if(kakao)2022

이게 돼요? 도커 없이 컨테이너 만들기

김삼영 sam.0
카카오엔터프라이즈

과정소개

실습안내

컨테이너 맛보기

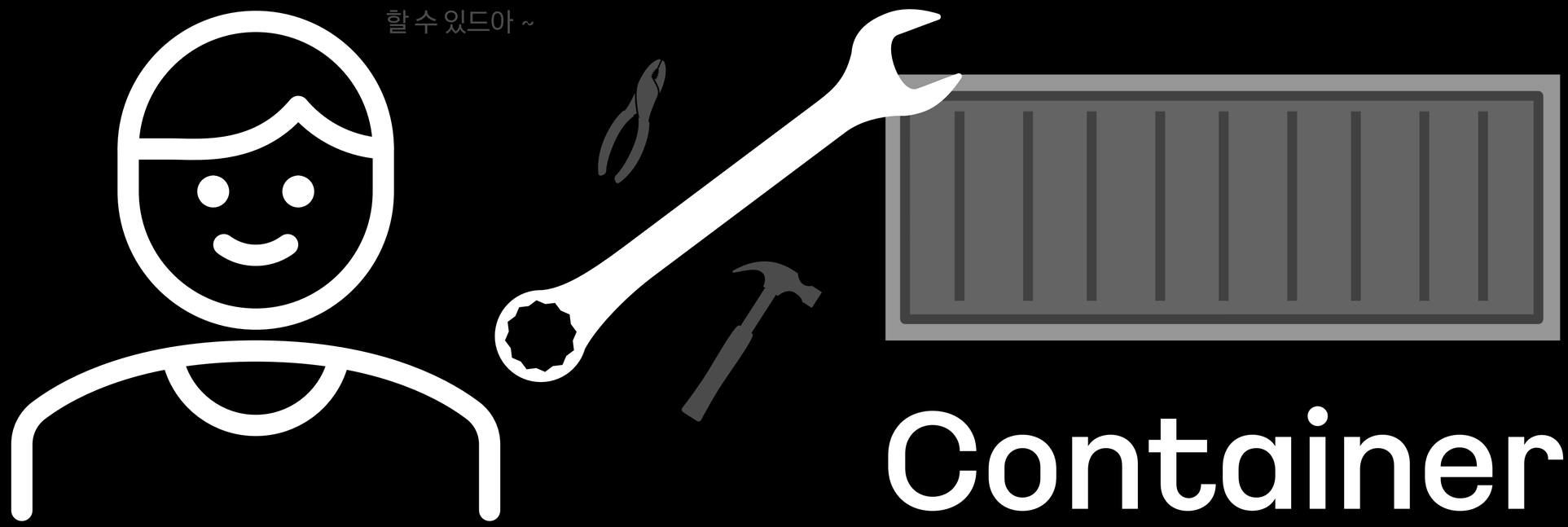
과정소개

실습안내

컨테이너 맛보기

도커 없이 ... 컨테이너 만들기

휘익~ docker



20억[1]

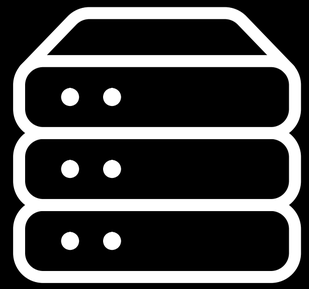
구글이 매주 배포하는 컨테이너 개수 (2014)

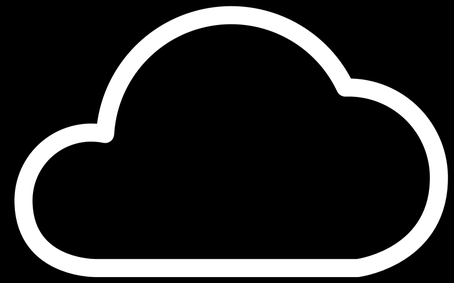
1) Containers at scale, https://speakerdeck.com/jbeda/containers-at-scale?slide=2

컨테이너 왜 써요?

컨테이너 왜 써요 ?

다양한 서버 환경





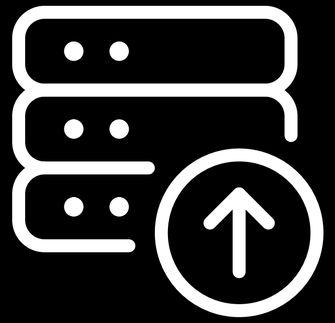
물리 머신 + 클라우드

서버사양, OS종류, 설치환경, ...

컨테이너 왜 써요 ?

운영 비용 증가

환경별 대응 어려움

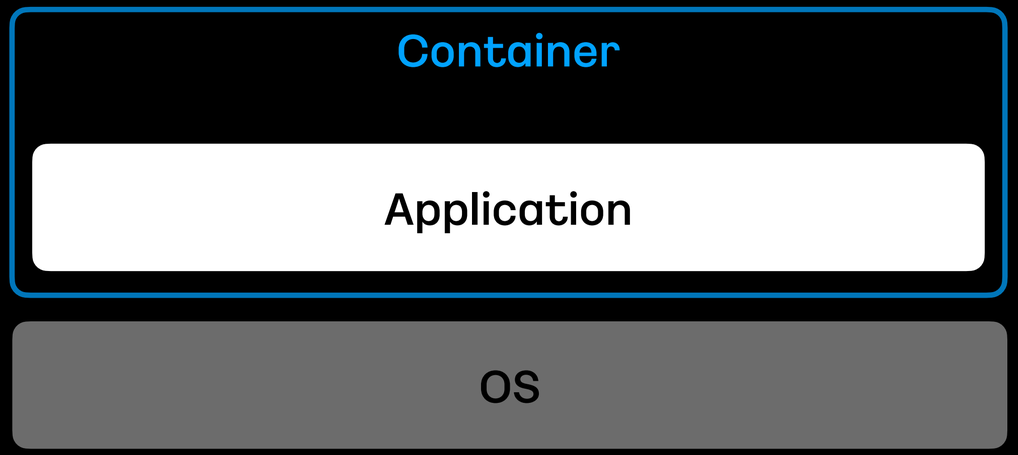


애플리케이션이 동작하는 환경

컨테이너 왜 써요 ?

서버 환경

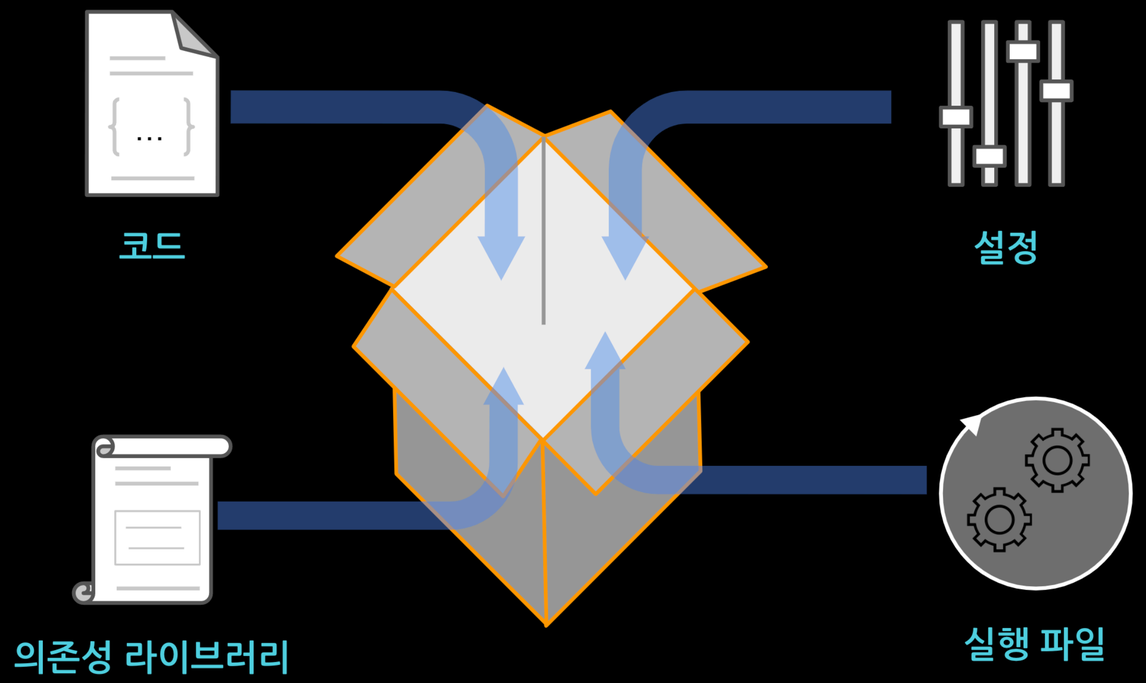
구애받지 않을 수 있을까



앱 전용환경 만들기

올인원 패키징

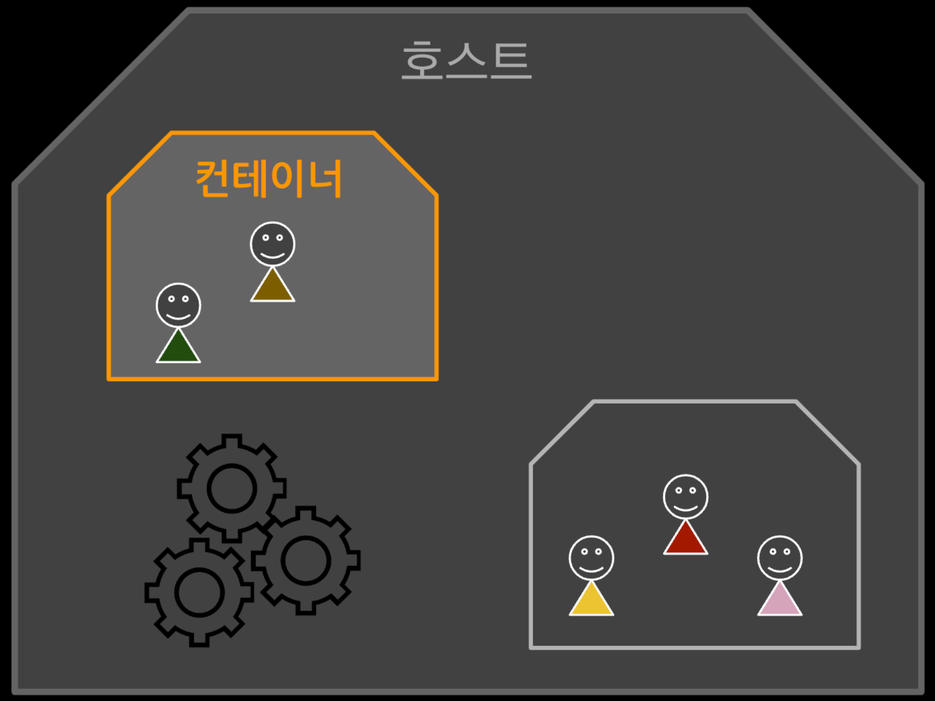
실행에 필요한 모든 걸 담아요



앱 전용환경 만들기

격리 하기

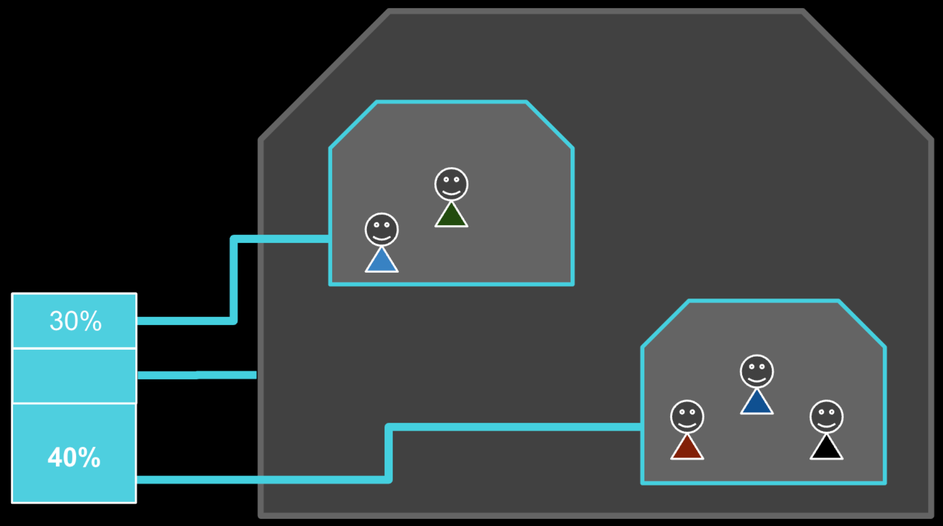
프로세스 주변 영향을 차단합니다



앱 전용환경 만들기

자원 할당

부여된 자원의 사용을 보장합니다



컨테이너 왜 써요 ?

올인원 패키징 + 격리 + 자원보장

애플리케이션 전용 환경 제공

컨테이너 사용 요건

쿠버네티스

컨테이너 오케스트레이션

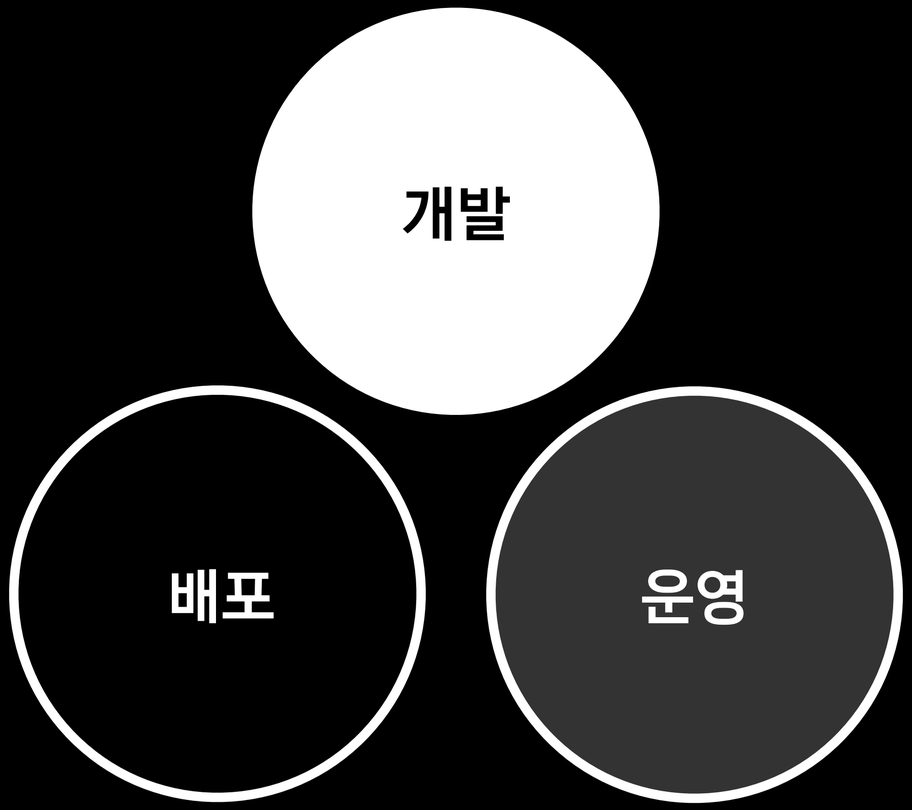
도커 ? 쿠버네티스 ?

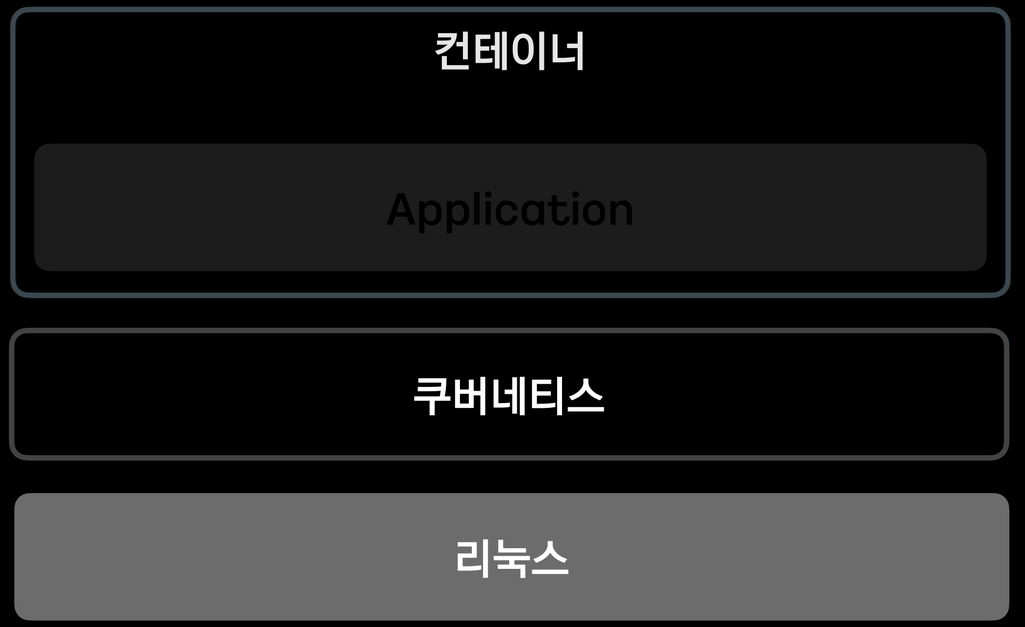
컨테이너

docker

kubernetes

컨테이너에 대한 이해 필요

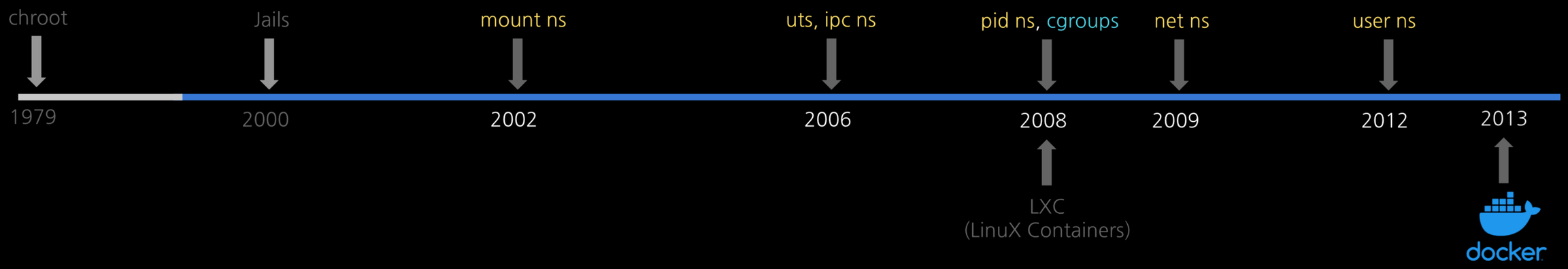




Goal

도커 없이 컨테이너 만들기

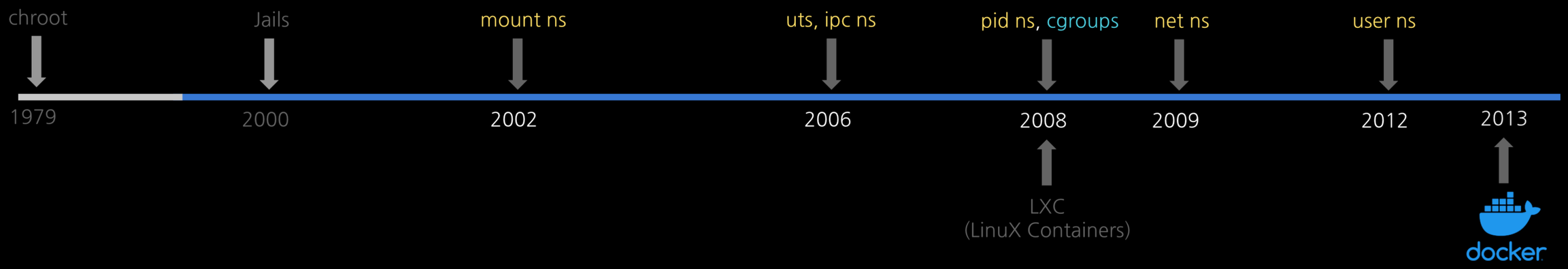
컨테이너의 역사


chroot
Jails
mount ns
uts, ipc ns
pid ns, cgroups
net ns
user ns
1979
2000
2002
2006
2008
2009
2012
2013
LXC
(LinuX Containers)
docker

과정 소개

- 컨테이너 파일시스템
- 컨테이너 격리와 자원

도커 없이 컨테이너 만들기



과정소개

실습안내

컨테이너 맛보기

실습 환경 준비

https://github.com/sam0kim/container-internal

- 맥/윈도우 환경 : Vagrant + Virtualbox
- 기타 환경 : Ubuntu 18.04 환경 준비

container-internal

컨테이너 인터널 실습 리포입니다

실습 환경 갖추기

실습 안내

- Mac 환경에서 VirtualBox 와 Vagrant 기반으로 테스트 되었습니다
- Mac 이외 OS 환경에서도 가급적 “Vagrant + VirtualBox” 사용을 권장드립니다.

> Vagrant 기반 실습환경 준비가 어려운 분들은 아래 "Vagrant 미지원 시 실습환경 갖추기"를 참고해주세요.

실습 환경 준비 : 맥/윈도우 사용자

https://github.com/sam0kim/container-internal

- Vagrant , Virtualbox 설치

실습 도구

- Virtualbox 공식 다운로드 및 설치 https://www.virtualbox.org/wiki/Downloads
- Vagrant 공식 다운로드 및 설치 https://www.vagrantup.com/downloads

맥 OS기준 (M1제외) Virtualbox 및 Vagrant 설치 방법은 다음과 같습니다.

설치 방법

```
# virtualbox 설치
brew install virtualbox --cask
# vagrant 설치
brew install vagrant --cask
```

Virtualbox 설정

- (설정1) 시스템 환경 설정에서 “Oracle” 허용 및 맥 재부팅

```
시스템 환경설정 > 보안 및 개인 정보 보호 > 일반 > 하단에 Oracle 관련 설치 허용
```

- (설정2) 네트워크 range 설정 (참고) https://www.virtualbox.org/manual/ch06.html#network_hostonly

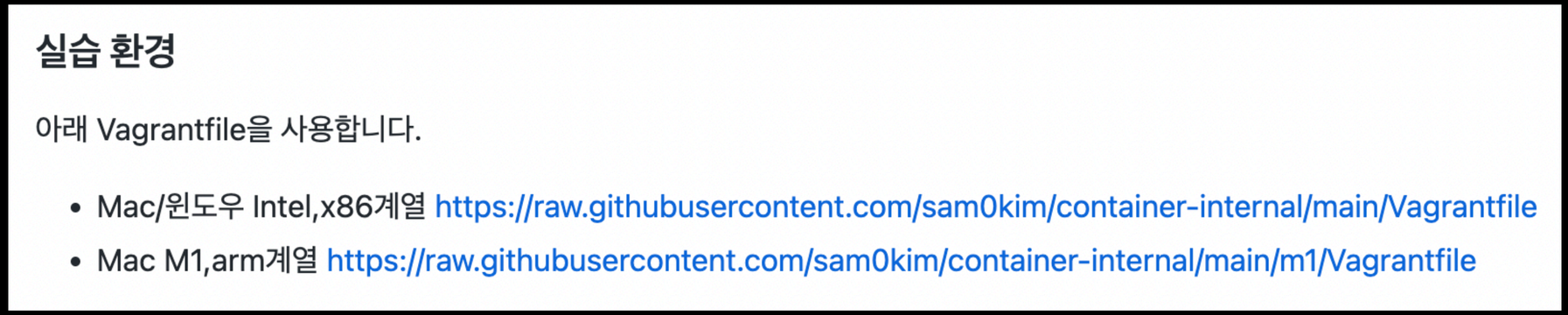
```
$ sudo -i
# mkdir -p /etc/vbox/
# vi /etc/vbox/networks.conf
* 0.0.0.0/0 ::/0
```

실습 환경 준비 : 맥/윈도우 사용자

https://github.com/sam0kim/container-internal

- Vagrantfile 복사
- Mac/윈도우 x86계열 Vagrantfile

  Mac M1, arm계열 Vagrantfile



실습 환경

아래 Vagrantfile을 사용합니다.

- Mac/윈도우 Intel,x86계열 https://raw.githubusercontent.com/sam0kim/container-internal/main/Vagrantfile
- Mac M1,arm계열 https://raw.githubusercontent.com/sam0kim/container-internal/main/m1/Vagrantfile

실습 환경 준비 : 맥/윈도우 사용자

https://github.com/sam0kim/container-internal

- VM 기동과 접속

Vagrant 기본 사용법. vagrant는 "Vagrantfile" 경로를 기준으로 동작합니다.

```
# VM 중지와 재개
$ vagrant suspend
$ vagrant resume

# VM 종료와 기동
$ vagrant halt
$ vagrant up

# VM 재기동
$ vagrant reload

# VM 재설정/기동
$ vagrant reload --provision

# VM 상태 확인 (VM 목록별 상태 출력)
$ vagrant status

# VM 터미널 접속
$ vagrant ssh <VM이름>
```

실습 환경 준비 : 기타 환경 사용자

https://github.com/sam0kim/container-internal

- Ubuntu 18.04 환경 준비
- Pre-requisite, Docker 설치

Vagrant 미지원 시 실습환경 갖추기

- 본 가이드는 Vagrant 기반 실습환경 구성이 어려운 경우에 해당합니다.
- Ubuntu(1804) 환경을 ~~27H~~준비해 주세요. * 무료클라우드 활용 등

실습 환경 갖추기

각자 준비된 Ubuntu(1804) 환경에서 아래 설치 스크립트를 실행해 주세요

- Pre-requisite

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

- Docker 설치

```
apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common >
&& curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - \
&& add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
&& apt-get update \
&& apt-get -y install docker-ce docker-ce-cli containerd.io > /dev/null 2>&1
```

용어사용

컨테이너 : 격리하는 환경을 통칭해 사용합니다

예시) chroot, pivot_root, namespace ... 등

프롬프트

```
/tmp# docker run -it busybox
```

루트 계정, 실습 경로

```
# docker run -it busybox
```

루트 계정, 경로 무관

```
$ docker run -it busybox
```

일반 계정, 경로 무관

```
docker run -it busybox
```

프롬프트 없음 - 이전 계정/경로

컨테이너 종료 안내

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

과정소개

실습안내

컨테이너 맛보기

컨테이너 기동

```
# docker run -it busybox
```

```
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
729ce43e2c91: Pull complete
Digest: sha256:ad9bd57a3a57cc95515c537b89aaa69d83a
ec0cd5
Status: Downloaded newer image for busybox:latest
/ #
```

BusyBox는 수많은 UNIX 명령행 유틸리티의 기능을 하나의 실행 파일 안에 통합시킨 소프트웨어이다. 공식 슬로건은 "임베디드 리눅스 업계의 스위스 아미 나이프(The Swiss Army Knife of Embedded Linux)".
1)

1) 나무위키 - BusyBox

루트 디렉토리 비교

```
/# ls /
```

파일시스템 비교

```
/# df -h
```

프로세스 비교

```
/# ps aux
```

네트워크 비교

```
/# ip l
```

호스트네임 비교

```
/# hostname
```

uid, gid 비교

```
/# id
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

if(kakao)2022

컨테이너 파일시스템

김삼영 sam.0
카카오엔터프라이즈

프로세스를 가두자

탈옥을 막아보자

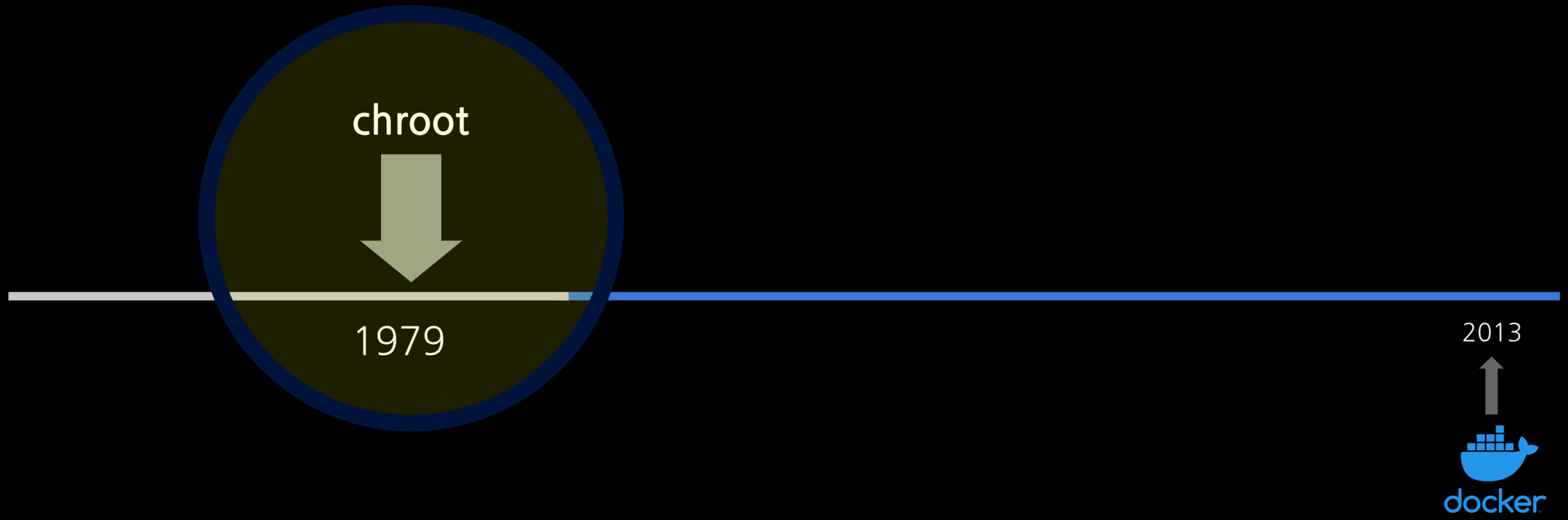
중복을 해결하자

프로세스를 가두자

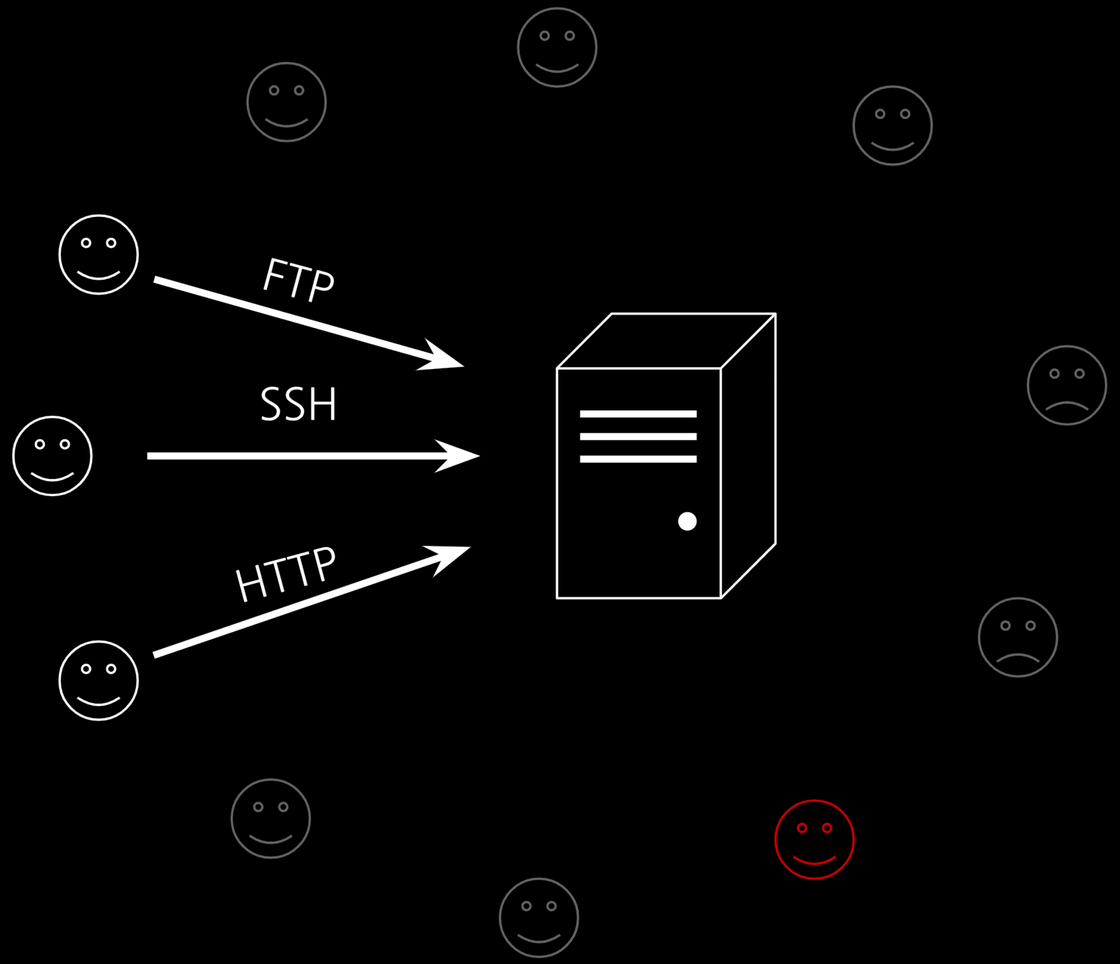
탈옥을 막아보자

중복을 해결하자

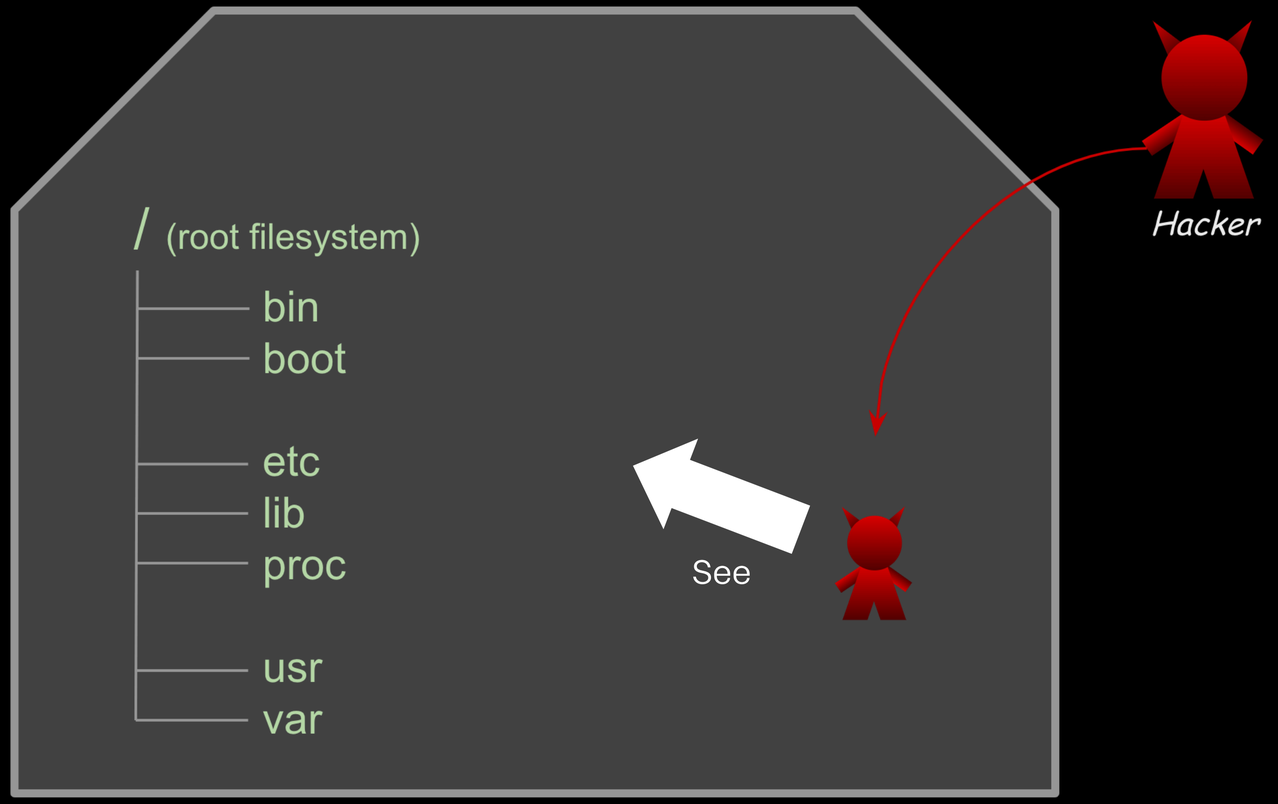
컨테이너의 기원



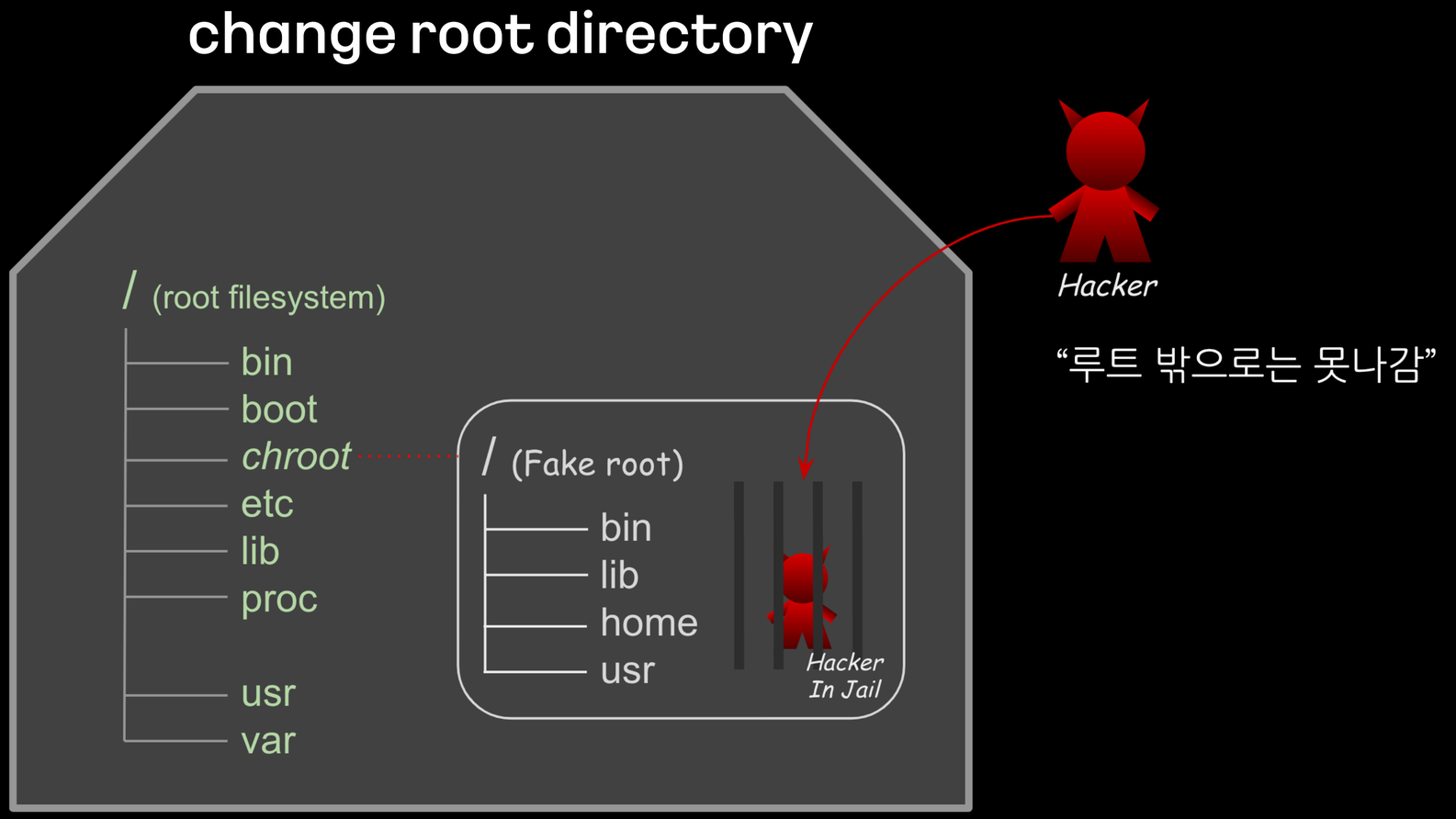
여러 사람들이 이용하는 서버



여러 사람들이 이용하는 서버



chroot



"루트 밖으로는 못나감"

chroot

(사용법) chroot [옵션] NEWROOT [커맨드]

(사용예) `chroot myroot /bin/sh`

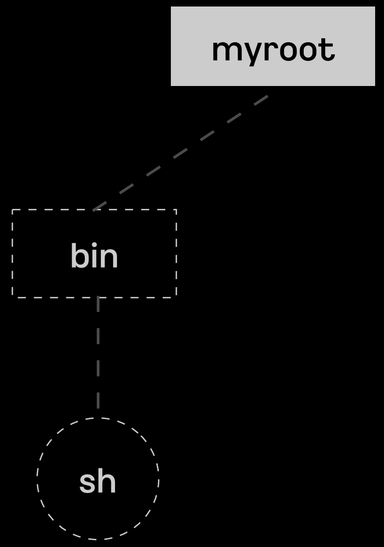
(참고) 커맨드를 지정하지 않으면 $SHELL이 기본값

[실습1] chroot

chroot 해서 sh 실행하기

