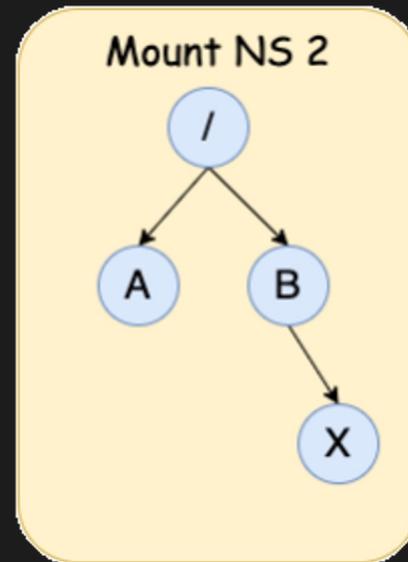
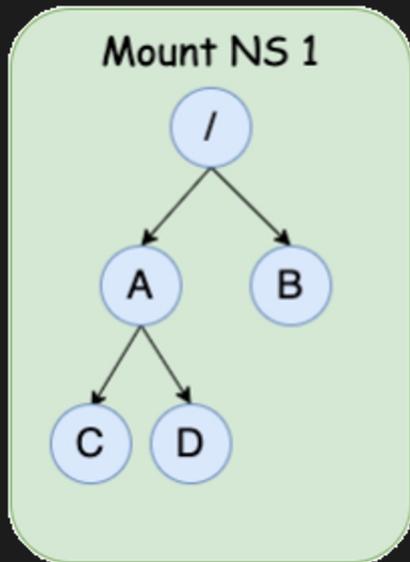
4-4 cgroup namespace

4-5 Network namespace

4-6 USER namespace

Mount namespace

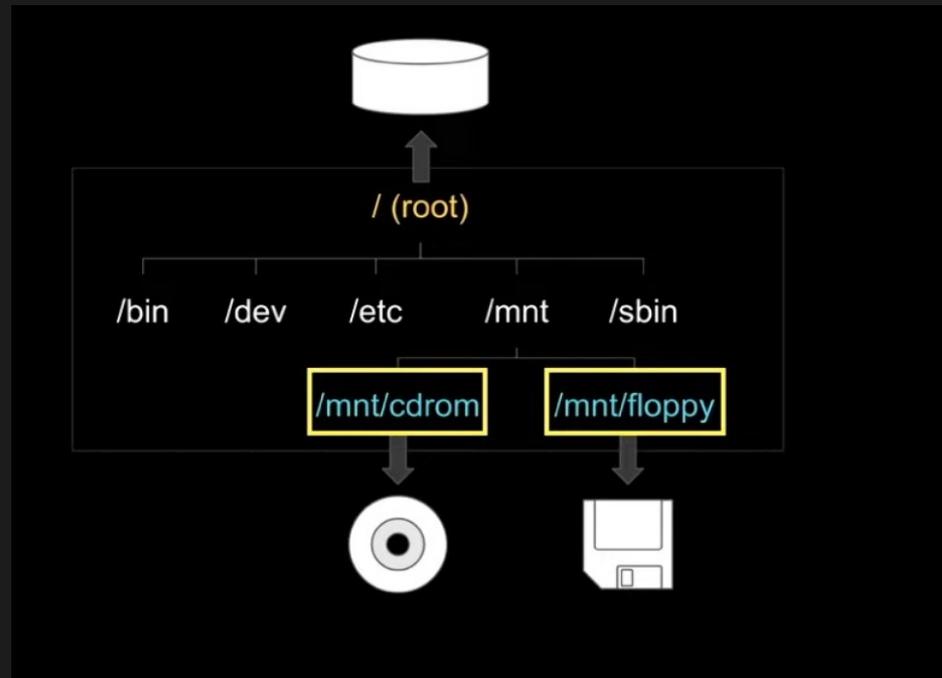
Isolates file systems, so that each namespace has its own unique view of the file system tree.



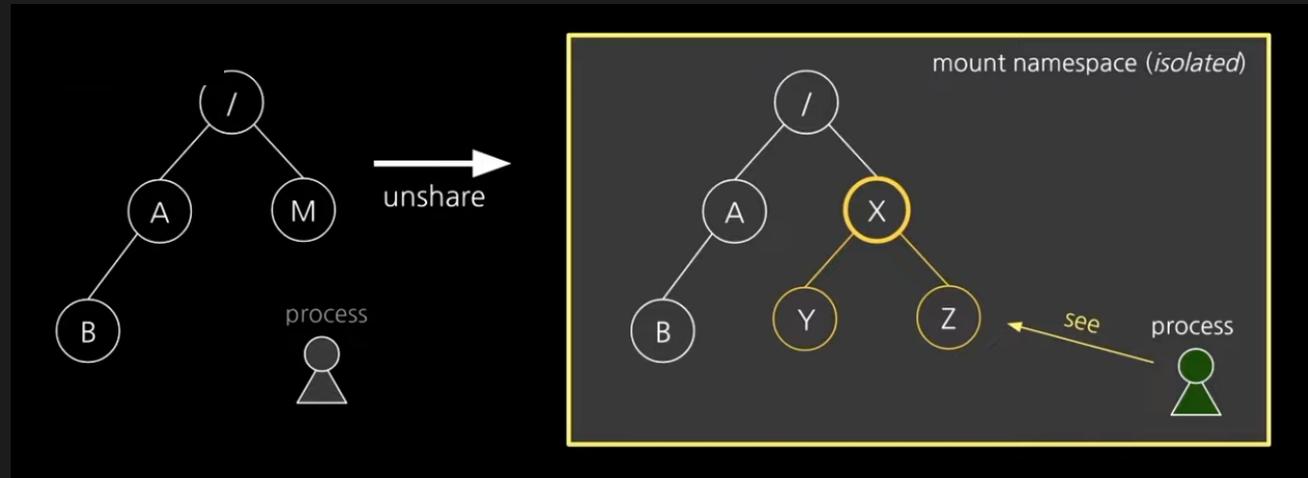
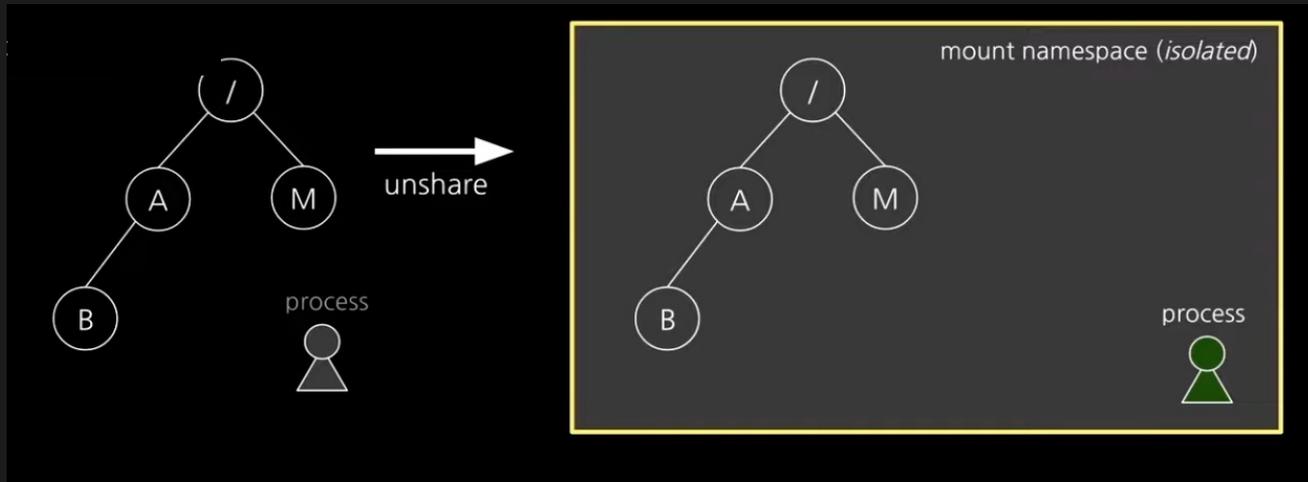
Mount

Process of making a file system available for use by the operating system and its users

Ex. external storage



Mount namespace



Mount namespace

Seems identical in initial state

```

ubuntu@ip-172-26-8-43: ~ (ssh)

Last login: Fri Mar 17 16:58:16 2023 from 203.230.52.5
ubuntu@ip-172-26-8-43:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            222M   0  222M   0% /dev
tmpfs           48M   800K  47M   2% /run
/dev/xvda1      20G   3.0G   17G  16% /
tmpfs           236M   0  236M   0% /dev/shm
tmpfs           5.0M   0   5.0M   0% /run/lock
tmpfs           236M   0  236M   0% /sys/fs/cgroup
/dev/loop0      117M  117M   0 100% /snap/core/14784
/dev/loop1       56M   56M   0 100% /snap/core18/2708
/dev/loop2       25M   25M   0 100% /snap/amazon-ssm-agent/6312
/dev/loop3       13M   13M   0 100% /snap/amazon-ssm-agent/495
/dev/loop4       56M   56M   0 100% /snap/core18/2714
tmpfs           48M   0   48M   0% /run/user/1000
ubuntu@ip-172-26-8-43:~$ █

root@ip-172-26-8-43: /tmp (ssh)
root@ip-172-26-8-43:/# cd /tmp
root@ip-172-26-8-43:/tmp# clear
root@ip-172-26-8-43:/tmp# unshare --mount /bin/sh
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1      20G   3.0G   17G  16% /
udev            222M   0  222M   0% /dev
tmpfs           236M   0  236M   0% /dev/shm
tmpfs           48M   792K  47M   2% /run
tmpfs           5.0M   0   5.0M   0% /run/lock
tmpfs           48M   0   48M   0% /run/user/1000
tmpfs           236M   0  236M   0% /sys/fs/cgroup
/dev/loop0      117M  117M   0 100% /snap/core/14784
/dev/loop1       56M   56M   0 100% /snap/core18/2708
/dev/loop2       25M   25M   0 100% /snap/amazon-ssm-agent/6312
/dev/loop3       13M   13M   0 100% /snap/amazon-ssm-agent/495
/dev/loop4       56M   56M   0 100% /snap/core18/2714
# █

```

Mount namespace

mount *new_root* can only be seen in mount namespace

```

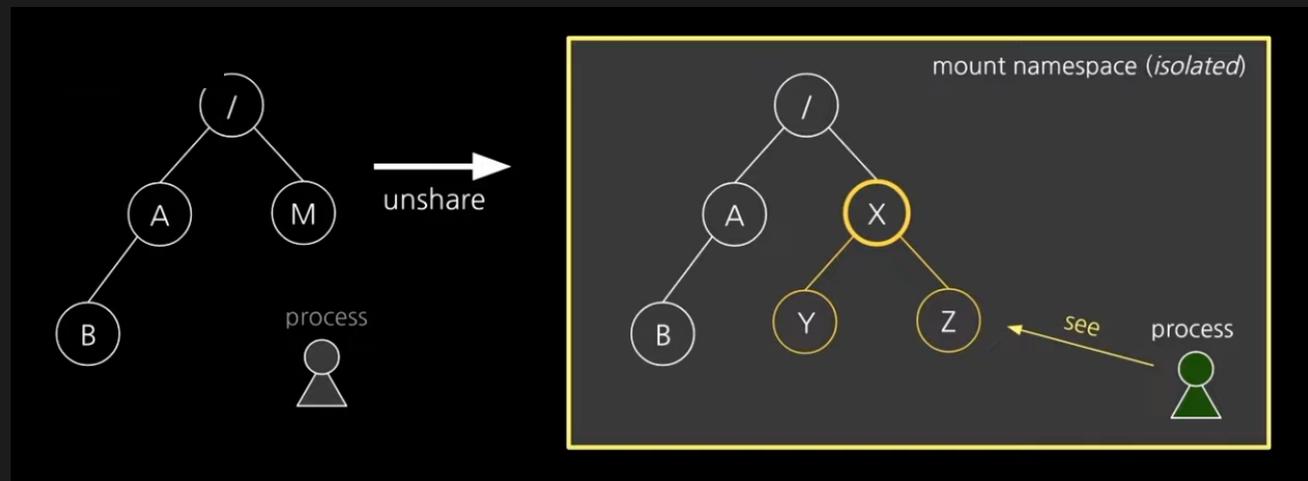
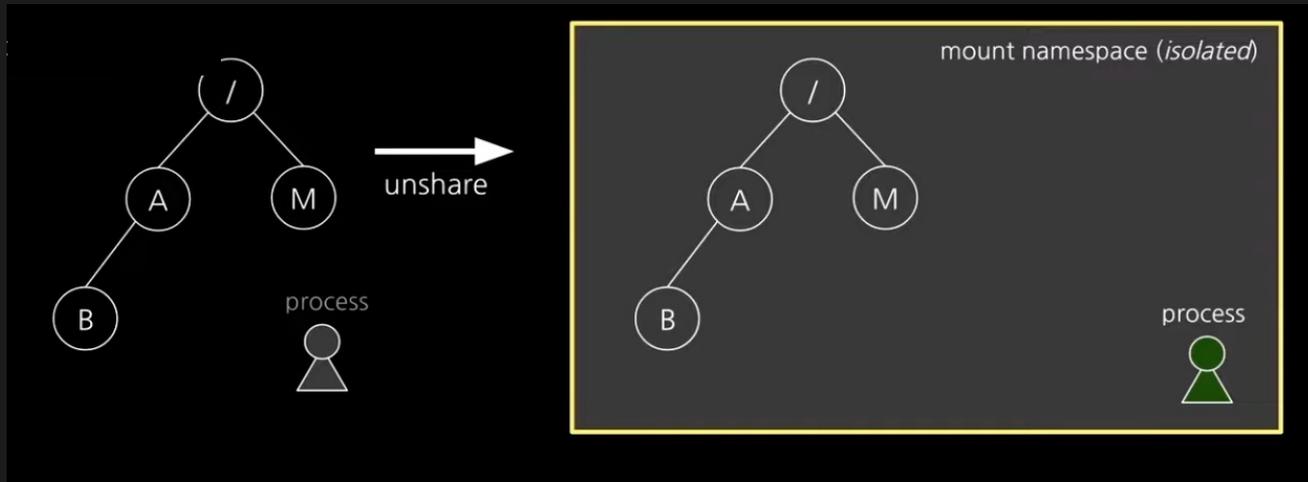
ubuntu@ip-172-26-8-43: /tmp (ssh)
ubuntu@ip-172-26-8-43:/tmp$ ls
chroot_ps.sh
escape_chroot.c
myroot
new_root
nginx-root
snap-private-tmp
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-resolved.service-E6CL5F
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-timesyncd.service-0RrYr7
ubuntu@ip-172-26-8-43:/tmp$ mount | grep new_root
ubuntu@ip-172-26-8-43:/tmp$

root@ip-172-26-8-43:/tmp# chroot myroot
# ls /
bin escape_chroot lib lib64 proc us
#

root@ip-172-26-8-43:/tmp (ssh)
/dev/loop3 13M 13M 0 100% /snap/amazon-ssm-agent/495
/dev/loop4 56M 56M 0 100% /snap/core18/2714
# mkdir new_root
# mount -t tmpfs none new_root
# df -h | grep new_root
none 236M 0 236M 0% /tmp/new_root
# ls
chroot_ps.sh
escape_chroot.c
myroot
new_root
nginx-root
snap-private-tmp
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-resolved.service-E6CL5F
systemd-private-a5d1b3db81c84dd3ab356eb0e8f5be2a-systemd-timesyncd.service-0RrYr7
# mount | grep new_root
none on /tmp/new_root type tmpfs (rw,relatime)
#