```
/tmp# mkdir myroot
/tmp# chroot myroot /bin/sh
```

chroot: failed to run command '/bin/sh': No such file or directory



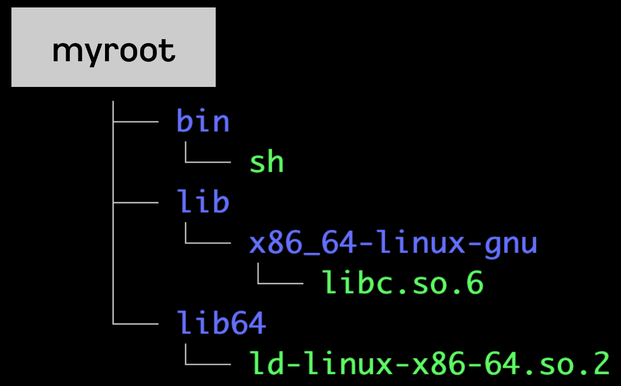
sh 파일 확인

```
/tmp# which sh
/tmp# ldd /bin/sh
```

```
root@ubuntu1804:/tmp# which sh
/bin/sh
root@ubuntu1804:/tmp# ldd /bin/sh
        linux-vdso.so.1 (0x00007ffe2abe7000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc92f3fa000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc92fa0b000)
```

sh 파일 복사

```
/tmp# mkdir -p myroot/bin;
/tmp# cp /bin/sh myroot/bin/;
/tmp# mkdir -p myroot/{lib64,lib/x86_64-linux-gnu};
/tmp# cp /lib/x86_64-linux-gnu/libc.so.6 myroot/lib/x86_64-linux-gnu/;
/tmp# cp /lib64/ld-linux-x86-64.so.2 myroot/lib64;
```



chroot 실행

```
/tmp# chroot myroot /bin/sh
# ls
```

디렉토리 확인 ?

```
# ls
/bin/sh: 1: ls: not found
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

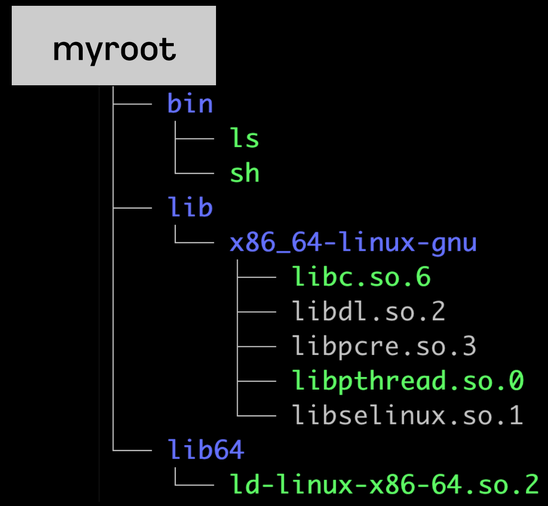
ls 파일 확인

```
/tmp# which ls
/tmp# ldd /bin/ls
```

```
root@container-internal-sam-1:/tmp# which ls
/bin/ls
root@container-internal-sam-1:/tmp# ldd /bin/ls
        linux-vdso.so.1 (0x00007ffd5636e000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f9e7785e000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9e7746d000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f9e771fb000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f9e76ff7000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9e77ca8000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f9e76dd8000)
```

ls 파일 복사

```
/tmp# cp /bin/ls myroot/bin/;
/tmp# 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/;

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



chroot 실행

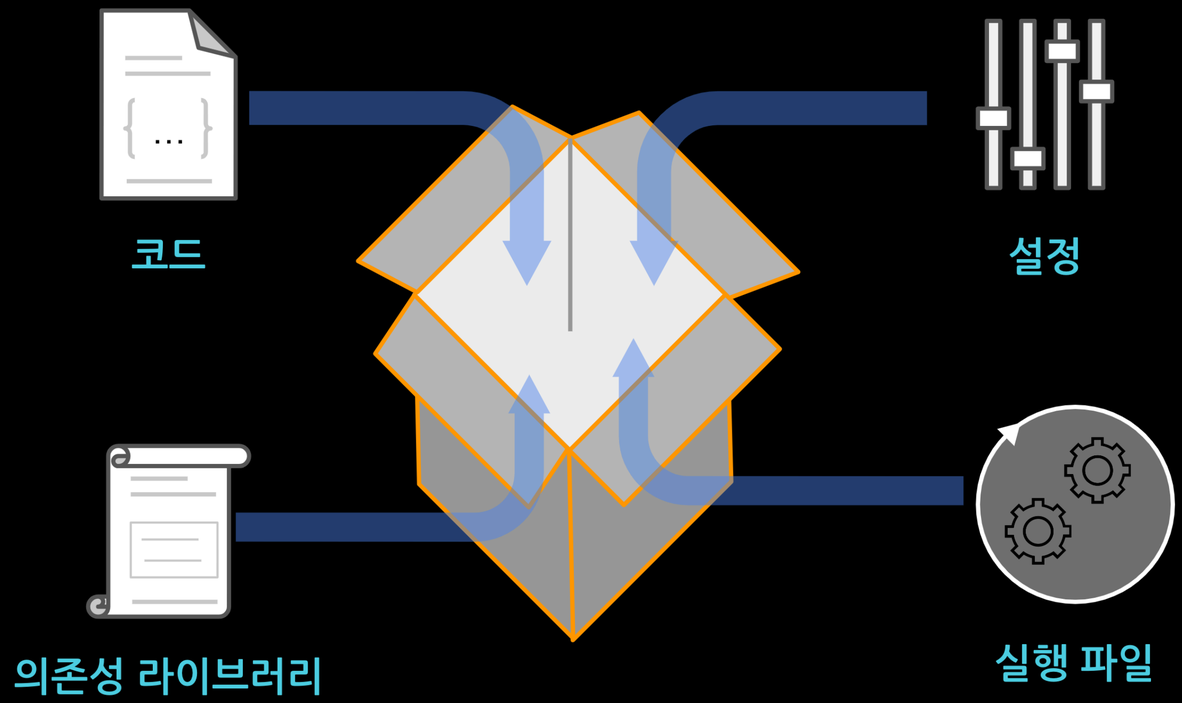
```
/tmp# chroot myroot /bin/sh
# ls
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

여기서 잠깐, chroot

- 경로에 모으고 (패키징)
- 경로에 가둬서 실행 (격리)



Mission

chroot 하여 ps 명령어를 실행해 보세요

chroot_ps.sh

```
/tmp#  wget https://raw.githubusercontent.com/sam0kim/container-internal/main/scripts/chroot_ps.sh;

/tmp#  bash chroot_ps.sh;
```

chroot > ps 실행해 봅시다

```
/tmp# chroot myroot /bin/sh
# ps
Error, do this: mount -t proc proc /proc
```

chroot > ps 실행해 봅시다

```
# mount -t proc proc /proc
mount: /proc: mount point does not exist.
# mkdir /proc
# mount -t proc proc /proc
```

chroot > ps 실행해 봅시다

```
# ps
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

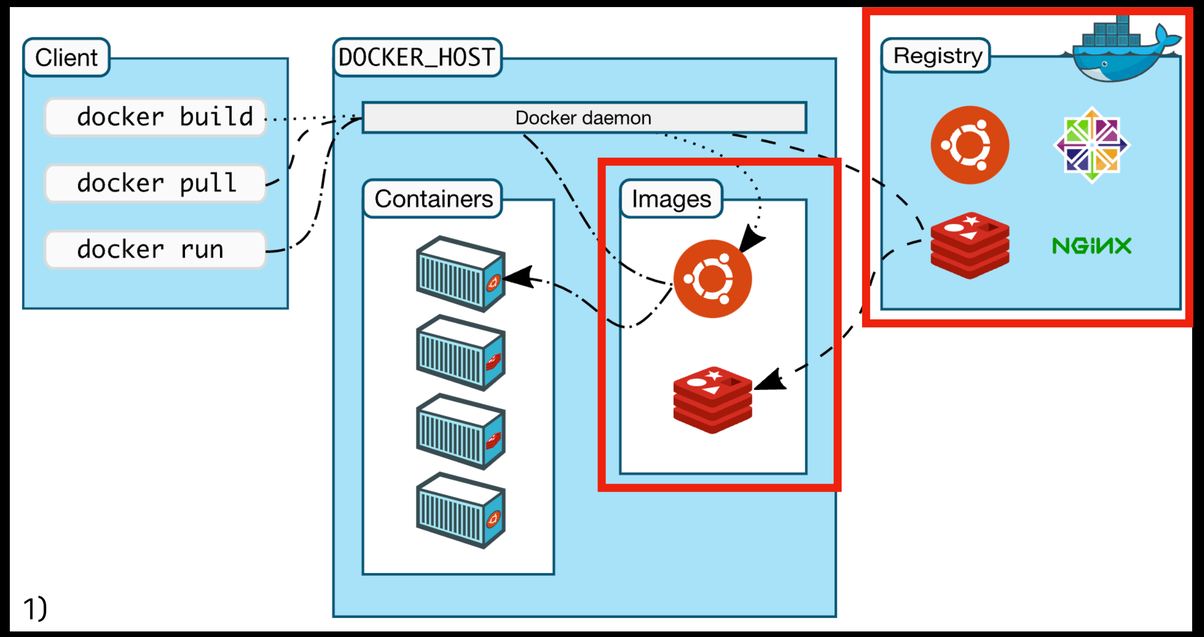
실습환경 초기화

myroot/proc 는 언마운트 해줍니다

```
# umount /tmp/myroot/proc
```

한땀 한땀 복사하기 힘드시죠

이미지



1) https://docs.docker.com/get-started/overview/

남이 만든 이미지로 해보자
chroot

nginx 이미지 chroot 해보기

```
/tmp# mkdir nginx-root;
/tmp# docker export $(docker create nginx) \
      | tar -C nginx-root -xvf -;
/tmp# chroot nginx-root /bin/sh;
```

호스트와 "루트디렉토리" 비교

```
# ls /
```

nginx 실행

```
# nginx -g "daemon off;"
```

👉 호스트에서 확인해 보세요

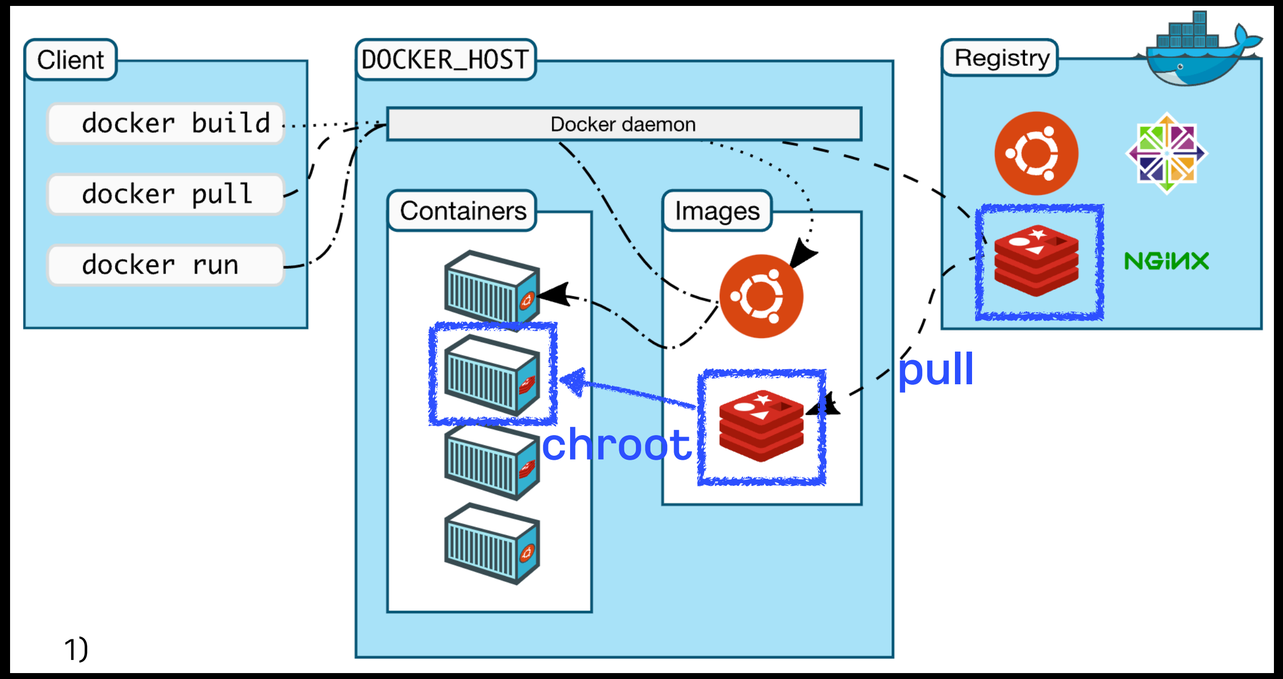
```
# curl localhost:80
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

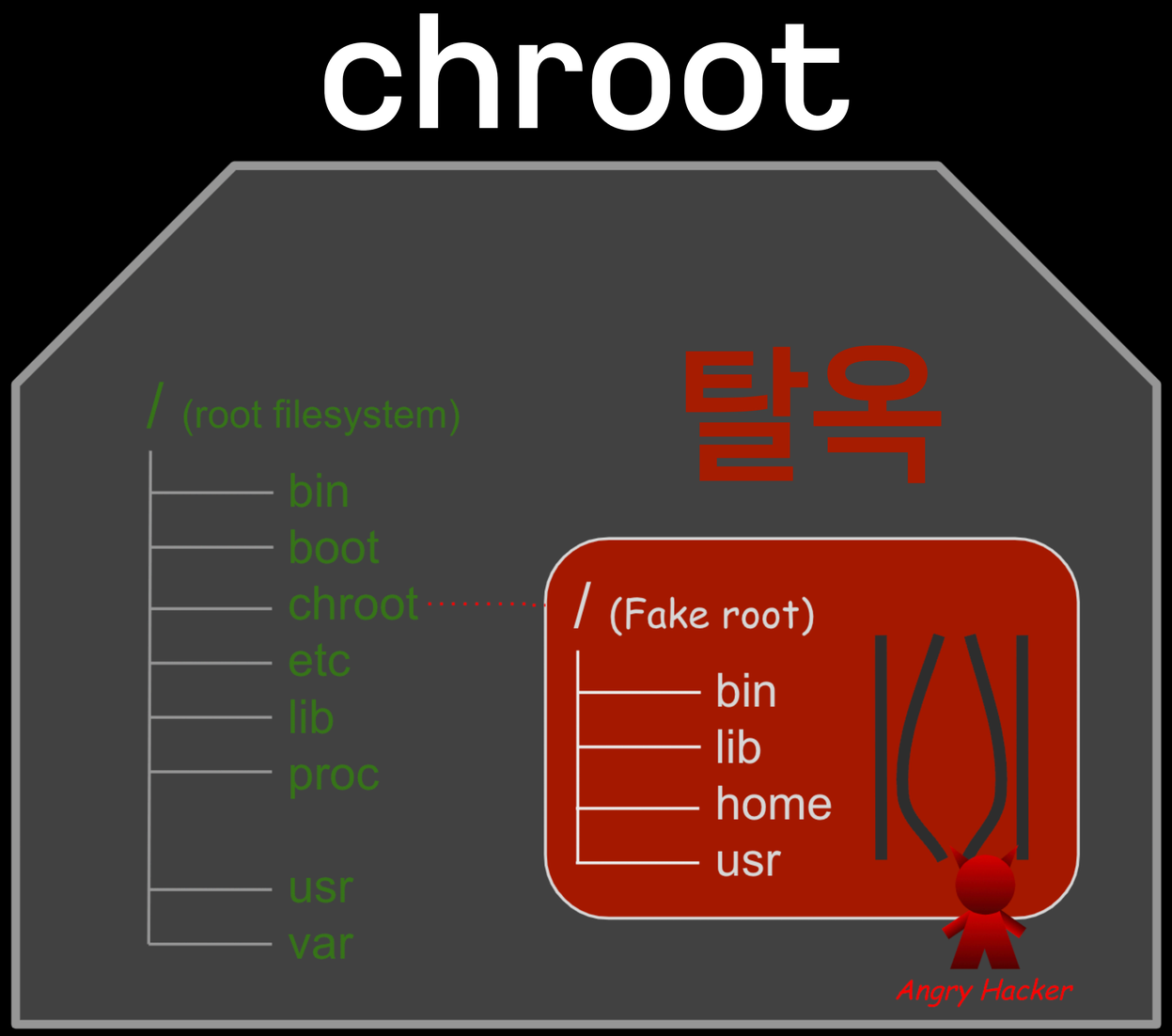
chroot 만 있어도 충분 ?



1)

1) https://docs.docker.com/get-started/overview/

그런데 말입니다 ...


chroot
/ (root filesystem)
bin
boot
chroot
etc
lib
proc
usr
var
탈옥
/ (Fake root)
bin
lib
home
usr
Angry Hacker

지금부터
chroot를 해킹해 보겠습니다

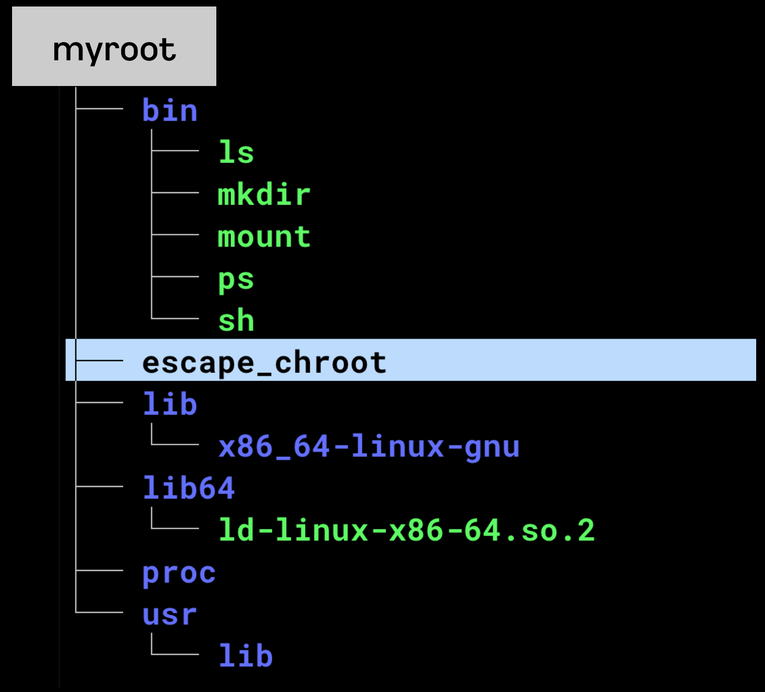
탈옥코드

/tmp/escape_chroot.c

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

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

탈옥코드 컴파일

```
/tmp# gcc -o myroot/escape_chroot \
      escape_chroot.c
```



chroot 실행

```
/tmp# chroot myroot /bin/sh
# ls /
bin  escape_chroot  lib  lib64 proc  usr
```

chroot 탈옥

```
# ./escape_chroot
# ls /
```

EXIT x2 다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

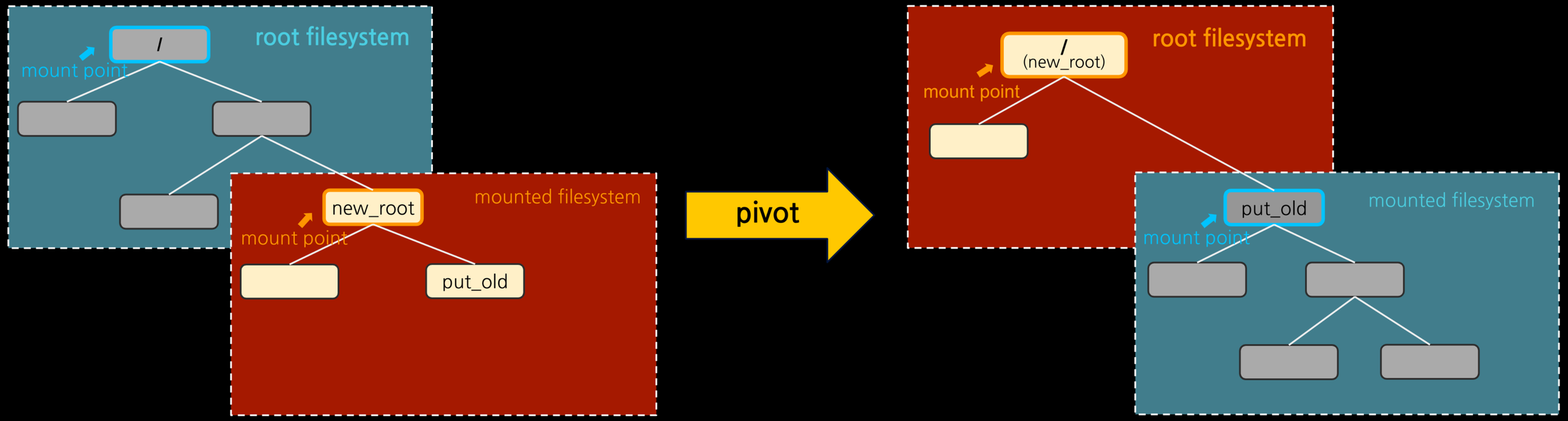
프로세스를 가두자

탈옥을 막아보자

중복을 해결하자

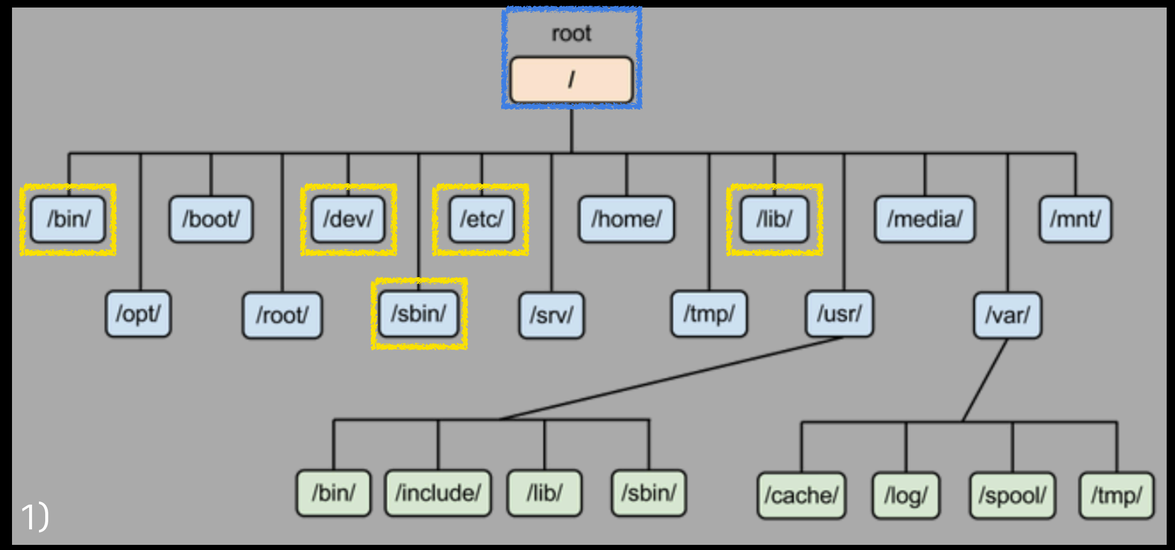
pivot_root

"루트파일시스템"을 피봇합니다



루트파일시스템?

- 최상위 파일시스템
- 루트디렉토리를 포함
- 하위의 모든 파일시스템들이 마운트



1)

1) https://dev.to/kcdchennai/linux-file-system-4idj

그런데, 루트 파일시스템을 피봇하면 ...

호스트에 영향이 가지 않을까요?

당연히 영향이 갑니다

호스트에 영향을 주지 않으려면 ?

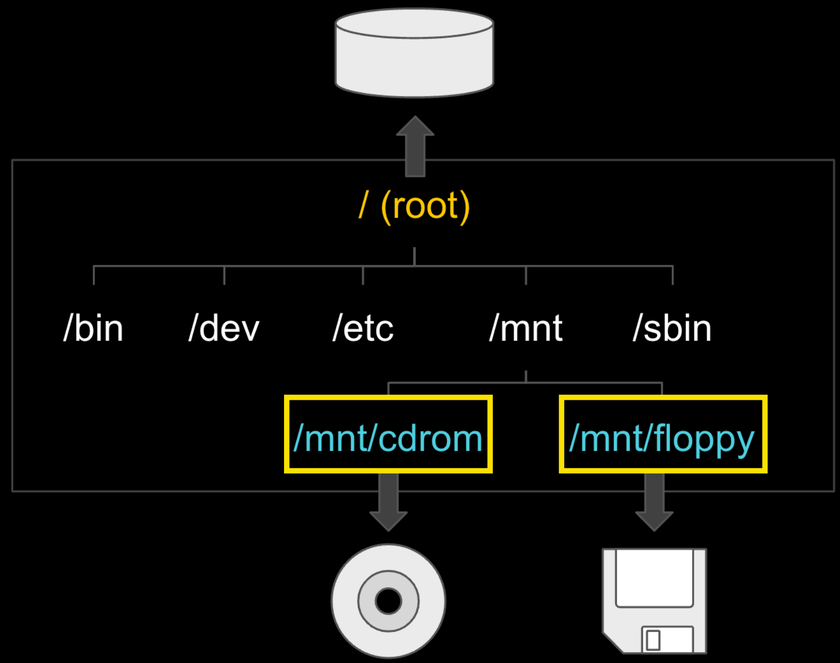
프로세스의 환경을 격리하는 "네임스페이스" 개발 (2002)

- "마운트 환경" 격리만 고려

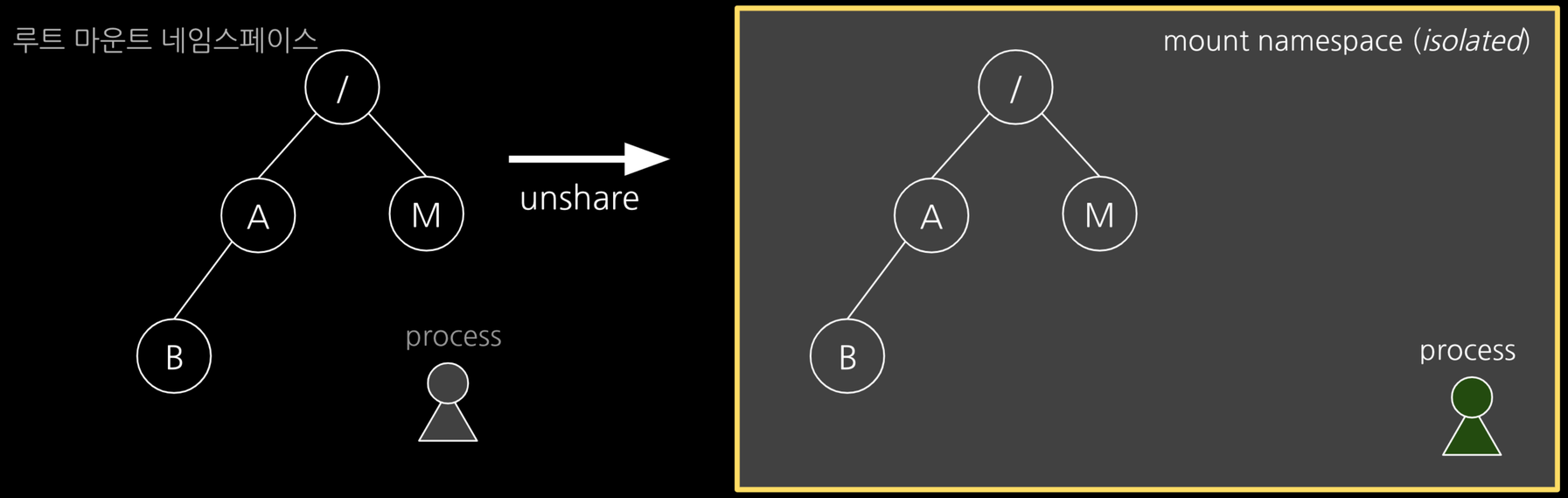
마운트

파일시스템을 루트파일시스템의 하위 디렉토리로 부착하는 시스템콜

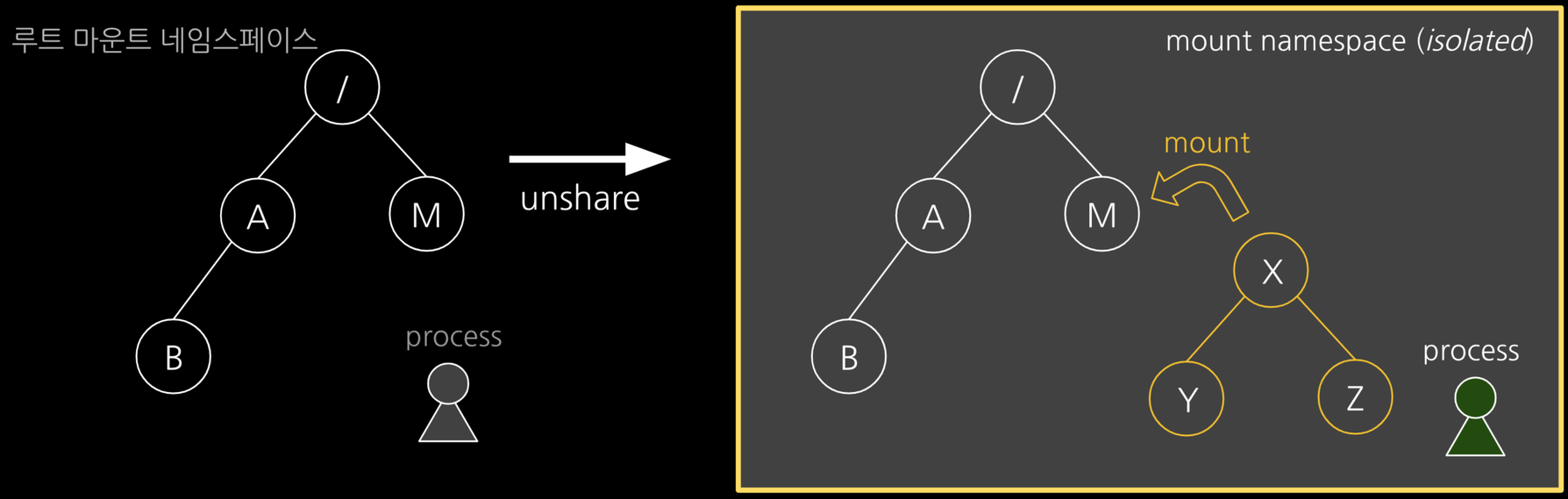
- 마운트 포인트 : 부착 지점, 접근 지점
- 예) USB, CDROM 마운트



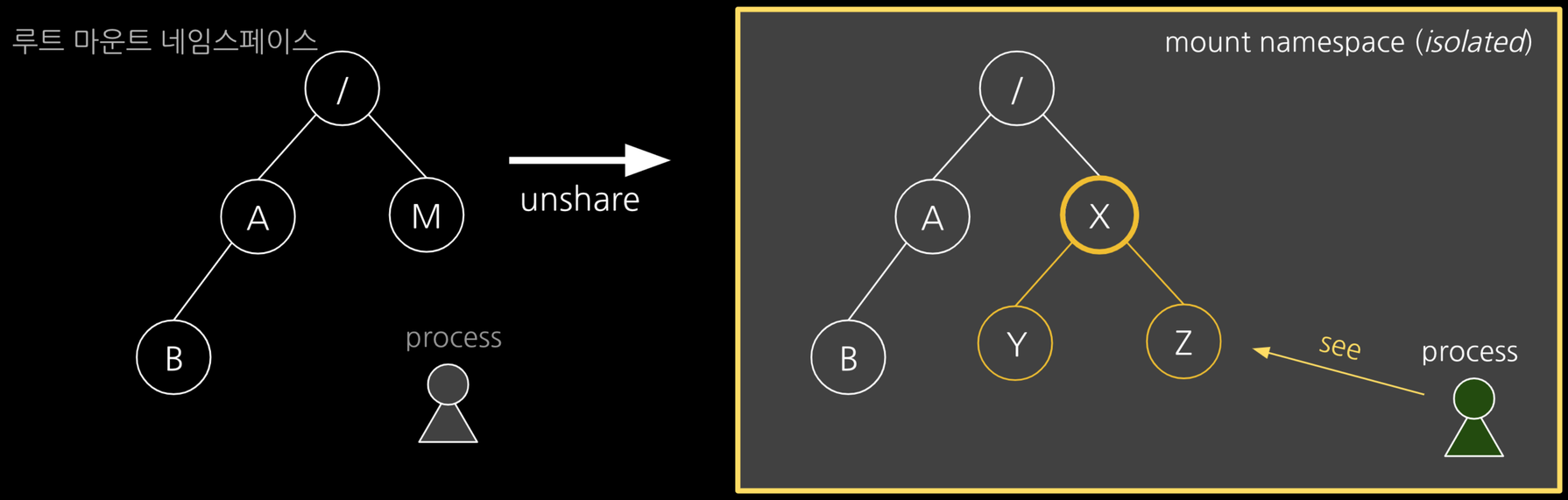
마운트 네임스페이스


루트 마운트 네임스페이스
/
A
M
B
process
unshare
mount namespace (*isolated*)
/
A
M
B
process

마운트 네임스페이스

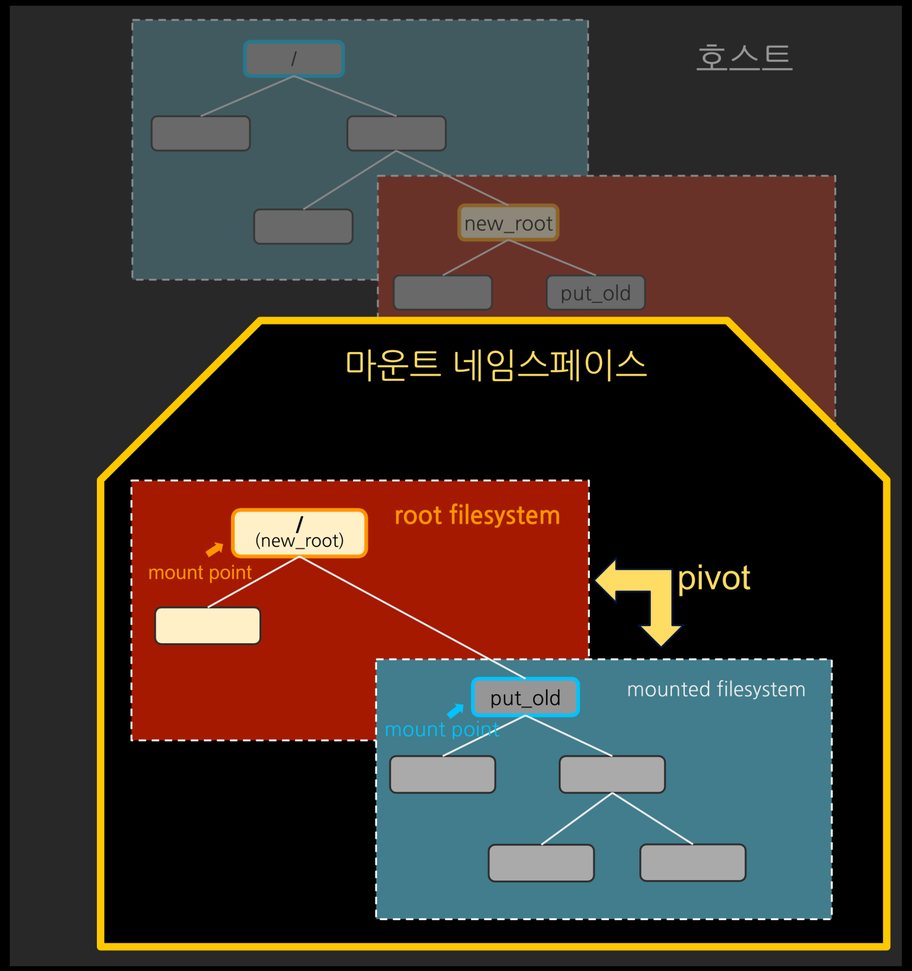

루트 마운트 네임스페이스
/
A
M
B
process
unshare
mount namespace (*isolated*)
/
A
M
B
mount
X
Y
Z
process

마운트 네임스페이스


루트 마운트 네임스페이스
/
A
M
B
process
unshare
mount namespace (*isolated*)
/
A
X
B
Y
Z
see
process

마운트 네임스페이스와 pivot_root

호스트에 영향 없음



마운트 네임스페이스를 사용하려면?

(사용법) `unshare [options] [<program> [<arguments>... ]]`

(옵션) `-m, --mount` 마운트 네임스페이스

(참고) 프로그램을 지정하지 않으면 `$SHELL` 기본값

[실습2] pivot_root

마운트 네임스페이스

```
/tmp# unshare --mount /bin/sh
```

👉 호스트와 비교해 보세요

```
# df -h
```

new_root 마운트

```
/tmp# mkdir new_root;
/tmp# mount -t tmpfs none new_root;
```

👉 호스트와 비교해 보세요

```
# df -h | grep new_root
```

myroot → new_root 복사

```
/tmp# cp -r myroot/* new_root
```

👉 호스트와 비교해 보세요

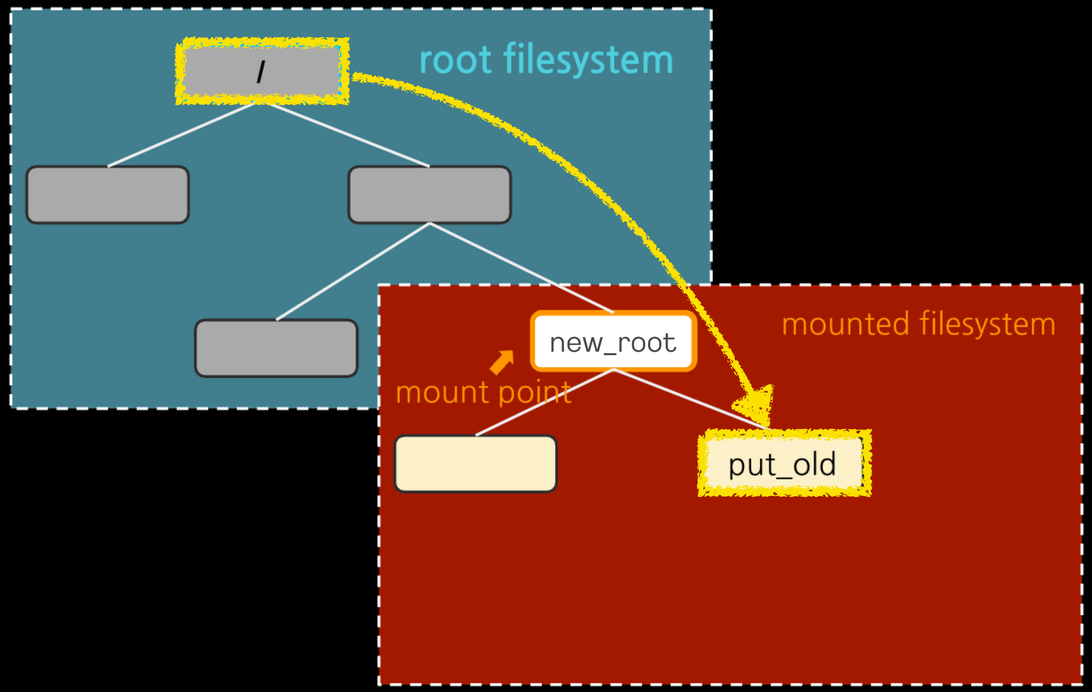
```
# tree new_root
```

put_old 디렉토리

```
/tmp# mkdir new_root/put_old
```

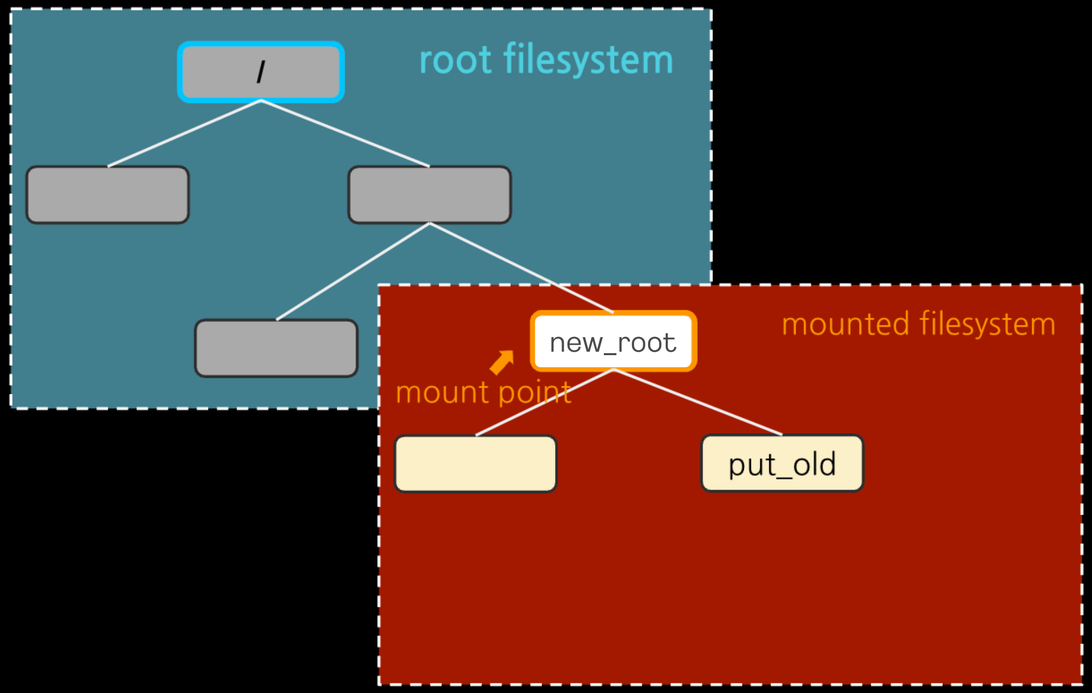
pivot_root <new_root> <put_old>

- new_root : 마운트 포인트 (신규루트)
- put_old : 디렉토리 (기존루트)



pivot_root 실행

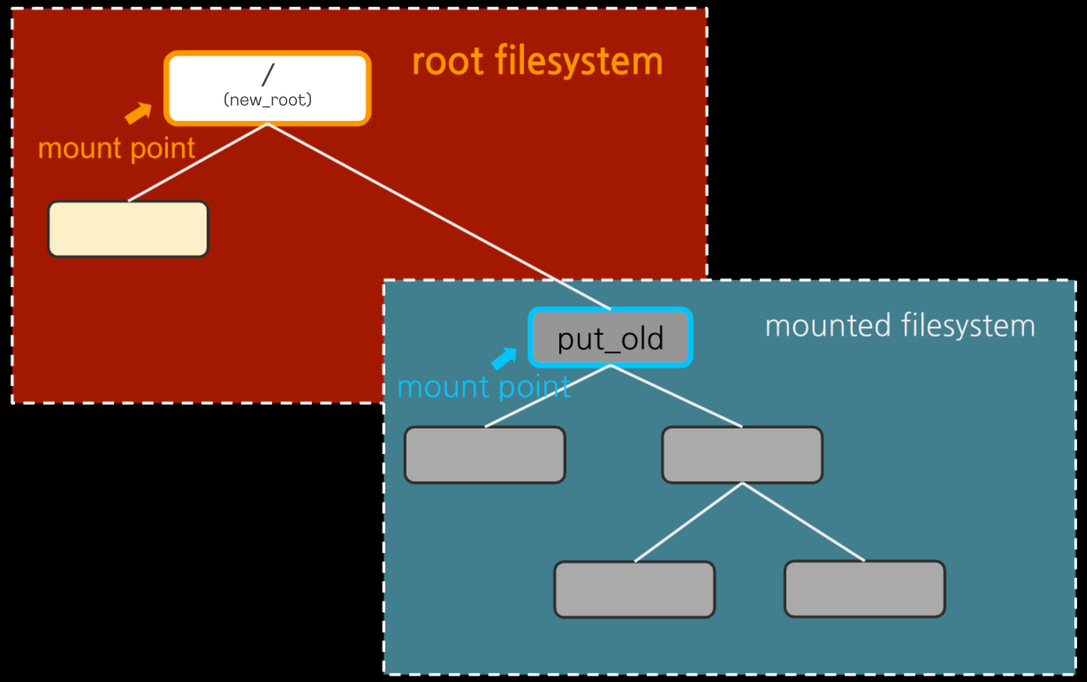
```
/tmp# cd new_root;
/tmp# pivot_root . put_old;
/tmp# cd /;
```



루트파일시스템이 바뀌었나요?

```
/# ls /
/# ls put_old
```

```
bin   home                      lib         mnt   run          swapfile  var
boot  initrd.img                lib64       opt   sbin         sys       vmlinuz
dev   initrd.img.old            lost+found  proc  srv          tmp       vmlinuz.old
etc   krb5.conf.20220422.120947 media       root  sshd_config  usr
```



탈옥이 되는지 확인해 보세요

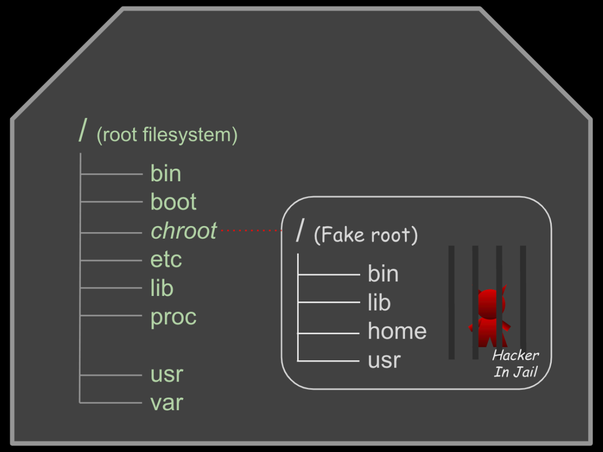
```
/# ./escape_chroot
```

EXIT x2 다음 실습을 위해 컨테이너를 종료해 주세요

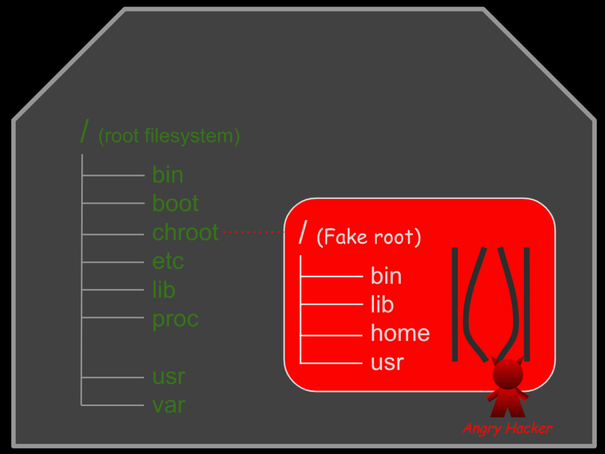
```
# exit
root@ubuntu1804:/tmp#
```

컨테이너 파일시스템

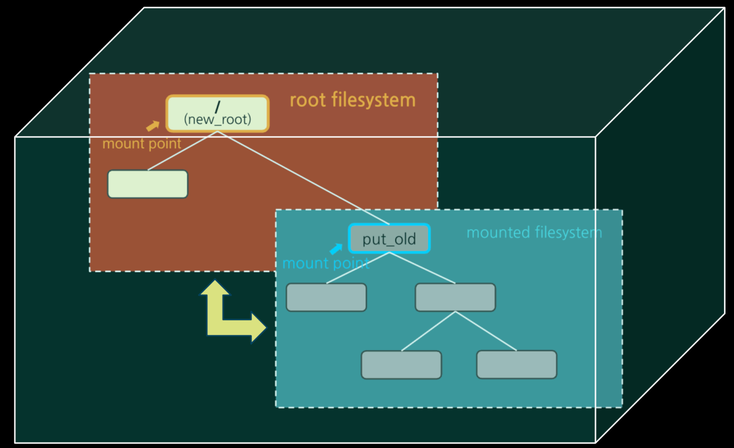
chroot



탈옥

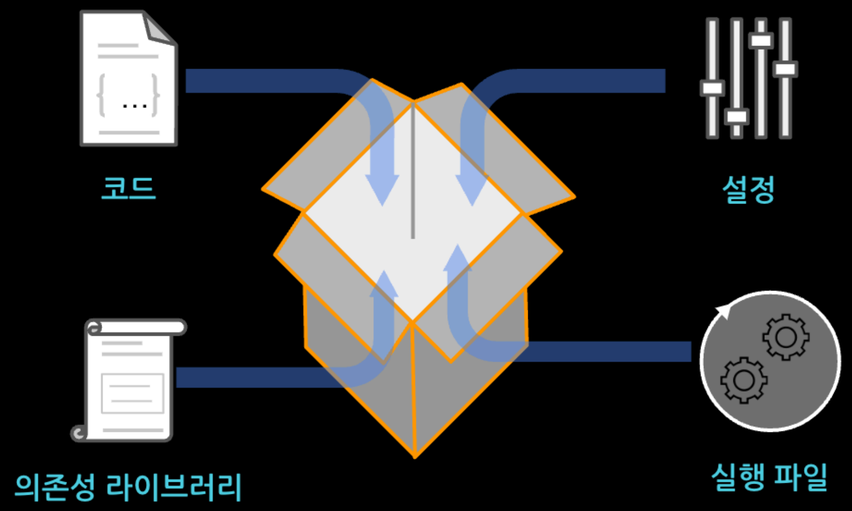


격리 + pivot_root



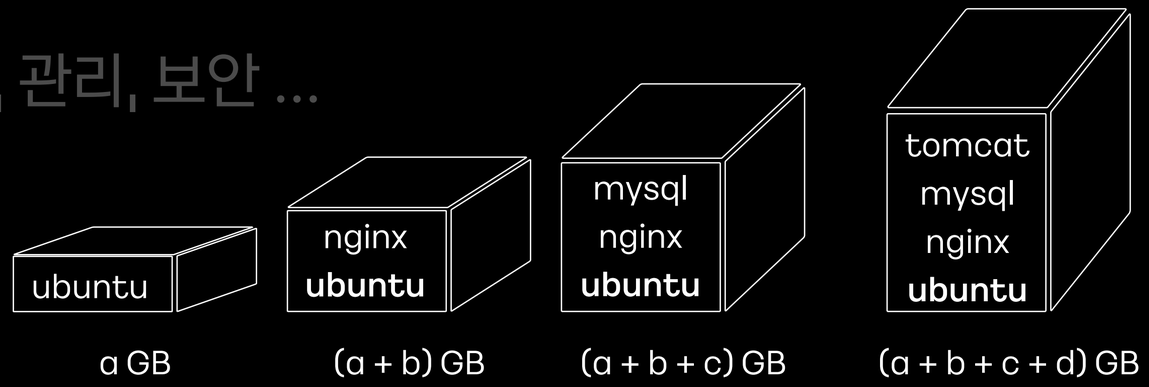
그런데 말입니다 ... 또?

패키징으로 잘 모으긴 했는데요



중복

이미지 저장, 유통, 관리, 보안 ...



해결 과제 추가요

탈옥 문제 pivot_root

중복 문제 • • •

프로세스를 가두자

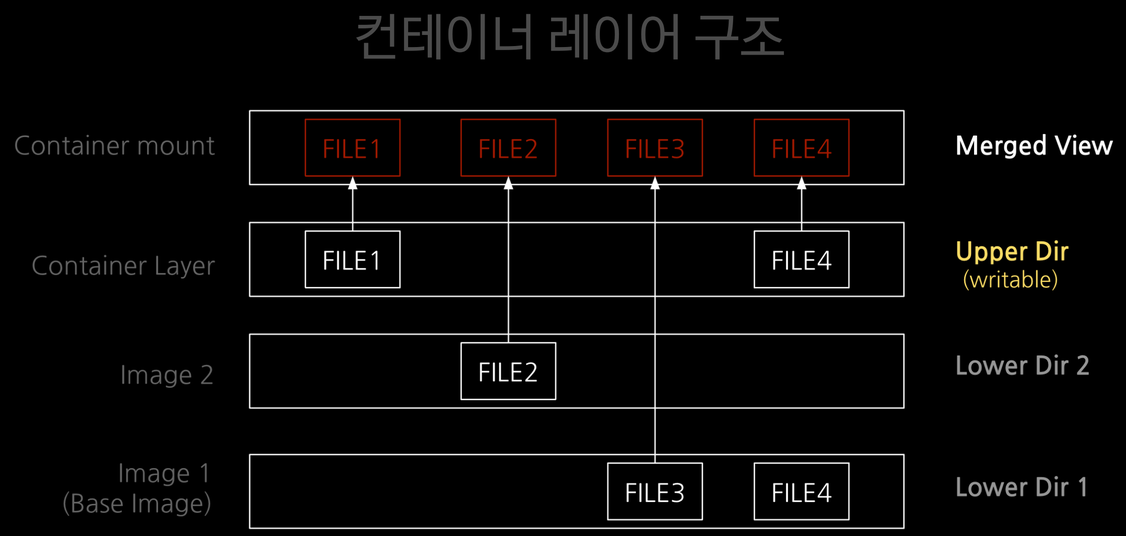
탈옥을 막아보자

중복을 해결하자

오버레이 파일시스템

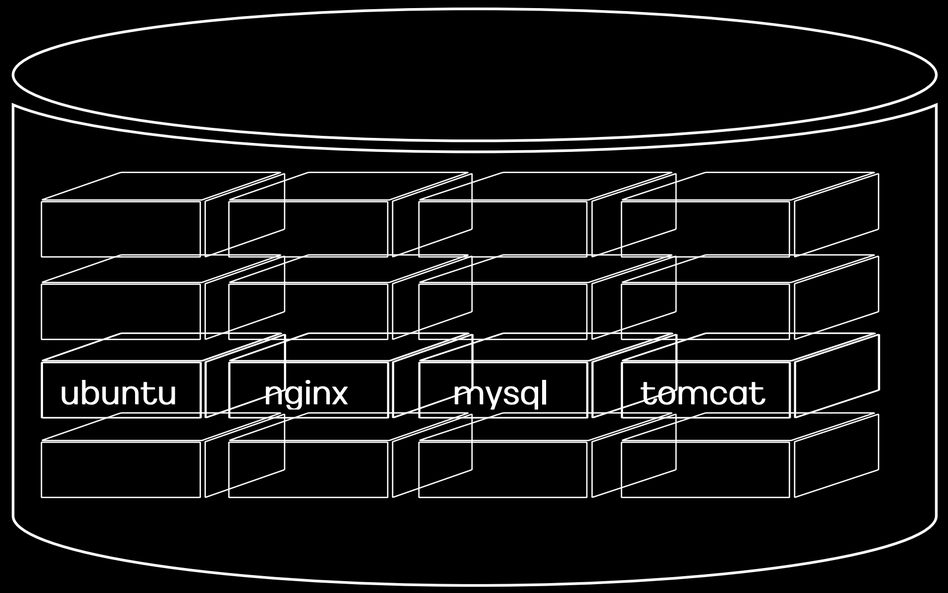
이미지 "중복 문제"를 해결

- 여러 이미지 레이어를 하나로 마운트
- Lower 레이어는 ReadOnly
- Upper 레이어는 Writable
- CoW, copy-on-write (원본유지)

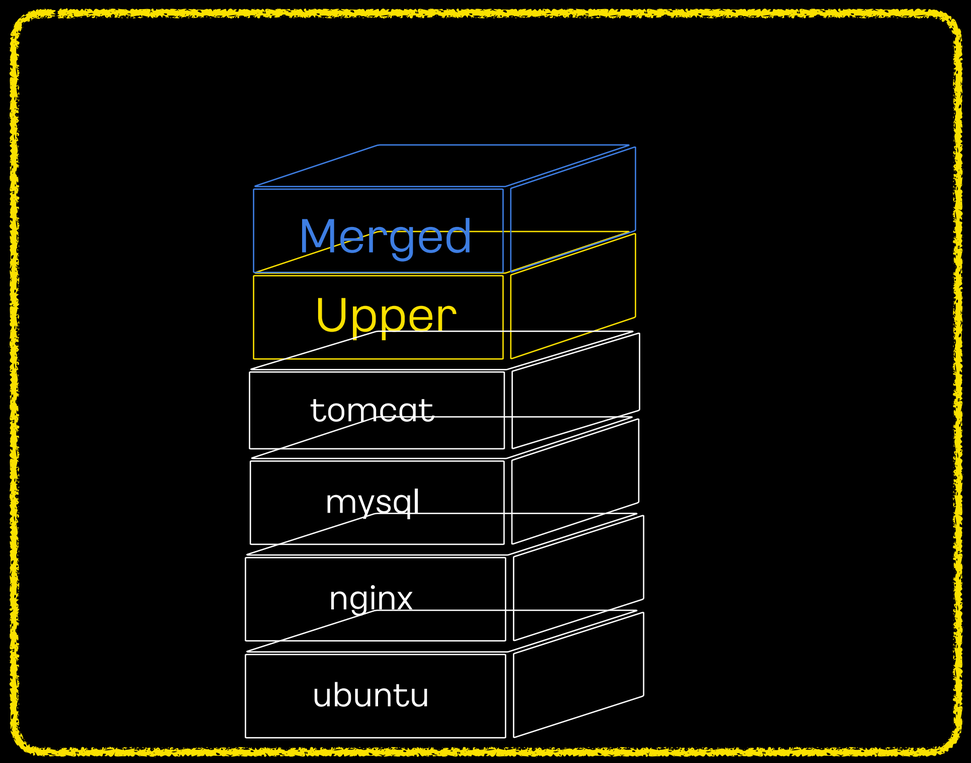


이미지 중복문제 해결

이미지 저장소

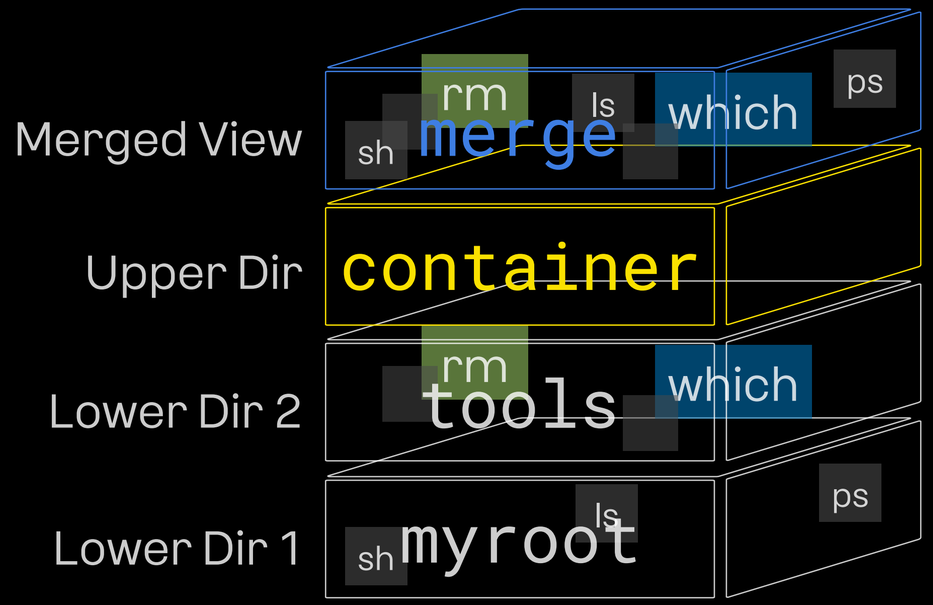


컨테이너



[실습3] overlay mount

오버레이 파일시스템


Merged View
rm
ls
which
ps
sh
merge
Upper Dir
container
Lower Dir 2
rm
tools
which
Lower Dir 1
ps
ls
sh
myroot

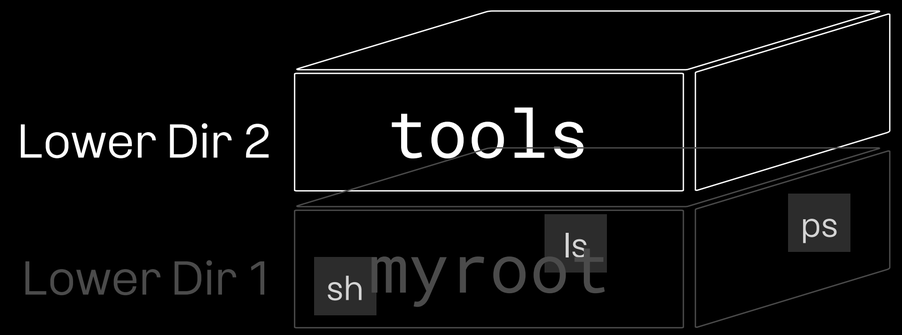
Lower Dir 1 준비

```
/tmp# tree myroot
```

```
root@ubuntu1804:/tmp# tree -L 2 myroot
myroot
├── bin
│   ├── ls
│   ├── mkdir
│   ├── mount
│   ├── ps
│   └── sh
├── escape_chroot
├── lib
│   └── x86_64-linux-gnu
├── lib64
│   └── ld-linux-x86-64.so.2
├── proc
└── usr
    └── lib
```

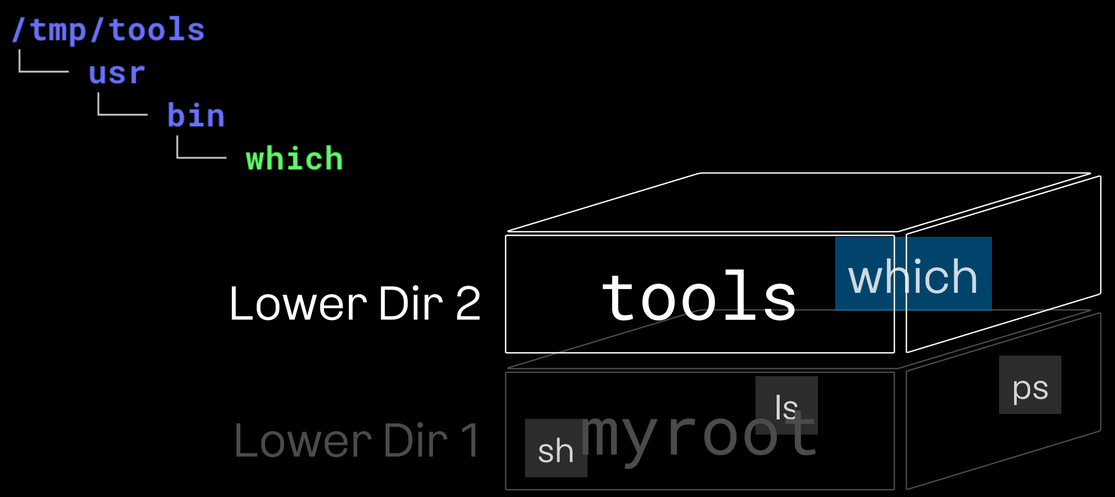
Lower Dir 2 준비

```
/tmp# mkdir tools
```



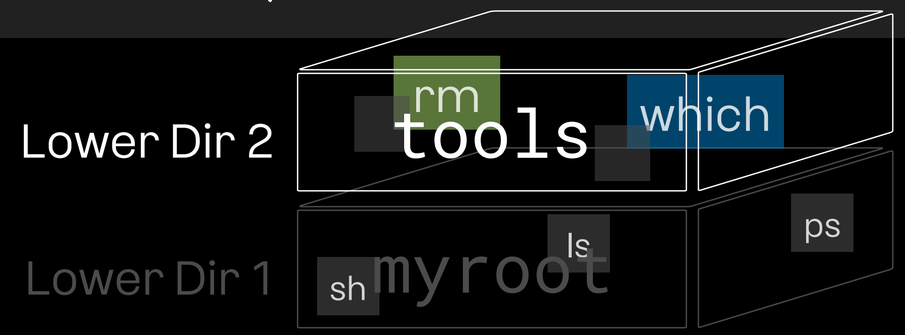
Lower Dir 2 준비 : which 복사

```
/tmp# which which
/tmp# ldd /usr/bin/which
    not a dynamic executable
/tmp# mkdir -p tools/usr/bin;
/tmp# cp /usr/bin/which tools/usr/bin/;
```



Lower Dir 2 준비 : rm 복사

```
/tmp# which rm;
/tmp# ldd /bin/rm;
/tmp# mkdir -p tools/{bin,lib64,lib/ \
      x86_64-linux-gnu};
/tmp# cp /bin/rm tools/bin/;
/tmp# cp /lib/x86_64-linux-gnu/libc.so.6 \
      tools/lib/x86_64-linux-gnu/;
/tmp# cp /lib64/ld-linux-x86-64.so.2 \
      tools/lib64;
```



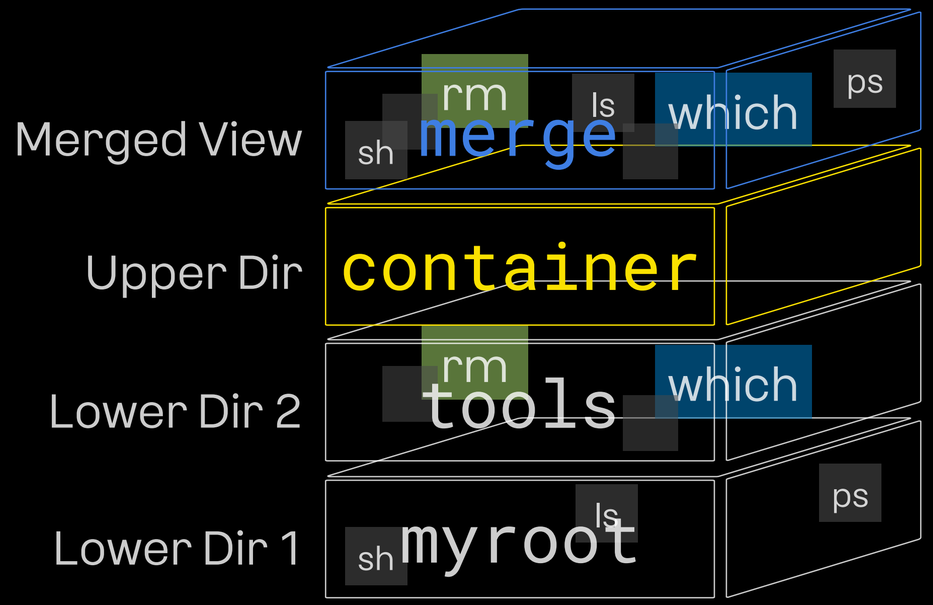
오버레이 마운트 준비

```
/tmp# mkdir -p rootfs/{container,work,merge}
```

```
root@ubuntu1804:/tmp# tree rootfs
rootfs
├── container
├── merge
└── work

3 directories, 0 files
```