```

Mount namespace

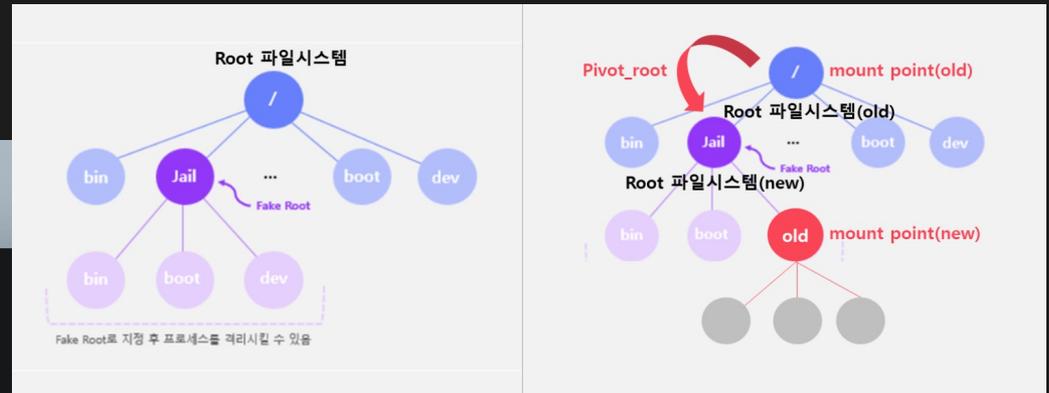


pivot_root



```
cp -r jail/* ./new_jail/
mkdir ./new_jail/old_jail
```

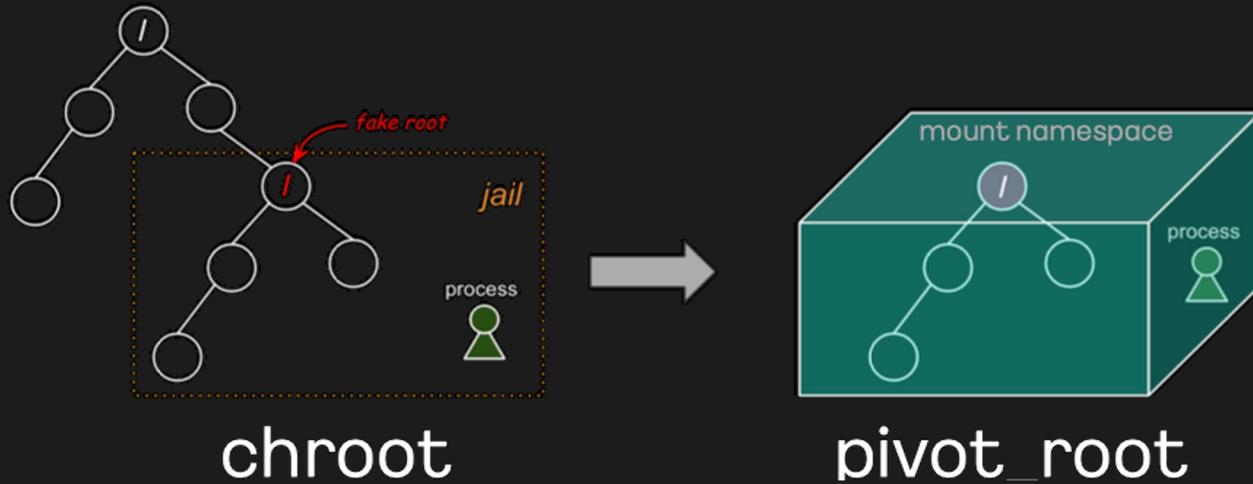
```
tree -L 1 ./new_jail/
---
./new_jail/
├── bin
├── ...
├── escape_chroot
├── ...
├── old_jail
├── ...
└── var
```



pivot_root

```
cd ./new_jail/  
# mount namespace  
unshare -m  
  
pivot_root . old_jail  
  
# try to escape  
./escape_chroot  
  
cd /  
ls  
---  
bin   docker-entrypoint.d  etc   lib64  old_jail  root  srv  usr  
boot  docker-entrypoint.sh home  media  opt     run   sys  var  
dev   escape_chroot       lib   mnt    proc     sbin  tmp
```

pivot_root



Mount namespace

1. Isolated Environment (as own root filesystem)
2. Safe in security manner
3. Can change mount freely and expand filesystem also

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

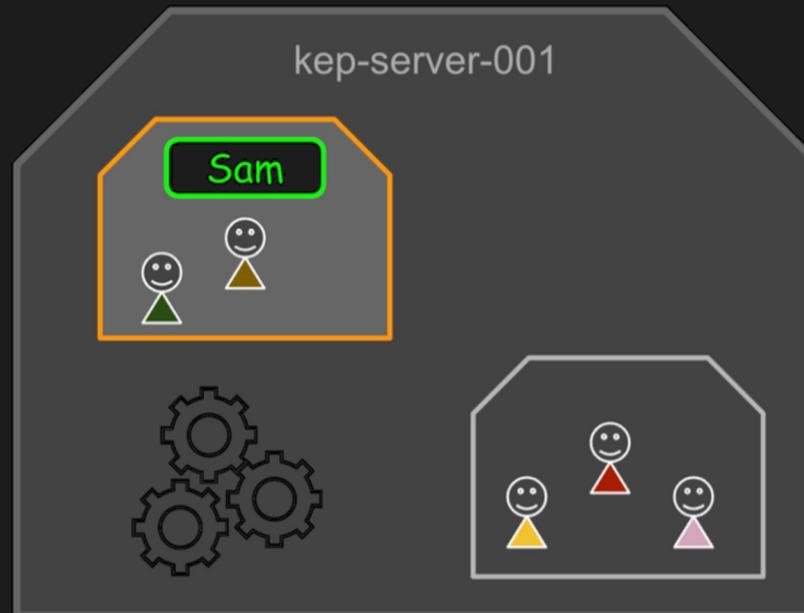
5-2 Container layout

6 Making own container

6-1 Let's do it!

UTS (Unix Time Sharing) namespace

Namespace that isolates
two system identifiers related to the system hostname



UTS (Unix Time Sharing) namespace

```
hostname  
---  
ubuntu
```

```
unshare --uts /bin/bash  
  
hostname jaehong21  
hostname  
---  
jaehong21
```

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

IPC (Inter-Process Communication) namespace

1. Pipe
2. Named-pipe
3. Message queue
4. Shared Memory
5. Memory map
6. Socket

IPC (Inter-Process Communication) namespace

Host(up), Container(down)

```
ubuntu@ip-172-26-8-43:~$ sudo ipcmk -M 1000
```

```
Shared memory id: 3
```

```
ubuntu@ip-172-26-8-43:~$ sudo ipcs -m
```

```
----- Shared Memory Segments -----  
key          shmid      owner      perms      bytes      nattch     status  
0x00000000  0          root       644        80         2            
0x00000000  1          root       644        16384      2            
0x00000000  2          root       644        280        2            
0xf58e45ec  3          root       644        1000       0          
```

```
root@ip-172-26-8-43:/tmp# unshare --ipc /bin/bash
```

```
root@ip-172-26-8-43:/tmp# ipcmk -M 2000
```

```
Shared memory id: 0
```

```
root@ip-172-26-8-43:/tmp# ipcs -m
```

```
----- Shared Memory Segments -----  
key          shmid      owner      perms      bytes      nattch     status  
0xeb4b79db  0          root       644        2000       0          
```

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

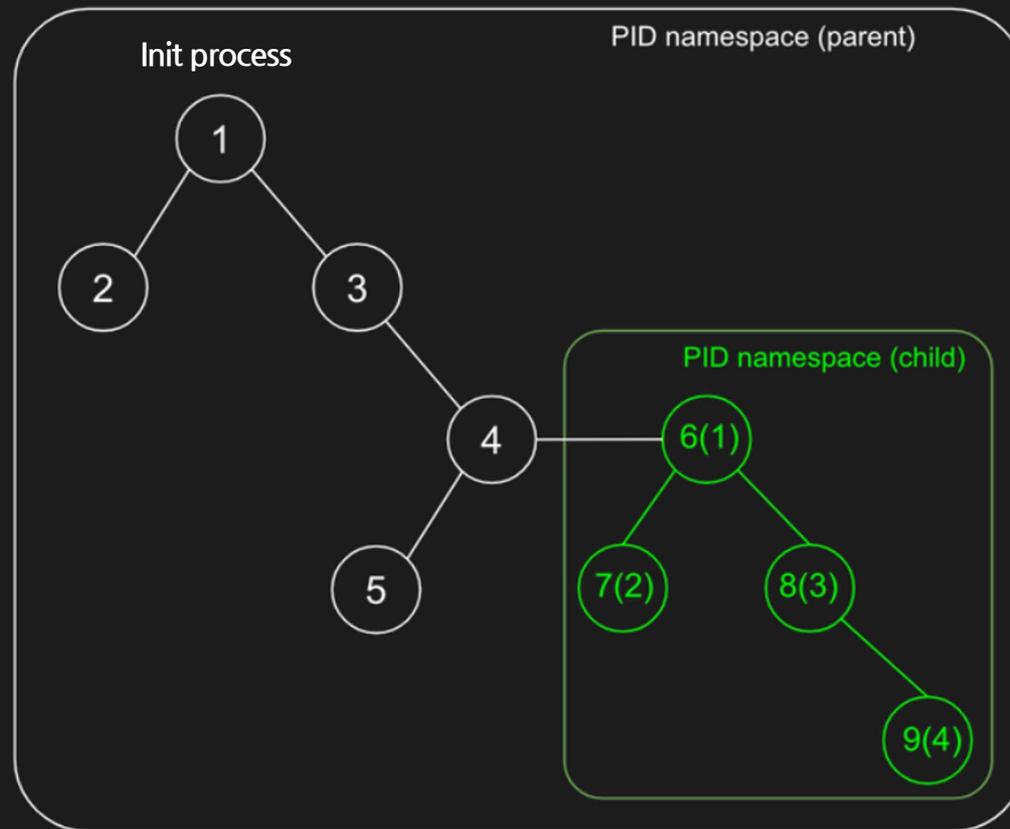
5-1 Overlay Filesystem mount

5-2 Container layout

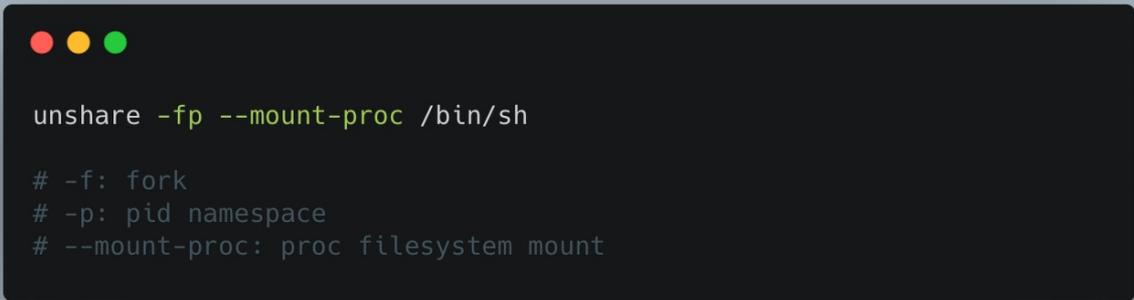
6 Making own container

6-1 Let's do it!