오버레이 마운트

```
/tmp# mount -t overlay overlay -o \
      lowerdir=tools:myroot,\
      upperdir=rootfs/container,\
      workdir=rootfs/work rootfs/merge
```



myroot 와 비교

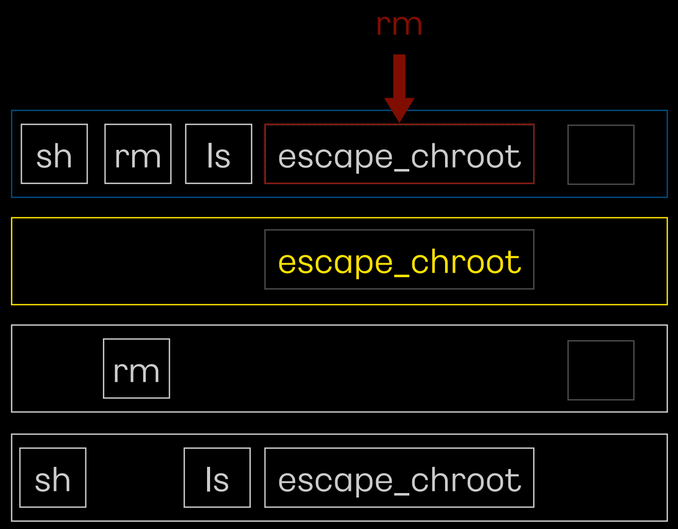
```
/tmp# tree -L 2 myroot/{bin,usr};
/tmp# tree -L 2 rootfs/merge/{bin,usr};
```

```
myroot/bin
├── ls
├── mkdir
├── mount
├── ps
└── sh
myroot/usr
└── lib
    └── x86_64-linux-gnu
```

```
rootfs/merge/bin
├── ls
├── mkdir
├── mount
├── ps
├── rm
└── sh
rootfs/merge/usr
├── bin
│   └── which
└── lib
    └── x86_64-linux-gnu
```

CoW, Copy On Write

```
/tmp# rm rootfs/merge/escape_chroot
/tmp# tree -L 2 rootfs
```



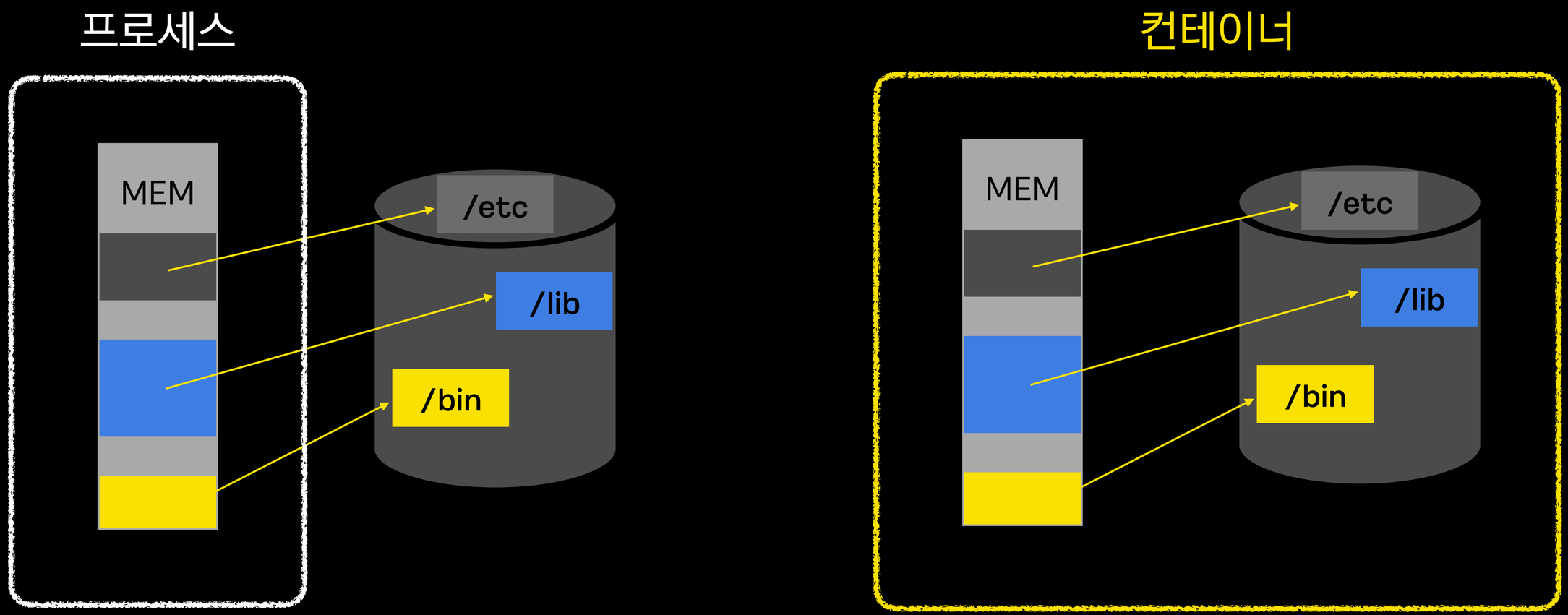

```
rootfs
├── container
├── merge
│   ├── bin
│   ├── escape_chroot
│   ├── lib
│   ├── lib64
│   ├── proc
│   └── usr
└── work
    └── work
```

실습환경 초기화

언마운트 해주세요

```
# umount /tmp/rootfs/merge
```

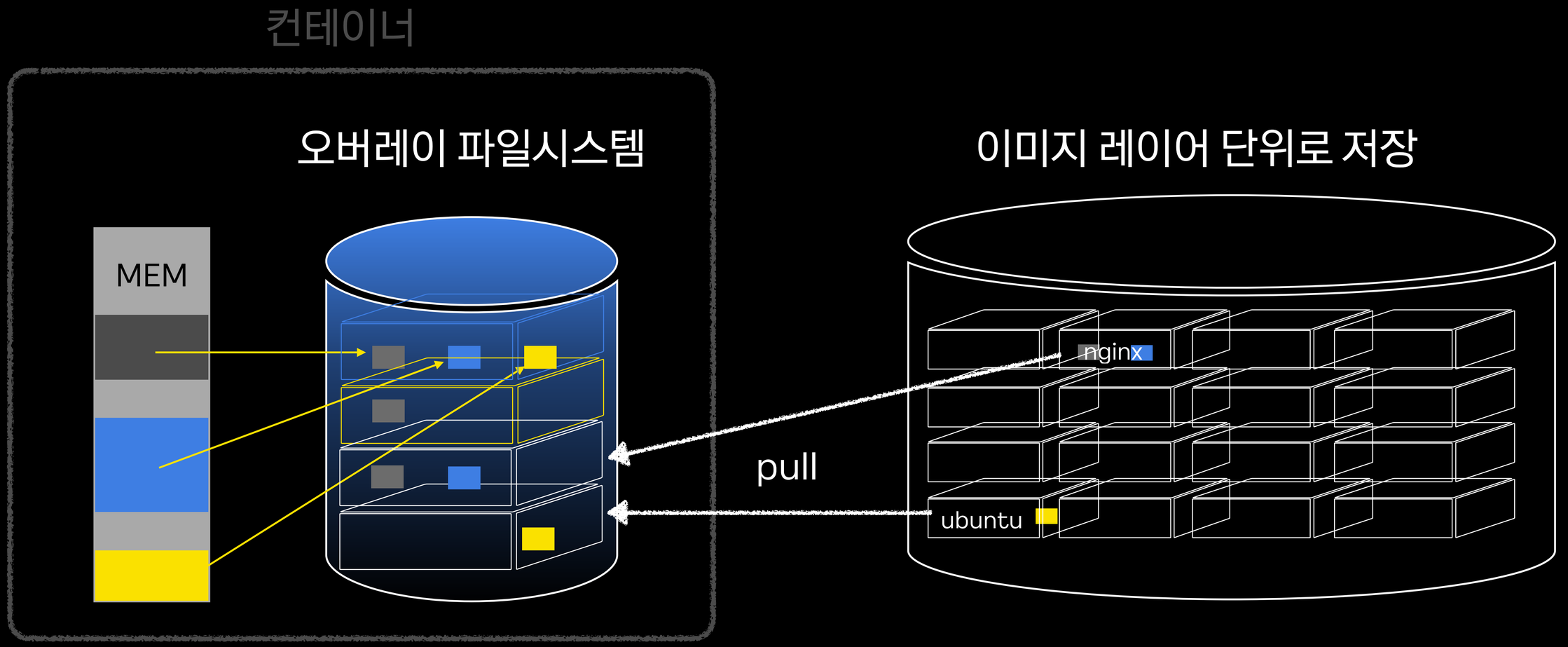
컨테이너 전용 루트파일시스템



1)

1) Container vs Process, https://sites.google.com/site/mytechnicalcollection/cloud-computing/docker/container-vs-process

이미지 중복 문제 해결



탈옥 말고 다른 문제는 없나요?

해결 과제

탈옥 문제	pivot_root
중복 문제	오버레이 파일시스템
격리 안됨	네임스페이스
루트 권한	
자원 보장	Cgroups

"컨테이너 격리와 자원" 편에서 다뤄 보겠습니다

if(kakao)2022

컨테이너 격리와 자원

김삼영 sam.0
카카오엔터프라이즈

컨테이너 격리

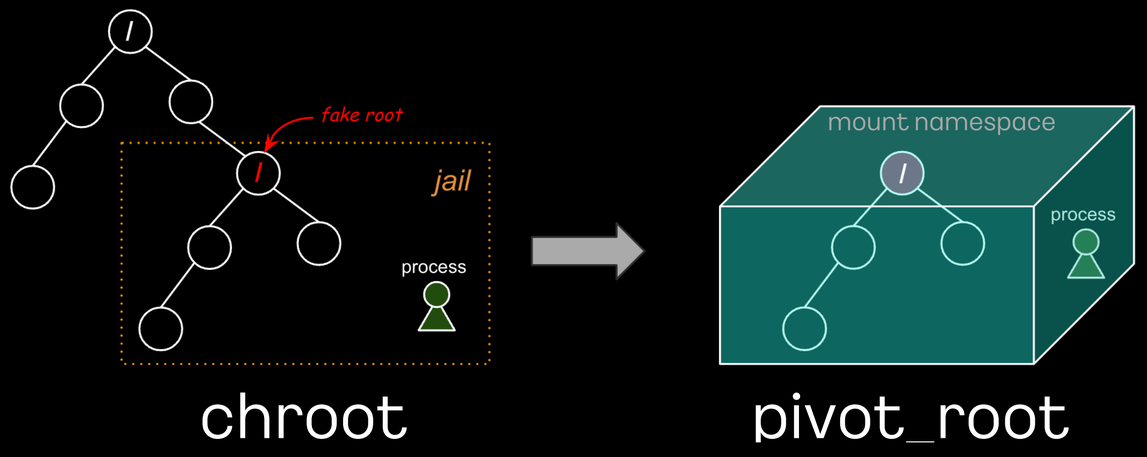
컨테이너 자원

컨테이너 격리

컨테이너 자원

초기의 컨테이너

"전용루트파일시스템"으로 충분하다고 생각

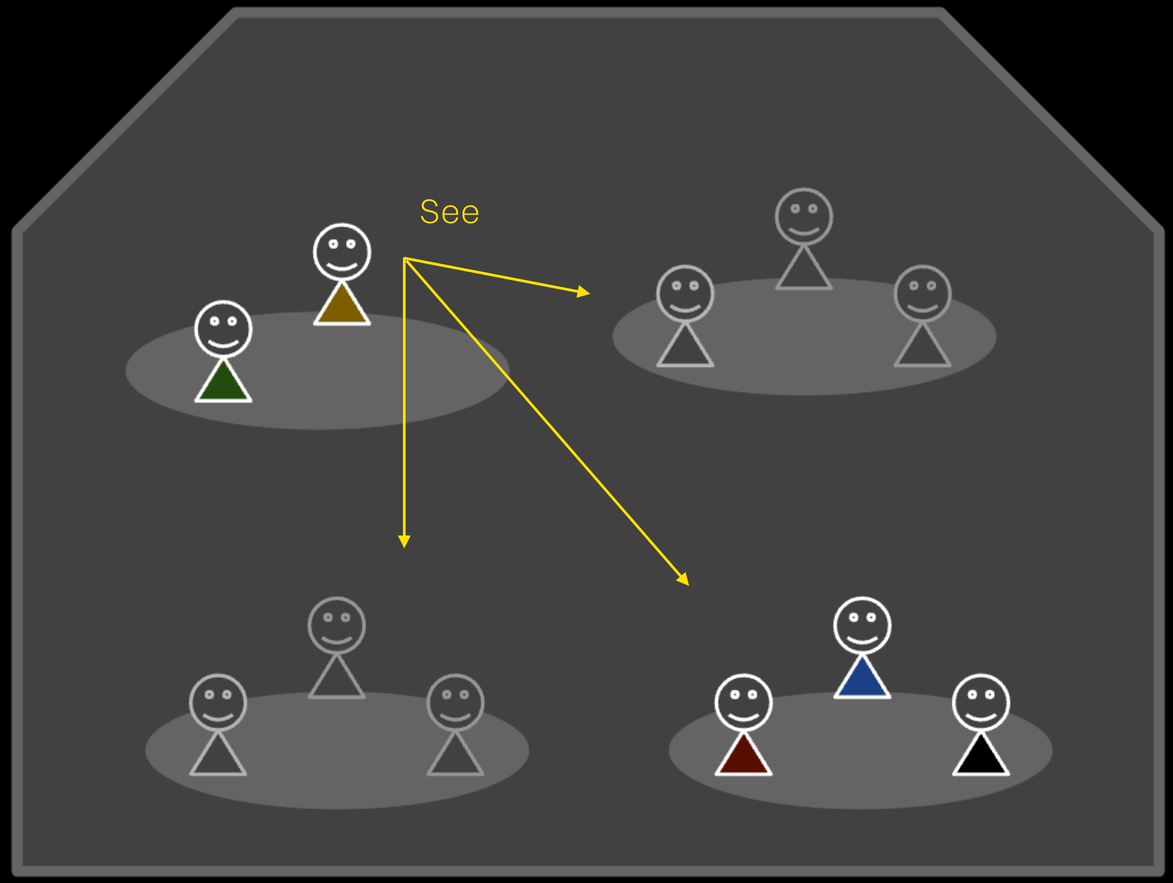


아니 근데 ...

컨테이너에서 호스트의 다른 프로세스들이 다 보여요

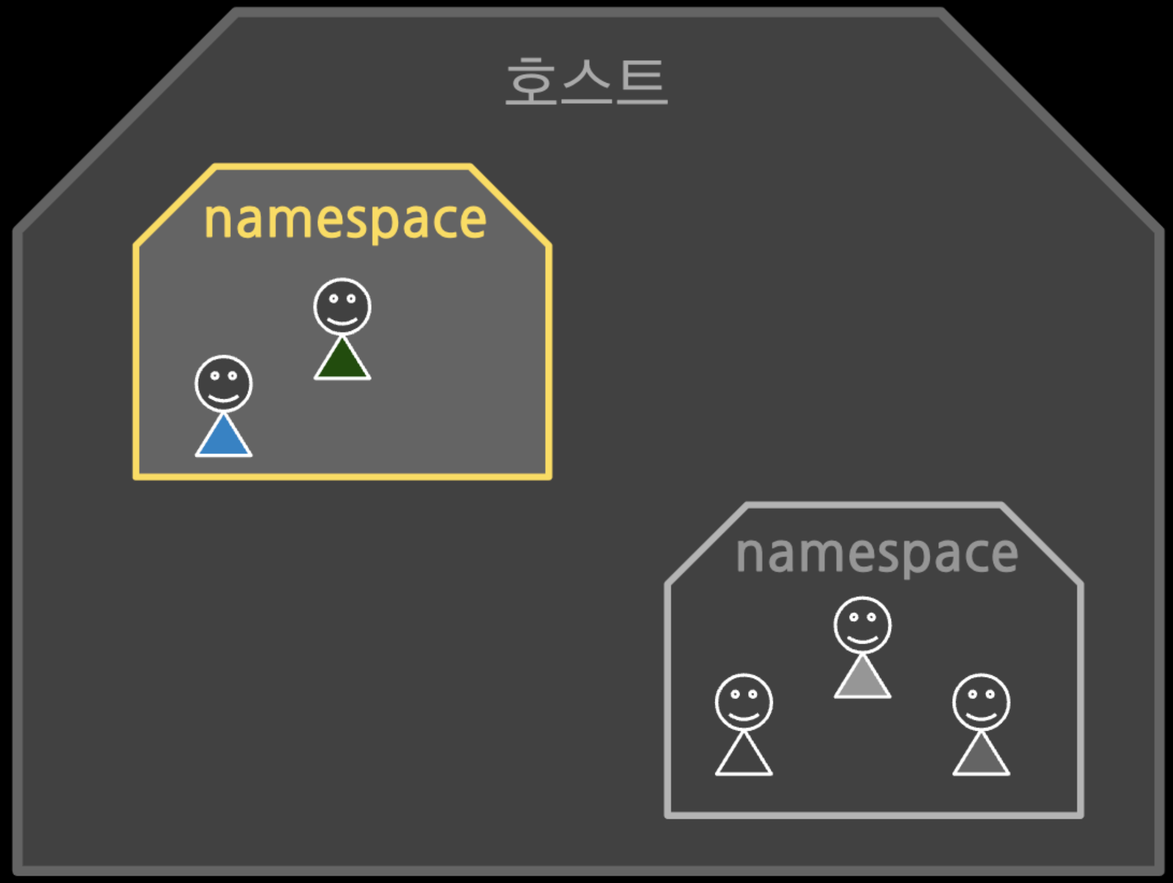
컨테이너에서 호스트의 포트를 사용해요

컨테이너에 루트 권한이 있어요



네임스페이스

- 프로세스에 격리된 환경 제공
- 리눅스 커널 기능 (2002, v2.4.19)
- 컨테이너 레디 (2013, v3.8)



네임스페이스 특징

- 모든 프로세스는 타입별로 네임스페이스에 속함
- 자식 프로세스는 부모의 네임스페이스를 상속함

네임스페이스 사용 방법

(사용법) unshare [옵션] [프로그램 [arguments ...]]

(옵션)

-m, --mount

-u, --uts

-i, --ipc

-p, --pid

-n, --net

-U, --user

[실습4] 네임스페이스

네임스페이스 확인 방법 #1

```
# ls -al /proc/$$/ns
  ipc -> 'ipc:[4026531839]'
  mnt -> 'mnt:[4026531840]'
  net -> 'net:[4026531993]'
  pid -> 'pid:[4026531836]'
  pid_for_children -> 'pid:[4026531836]'
  user -> 'user:[4026531837]'
  uts -> 'uts:[4026531838]'

# readlink /proc/$$/ns/mnt
  mnt:[4026531840]
```

네임스페이스 확인 방법 #2

```
# lsns -p 1
        4026531836 pid        115   1 root /sbin/init
        4026531837 user       115   1 root /sbin/init
        4026531838 uts        115   1 root /sbin/init
        4026531839 ipc        115   1 root /sbin/init
        4026531840 mnt        111   1 root /sbin/init
        4026531993 net        115   1 root /sbin/init

# lsns -t mnt -p 1
                NS TYPE NPROCS PID USER COMMAND
        4026531840 mnt      97   1 root /sbin/init
```

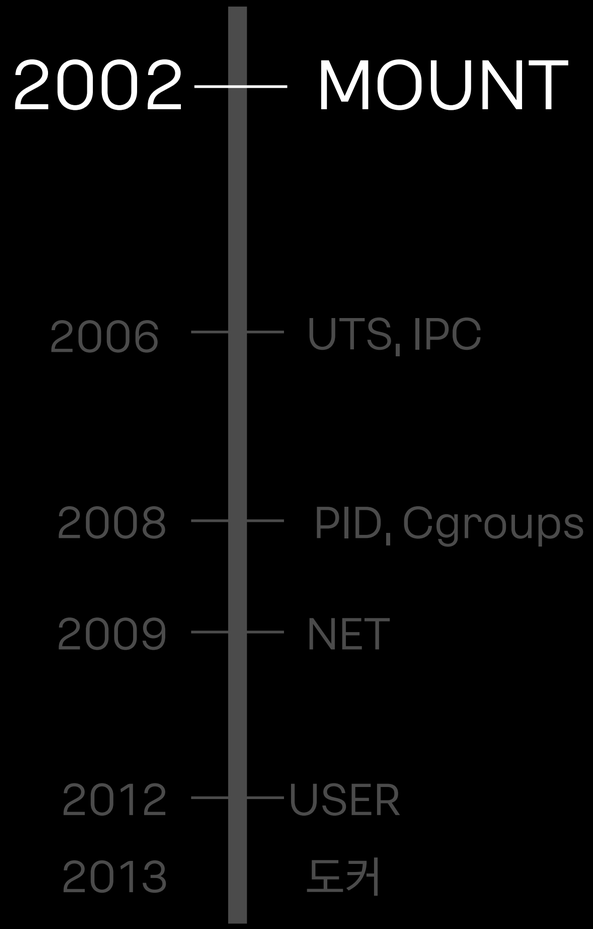
lsns, list namespace

-t : 네임스페이스 타입 예) pid, mnt, uts, ...

-p : 조회할 process id

마운트 네임스페이스

- 마운트 포인트 격리
- 최초의 네임스페이스



마운트 네임스페이스

```
# unshare -m
# lsns -p $$
```

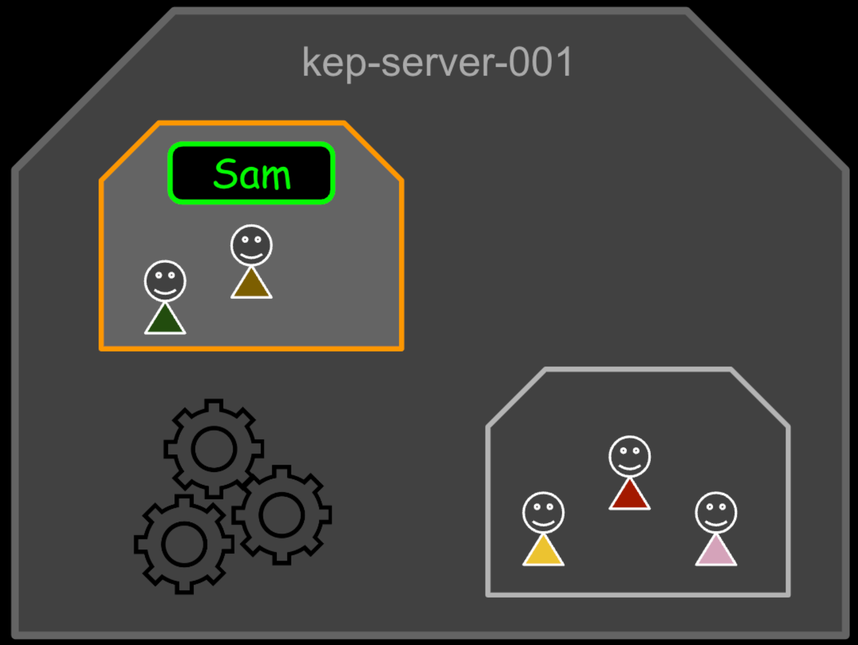
```
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    102     1 root /sbin/init
4026531836 pid       102     1 root /sbin/init
4026531837 user      102     1 root /sbin/init
4026531838 uts       102     1 root /sbin/init
4026531839 ipc       102     1 root /sbin/init
4026531993 net       102     1 root /sbin/init
4026532193 mnt         2 27163 root -bash
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

UTS 네임스페이스

- UTS, Unix Time Sharing (서버 나눠쓰기)
- 호스트명, 도메인명 격리



UTS 네임스페이스

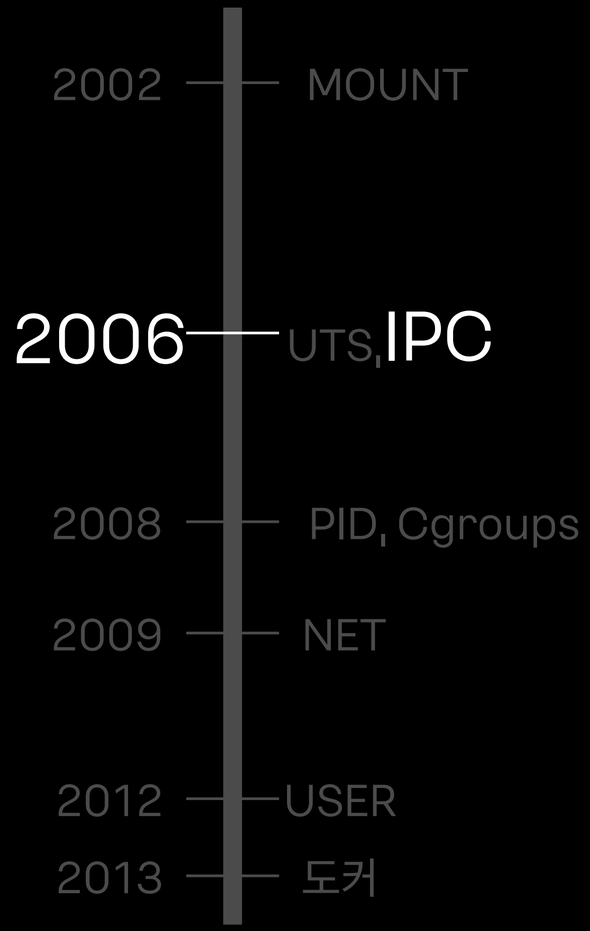
```
# unshare -u
# lsns -p $$
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    104     1 root /sbin/init
4026531836 pid       104     1 root /sbin/init
4026531837 user      104     1 root /sbin/init
4026531839 ipc       104     1 root /sbin/init
4026531840 mnt       100     1 root /sbin/init
4026531993 net       104     1 root /sbin/init
4026532193 uts         2 27207 root -bash

# hostname
ubuntu1804
# hostname Sam
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

IPC 네임스페이스

- IPC, Inter-Process Communication 격리
- Shared Memory, Pipe, Message Queue 등



IPC 네임스페이스

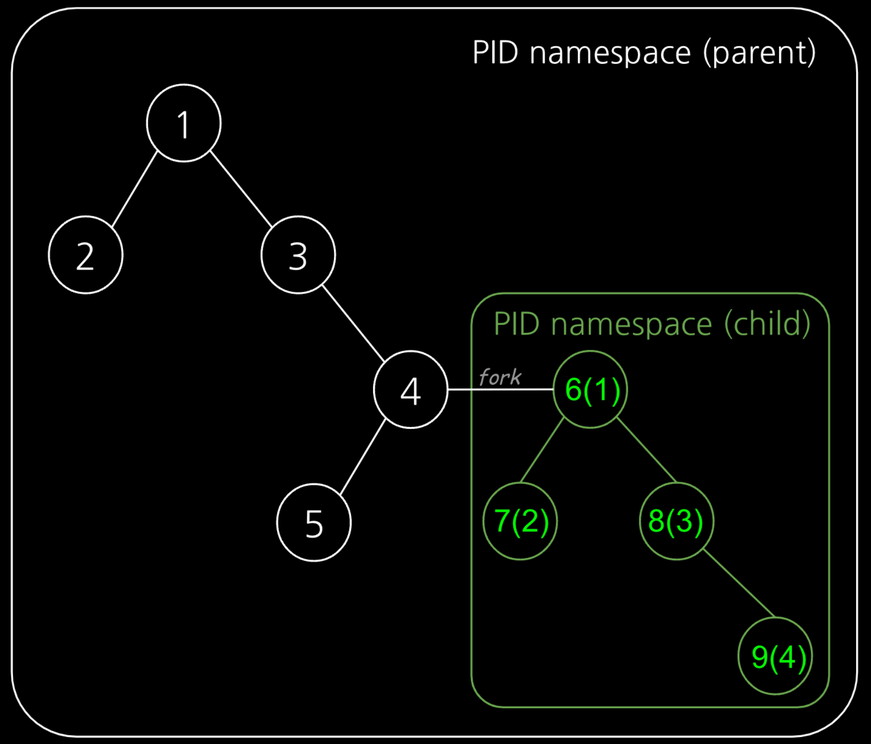
```
# unshare -i
# lsns -p $$
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    103     1 root /sbin/init
4026531836 pid       103     1 root /sbin/init
4026531837 user      103     1 root /sbin/init
4026531838 uts       103     1 root /sbin/init
4026531840 mnt        99     1 root /sbin/init
4026531993 net       103     1 root /sbin/init
4026532193 ipc         2 27221 root -bash
```

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

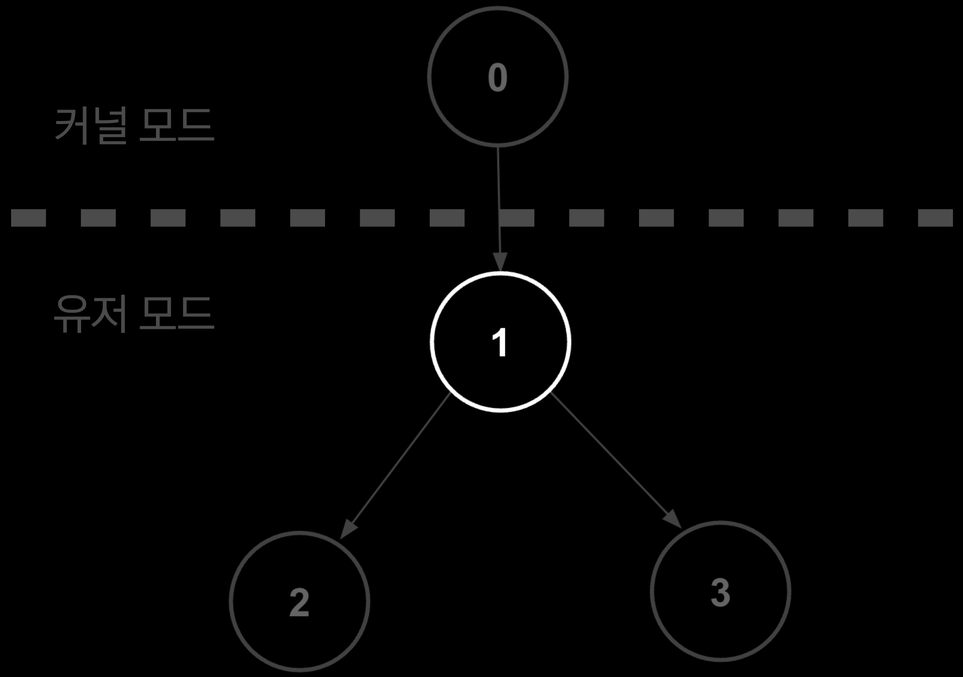
PID 네임스페이스

- PID (Process ID) 넘버스페이스를 격리
- 부모-자식 네임스페이스 중첩 구조
- 부모 네임스페이스 - (see) → 자식네임스페이스



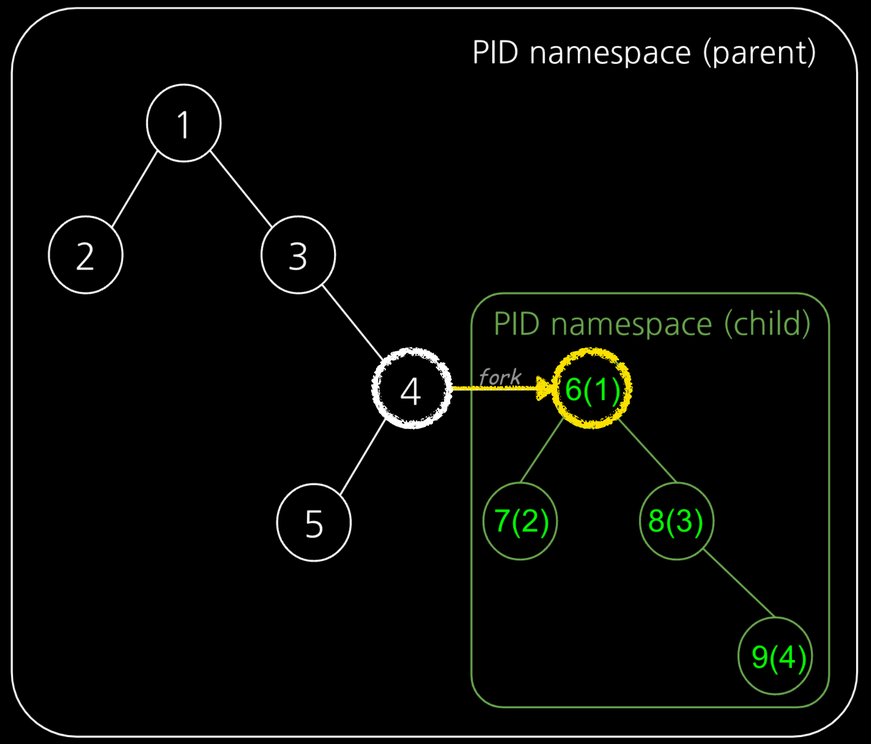
pid 1

- init 프로세스 (커널이 생성)
- 시그널 처리
- 좀비, 고아 프로세스 처리
- 죽으면 시스템 패닉 (reboot)



컨테이너 pid 1

- unshare 할 때 fork 하여 자식 PID 네임스페이스의 pid1 로 실행
- 시그널 처리
- 좀비, 고아 프로세스 처리
- 죽으면 컨테이너 종료



PID 네임스페이스 만들기

```
# unshare -fp --mount-proc /bin/sh
```

unshare 옵션 설명

-p : pid namespace

-f : fork

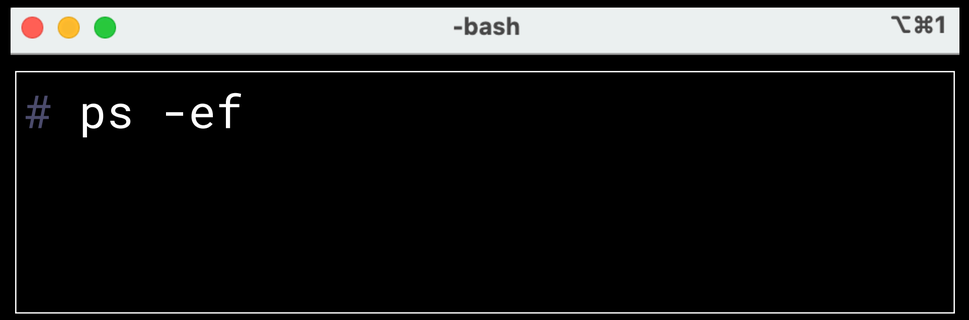
--mount-proc : proc 파일시스템 마운트

/proc

- 메모리 기반의 가상파일시스템
- 커널이 관리하는 시스템 정보 제공
- 시스템 모니터링과 분석에 활용

프로세스 확인

컨테이너



호스트

* pid (2611) 값은 실습환경 마다 다릅니다.

PID 네임스페이스 비교

컨테이너

-bash ⌥⌘1

```
# lsns -t pid -p 1

        NS TYPE NPROCS PID USER COMMAND
4026532204 pid       2   1 root /bin/sh
```

호스트

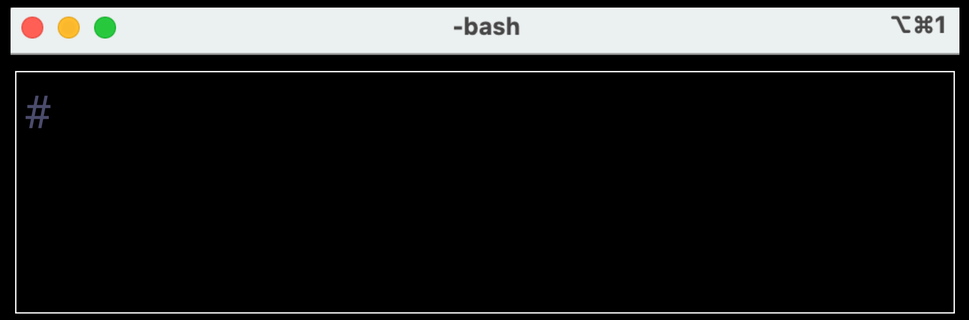
-bash ⌥⌘1

```
# lsns -t pid -p 2611

        NS TYPE NPROCS   PID USER COMMAND
4026532204 pid       1  2611 root /bin/sh
```

* pid (2611) 값은 실습환경 마다 다릅니다.

컨테이너 pid1 KILL ?