PID (process id) namespace



PID (process id) namespace



```
unshare -fp --mount-proc /bin/sh  
  
# -f: fork  
# -p: pid namespace  
# --mount-proc: proc filesystem mount
```

/proc

1. Memory-based virtual filesystem
2. System information that kernel manages
3. Used for System monitoring and analysis

PID (process id) namespace

```
root@ip-172-26-8-43:/tmp# echo $$
14466
root@ip-172-26-8-43:/tmp# unshare --pid --fork --mount-proc /bin/bash
root@ip-172-26-8-43:/tmp# echo $$
1
root@ip-172-26-8-43:/tmp# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	1.0	23028	4932	pts/0	S	15:21	0:00	/bin/bash
root	9	0.0	0.6	37804	3308	pts/0	R+	15:22	0:00	ps aux

```
ubuntu@ip-172-26-8-43:~$ ps aux | grep '/bin/bash'
```

root	14466	0.0	1.0	23028	4980	pts/0	S	15:20	0:00	/bin/bash
root	14476	0.0	0.1	7920	828	pts/0	S	15:21	0:00	unshare --pid --fork --mount-proc /bin/bash
root	14477	0.0	1.0	23028	4932	pts/0	S+	15:21	0:00	/bin/bash
ubuntu	14618	0.0	0.2	14860	1120	pts/1	S+	15:25	0:00	grep --color=auto /bin/bash

PID (process id) namespace

```
ubuntu@ip-172-26-8-43:~$ ps aux | grep '/bin/bash'
root      14466  0.0  1.0 23028 4980 pts/0    S   15:20   0:00 /bin/bash
root      14476  0.0  0.1  7920   828 pts/0    S   15:21   0:00 unshare --pid --fork --mount-proc /bin/bash
root      14477  0.0  1.0 23028 4932 pts/0    S+  15:21   0:00 /bin/bash
ubuntu    14618  0.0  0.2 14860 1120 pts/1    S+  15:25   0:00 grep --color=auto /bin/bash
```

```
readlink /proc/<target-pid>/ns/pid
```

```
root@ip-172-26-8-43:~# readlink /proc/14768/ns/pid
pid:[4026532224]
```

```
root@ip-172-26-8-43:/tmp# readlink /proc/1/ns/pid
pid:[4026532224]
```

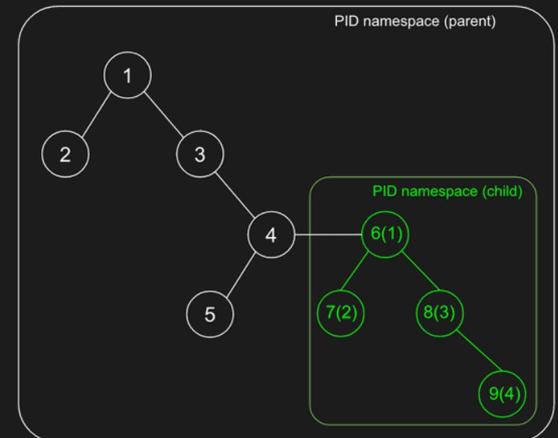


Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

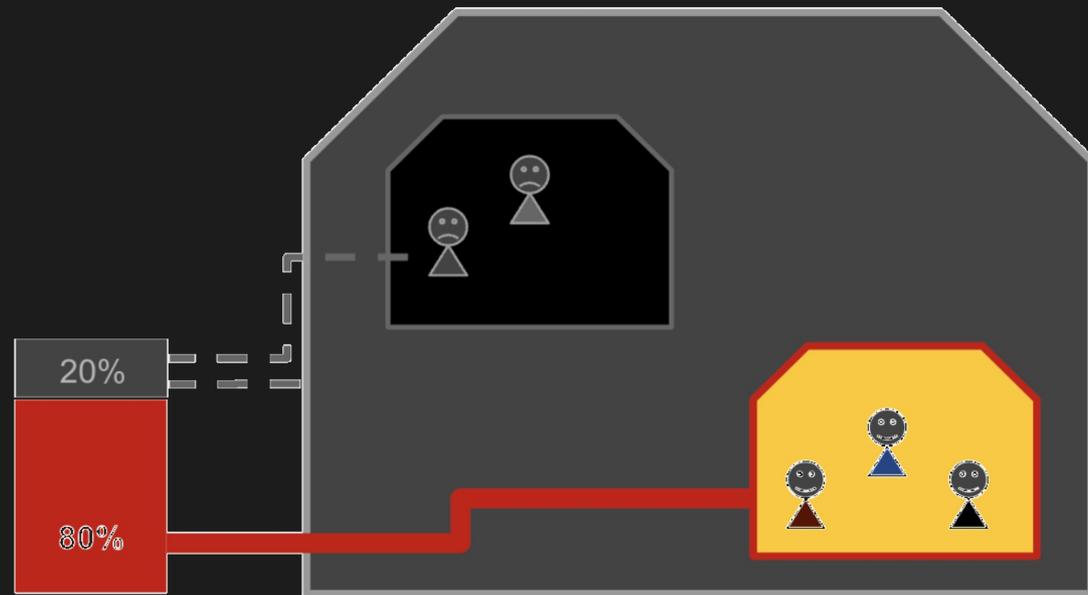
5-2 Container layout

6 Making own container

6-1 Let's do it!

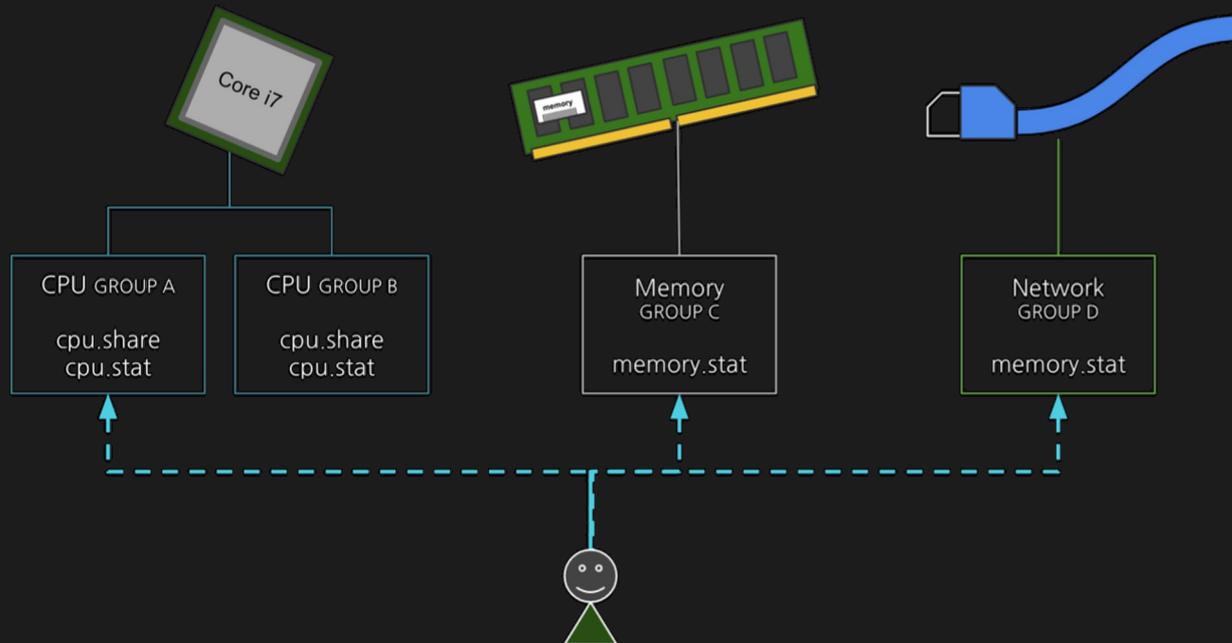
cgroup (control group) **namespace**

Namespace isolates and manages system resources, such as CPU, memory, disk I/O, and network bandwidth, for different groups of processes



cgroup (control group) namespace

Namespace isolates and manages system resources, such as CPU, memory, disk I/O, and network bandwidth, for different groups of processes



cgroup (control group) namespace

cgroup filesystem: `/sys/fs/cgroup`

```
ubuntu@ip-172-26-8-43:~$ tree -L 1 /sys/fs/cgroup
/sys/fs/cgroup
├── blkio
├── cpu -> cpu,cpuacct
├── cpu,cpuacct
├── cpuacct -> cpu,cpuacct
├── cpuset
├── devices
├── freezer
├── hugetlb
├── memory
├── net_cls -> net_cls,net_prio
├── net_cls,net_prio
├── net_prio -> net_cls,net_prio
├── perf_event
├── pids
├── rdma
├── systemd
└── unified

17 directories, 0 files
```

cgroup (control group) namespace

cgroup filesystem: `/sys/fs/cgroup`

```
ubuntu@ip-172-26-8-43:~$ tree -L 1 /sys/fs/cgroup/cpu
/sys/fs/cgroup/cpu
├── cgroup.clone_children
├── cgroup.procs
├── cgroup.sane_behavior
├── cpu.cfs_period_us
├── cpu.cfs_quota_us
├── cpu.shares
├── cpu.stat
├── cpuacct.stat
├── cpuacct.usage
├── cpuacct.usage_all
├── cpuacct.usage_percpu
├── cpuacct.usage_percpu_sys
├── cpuacct.usage_percpu_user
├── cpuacct.usage_sys
├── cpuacct.usage_user
├── notify_on_release
├── release_agent
├── system.slice
├── tasks
└── user.slice
```

2 directories, 18 files

cgroup (control group) namespace

```
root@ip-172-26-8-43:~# cgcreate -a root -g cpu:jaehong21
root@ip-172-26-8-43:~# tree /sys/fs/cgroup/cpu/jaehong21
/sys/fs/cgroup/cpu/jaehong21
├── cgroup.clone_children
├── cgroup.procs
├── cpu.cfs_period_us
├── cpu.cfs_quota_us
├── cpu.shares
├── cpu.stat
├── cpu.uclamp.max
├── cpu.uclamp.min
├── cpuacct.stat
├── cpuacct.usage
├── cpuacct.usage_all
├── cpuacct.usage_percpu
├── cpuacct.usage_percpu_sys
├── cpuacct.usage_percpu_user
├── cpuacct.usage_sys
├── cpuacct.usage_user
├── notify_on_release
└── tasks

0 directories, 18 files
root@ip-172-26-8-43:~#
root@ip-172-26-8-43:~# cgset -r cpu.cfs_quota_us=30000 jaehong21
```

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

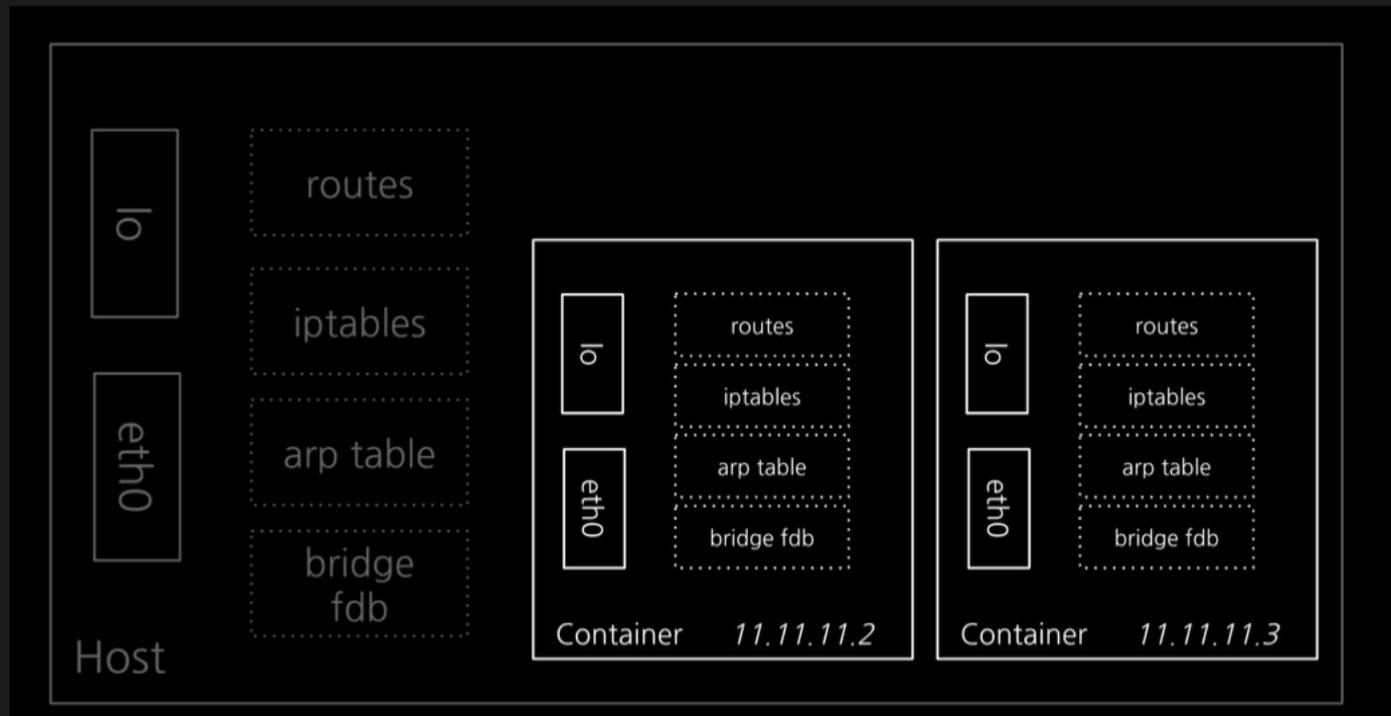
5-2 Container layout

6 Making own container

6-1 Let's do it!

Network namespace

Namespace provides an isolated network stack for a group of processes.



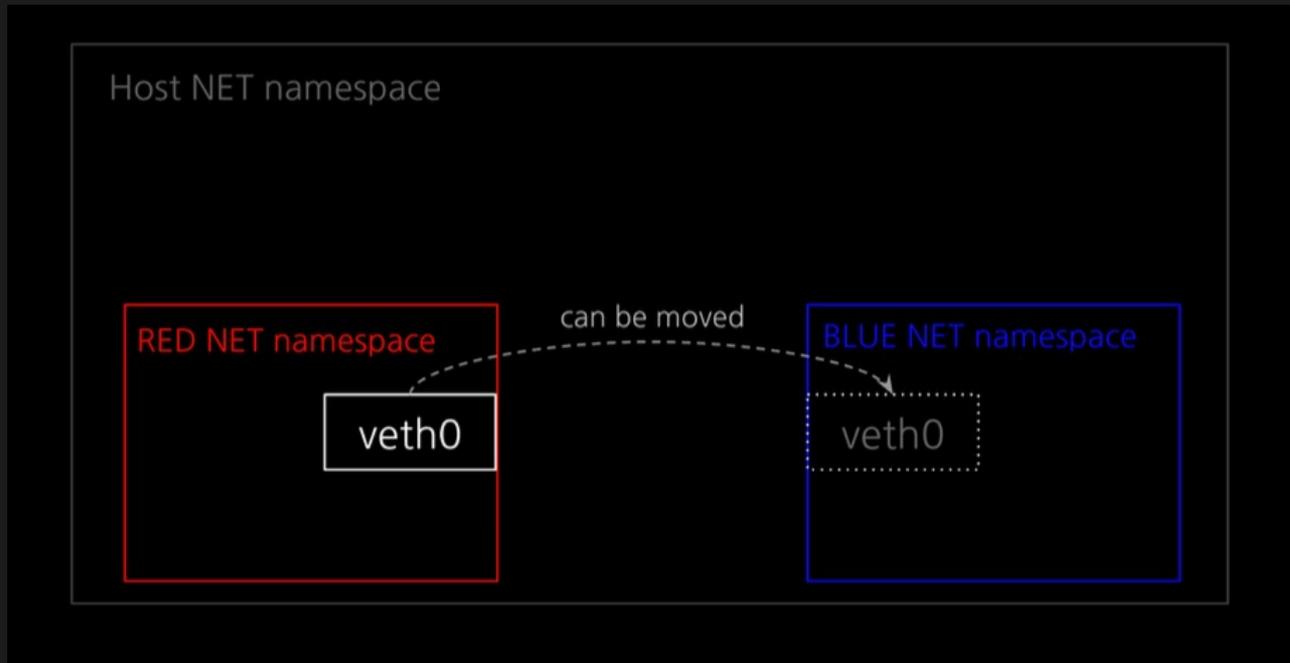
Network namespace

Namespace provides an isolated network stack for a group of processes.