컨테이너



호스트

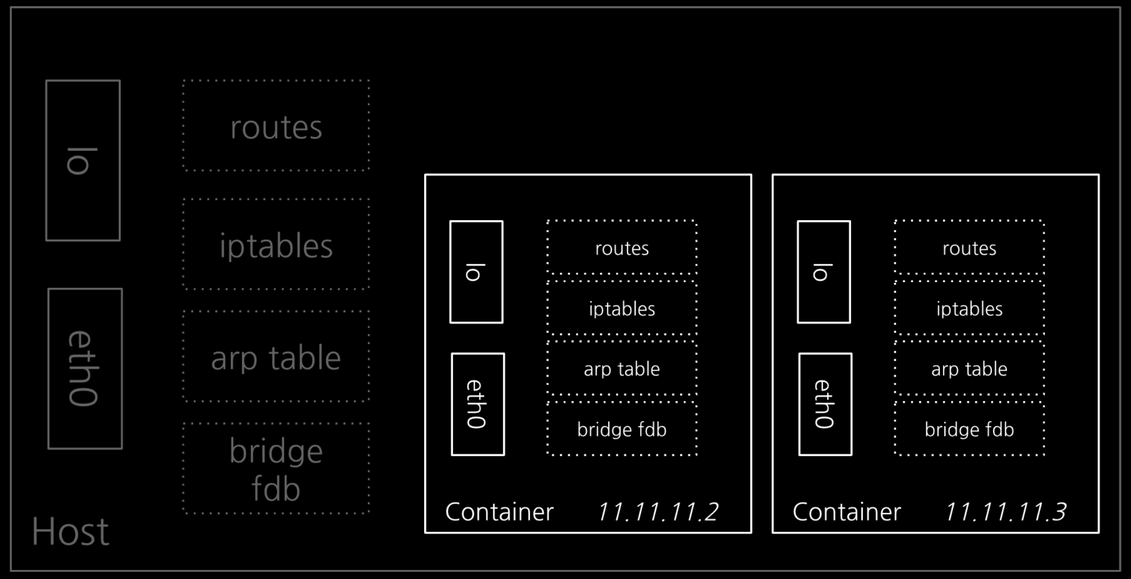
* pid (2611) 값은 실습환경 마다 다릅니다.

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

네트워크 네임스페이스

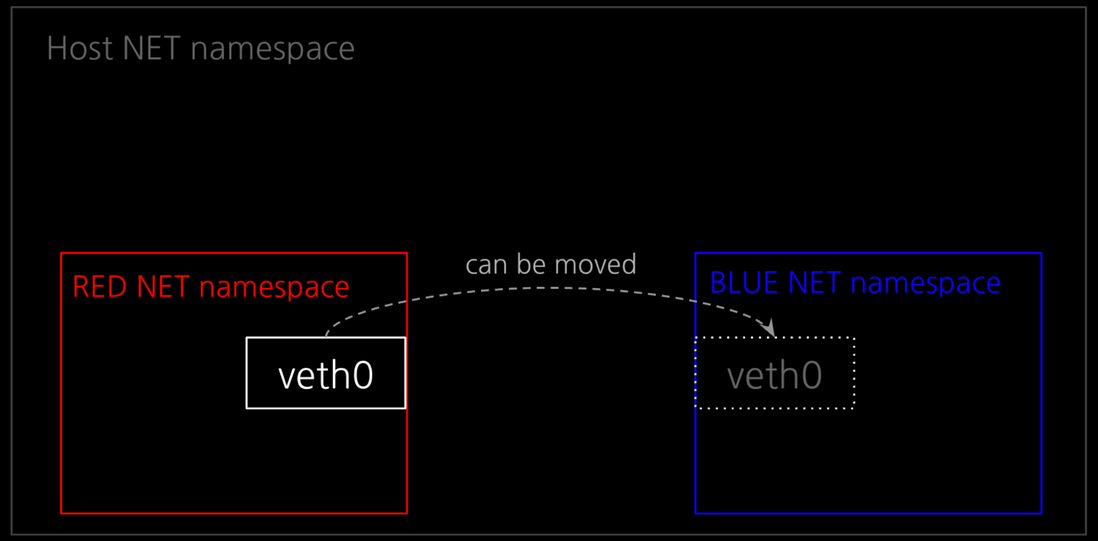
- 네트워크 스택 격리
- 네트워크 가상화. 가상 인터페이스(장치) 사용



IP주소, 라우팅테이블, 소켓, 방화벽, … 등 모든게 별도로 가상화 네트워크에 준비됩니다

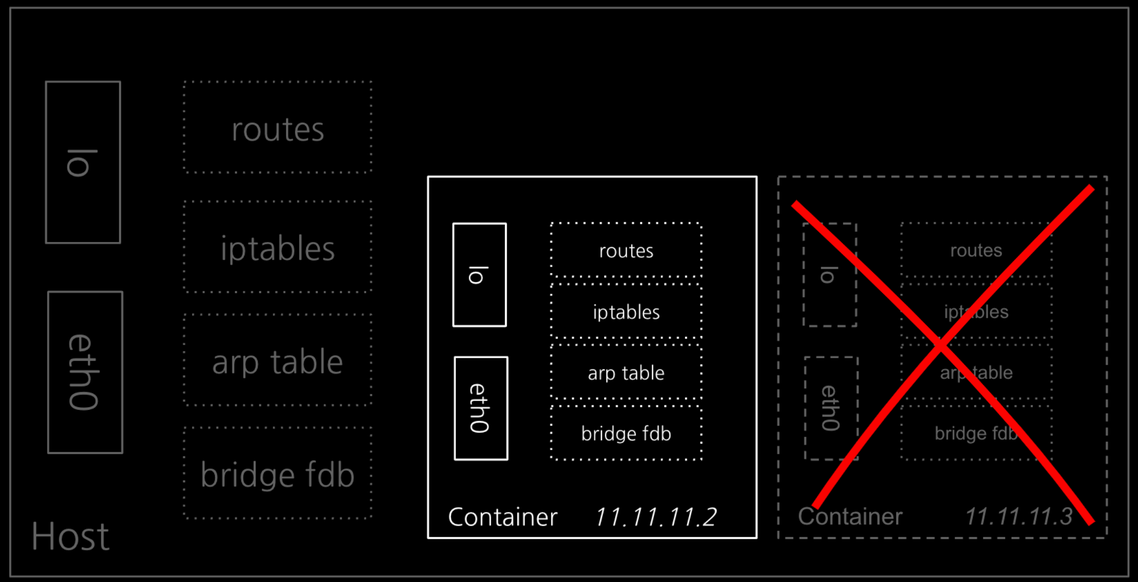
네트워크 인터페이스

- 여러 네트워크 네임스페이스에 걸쳐 있을 수 없음
- 다른 네트워크 네임스페이스로 이동할 수 있음
- 예) veth, bridge, vxlan, ...



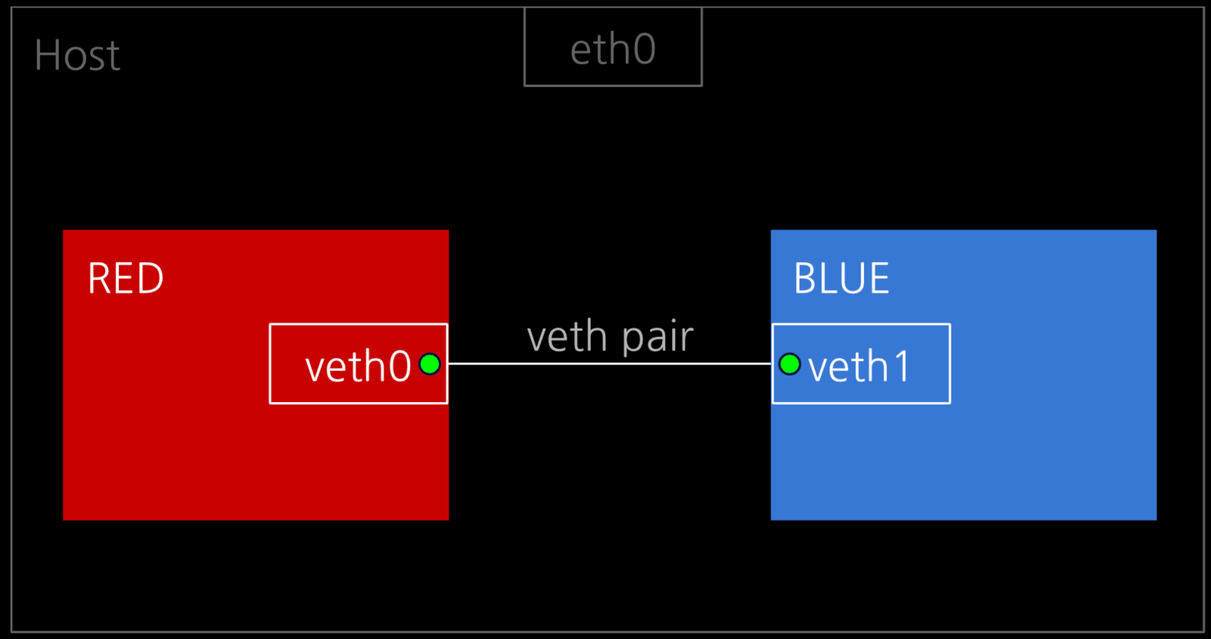
네트워크 네임스페이스 삭제

- 가상 인터페이스 : 삭제됨
- 물리 인터페이스 : 기존 네임스페이스로 복원됨



1:1 통신 실습

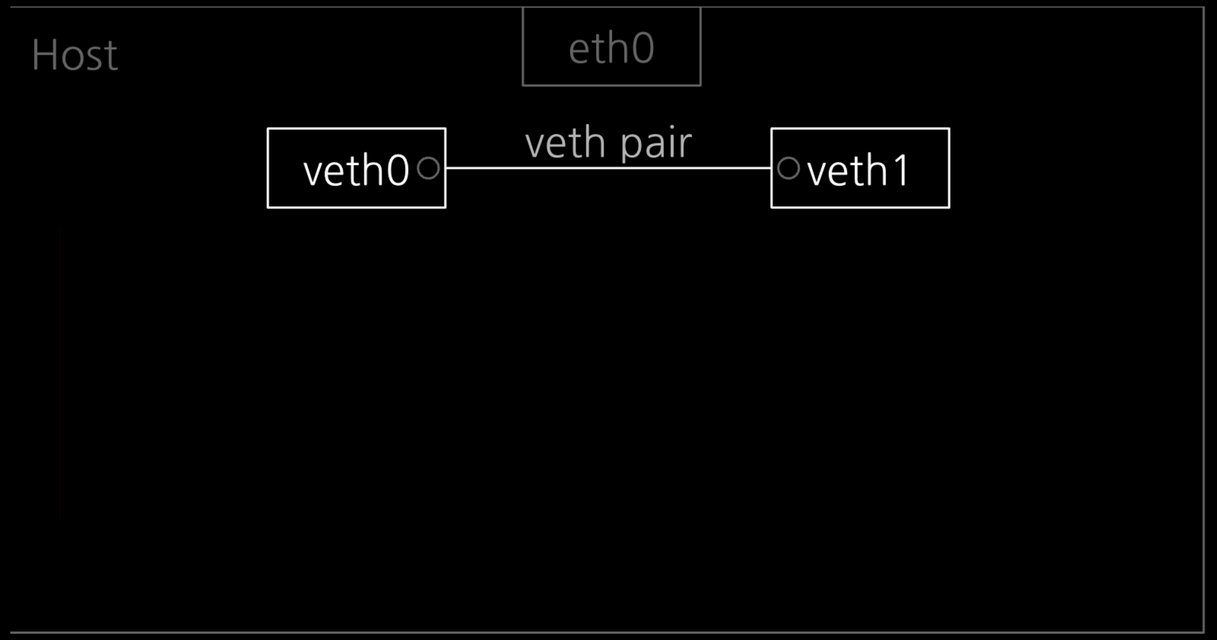
- 네트워크 네임스페이스를 격리하여
- RED/BLUE 컨테이너 간에 통신을 해봅시다



veth pair 설정

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

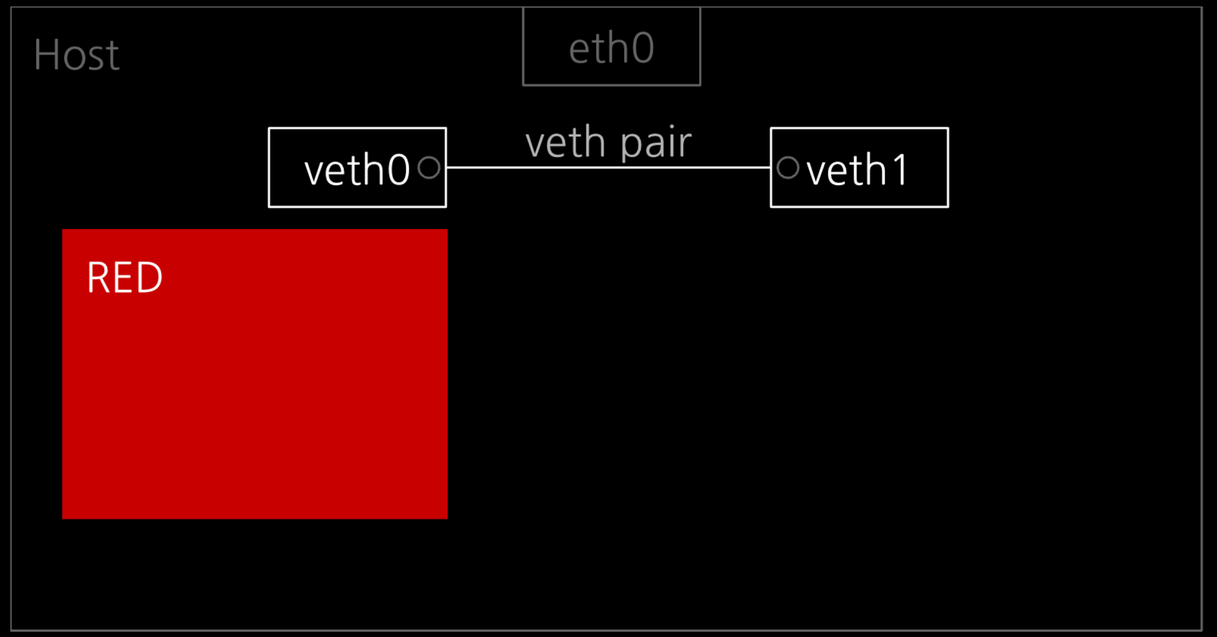
1)



1) https://man7.org/linux/man-pages/man8/ip-link.8.html

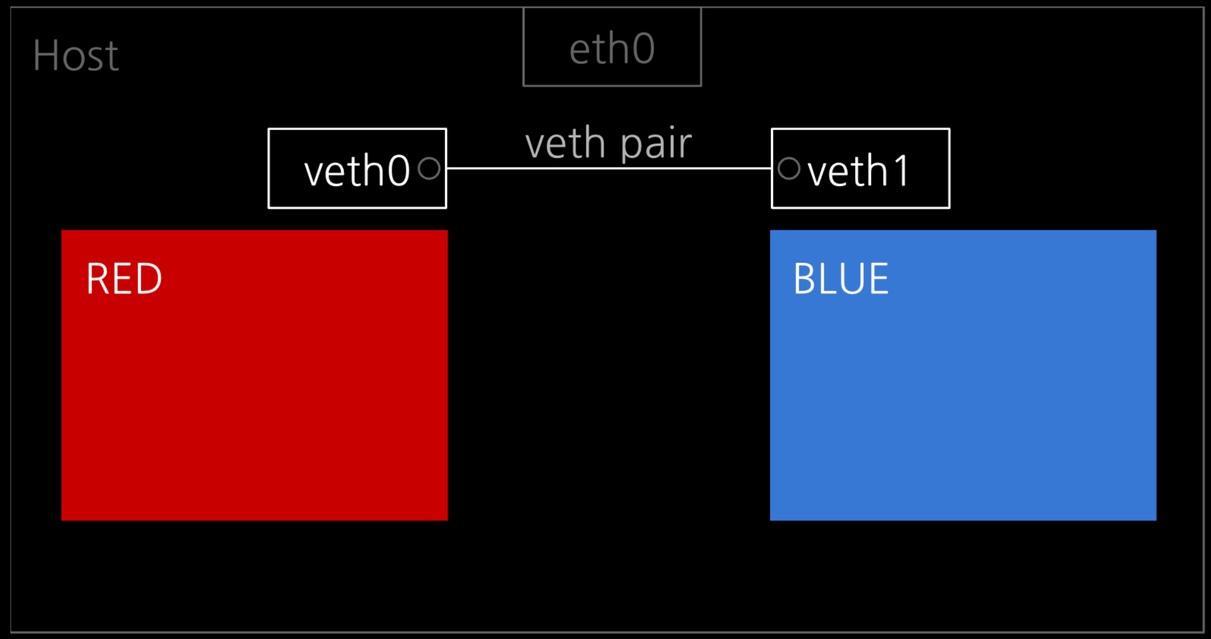
RED 네트워크 네임스페이스

```
# ip netns add RED
```



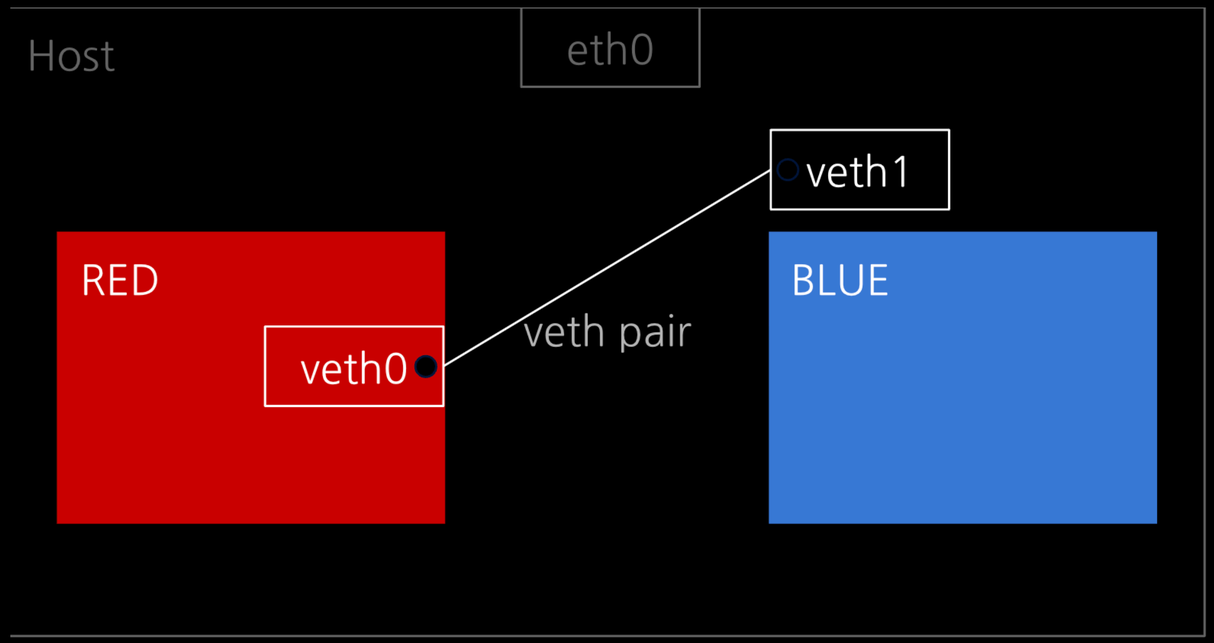
BLUE 네트워크 네임스페이스

```
# ip netns add BLUE
```



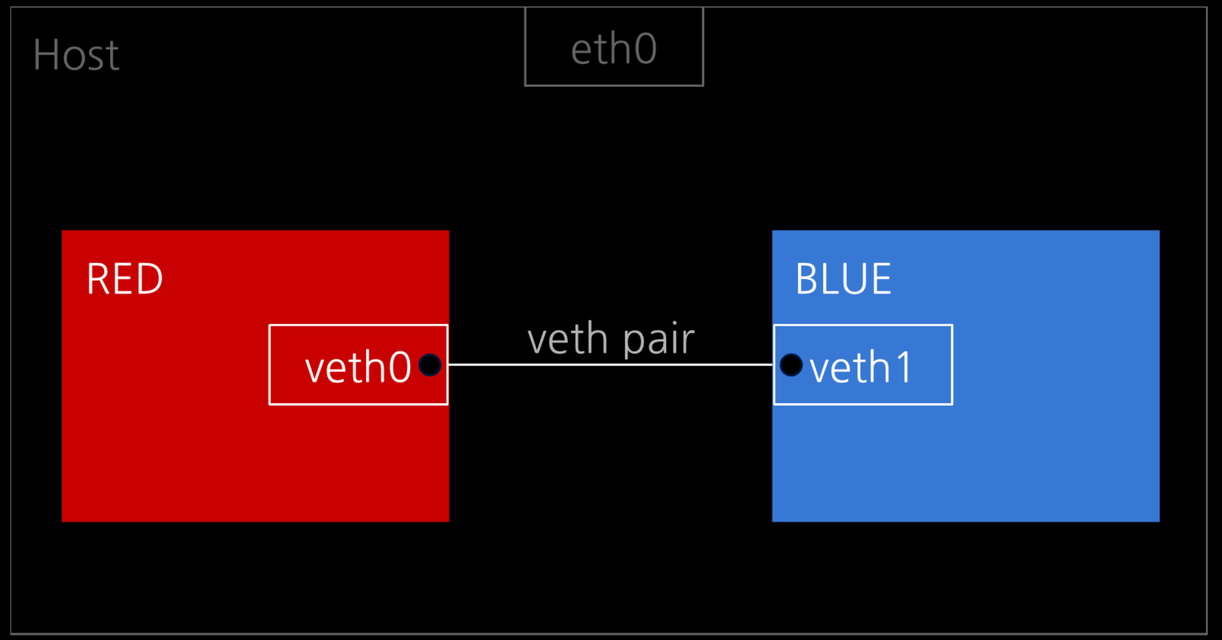
veth0→RED

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



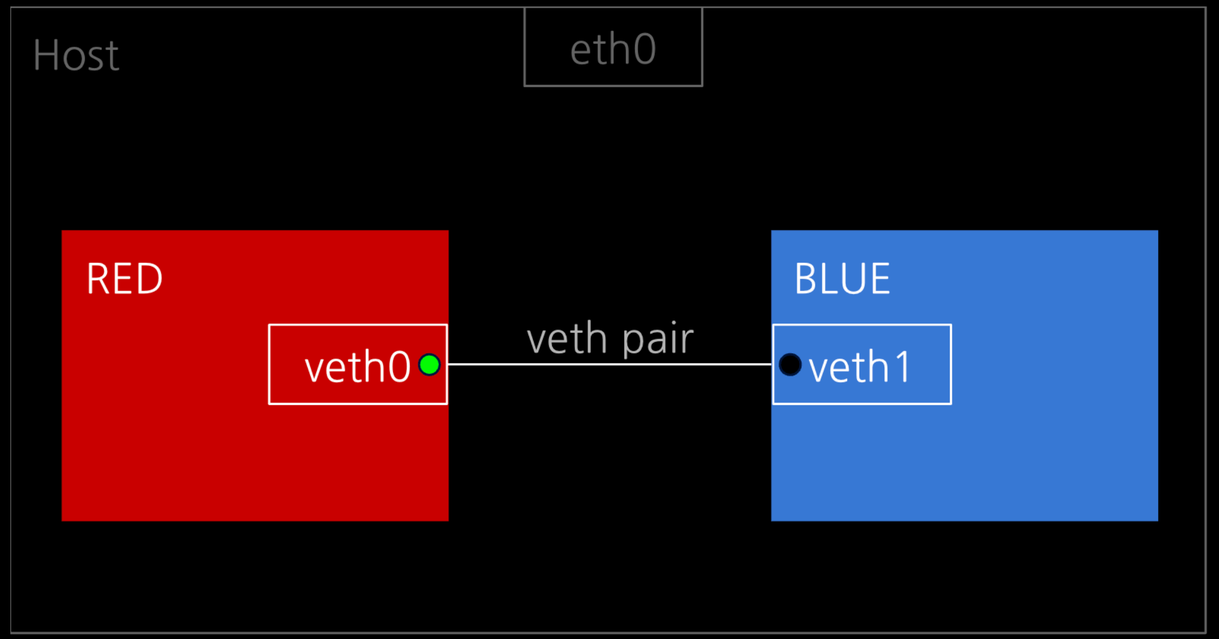
veth1→BLUE

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



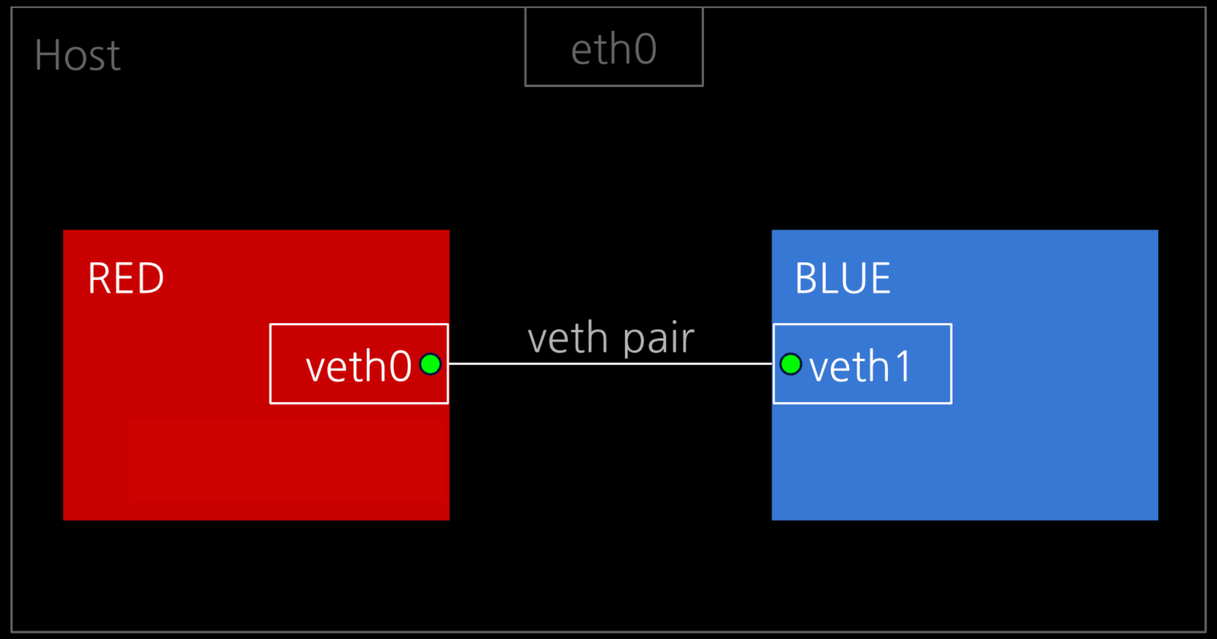
veth0 UP

```
# ip netns exec RED ip link set veth0 up
```



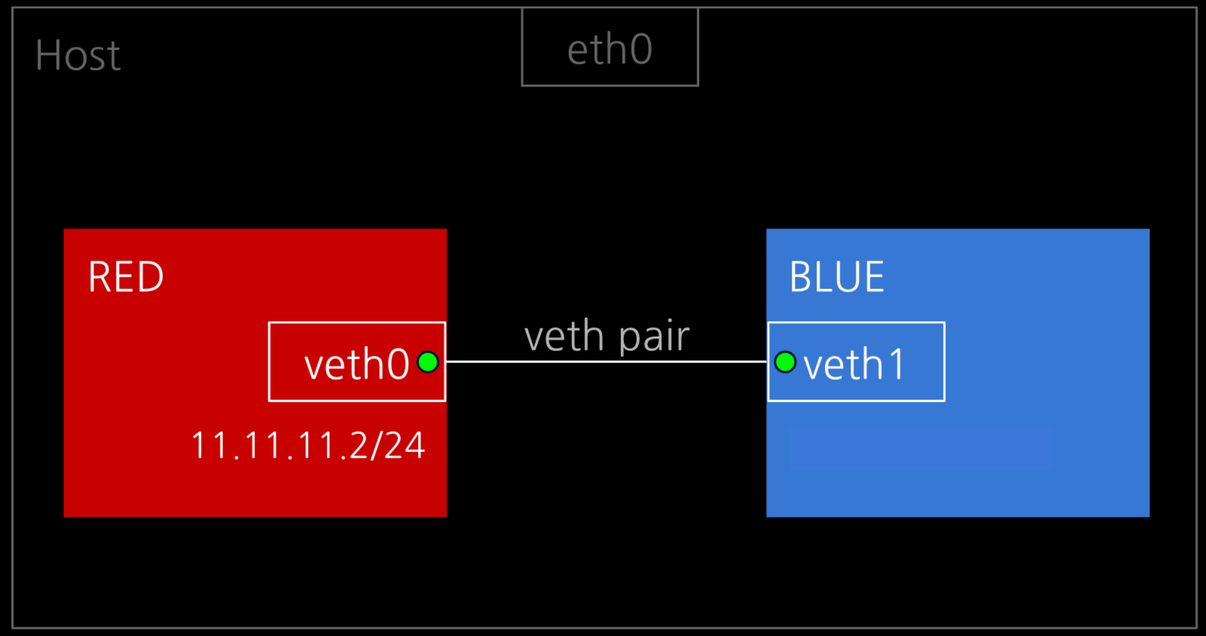
veth1 UP

```
# ip netns exec BLUE ip link set veth1 up
```



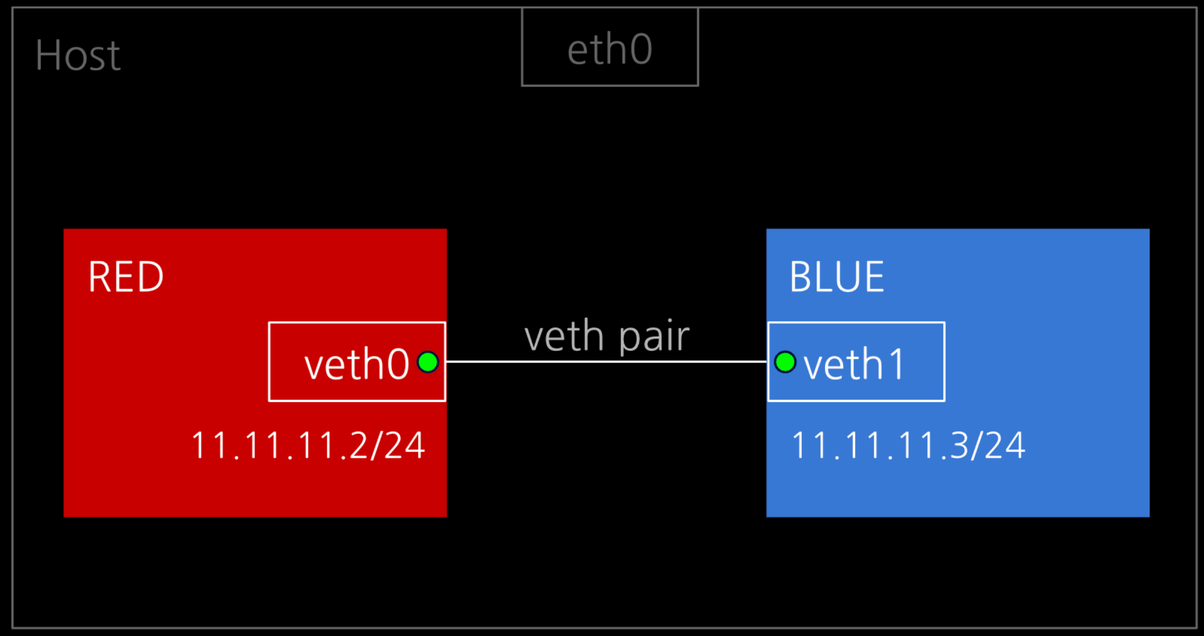
veth0 IP설정

```
# ip netns exec RED ip addr add 11.11.11.2/24 dev veth0
```



veth1 IP설정

```
# ip netns exec BLUE ip addr add 11.11.11.3/24 dev veth1
```



RED/BLUE 준비완료

RED/BLUE에 들어가보아요

```
# nsenter --net=/var/run/netns/{NS명}
```

nsenter 네임스페이스 진입

--net : 네트워크 네임스페이스

네트워크 비교 : RED vs 호스트

RED

-bash ⌥⌘1

```
# nsenter --net=/var/run/netns/RED
# ip a
# ip route
```

호스트

-bash ⌥⌘1

```
# ip a
# ip route
```

네트워크 비교 : RED vs BLUE

RED

-bash ⌥⌘1

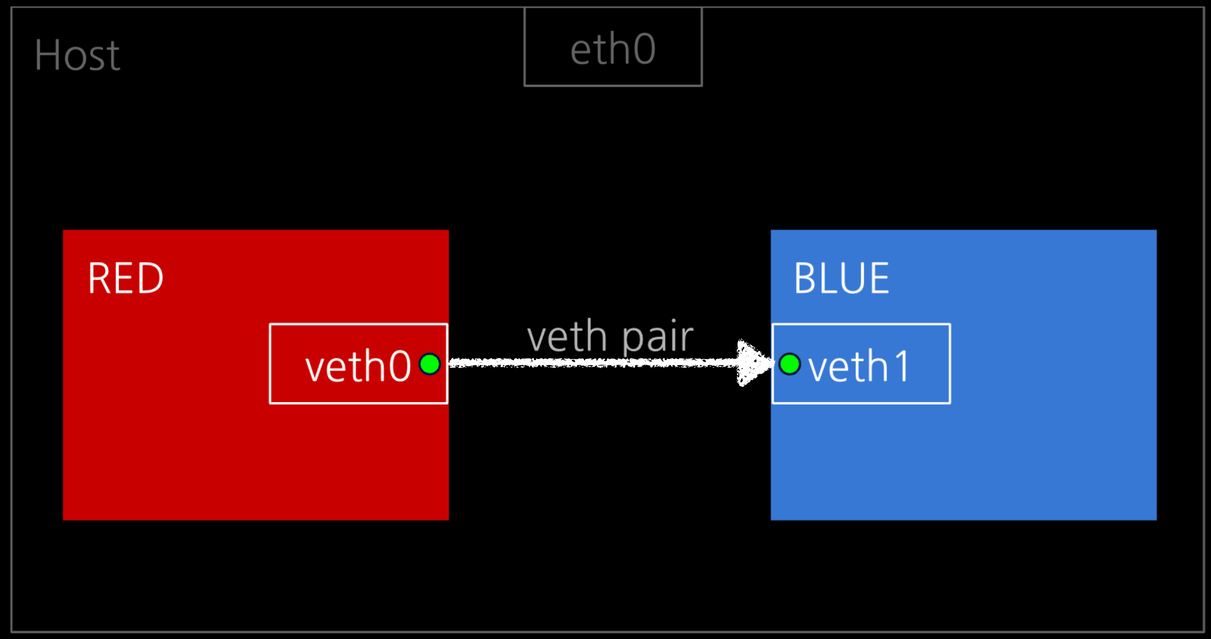
```
# nsenter --net=/var/run/netns/RED
# ip a
# ip route
```