```
root@ip-172-26-8-43:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
   link/ether 02:07:39:a7:61:80 brd ff:ff:ff:ff:ff:ff
   inet 172.26.8.43/20 brd 172.26.15.255 scope global dynamic eth0
       valid_lft 2549sec preferred_lft 2549sec
   inet6 2406:da12:6ab:ee00:109e:caca:eef1:e04f/128 scope global dynamic noprefixroute
       valid_lft 437sec preferred_lft 127sec
   inet6 fe80::7:39ff:fea7:6180/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
   link/ether 02:42:12:ab:d4:51 brd ff:ff:ff:ff:ff:ff
   inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
root@ip-172-26-8-43:~# unshare --net /bin/bash
root@ip-172-26-8-43:~# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
root@ip-172-26-8-43:~#
```

Network Interface

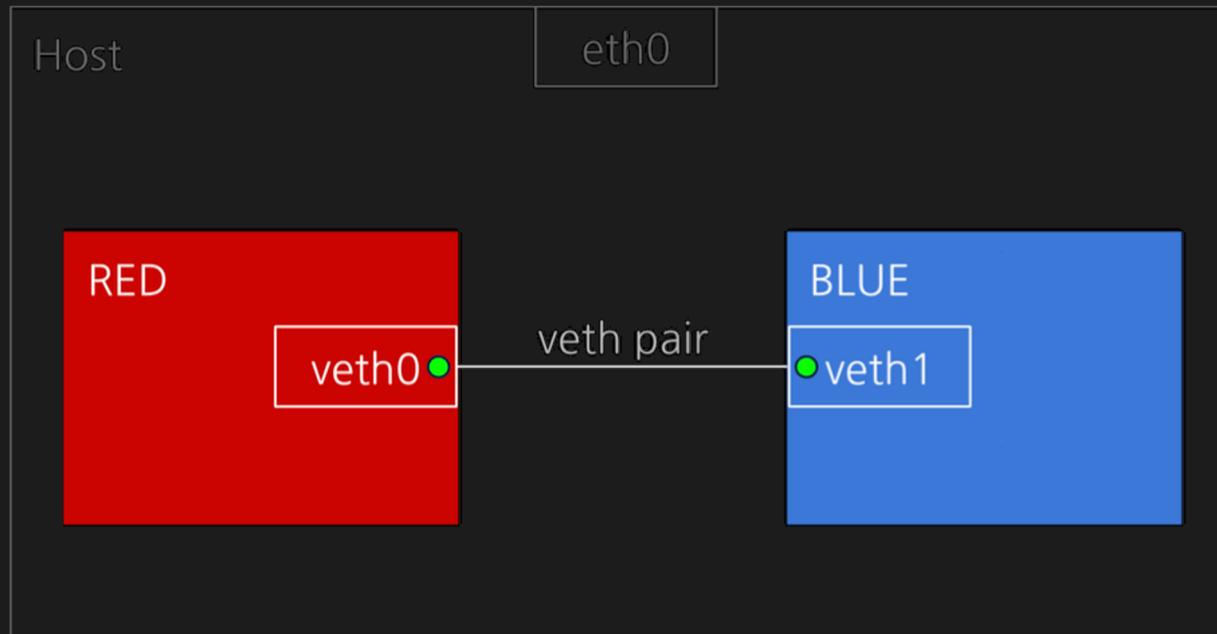
ex. veth, bridge, vxlan, ...



1. Cannot be present in multiple network namespaces
2. Can move to another network namespaces

Network Interface

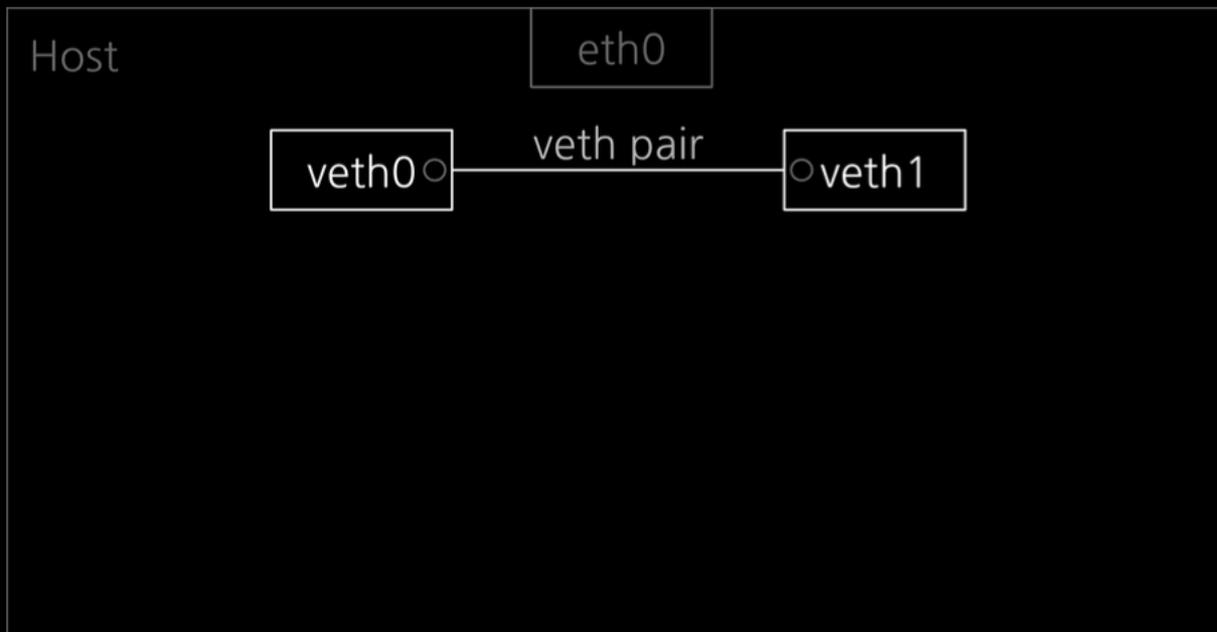
1. Isolate Network namespaces
2. Communication between RED & BLUE container



Network namespace



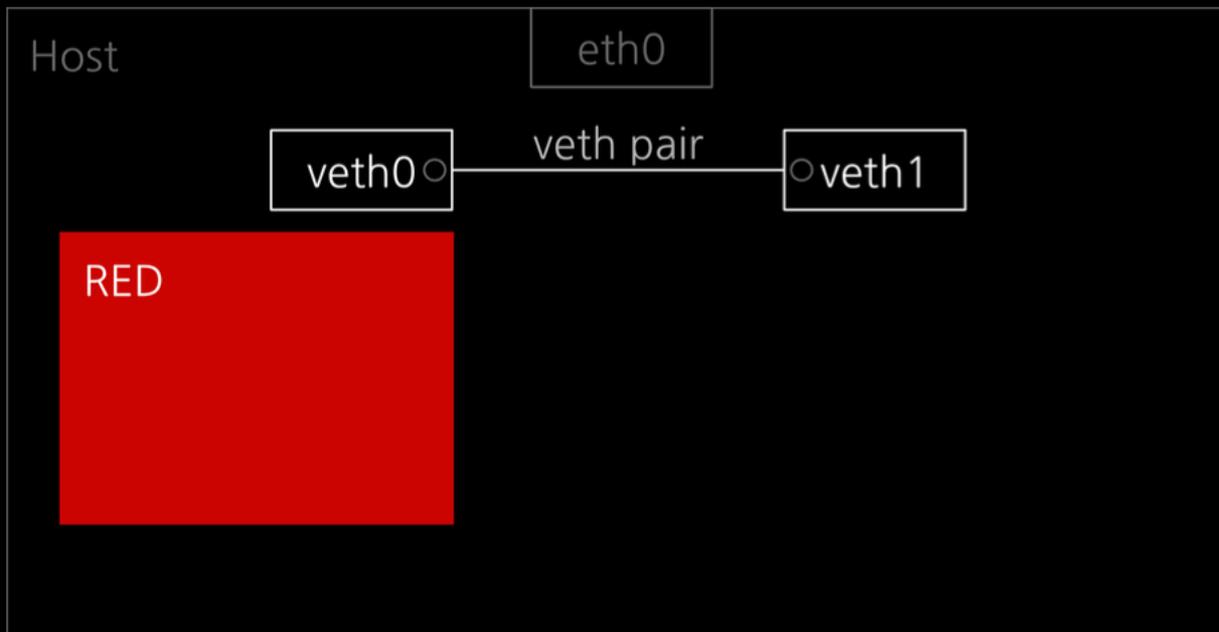
```
ip link add veth0 type veth peer name veth1
```



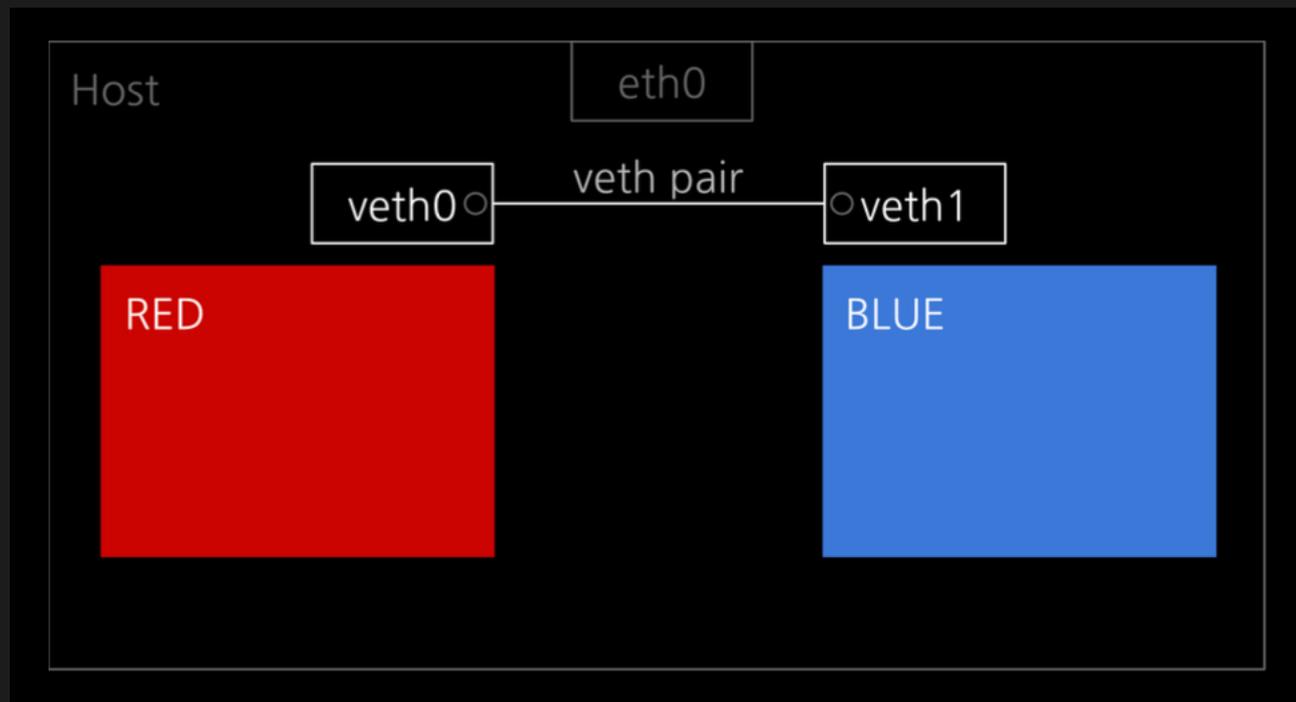
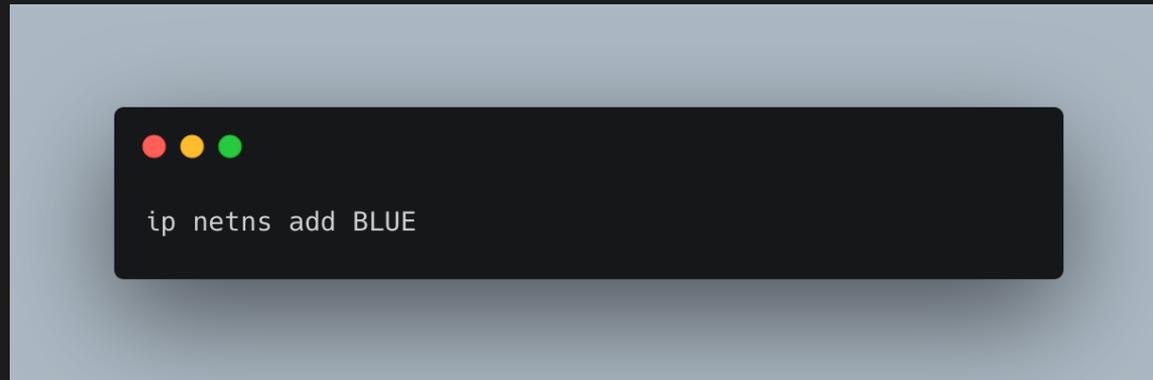
Network namespace



```
ip netns add RED
```



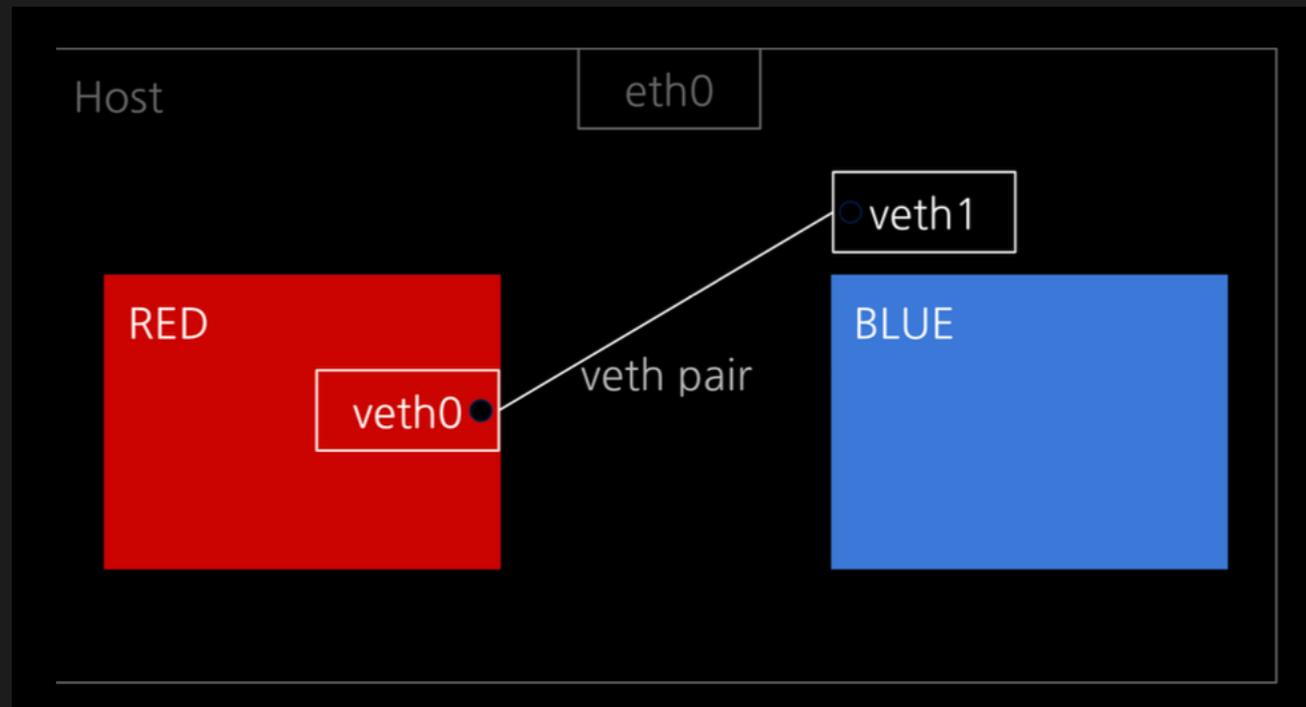
Network namespace



Network namespace



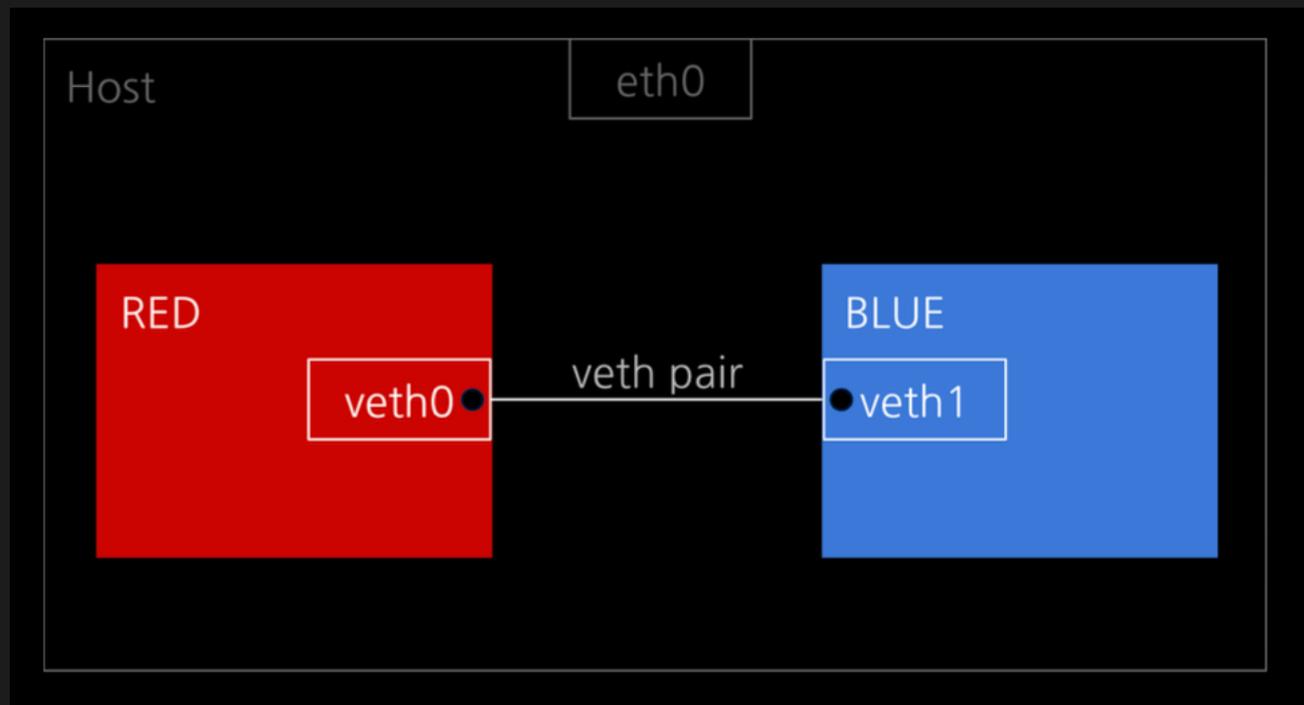
```
ip link set veth0 netns RED
```



Network namespace



```
ip link set veth1 netns BLUE
```



Network namespace



```
ip link set veth1 netns BLUE
```

Host

eth0

RED

BLUE

veth0

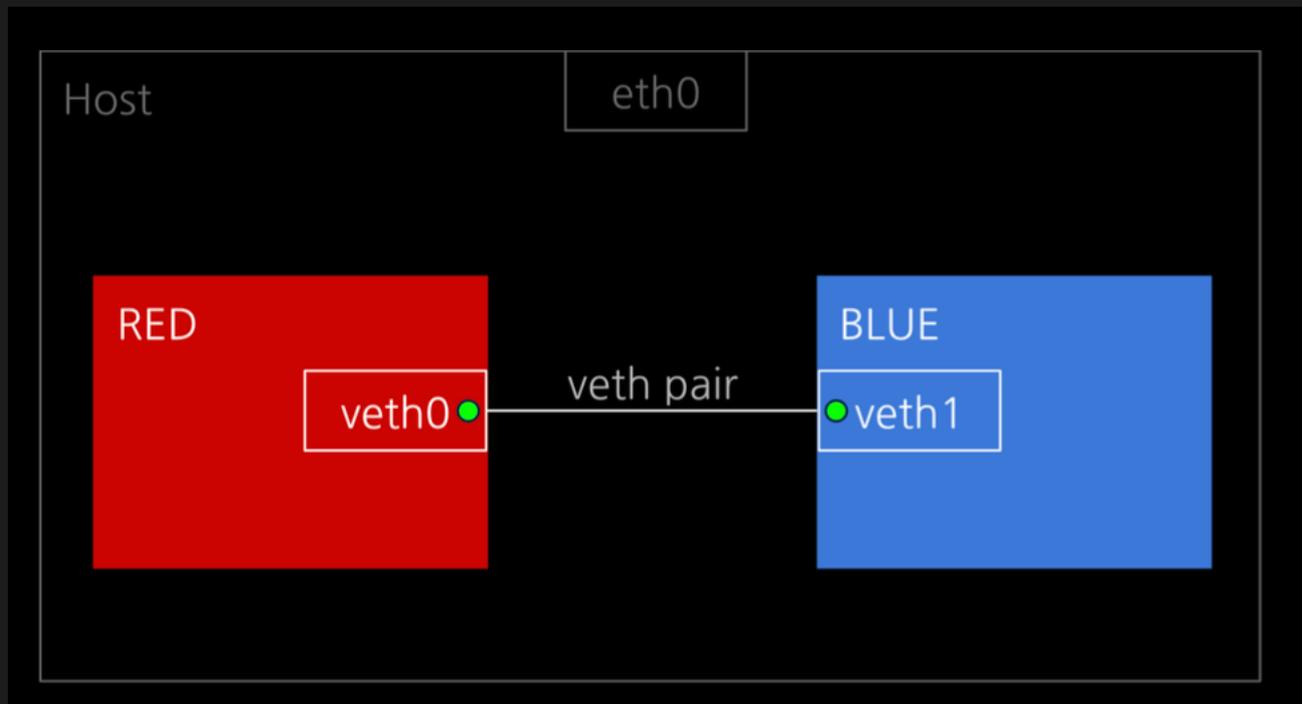
veth pair

veth1

Network namespace



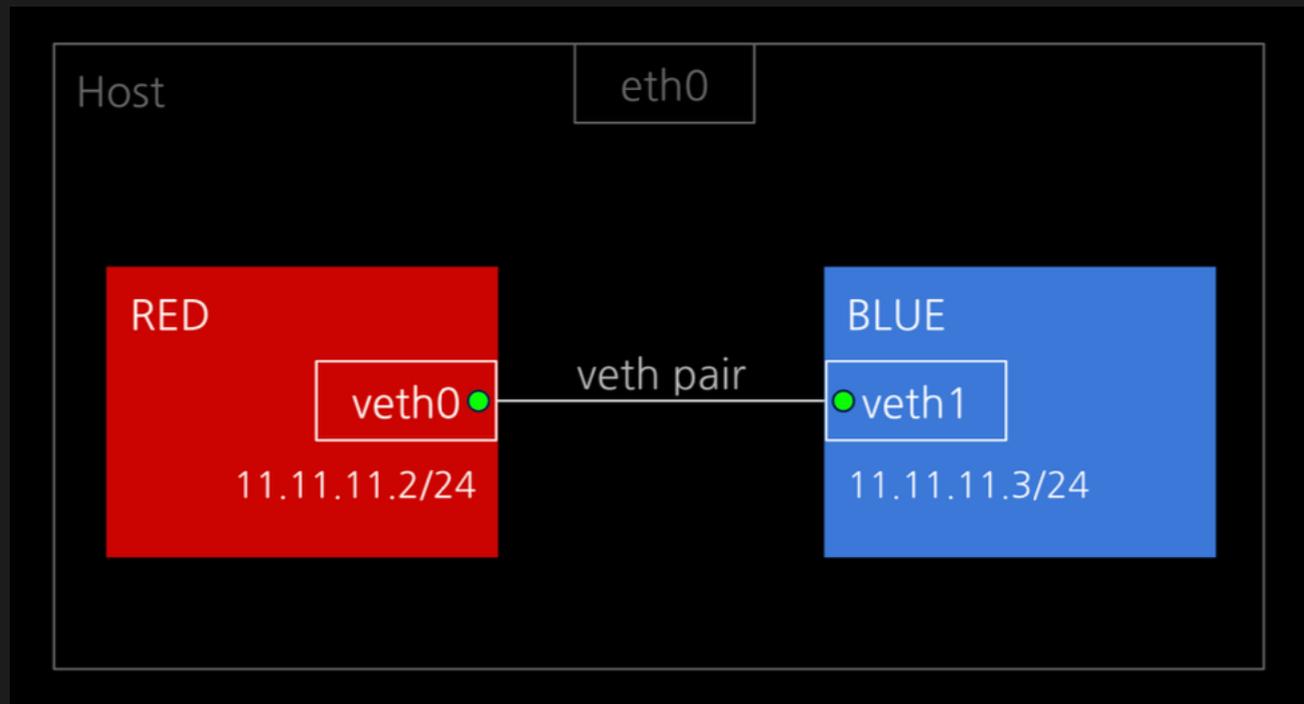
```
ip netns exec RED ip link set veth0 up  
ip netns exec BLUE ip link set veth1 up
```



Network namespace



```
ip netns exec RED ip addr add 11.11.11.2/24 dev veth0  
ip netns exec BLUE ip addr add 11.11.11.3/24 dev veth1
```



Network namespace

```
ubuntu@ip-172-26-8-43:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 02:07:39:a7:61:80 brd ff:ff:ff:ff:ff:ff
    inet 172.26.8.43/20 brd 172.26.15.255 scope global dynamic eth0
        valid_lft 1949sec preferred_lft 1949sec
    inet6 2406:da12:6ab:ee00:109e:caca:eef1:e04f/128 scope global dynamic noprefixroute
        valid_lft 417sec preferred_lft 107sec
    inet6 fe80::7:39ff:fea7:6180/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:12:ab:d4:51 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
ubuntu@ip-172-26-8-43:~$ ip route
default via 172.26.0.1 dev eth0 proto dhcp src 172.26.8.43 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
172.26.0.0/20 dev eth0 proto kernel scope link src 172.26.8.43
172.26.0.1 dev eth0 proto dhcp scope link src 172.26.8.43 metric 100
```

Network namespace