BLUE

-bash ⌥⌘1

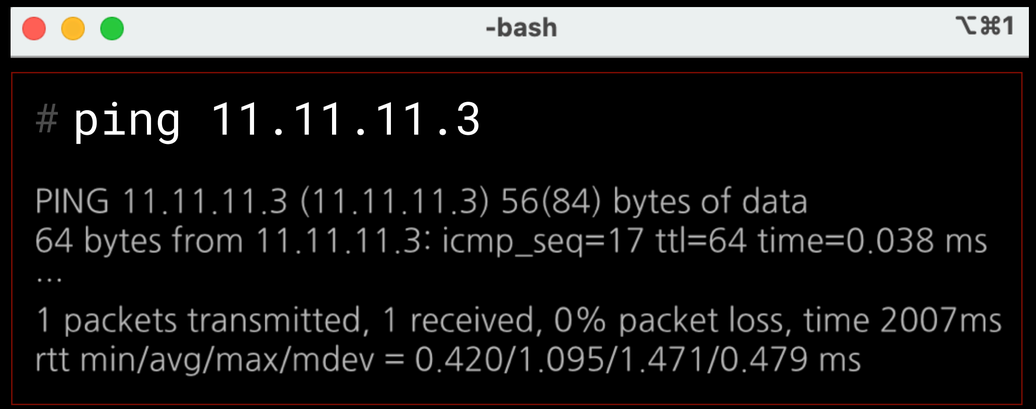
```
# nsenter --net=/var/run/netns/BLUE
# ip a
# ip route
```

ping 테스트, RED → BLUE



1:1 통신 성공

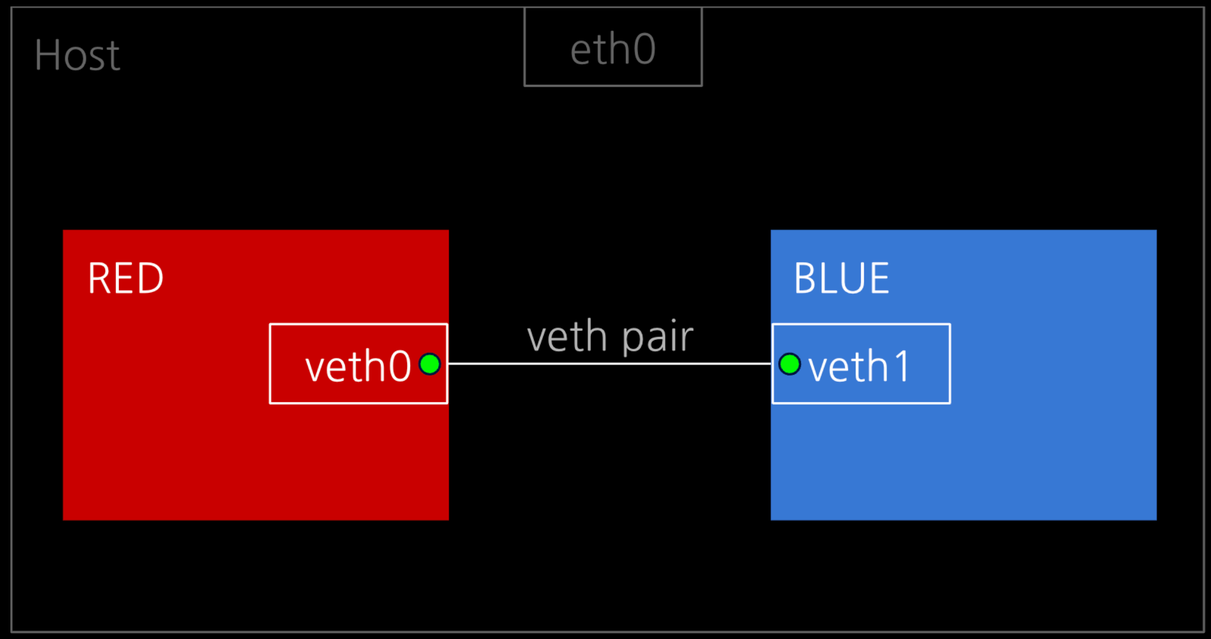
RED



BLUE

1:1 통신 실습 요약

- RED/BLUE 네트워크 격리 확인
- 격리된 RED/BLUE 간의 통신 확인



컨테이너 종료

RED

-bash ⌥⌘1

```
# exit

#
```

BLUE

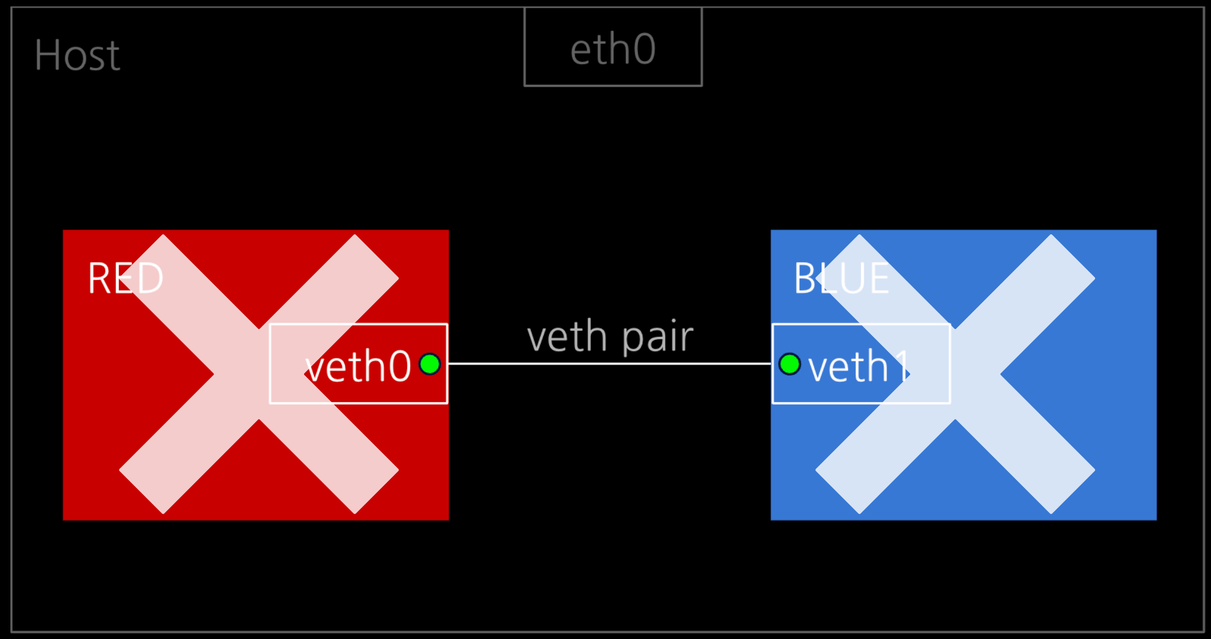
-bash ⌥⌘1

```
# exit

#
```

실습환경 초기화

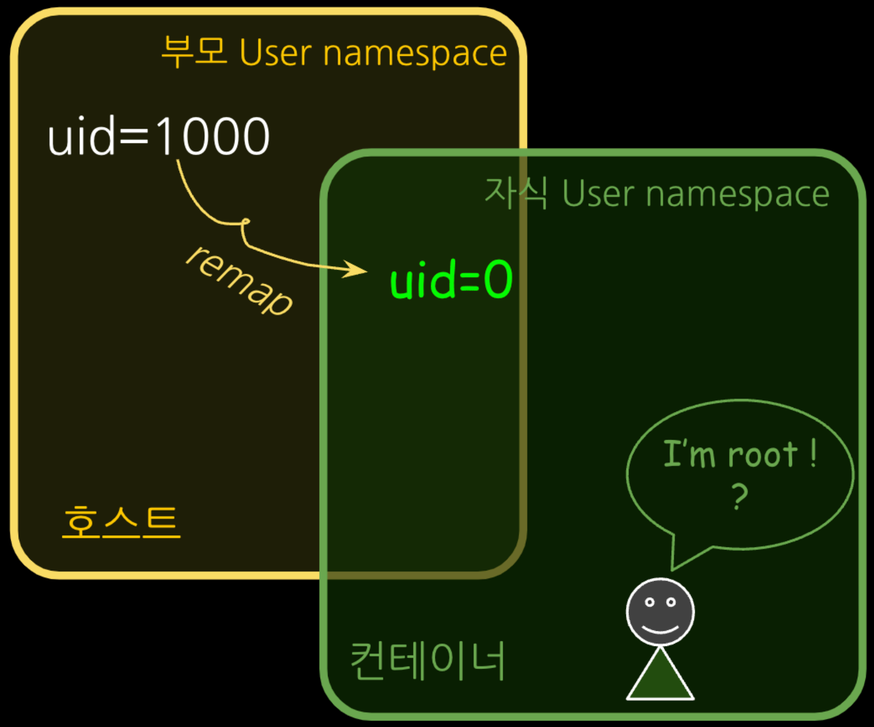
네트워크 네임스페이스 삭제

```
# ip netns del RED;
# ip netns del BLUE;
```



USER 네임스페이스

- UID/GID 넘버스페이스 격리
- 컨테이너의 루트권한 문제를 해결함
- 부모-자식 네임스페이스의 중첩 구조
- UID/GID Remap



(안내) 일반계정으로 전환해 주세요

```
root@ubuntu1804:/tmp# exit
exit
vagrant@ubuntu1804:~$
```

(안내) 일반계정에서 도커 사용하기

```
$ sudo usermod -aG docker {계정명}
```

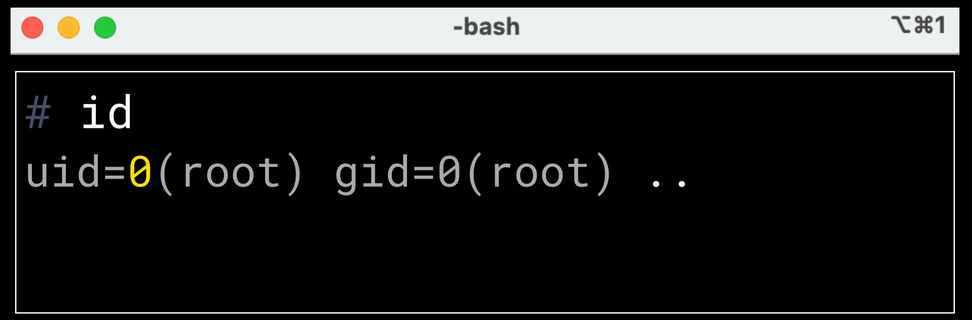
- 일반계정에서 docker CLI 사용 ~ docker 그룹에 추가
- 권한 적용을 위해 터미널을 종료 후 재접속 해주세요

일반계정으로 컨테이너 실행

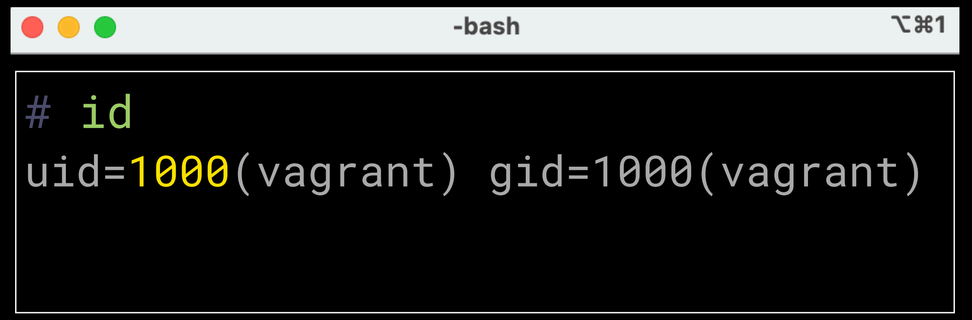
```
$ docker run -it ubuntu /bin/sh
```

컨테이너와 호스트 계정 비교

컨테이너



호스트



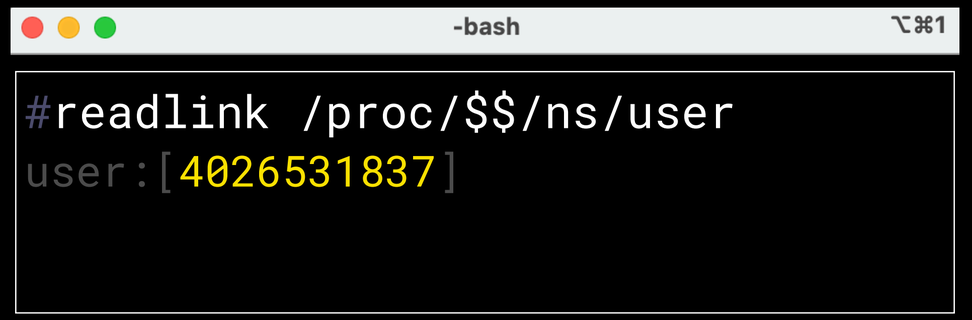
컨테이너 프로세스 계정 확인

컨테이너

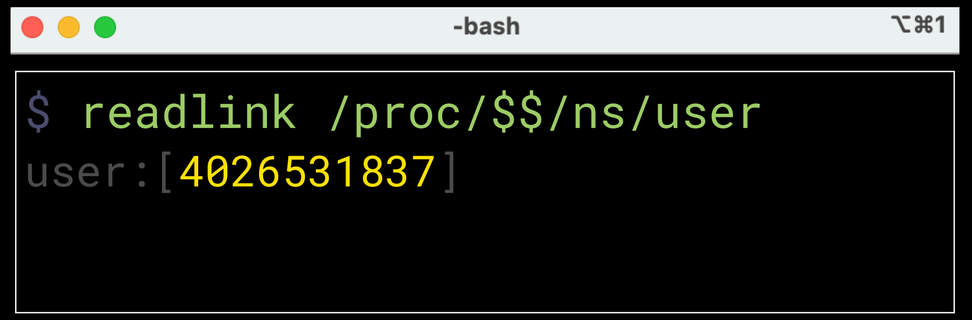
호스트

USER 네임스페이스 비교 (동일)

컨테이너



호스트



* inode (4026531837) 값은 실습환경 마다 다릅니다.

다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

도커의 root 사용

- 패키지 인스톨이 쉽다
- 시스템 리소스 이용에 제약이 없다

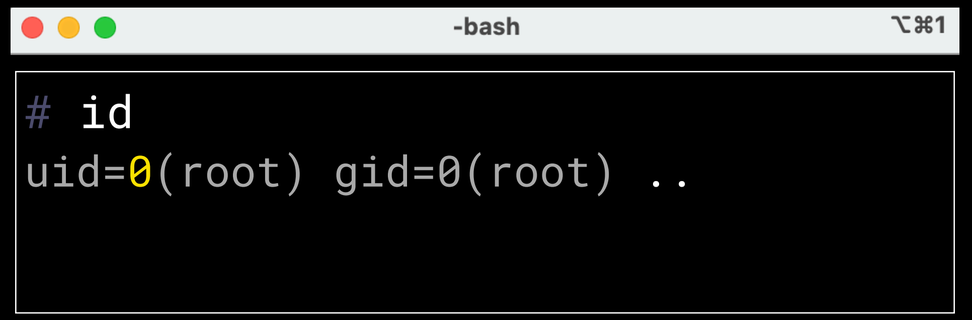
But ... 보안에 취약

USER 네임스페이스 격리

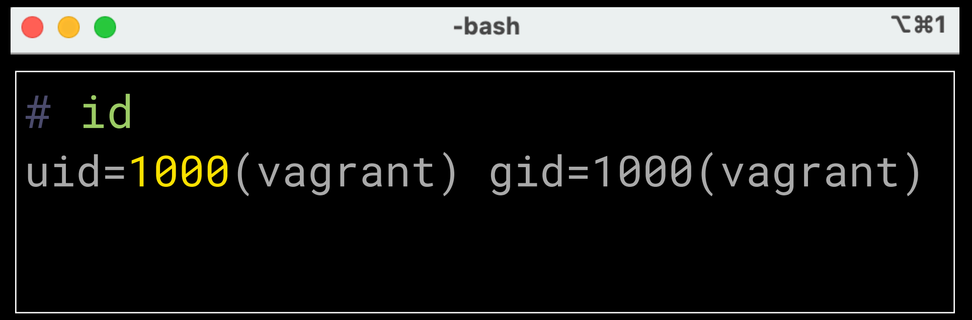
```
$ unshare -U --map-root-user /bin/sh
```

컨테이너와 호스트 계정 비교

컨테이너



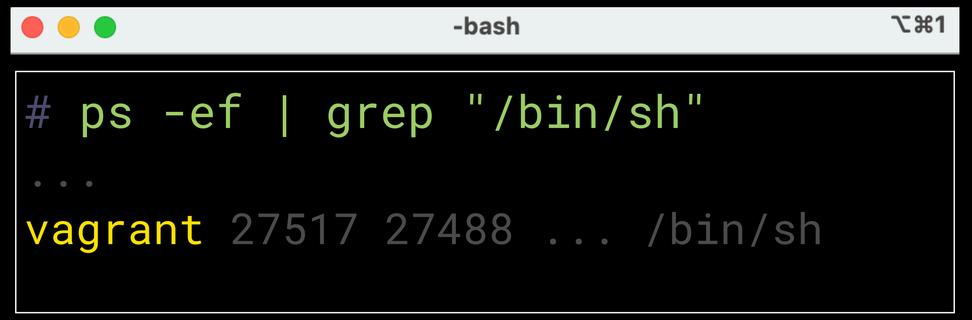
호스트



컨테이너 프로세스 계정 확인

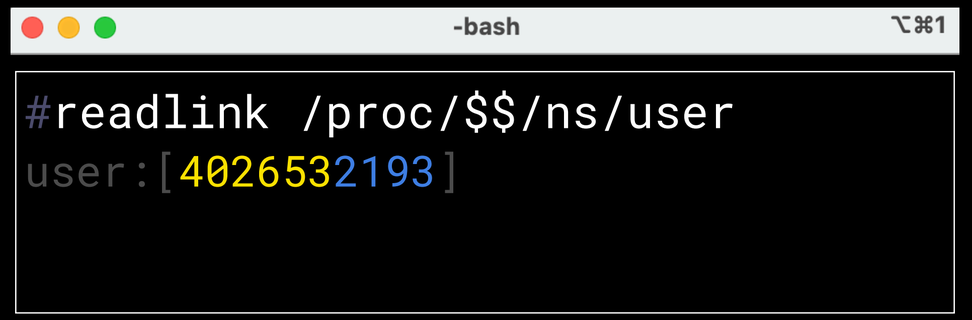
컨테이너

호스트

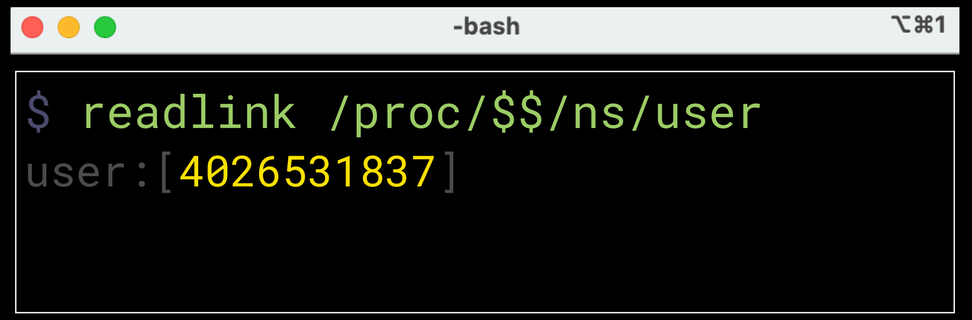


USER 네임스페이스 비교 (다름)

컨테이너



호스트

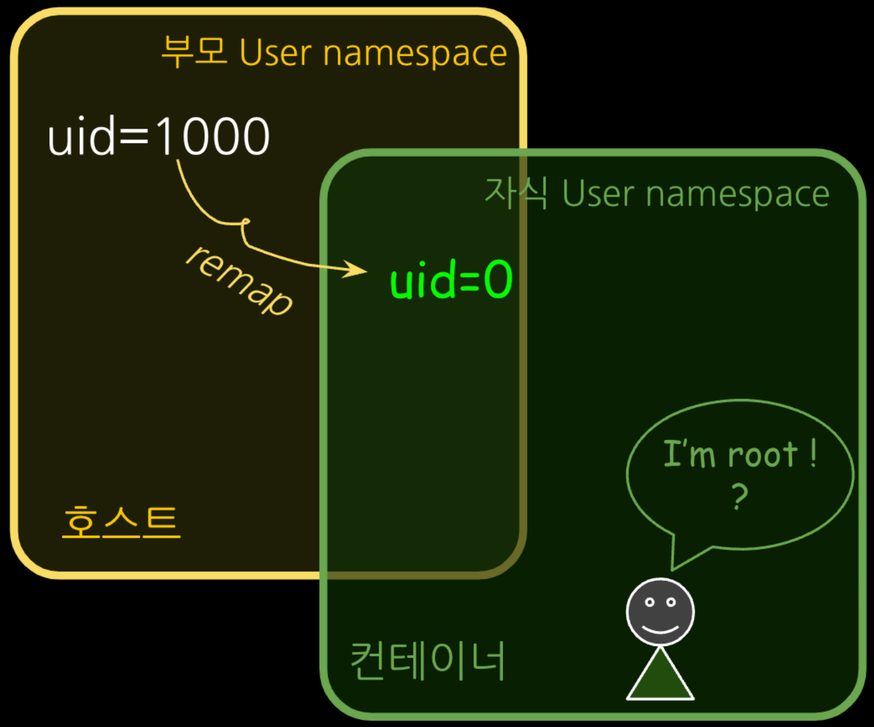


* inode (4026531837) 값은 실습환경 마다 다릅니다.

USER 네임스페이스

컨테이너 안에서만 root

- USER 네임스페이스 간 UID/GID Remap



다음 실습을 위해 컨테이너를 종료해 주세요

```
# exit
root@ubuntu1804:/tmp#
```

도커의 USER 네임스페이스 지원

- 도커 v1.10+
- 호스트 UID/GID Remap
- 보안관점에서 큰 진보
- But, 기본 설정은 USER 네임스페이스를 쓰지 않음

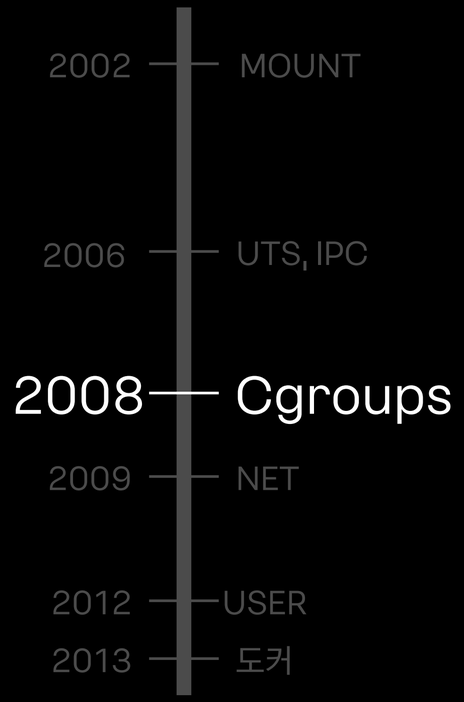
[요약] 네임스페이스 종류

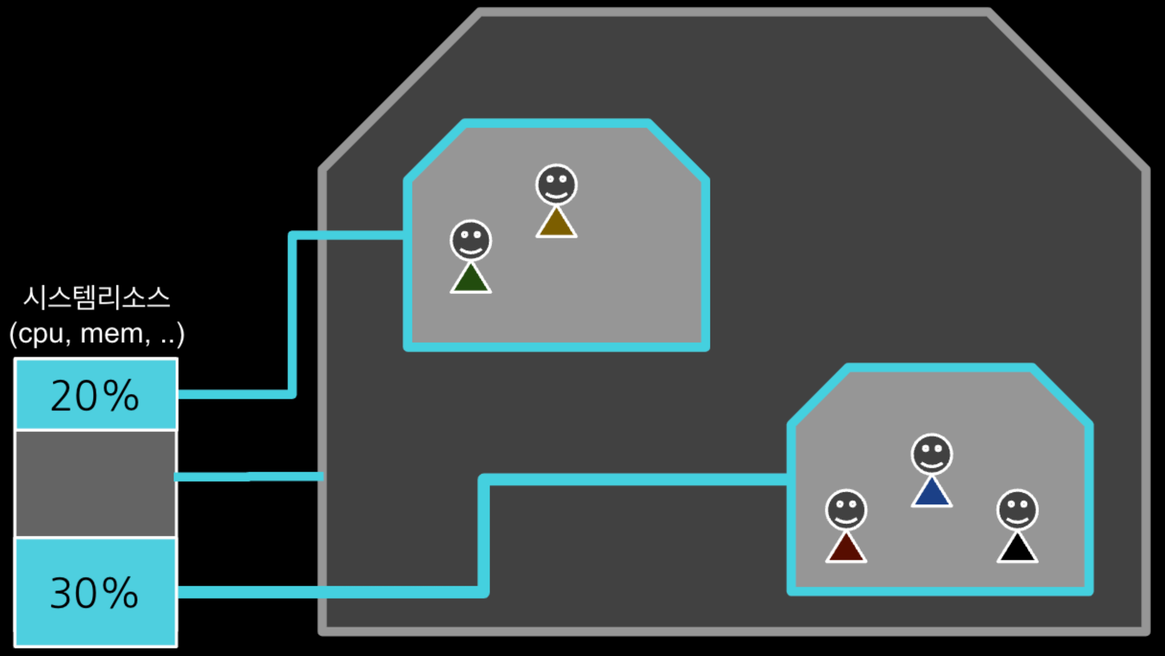
네임스페이스 구분	설명
마운트 네임스페이스	마운트 포인트 격리 (2002)
UTS 네임스페이스	hostname, domain name 격리 (2006)
IPC 네임스페이스	IPC 격리 (2006)
PID 네임스페이스	pid 넘버스페이스 격리 (2008)
네트워크 네임스페이스	네트워크 스택 가상화 및 격리 (2009)
USER 네임스페이스	UID/GID 넘버스페이스 격리 (2012)

컨테이너 자원

Cgroups

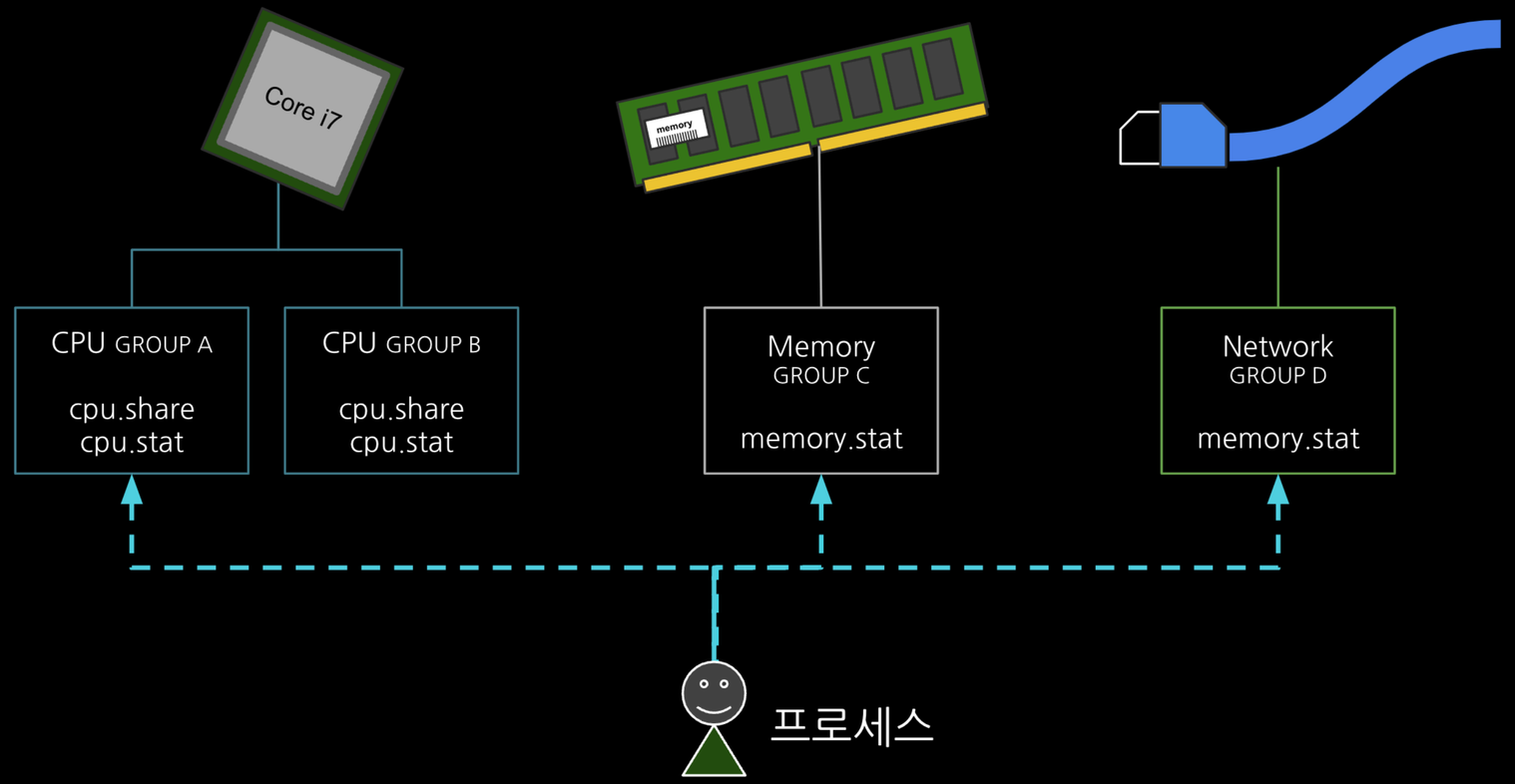
- Cgroups, Control Groups
- 컨테이너 별로 자원을 분배하고 limit 내에서 운용





Cgroups

- 하나 또는 복수의 장치를 묶어서 그룹
- 프로세스가 사용하는 리소스 통제



Cgroup 파일시스템

- 자원 할당과 제어를 파일시스템으로 제공합니다
- Cgroup 네임스페이스로 격리할 수 있습니다

```
/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
```

1)

1) man cgroups, https://man7.org/linux/man-pages/man7/cgroups.7.html

Cgroup 파일시스템

```
tree -L 1 /sys/fs/cgroup/cpu
```

```
/sys/fs/cgroup/cpu
├── cgroup.clone_children
├── cgroup.procs
├── cgroup.sane_behavior
├── cpuacct.stat
├── cpuacct.usage
├── cpuacct.usage_all
├── cpuacct.usage_percpu
├── cpuacct.usage_percpu_sys
├── cpuacct.usage_percpu_user
├── cpuacct.usage_sys
├── cpuacct.usage_user
├── cpu.cfs_period_us
├── cpu.cfs_quota_us
├── cpu.shares
├── cpu.stat
├── notify_on_release
├── release_agent
├── system.slice
├── tasks
└── user.slice
```

[실습5] Cgroups

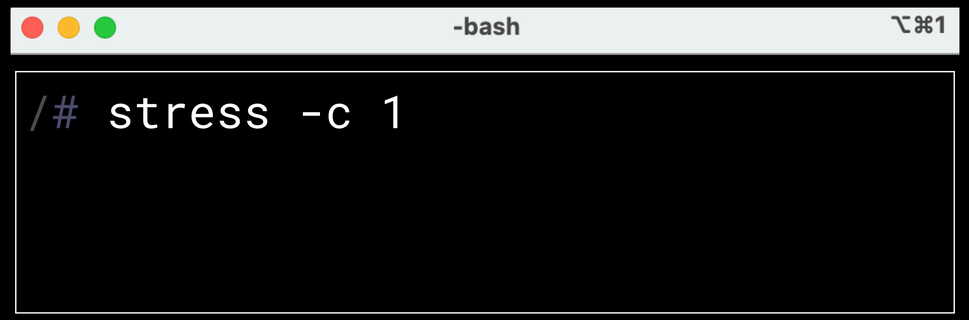
실습 준비

- 실습은 root 계정으로 진행합니다
- 실습을 위해 아래 툴을 설치해 주세요

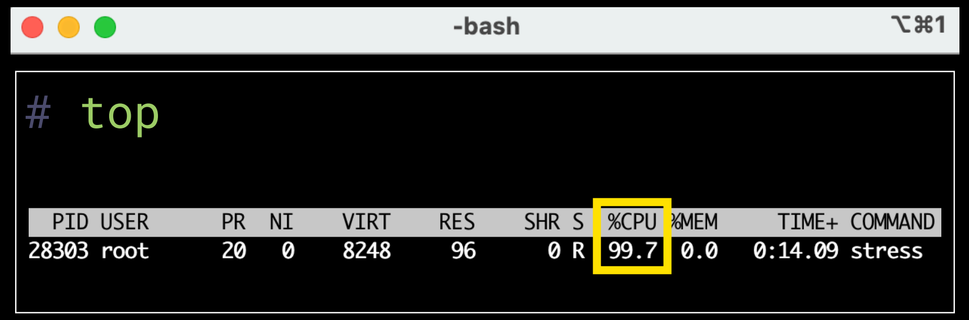
```
$ sudo -Es
# apt install -y cgroup-tools
# apt install -y stress
```

stress 실행 및 CPU 사용량 확인

호스트

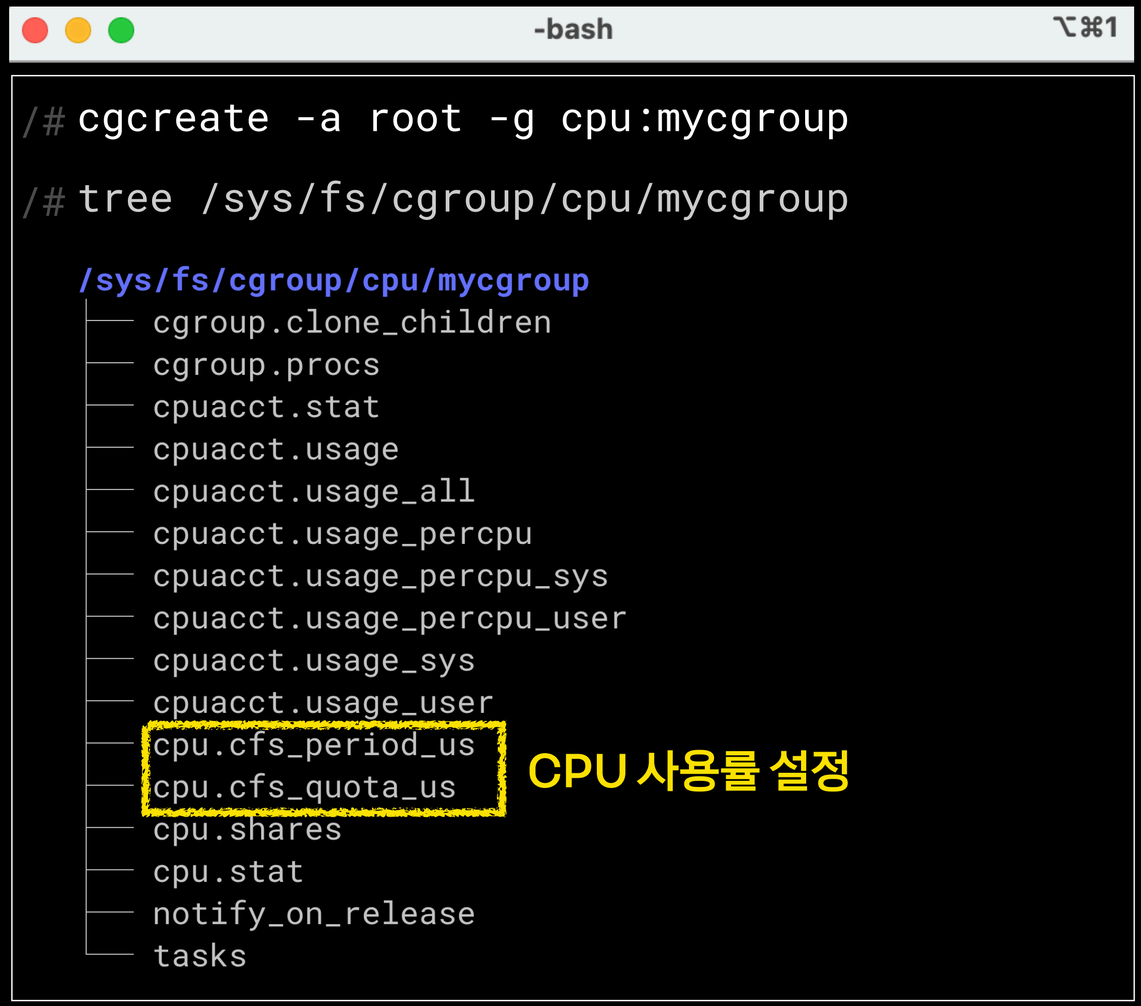


호스트



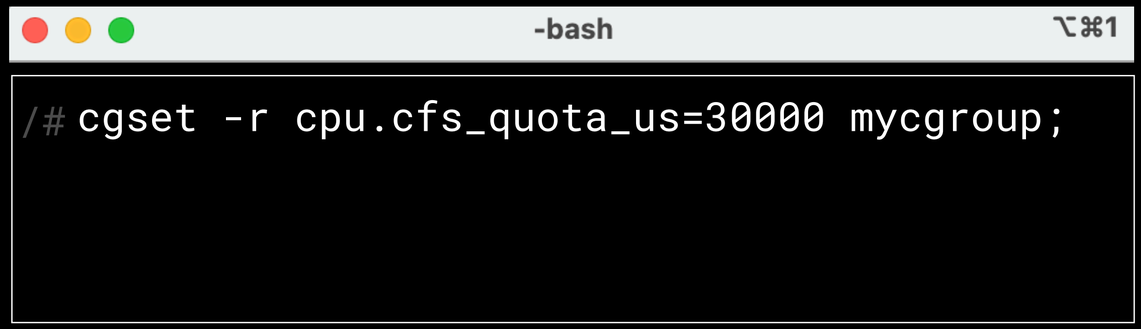
mycgroup 생성

호스트



mycgroup CPU사용률 설정

컨테이너



CPU 사용률 계산 (%CPU)

```
cpu.cfs_quota_us / cpu.cfs_period_us * 100
```

stress 실행 (mycgroup)