```
root@ip-172-26-8-43:~# nsenter --net=/var/run/netns/RED
root@ip-172-26-8-43:~# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: veth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
   link/ether e2:c7:92:10:ad:d3 brd ff:ff:ff:ff:ff:ff link-netnsid 1
   inet 11.11.11.2/24 scope global veth0
       valid_lft forever preferred_lft forever
   inet6 fe80::e0c7:92ff:fe10:ad3/64 scope link
       valid_lft forever preferred_lft forever
root@ip-172-26-8-43:~# ip route
11.11.11.0/24 dev veth0 proto kernel scope link src 11.11.11.2
root@ip-172-26-8-43:~#
```

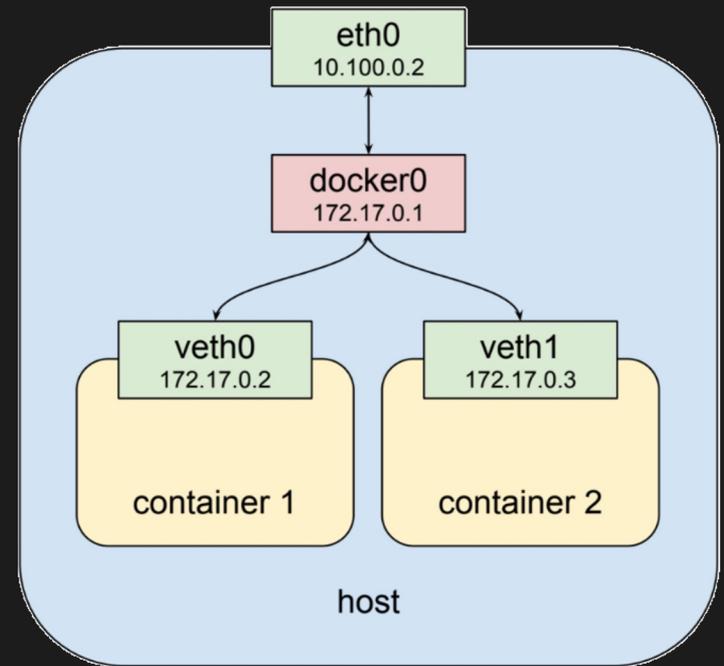
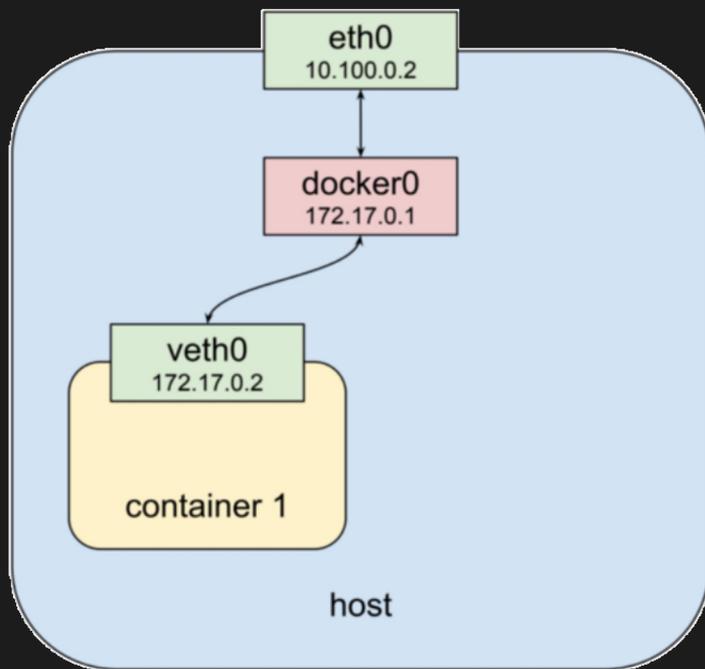
```
root@ip-172-26-8-43:~# nsenter --net=/var/run/netns/BBLUE
root@ip-172-26-8-43:~# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4: veth1@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
   link/ether aa:68:e9:18:4c:7e brd ff:ff:ff:ff:ff:ff link-netnsid 0
   inet 11.11.11.3/24 scope global veth1
       valid_lft forever preferred_lft forever
   inet6 fe80::a868:e9ff:fe18:4c7e/64 scope link
       valid_lft forever preferred_lft forever
root@ip-172-26-8-43:~# ip route
11.11.11.0/24 dev veth1 proto kernel scope link src 11.11.11.3
root@ip-172-26-8-43:~#
```

Network namespace

```
root@ip-172-26-8-43:~# ping 11.11.11.3
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data.
64 bytes from 11.11.11.3: icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from 11.11.11.3: icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from 11.11.11.3: icmp_seq=3 ttl=64 time=0.045 ms
^C
--- 11.11.11.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.032/0.040/0.045/0.007 ms
```

```
root@ip-172-26-8-43:~# tcpdump -li veth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth1, link-type EN10MB (Ethernet), capture size 262144 bytes
20:38:42.964454 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 17264, seq 1, length 64
20:38:42.964473 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 17264, seq 1, length 64
20:38:43.990744 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 17264, seq 2, length 64
20:38:43.990766 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 17264, seq 2, length 64
20:38:45.014749 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 17264, seq 3, length 64
20:38:45.014772 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 17264, seq 3, length 64
20:38:48.214682 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
20:38:48.214694 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
20:38:48.214730 ARP, Reply 11.11.11.3 is-at aa:68:e9:18:4c:7e (oui Unknown), length 28
20:38:48.214726 ARP, Reply 11.11.11.2 is-at e2:c7:92:10:ad:d3 (oui Unknown), length 28
```

Network namespace



Network namespace

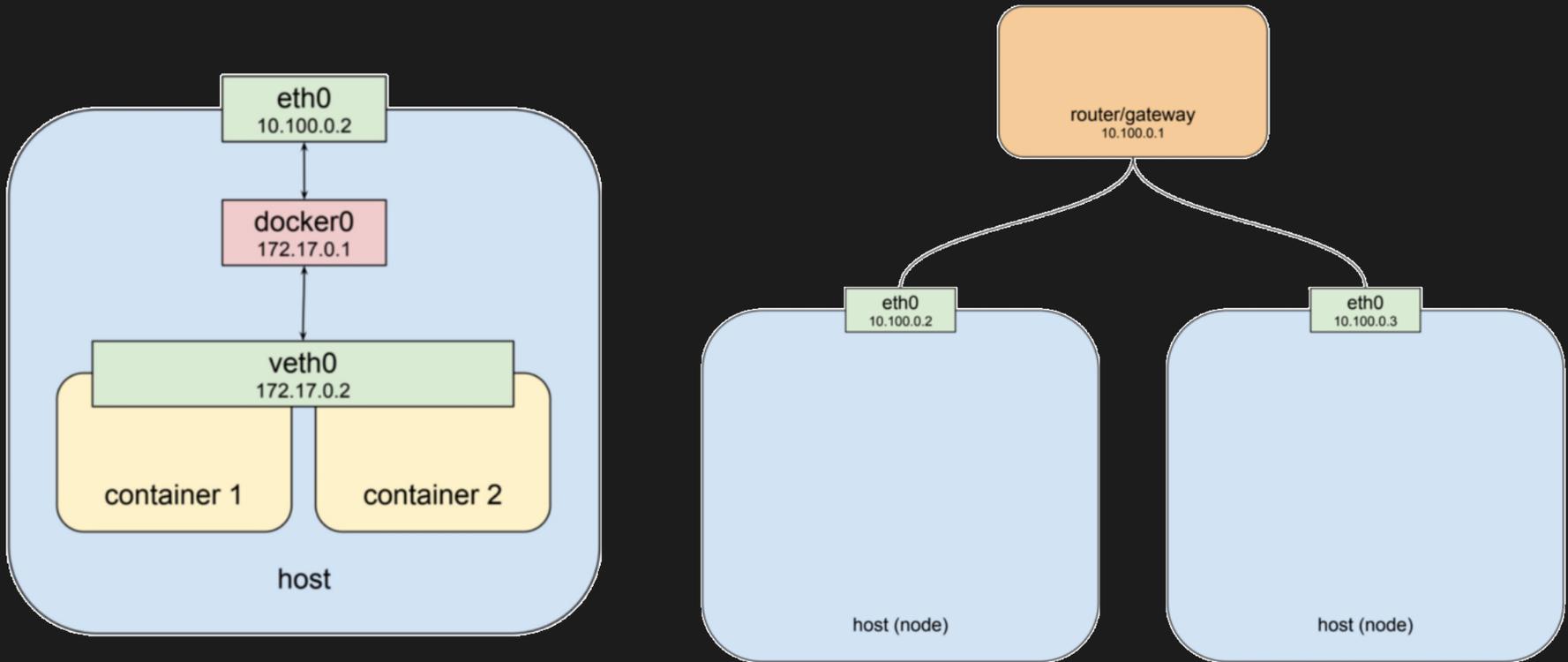


Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

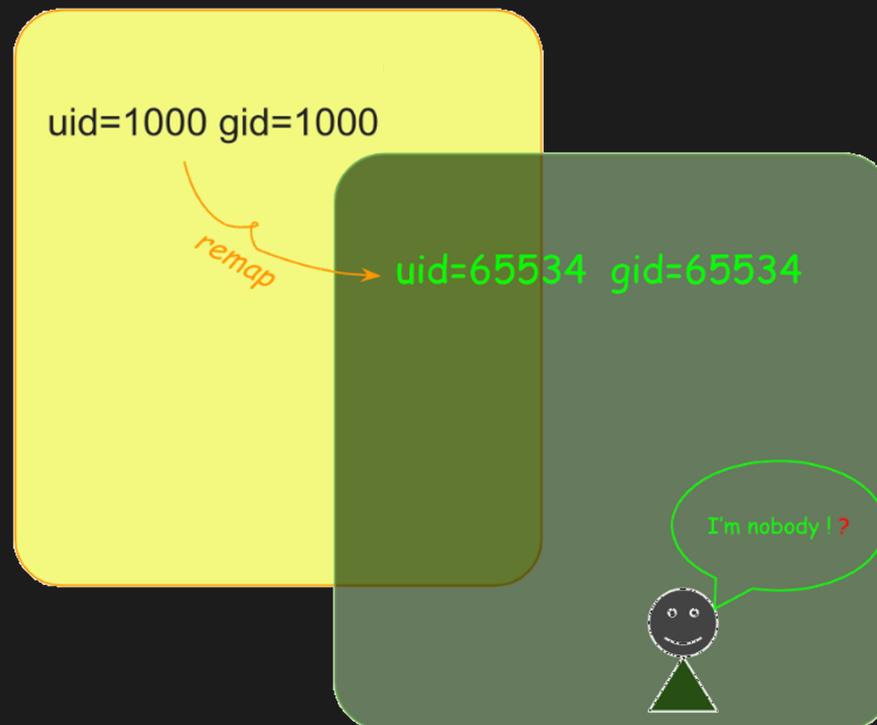
5-2 Container layout

6 Making own container

6-1 Let's do it!

USER namespace

Namespace provides an
isolated ID space for group of processes

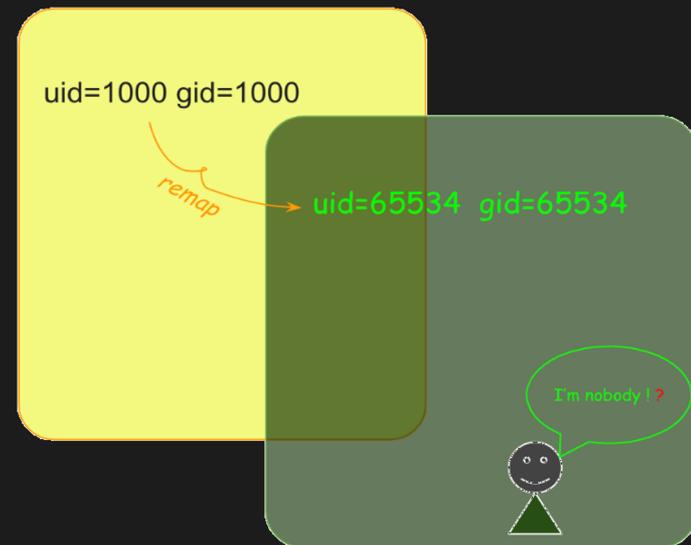


USER namespace

```
ubuntu@ip-172-26-8-43:~$ sudo -Es
root@ip-172-26-8-43:~# id
uid=0(root) gid=0(root) groups=0(root)
```

```
ubuntu@ip-172-26-8-43:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),
24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),108
(lxd),114(netdev),999(docker)
```

```
ubuntu@ip-172-26-8-43:~$ unshare --user /bin/bash
nobody@ip-172-26-8-43:~$ id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
```



USER namespace

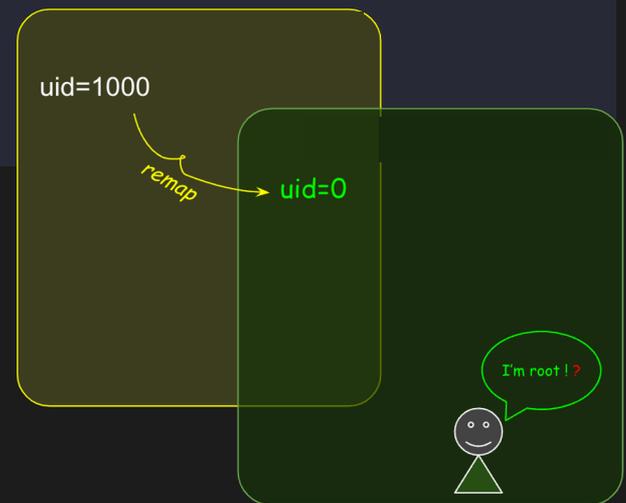
```
nobody@ip-172-26-8-43:~$ id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
nobody@ip-172-26-8-43:~$
nobody@ip-172-26-8-43:~$ unshare --uts /bin/bash
unshare: unshare failed: Operation not permitted
nobody@ip-172-26-8-43:~$
nobody@ip-172-26-8-43:~$ capsh --print | grep Current
Current: =
nobody@ip-172-26-8-43:~$
nobody@ip-172-26-8-43:~$ echo $$
18864
```

```
ubuntu@ip-172-26-8-43:~$ ps -ef | grep '/bin/bash'
ubuntu    18864  18616  0 06:23 pts/0    00:00:00 /bin/bash
ubuntu    19132  19115  0 06:41 pts/1    00:00:00 grep --color=auto /bin/bash
```

USER namespace