컨테이너

-bash ⌥⌘1

```
/# cgexec -g cpu:mycgroup stress -c 1
```

호스트

-bash ⌥⌘1

```
# top
```

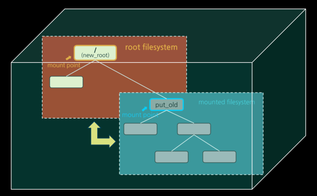
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
28630	root	20	0	8248	92	0	R	29.9	0.0	0:04.46	stress

Cgroups 실습 요약

- "Cgroup 파일시스템"으로 리소스 관리
- "제어그룹" (mycgroup) 생성
- "제어그룹" (mycgroup) 리소스 설정
- "제어그룹" (mycgroup) 프로세스 할당

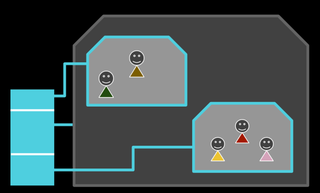
단원 마무리

chroot

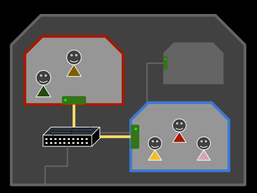


mount

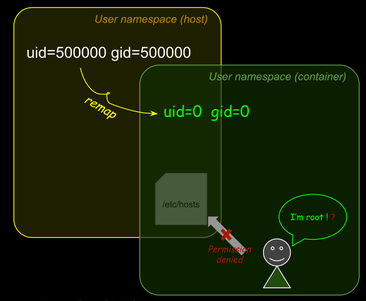
uts, ipc



pid, cgroups



net



user

1979 2002 2006 2008 2009 2012 2013 2015

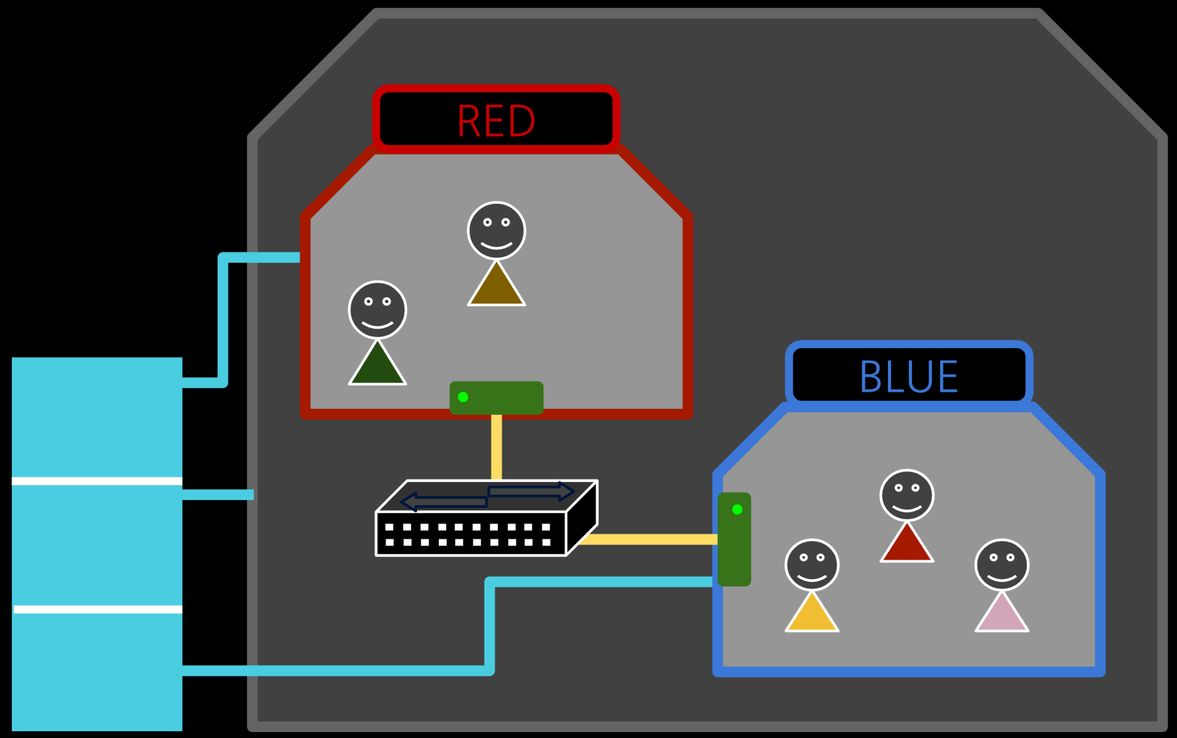
if(kakao)2022

도커 없이 컨테이너 만들기

김삼영 sam.0
카카오엔터프라이즈

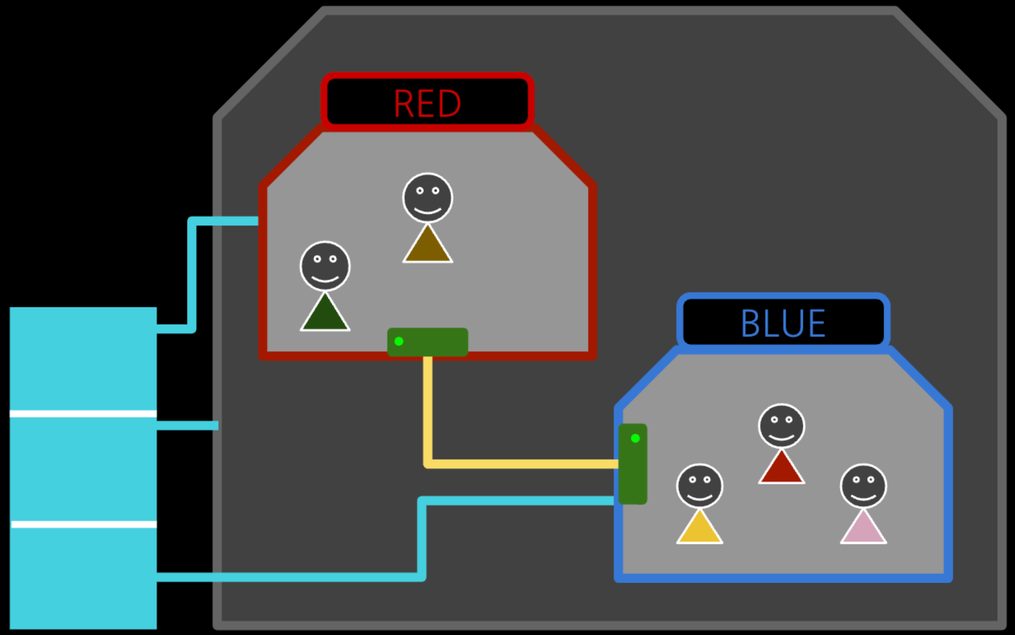
도커 없이 컨테이너 만들기

컨테이너 RED와 BLUE를 만들어 보아요

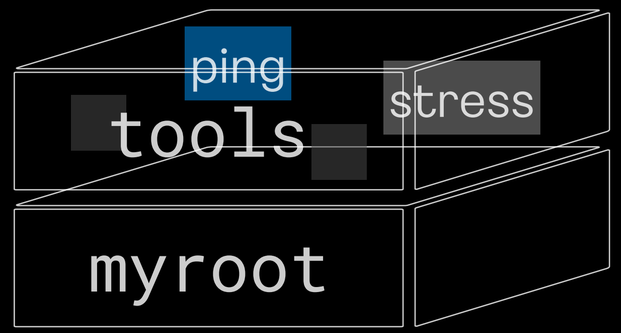


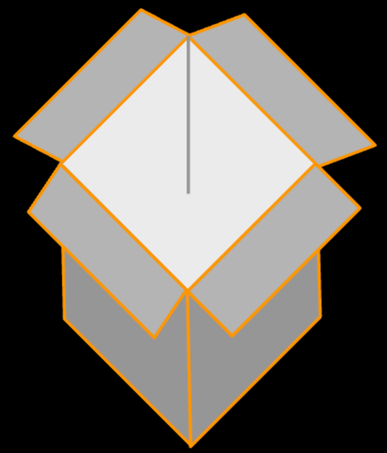
[실습6] 도커 없이 컨테이너 만들기

Goal



이미지 준비





이미지 준비 : myroot

컨테이너 파일시스템에서 준비한 myroot 를 사용합니다

```
# tree /tmp/myroot
```

이미지 준비 : tools

실습용 tools 준비

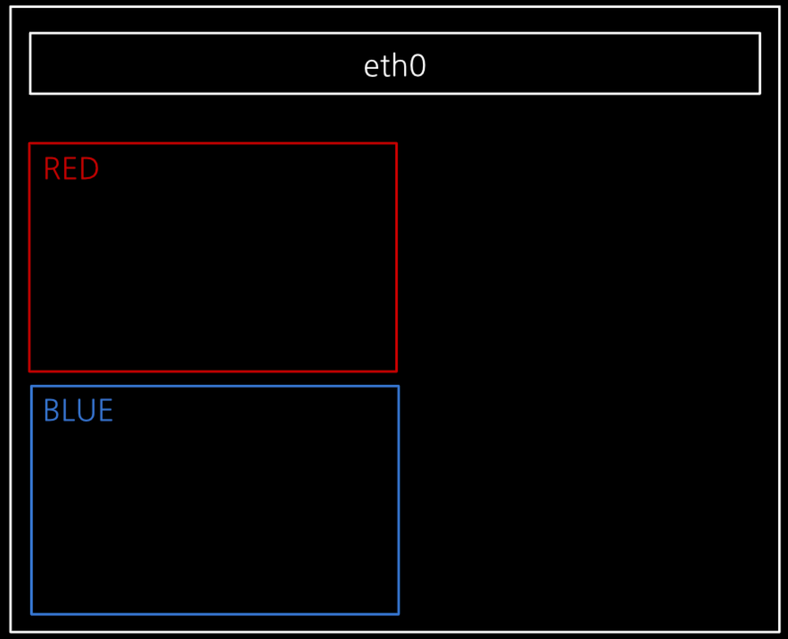
- ping : 컨테이너 통신 테스트
- stress : 컨테이너 부하 테스트
- hostname : 호스트네임 변경
- umount : put_old 제거

이미지 준비 : tools 복사

```
# wget https://raw.githubusercontent.com/sam0kim/container-internal/main/scripts/copy_tools.sh;
  bash copy_tools.sh;



#
```

컨테이너 네트워크

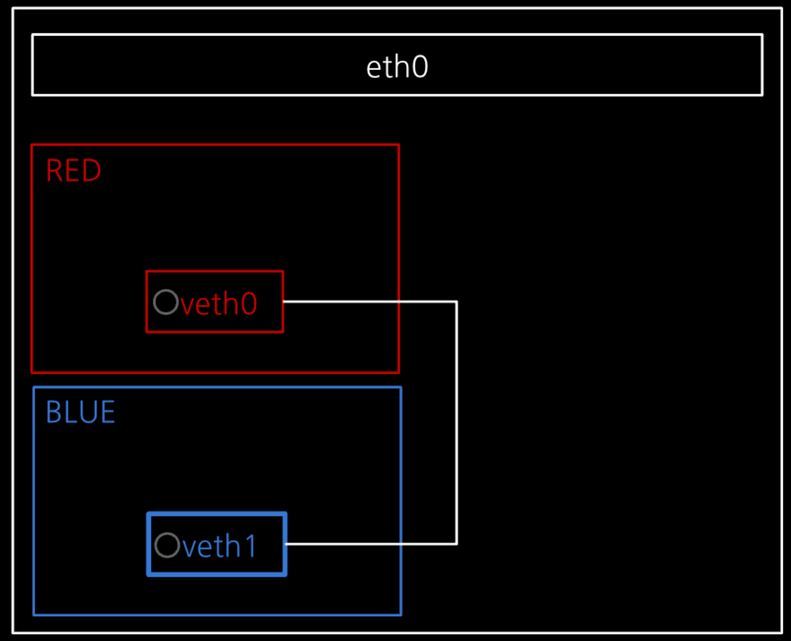
```
# ip netns add RED;
# ip netns add BLUE;
```



컨테이너 네트워크

```
# ip link add veth0 netns RED type veth peer name veth1 netns BLUE
```

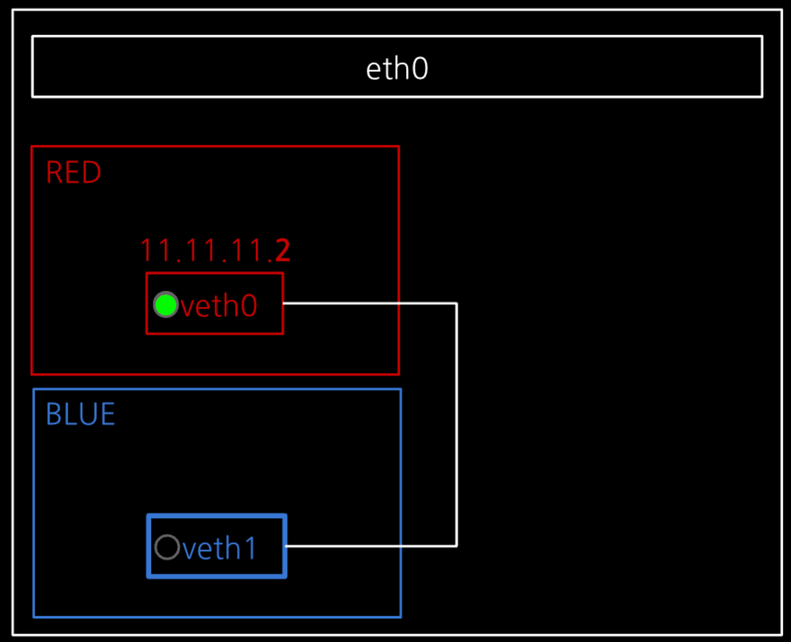
1)



1) https://man7.org/linux/man-pages/man8/ip-link.8.html

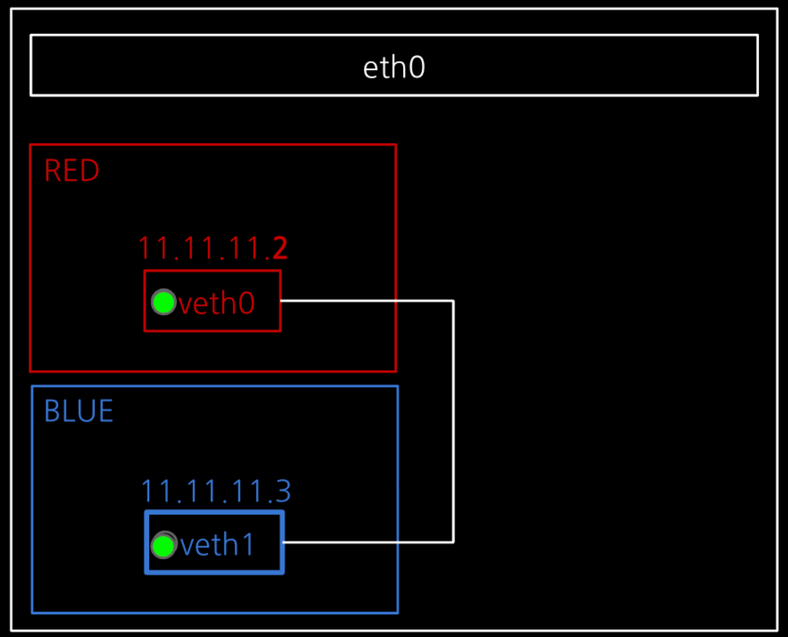
컨테이너 네트워크

```
# ip netns exec RED ip addr add dev veth0 11.11.11.2/24;
# ip netns exec RED ip link set veth0 up;
```

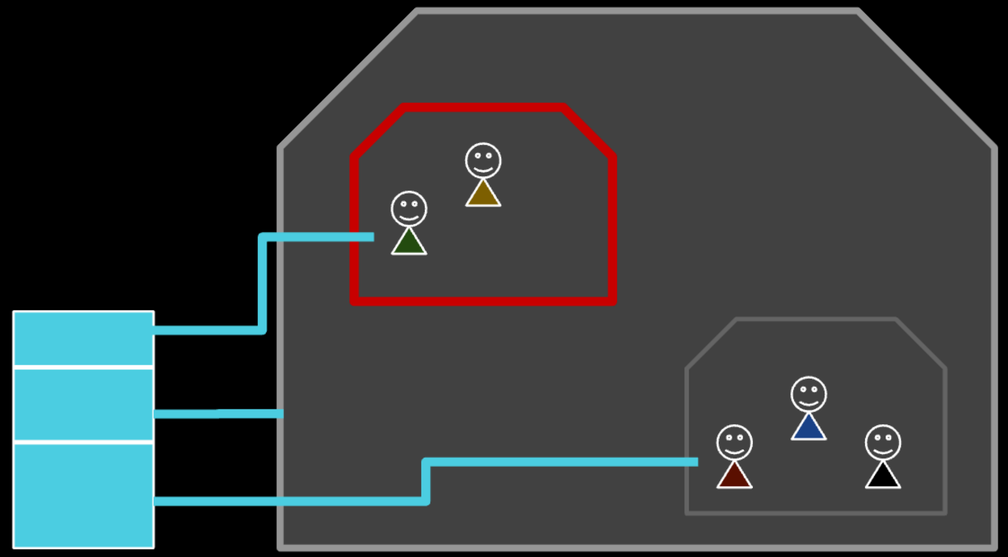


컨테이너 네트워크

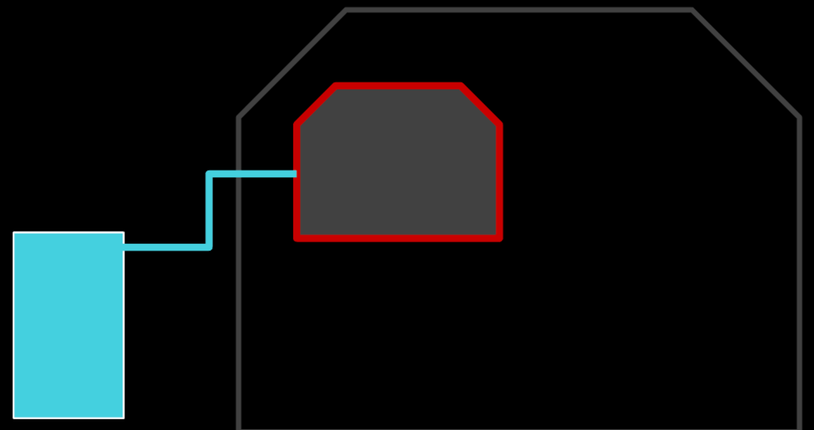
```
# ip netns exec BLUE ip addr add dev veth1 11.11.11.3/24;
# ip netns exec BLUE ip link set veth1 up;
```



RED 컨테이너를 만들어 보아요



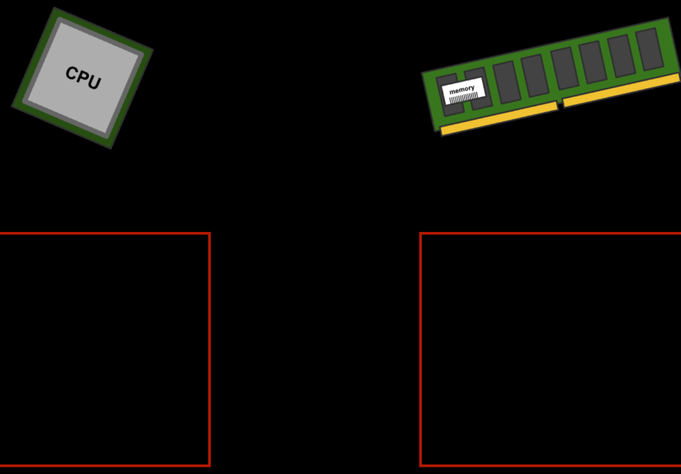
RED 격리와 자원



namespaces : mount, uts, net, ipc, pid
cgroups (red) : cpu 40%, memory 200M

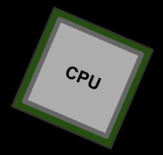
RED Cgroups 생성

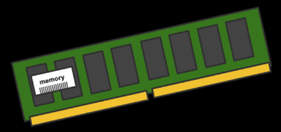
```
# mkdir /sys/fs/cgroup/cpu/red;
# mkdir /sys/fs/cgroup/memory/red;
```



RED Cgroups 설정

```
#  echo 40000 > /sys/fs/cgroup/cpu/red/cpu.cfs_quota_us;
#  echo 209715200 > /sys/fs/cgroup/memory/red/memory.limit_in_bytes;
#  echo 0 > /sys/fs/cgroup/memory/red/memory.swappiness;
```





40%

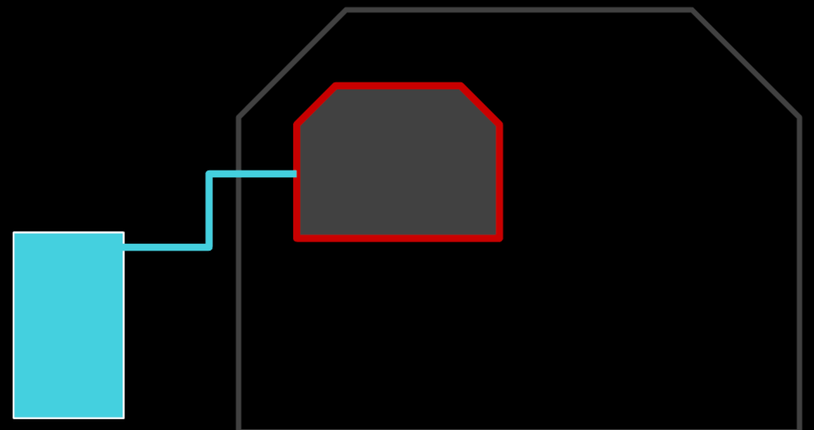
200MB
swap off

RED 격리

```
# unshare -m -u -i -fp nsenter --net=/var/run/netns/RED /bin/sh;
```

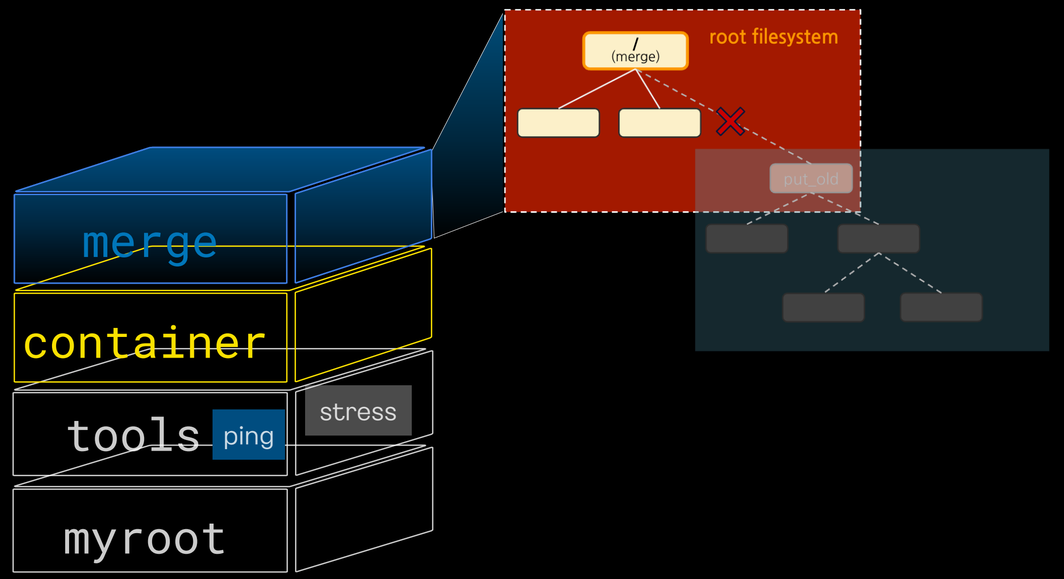
RED Cgroups 할당

```
# echo "1" > /sys/fs/cgroup/cpu/red/cgroup.procs;
# echo "1" > /sys/fs/cgroup/memory/red/cgroup.procs;
```



namespaces : mount, uts, net, ipc, pid
cgroups (red) : cpu 40%, memory 200M

RED 파일시스템


root filesystem
/
(merge)
put_old
merge
container
tools
ping
stress
myroot

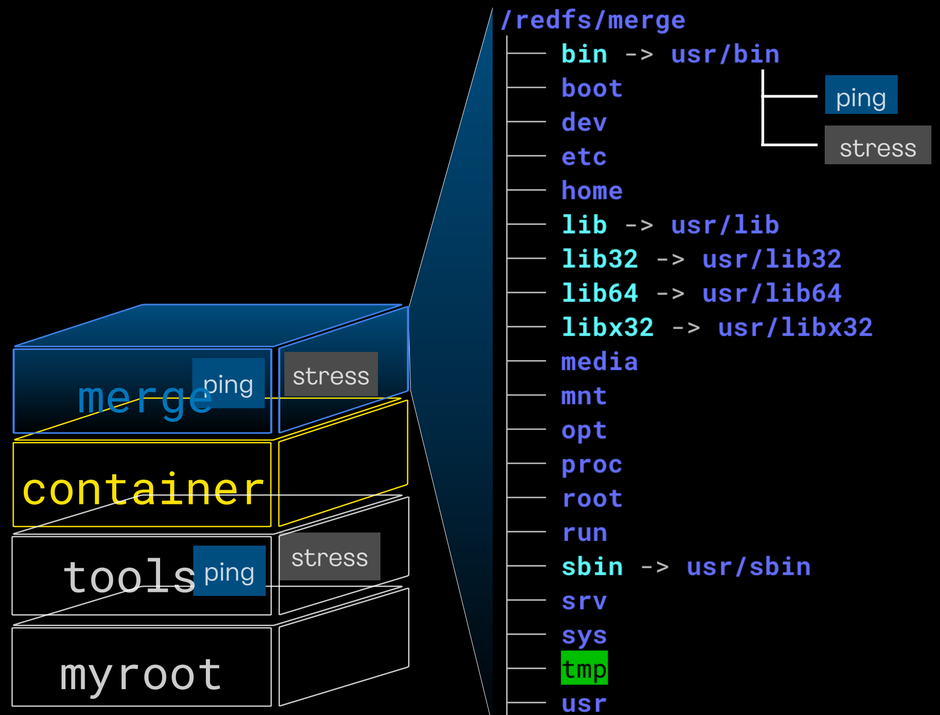
오버레이 마운트

```
# mkdir /redfs;
# mkdir /redfs/container;
# mkdir /redfs/work;
# mkdir /redfs/merge;
```

```
# tree /redfs
/redfs
├── container
├── merge
└── work
```

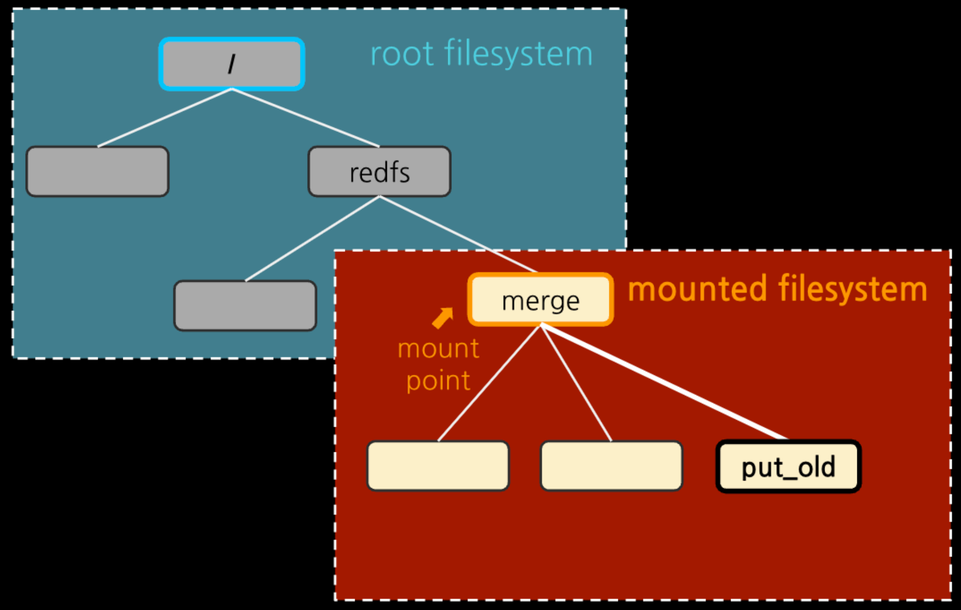
오버레이 마운트

```
# mount -t overlay overlay -o \
  lowerdir=/tmp/tools:/tmp/myroot,\
  upperdir=/redfs/container,workdir=/redfs/work \
  /redfs/merge
```



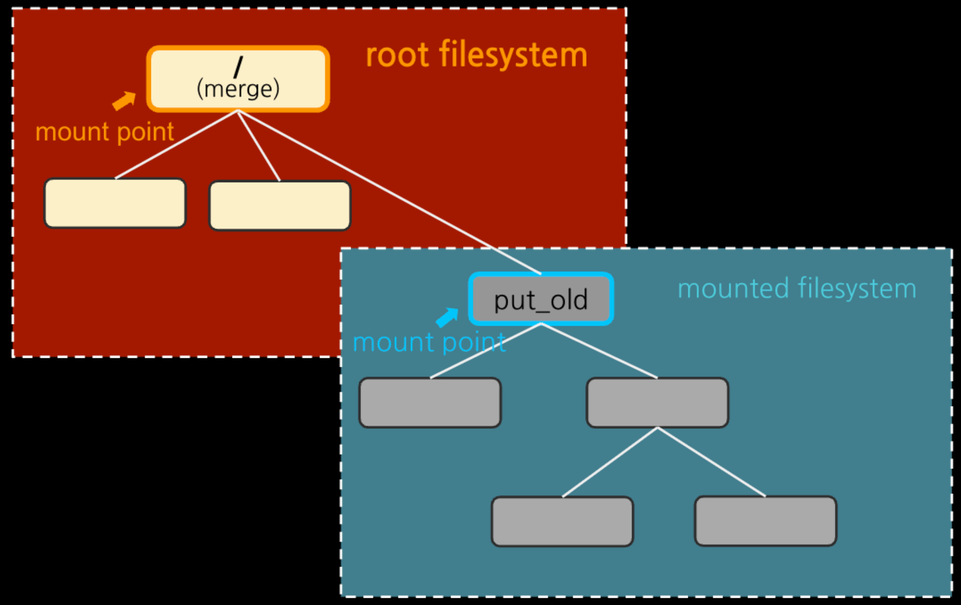
pivot_root

```
# mkdir -p /redfs/merge/put_old
```



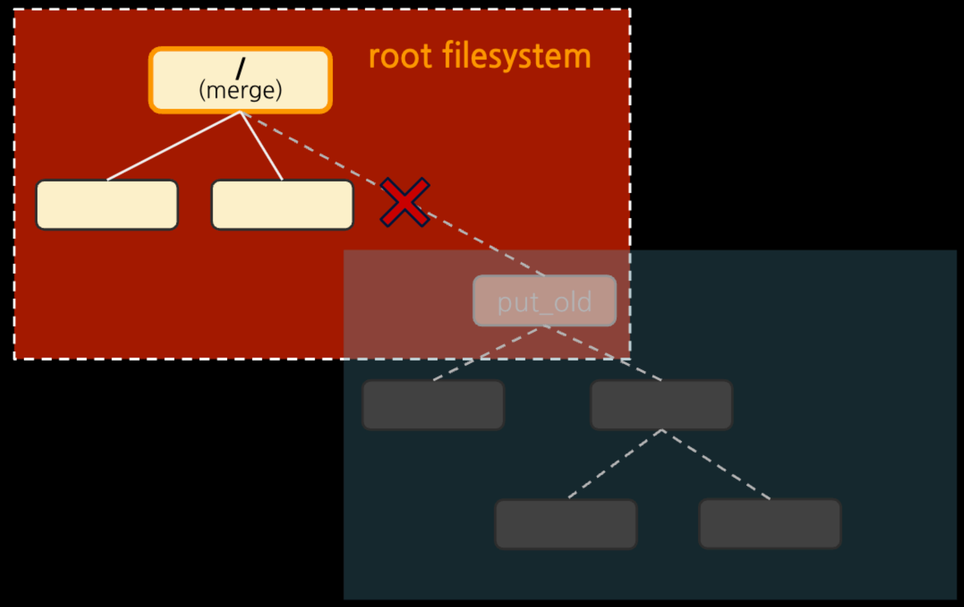
pivot_root

```
# cd /redfs/merge;
# pivot_root . put_old;
# cd /;
```



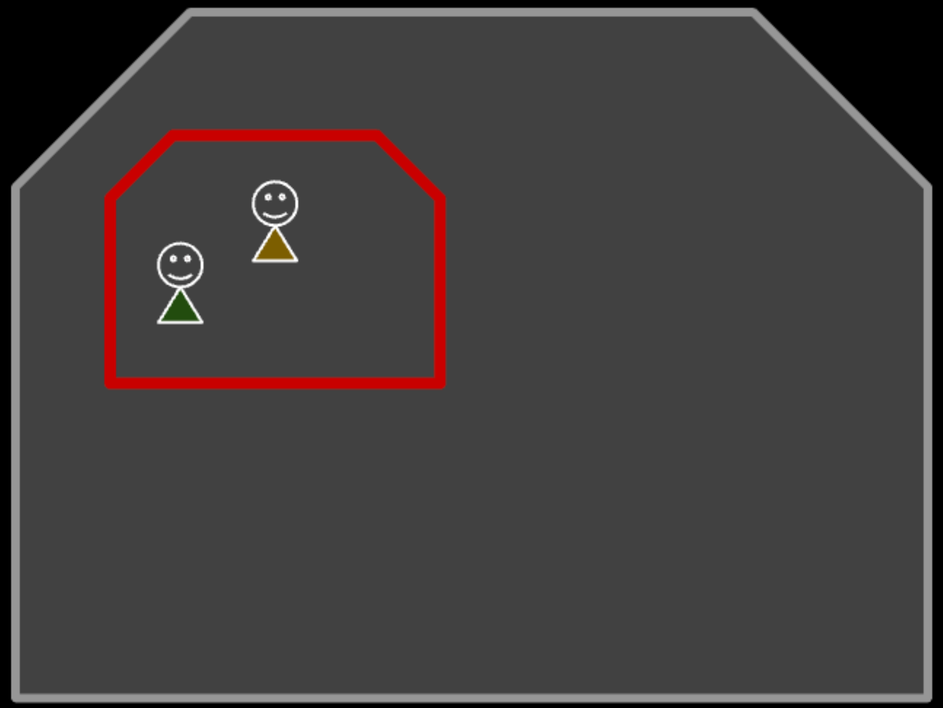
put_old 제거

```
# mount -t proc proc /proc;
# umount -l put_old;
# rm -rf put_old;
```