```
sudo echo '0 1000 1' > /proc/18864/uid_map
```

```
nobody@ip-172-26-8-43:~$ id
uid=0(root) gid=65534(nogroup) groups=65534(nogroup)
nobody@ip-172-26-8-43:~$
nobody@ip-172-26-8-43:~$ capsh --print | grep Current
Current: = cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read+ep
nobody@ip-172-26-8-43:~$
nobody@ip-172-26-8-43:~$ unshare --uts /bin/bash
root@ip-172-26-8-43:~#
root@ip-172-26-8-43:~# hostname jaehong21
root@ip-172-26-8-43:~# hostname
jaehong21
```



USER namespace



[Home](#)
[Guides](#)
[Manuals](#)
[Reference](#)
[Samples](#)
[Contribute](#)

[Home](#) / [Manuals](#) / [Docker Engine](#) / [Security](#) / [Overview](#)

- Docker Desktop
- Docker Engine
 - Overview
 - Install
 - Storage
 - Networking
 - Working with Docker Engine
 - Logging
 - Security
 - Overview
 - Rootless mode
 - Docker security non-events
 - Protect the Docker daemon socket
 - Using certificates for repository client verification
 - Use trusted images
 - Antivirus software

you can use them out of the box. For instance, we ship a template that works with AppArmor and Red Hat comes with SELinux policies for Docker. These templates provide an extra safety net (even though it overlaps greatly with capabilities).

- You can define your own policies using your favorite access control mechanism.

Just as you can use third-party tools to augment Docker containers, including special network topologies or shared filesystems, tools exist to harden Docker containers without the need to modify Docker itself.

As of Docker 1.10 User Namespaces are supported directly by the docker daemon. This feature allows for the root user in a container to be mapped to a non uid-0 user outside the container, which can help to mitigate the risks of container breakout. This facility is available but not enabled by default.

Refer to the [daemon command](#) in the command line reference for more information on this feature. Additional information on the implementation of User Namespaces in Docker can be found in [this blog post](#).

Conclusions

Docker containers are, by default, quite secure; especially if you run your processes as non-privileged users inside the container.

You can add an extra layer of safety by enabling AppArmor, SELinux, GRSEC, or another appropriate hardening system.

If you think of ways to make docker more secure, we welcome feature requests, pull

Contents:

Page details

-  11 minute read
-  [Edit this page](#)
-  [Request changes](#)

Tags

- [Docker](#)
- [Docker documentation](#)
- [Security](#)

Contents

- [Kernel namespaces](#)
- [Control groups](#)
- [Docker daemon attack surface](#)
- [Linux kernel capabilities](#)
- [Docker Content Trust Signature Verification](#)
- [Other kernel security features](#)
- [Conclusions](#)
- [Related information](#)

Give feedback

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

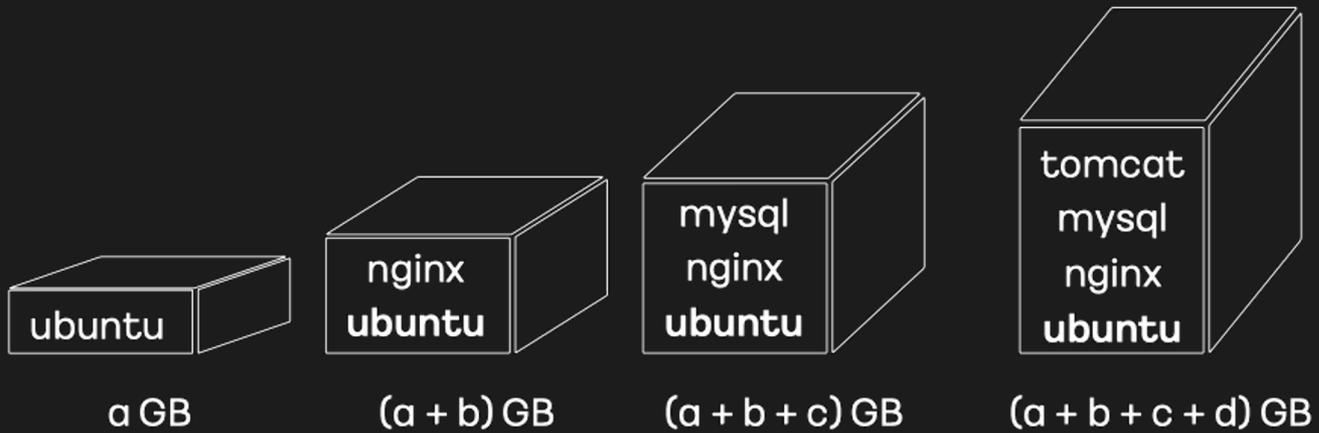
5-2 Container layout

6 Making own container

6-1 Let's do it!

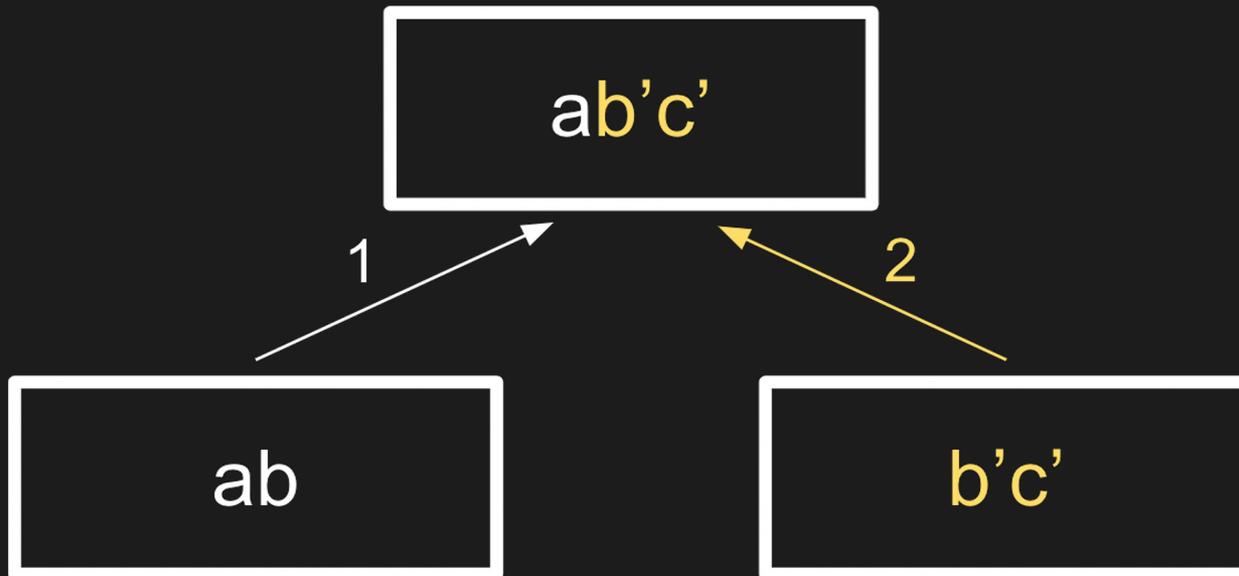
Overlay Filesystem

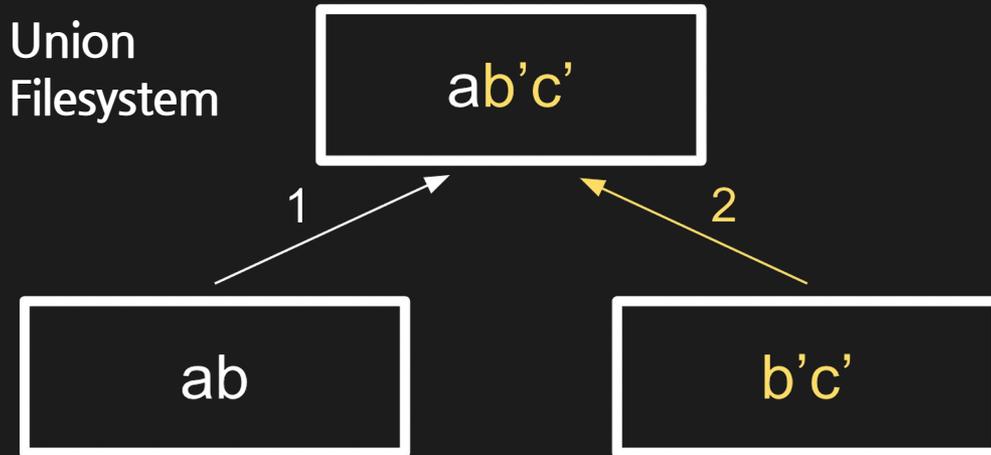
Duplicated File Problem



Overlay Filesystem

Union File system allows files and directories from multiple sources to be combined into a single, virtual file system

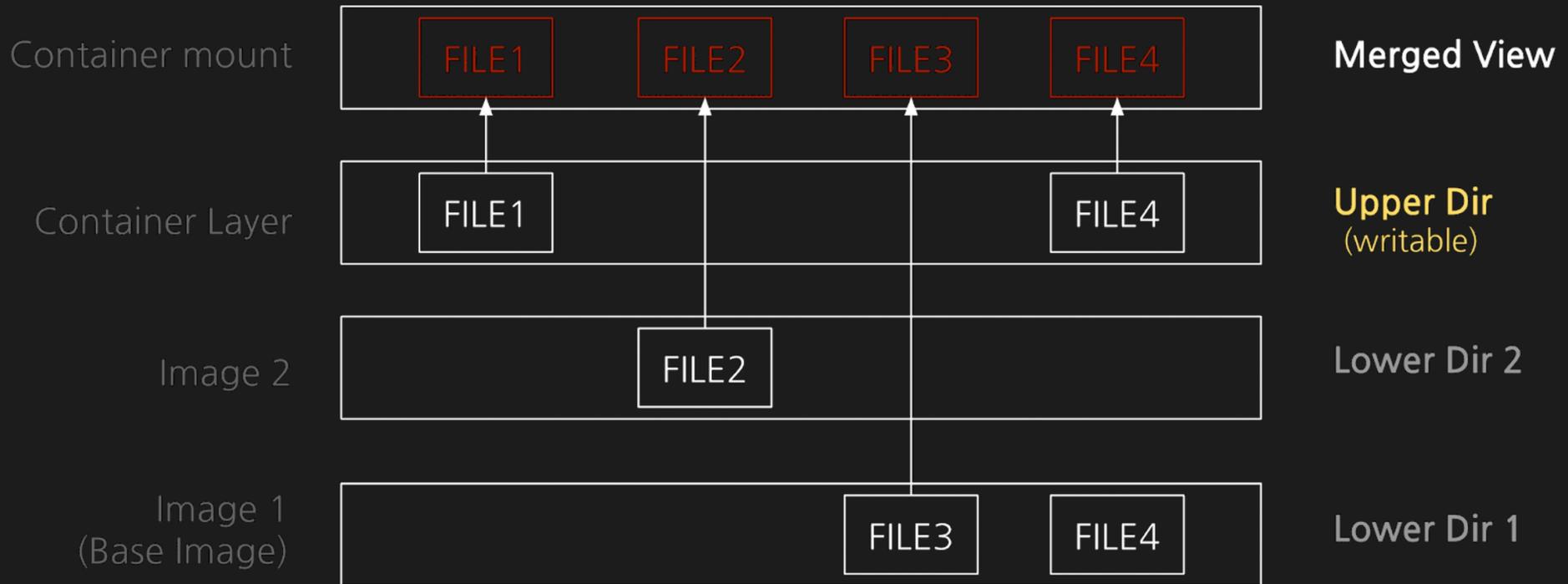




1. Combines multiple file systems into one by using a union mount
2. When two layers contain files with the same name, the file from the higher (more recently mounted) layer is overlaid on top of the file from the lower layer
3. When a write operation is performed on a file in a lower (read-only) layer, a copy is made before the write operation uses CoW(Copy-on-Write)

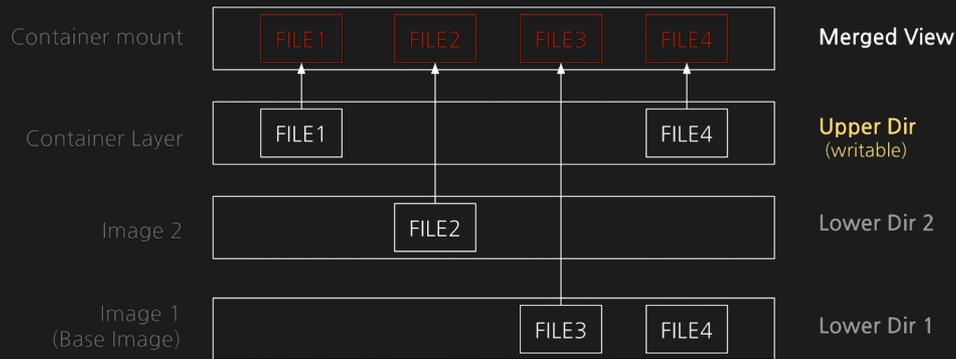
OverlayFS2

(AUFS → OverlayFS → OverlayFS2)



OverlayFS2

(AUFS → OverlayFS → OverlayFS2)



```
root@ubuntu1804:~# docker pull nginx@sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
docker.io/library/nginx@sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412: Pulling from library/nginx
f7ec5a41d630: Pull complete
aa1efa14b3bf: Pull complete
b78b95af9b17: Pull complete
c7d6bca2b8dc: Pull complete
cf16cd8e71e0: Pull complete
0241c68333ef: Pull complete
Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
Status: Downloaded newer image for nginx@sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
docker.io/library/nginx@sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
```

Overlay Filesystem

```
mkdir rootfs
cd rootfs

mkdir image1 image2 container work merge
touch image1/a image1/b image2/c

tree .
---
```

```
root@ip-172-26-8-43:~/tmp/rootfs# tree .
```

```
.
├── container
├── image1
│   ├── a
│   └── b
├── image2
│   └── c
├── merge
└── work
```

```
5 directories, 3 files
```

Overlay Filesystem



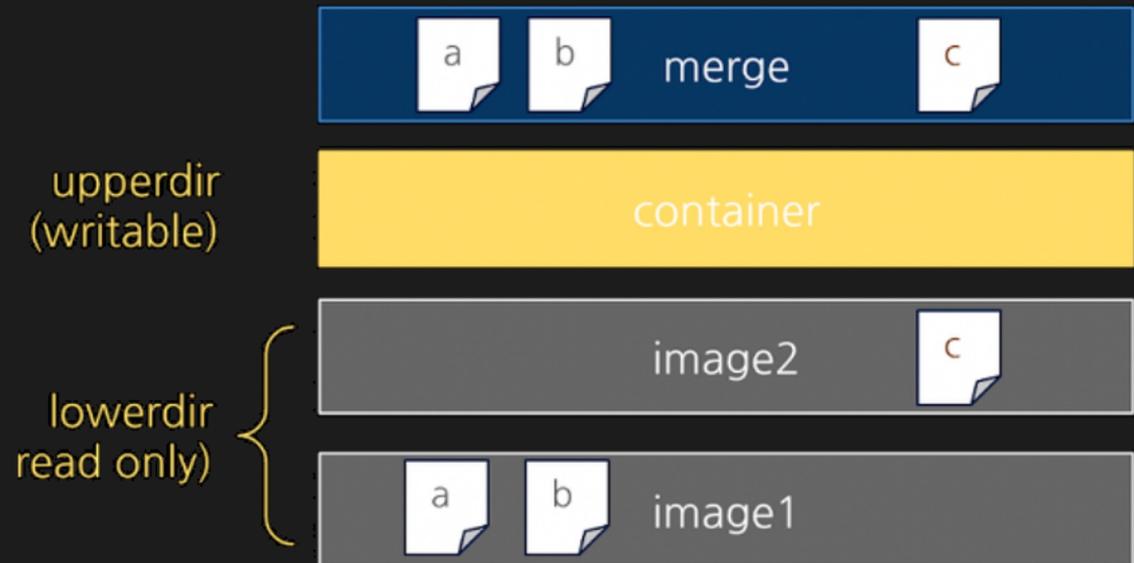
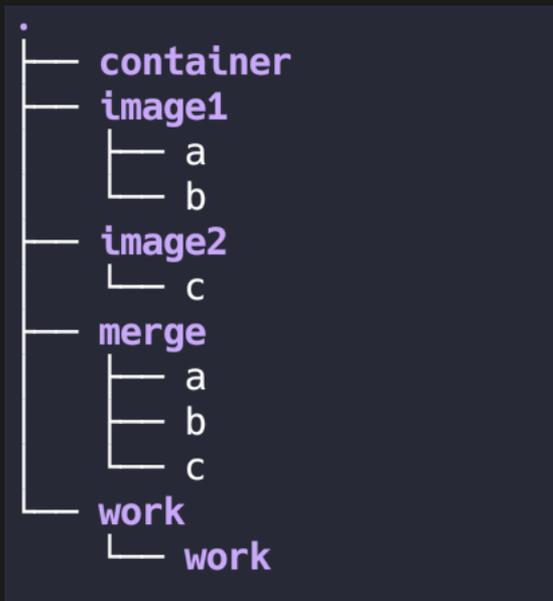
```
mount -t overlay overlay -o lowerdir=image2:image1,upperdir=container,workdir=work merge
```

```
root@ip-172-26-8-43:/tmp/rootfs# tree .
```

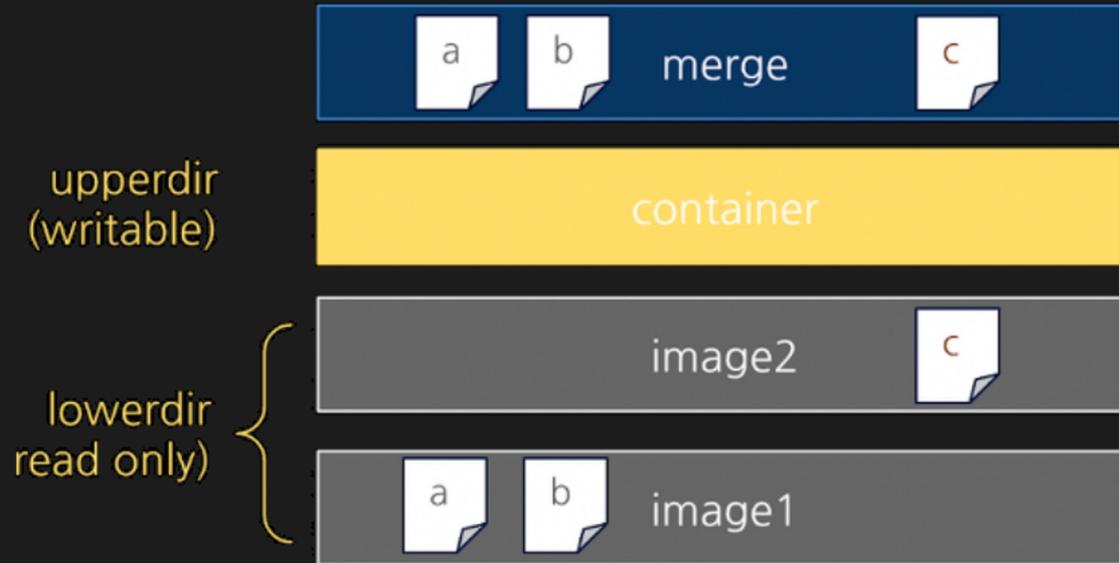
```
.
├── container
├── image1
│   ├── a
│   └── b
├── image2
│   └── c
├── merge
│   ├── a
│   ├── b
│   └── c
└── work
    └── work
```

```
6 directories, 6 files
```

Overlay Filesystem



Overlay Filesystem



1. What if delete merge/a ?
2. What if we edit merge/b?
3. What if we edit image1/b?

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

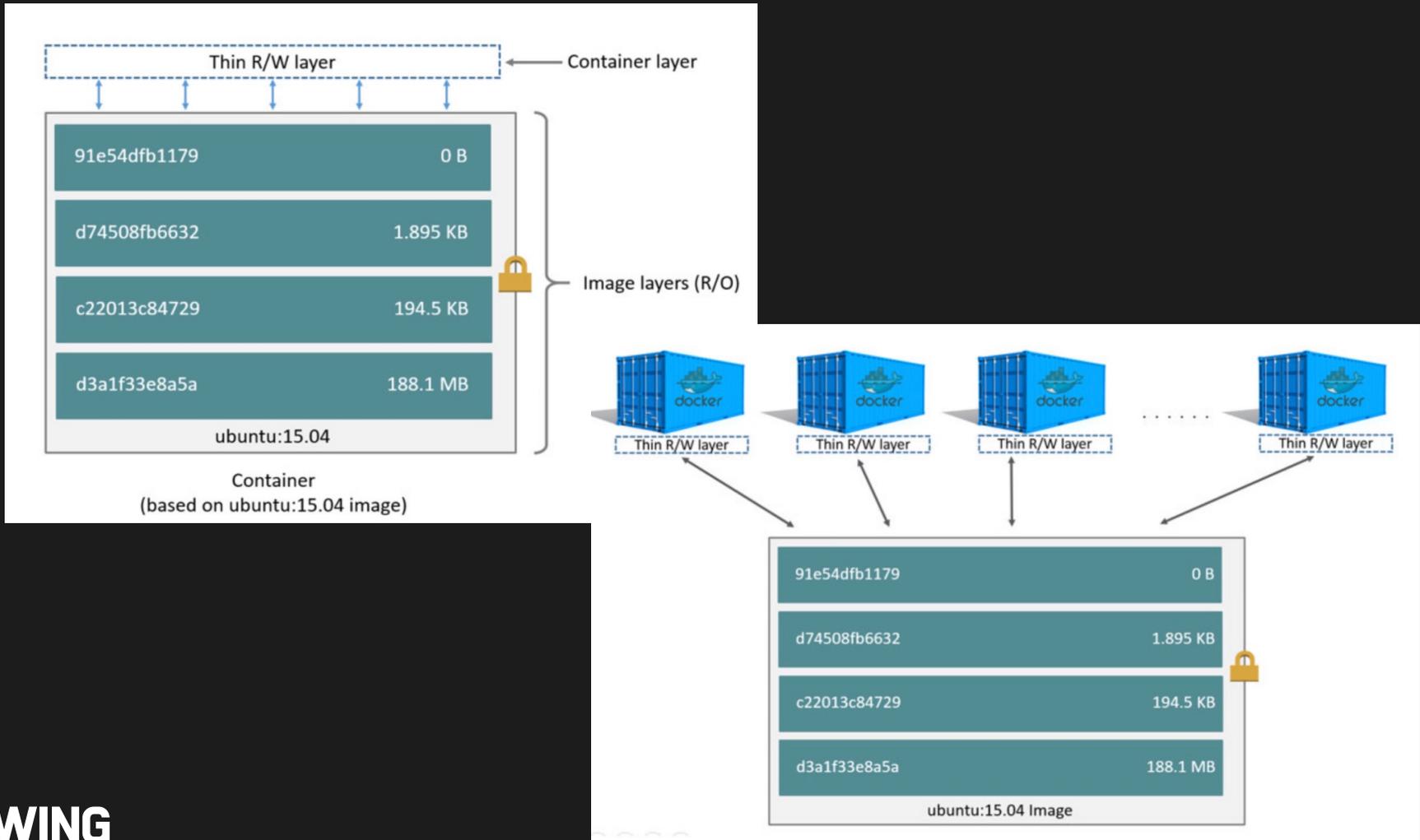
5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

Docker Layer



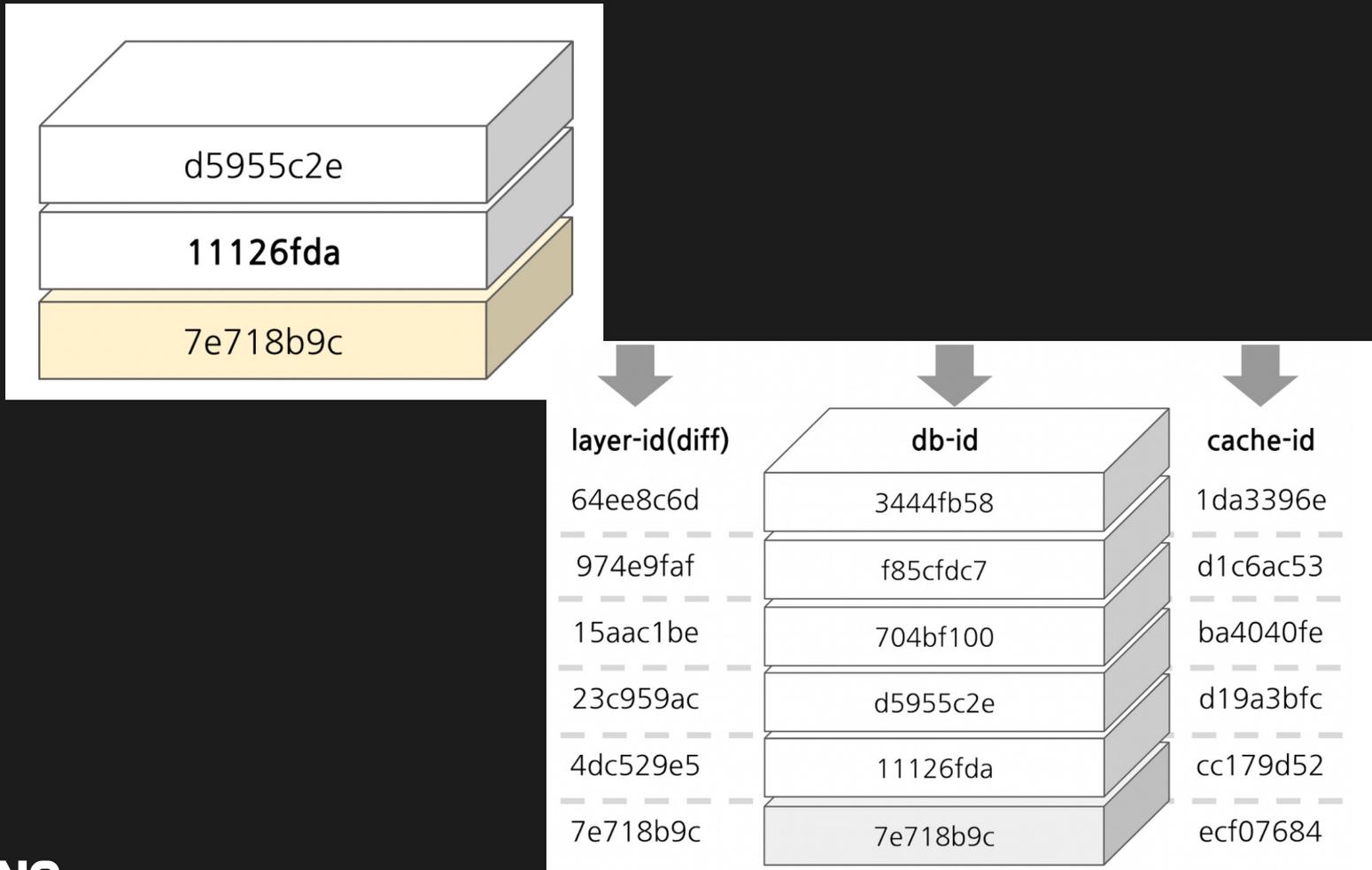
Docker
Layer

Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!

Let's do it

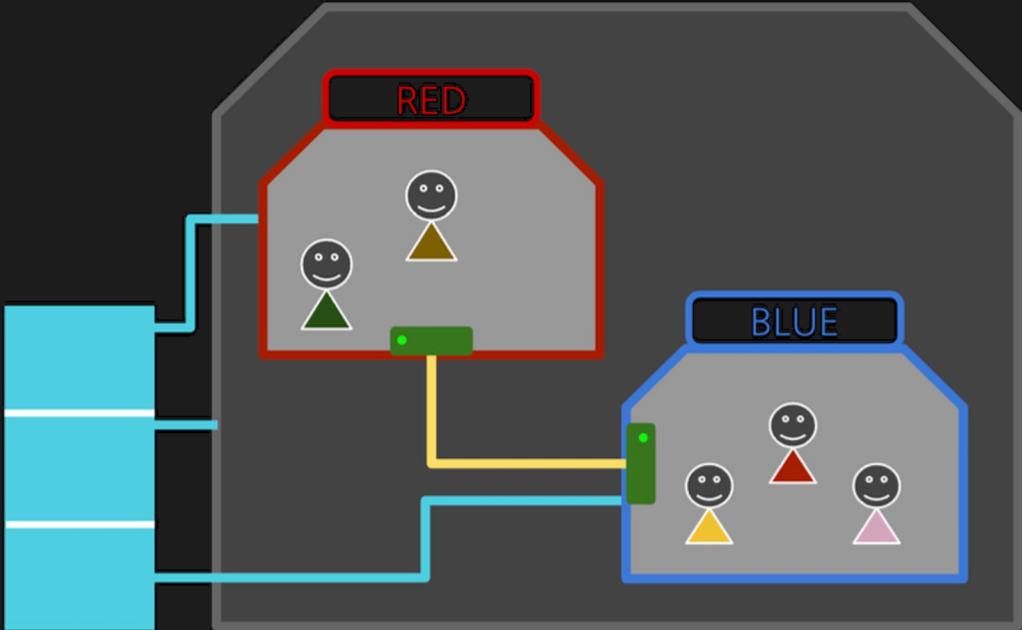


Table of Contents

1 Intro

1-1 What is Docker

1-2 Why Docker

2 chroot

2-1 What is chroot

2-2 Breaking out chroot jail

3 File system

3-1 mount and root filesystem

3-2 pivot_root

4 namespace

4-1 Mount namespace

4-2 UTS namespace

4-3 IPC namespace

4-4 PID namespace

4-5 cgroup namespace

4-6 Network namespace

4-7 USER namespace

5 Overlay Filesystem

5-1 Overlay Filesystem mount

5-2 Container layout

6 Making own container

6-1 Let's do it!



```
fn main() -> anyhow::Result<()> {  
    let a = String::from("I think we are all done");  
    println!("{}", a);  
  
    panic!("I'm done!");  
    Ok(())  
}
```