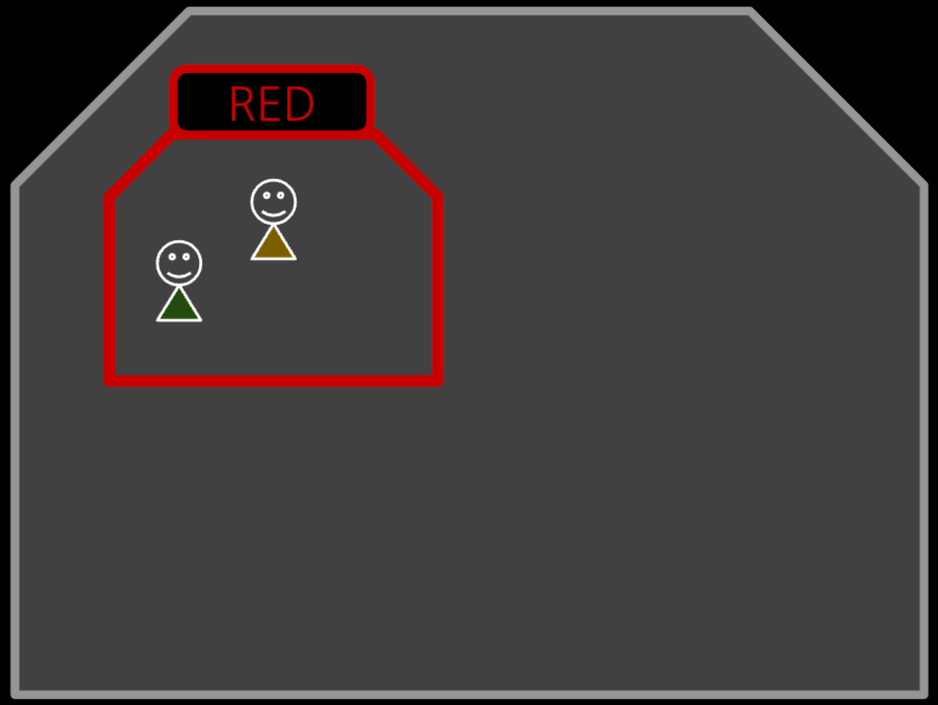
RED 프로세스

```
# ps -ef
```

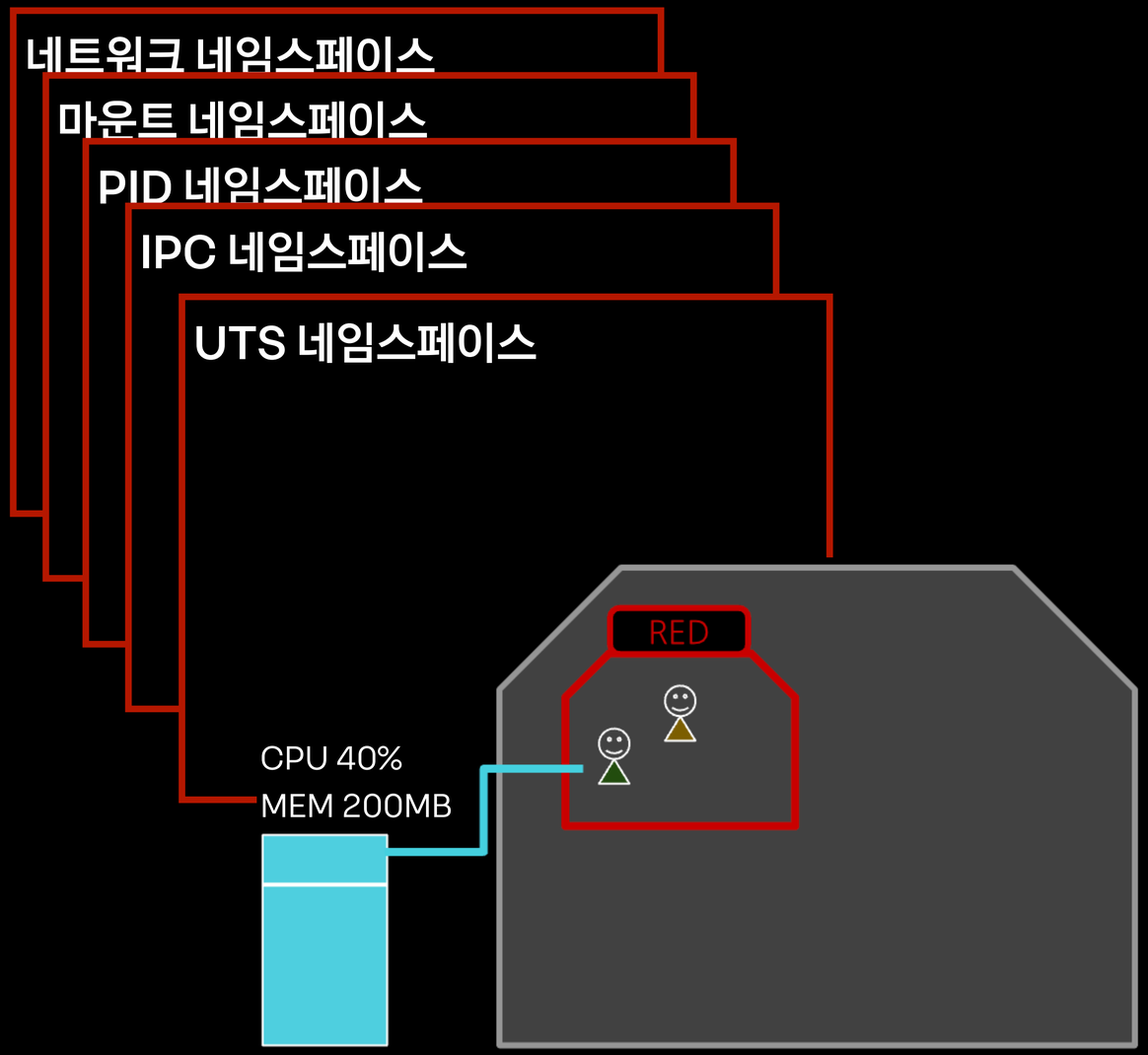


RED 호스트네임

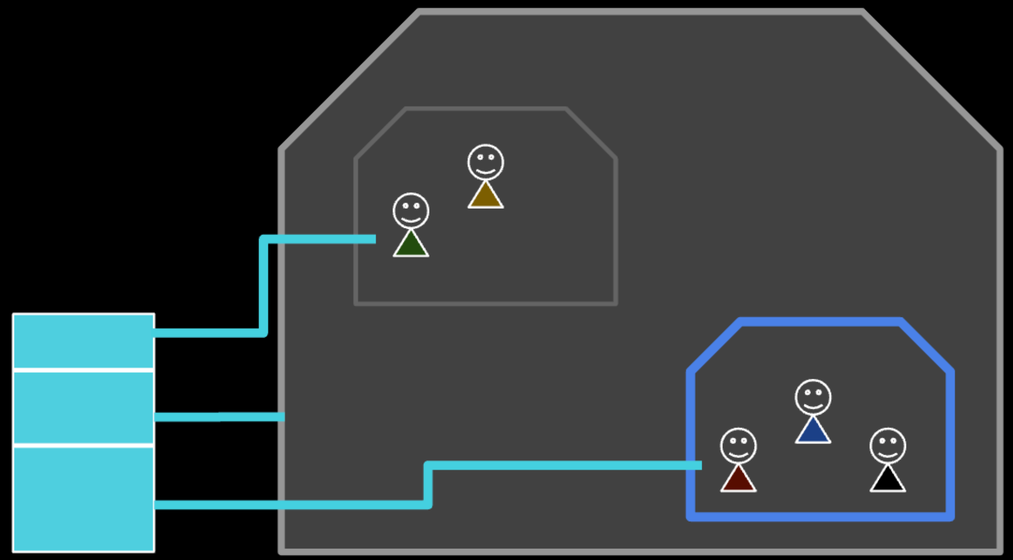
```
# hostname RED
```



RED 컨테이너 완성

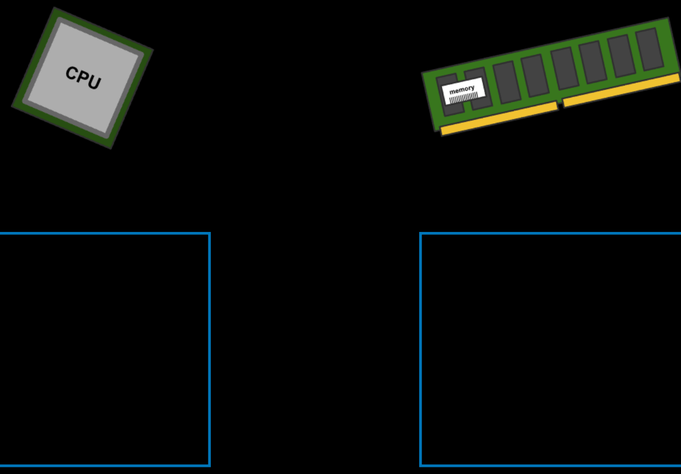

네트워크 네임스페이스
마운트 네임스페이스
PID 네임스페이스
IPC 네임스페이스
UTS 네임스페이스
RED
CPU 40%
MEM 200MB

BLUE 컨테이너를 만들어 보아요



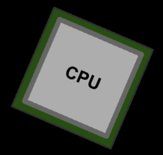
BLUE Cgroups 생성

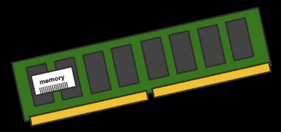
```
# mkdir /sys/fs/cgroup/cpu/blue;
# mkdir /sys/fs/cgroup/memory/blue;
```



BLUE Cgroups 설정

```
# echo 40000 > /sys/fs/cgroup/cpu/blue/cpu.cfs_quota_us;
# echo 209715200 > /sys/fs/cgroup/memory/blue/memory.limit_in_bytes;
# echo 0 > /sys/fs/cgroup/memory/blue/memory.swappiness;
```





40%

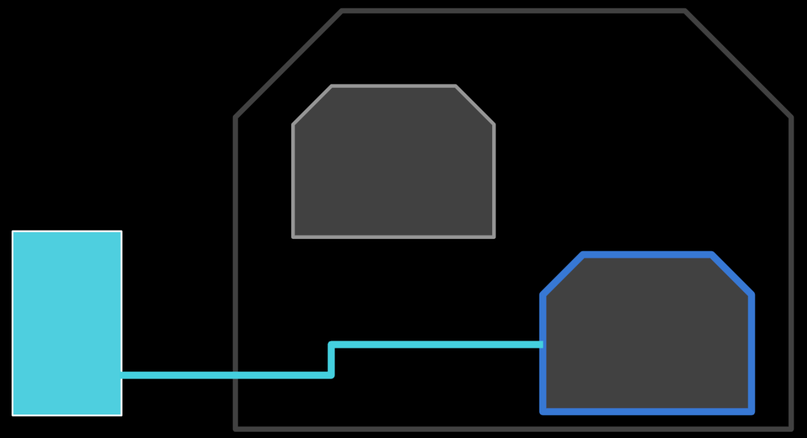
200MB
swap off

BLUE 격리

```
# unshare -m -u -i -fp nsenter --net=/var/run/netns/BLUE /bin/sh
```

BLUE Cgroups 할당

```
# echo "1" > /sys/fs/cgroup/cpu/blue/cgroup.procs;
# echo "1" > /sys/fs/cgroup/memory/blue/cgroup.procs;
```



namespaces : mount, uts, net, ipc, pid
cgroups (blue) : cpu 40%, memory 200M

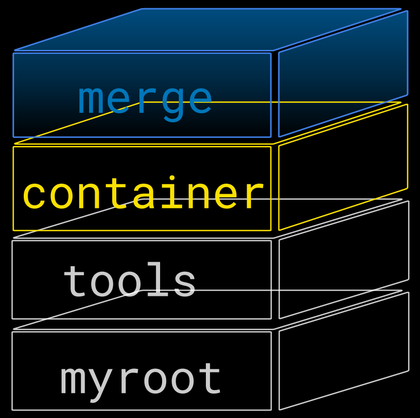
오버레이 마운트

```
# mkdir /bluefs;
# mkdir /bluefs/container;
# mkdir /bluefs/work;
# mkdir /bluefs/merge;
```

```
# tree /bluefs
/bluefs
├── container
├── merge
└── work

3 directories, 0 files
```

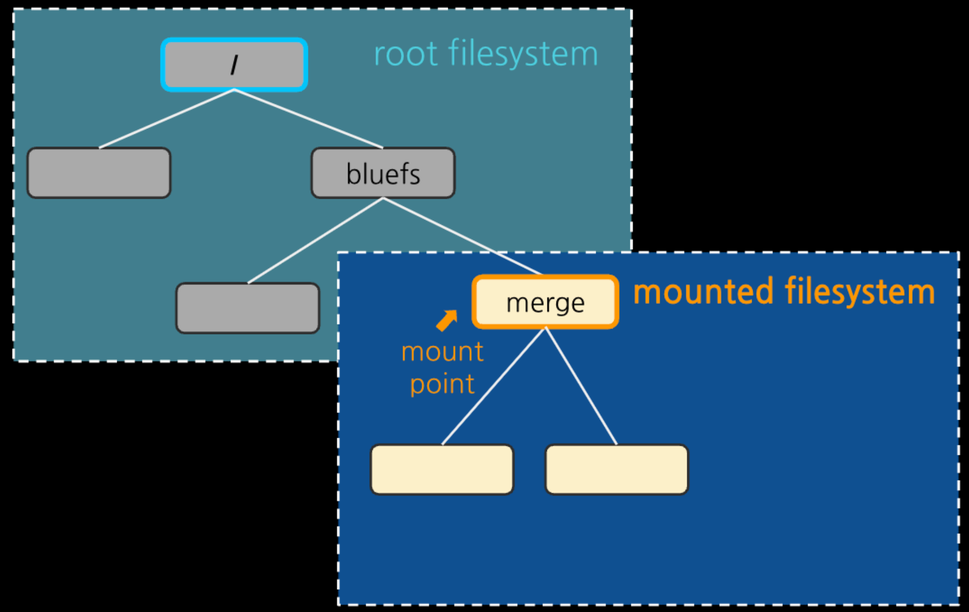
오버레이 마운트

```
# mount -t overlay overlay -o \
  lowerdir=/tmp/tools:/tmp/myroot,\
  upperdir=/bluefs/container,workdir=/bluefs/work \
  /bluefs/merge
```



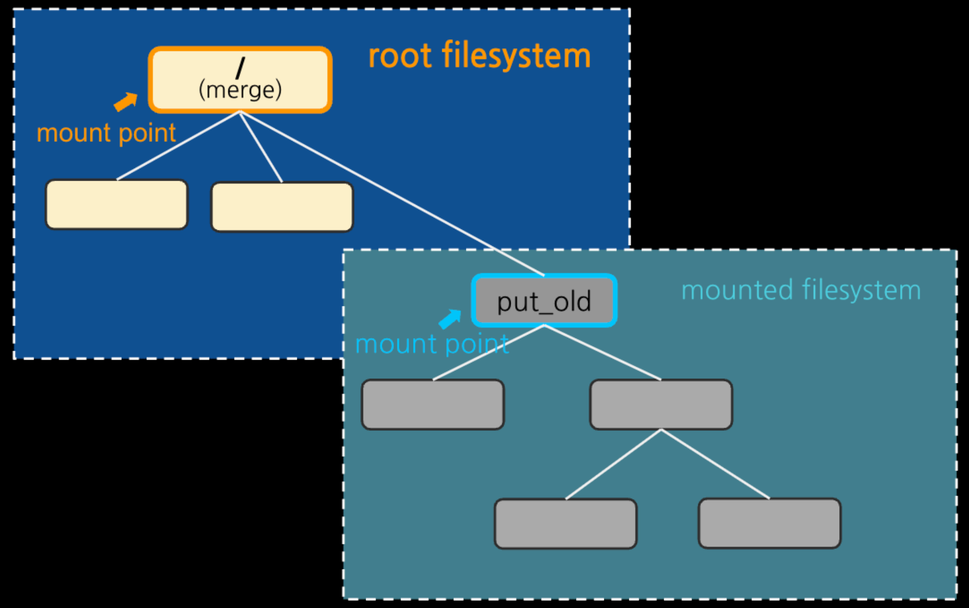
pivot_root : put_old

```
# mkdir -p /bluefs/merge/put_old
```



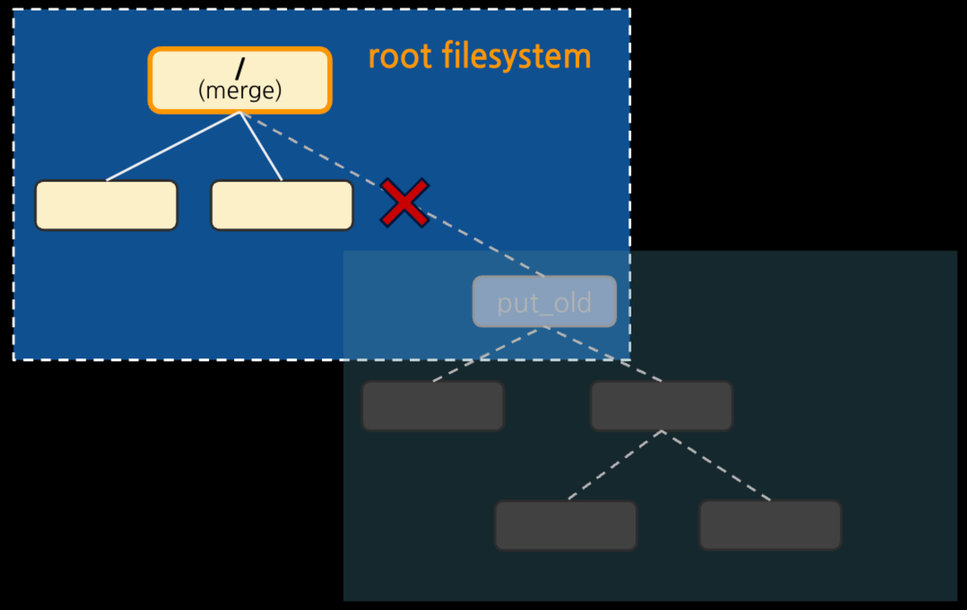
pivot_root

```
# cd /bluefs/merge;
# pivot_root . put_old;
# cd /;
```



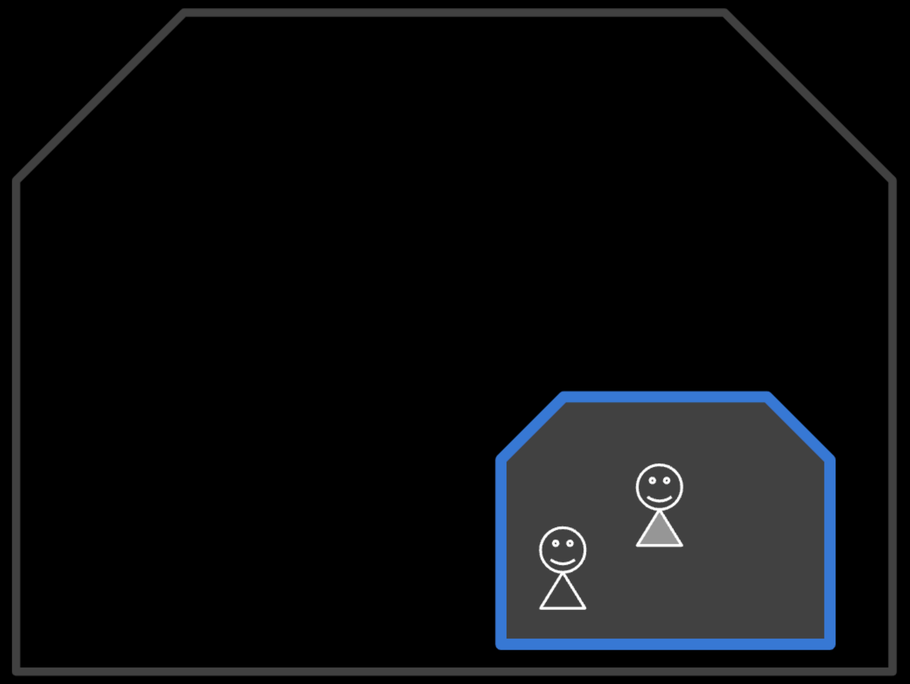
put_old 제거

```
# mount -t proc proc /proc;
# umount -l put_old;
# rm -rf put_old;
```



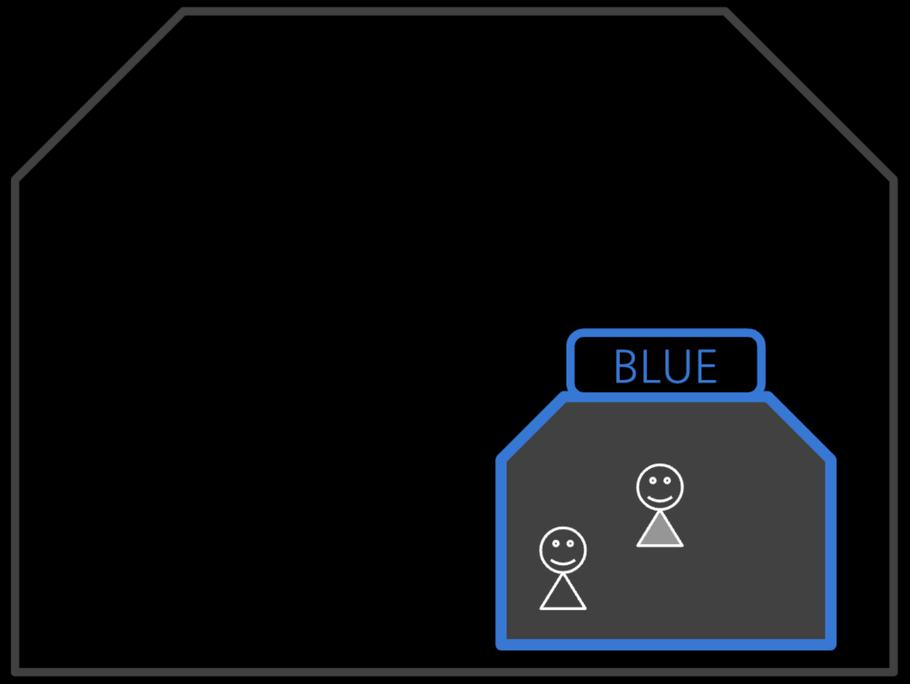
BLUE 프로세스 확인

```
# ps -ef
```

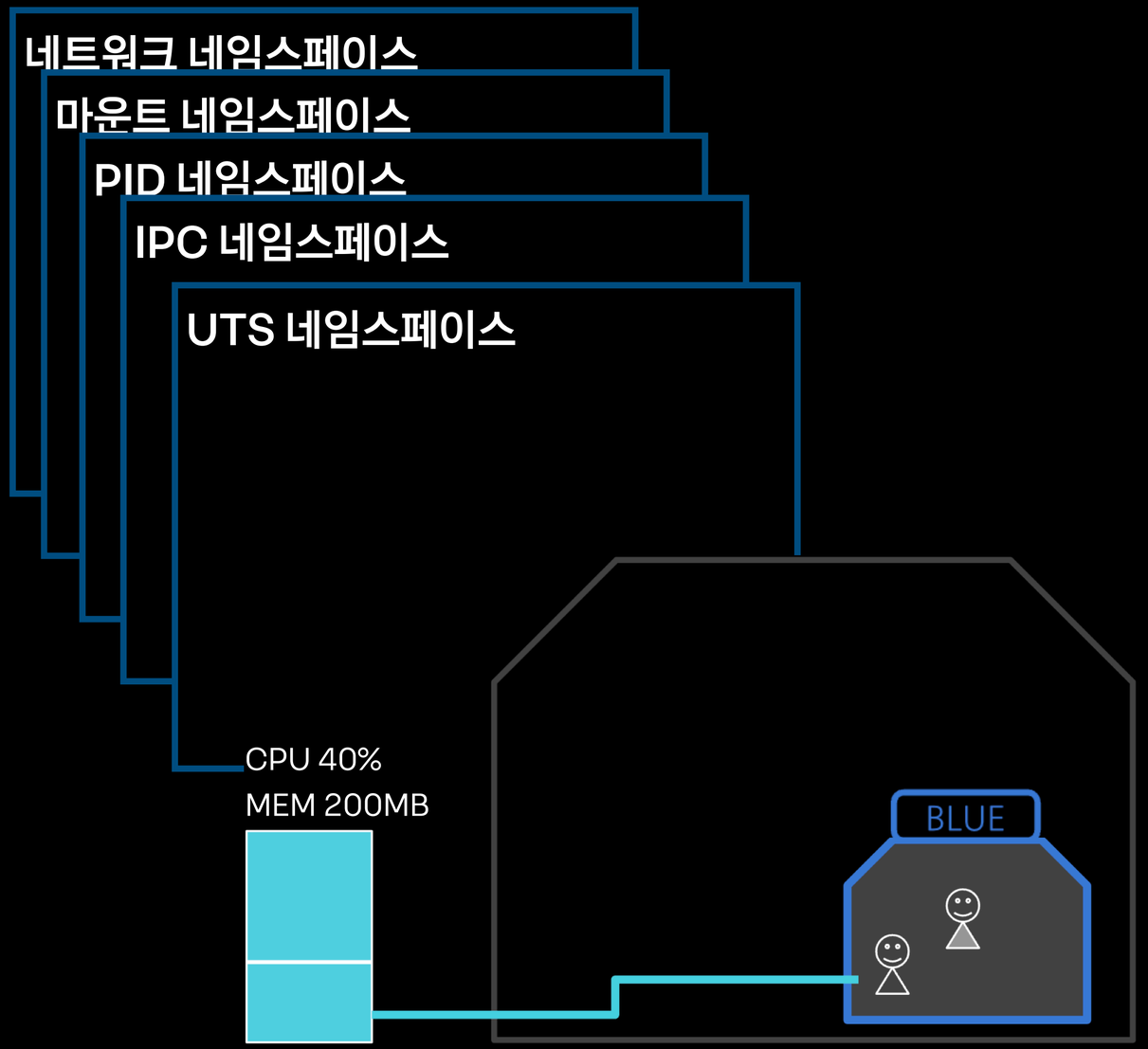


BLUE 호스트네임 변경

```
# hostname BLUE
```

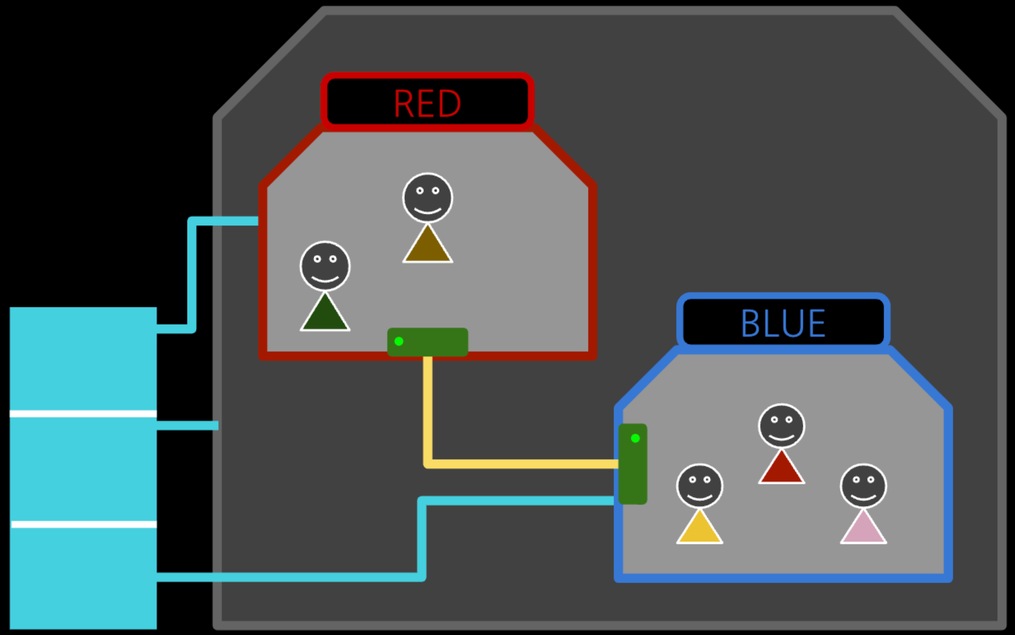


BLUE 컨테이너 완성


네트워크 네임스페이스
마운트 네임스페이스
PID 네임스페이스
IPC 네임스페이스
UTS 네임스페이스
CPU 40%
MEM 200MB
BLUE

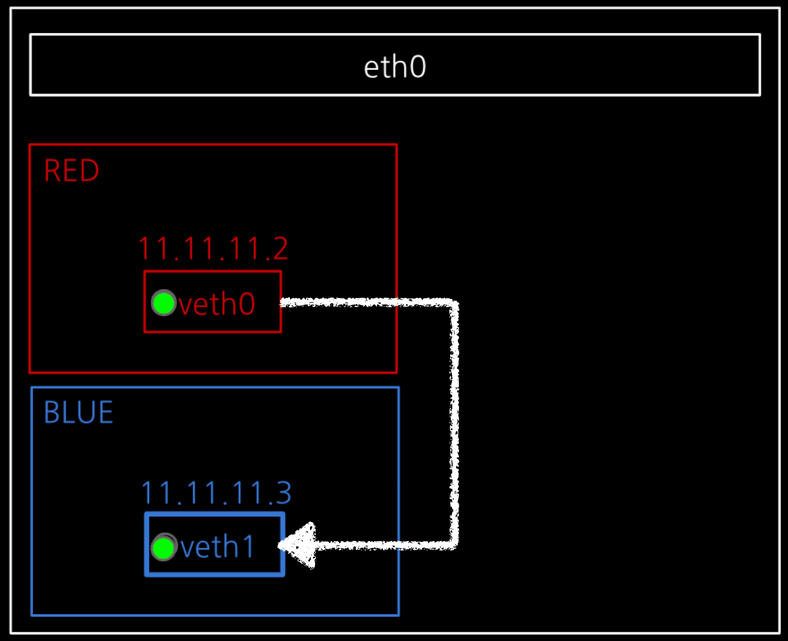
컨테이너 테스트

- RED/BLUE 통신 테스트
- stress 테스트



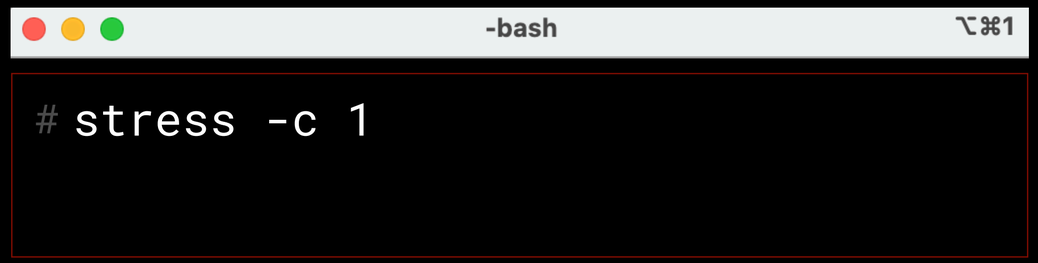
RED → BLUE

```
# ping 11.11.11.3
```

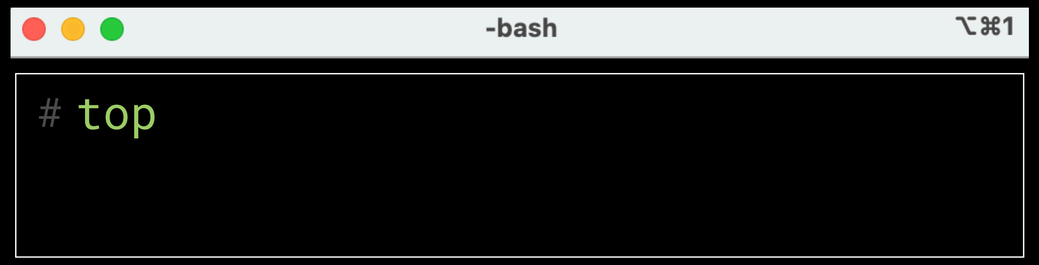


RED CPU 리소스 확인

RED



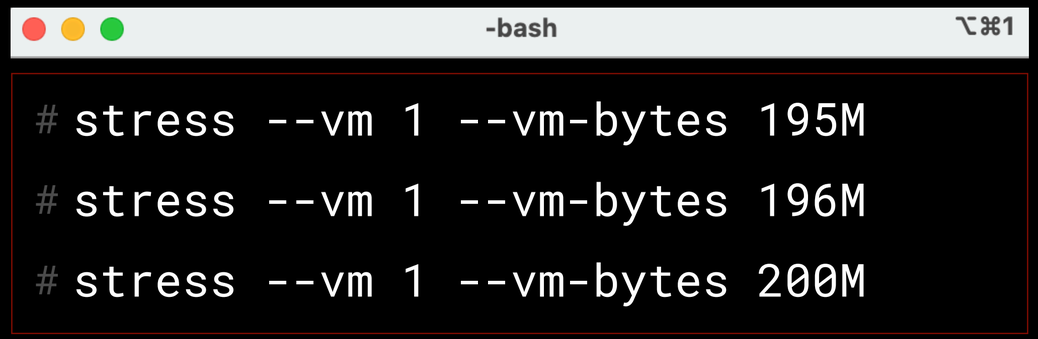
호스트



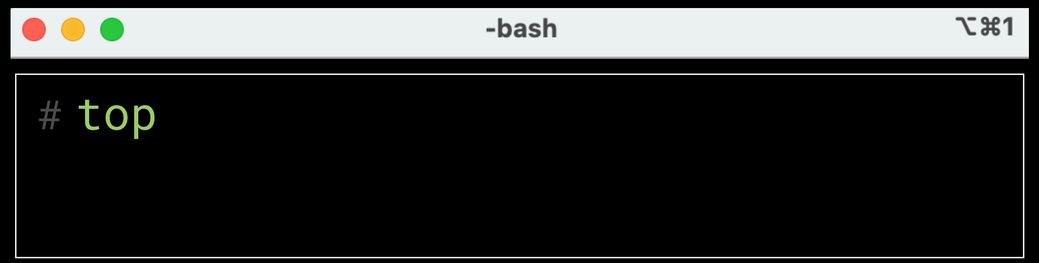
RED : cpu 40%

RED 메모리 리소스 확인

RED



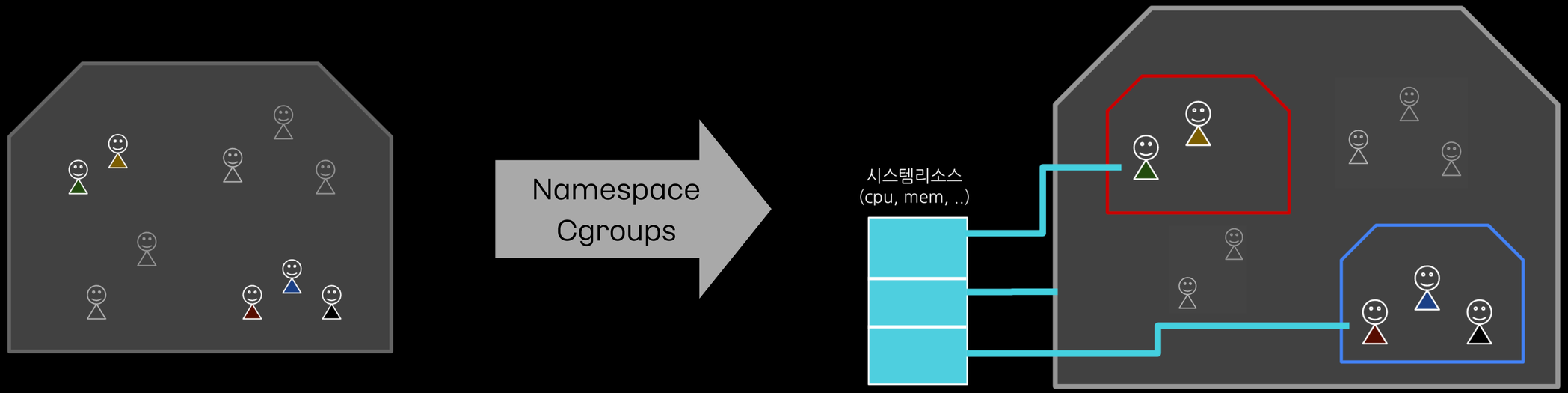
호스트



RED : memory 200MB, swap off

컨테이너 완성

올인원 패키징 + 격리 + 자원보장



마치며

지금까지 ...

- chroot ... and 문제들
- 컨테이너 전용 루트파일시스템
- 컨테이너 격리와 자원
- 직접 만들어 본 컨테이너

더 봐야 할 것들

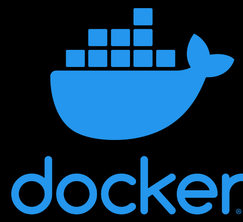
- 컨테이너 네트워크 — 가상네트워크 통신
- 컨테이너 표준화 — 인터페이스
- 컨테이너 오케스트레이션 — 쿠버네티스

마치며 ... 컨테이너 ≠ 도커


cri-o

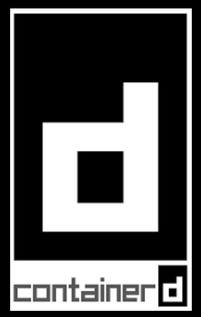

kata
containers


Firecracker


docker


LXC


podman


containerd


RUNC


gVisor

Thank you