실습과 함께 완성해보는

도커 없이 컨테이너 만들기

2편

Sam.0

시작하기에 앞서 ...

본 컨텐츠는 1편을 보았다고 가정하고 준비되었습니다.

원활한 이해 및 실습을 위하여 1편을 먼저 보시기를 추천드립니다

1편 링크 클릭

실습을 위한 사전 준비 사항

실습은 . . .

- 맥 환경에서 VirtualBox + Vagrant 기반으로 준비되었습니다
  - 맥 이외의 OS 환경도 괜찮습니다만
  - 원활한 실습을 위해서
  - “VirtualBox or VMware + Vagrant”는 권장드립니다.
- 실습환경 구성을 위한 Vagrantfile을 제공합니다.

실습을 위한 사전 준비 사항

Vagrantfile

- 오른쪽의 텍스트를 복사하신 후
- 로컬에 Vagrantfile로 저장하여 사용하면 됩니다.
- vagrant 사용법은 공식문서를 참고해 주세요 https://www.vagrantup.com/docs/index

```
BOX_IMAGE = "bento/ubuntu-18.04"
HOST_NAME = "ubuntu1804"

$pre_install = <<-SCRIPT
  echo ">>>> pre-install <<<<<<"
  sudo apt-get update &&
  sudo apt-get -y install gcc &&
  sudo apt-get -y install make &&
  sudo apt-get -y install pkg-config &&
  sudo apt-get -y install libseccomp-dev &&
  sudo apt-get -y install tree &&
  sudo apt-get -y install jq &&
  sudo apt-get -y install bridge-utils

  echo ">>>> install go <<<<<<"
  curl -O https://storage.googleapis.com/golang/go1.15.7.linux-amd64.tar.gz > /dev/null 2>&1 &&
  tar xf go1.15.7.linux-amd64.tar.gz &&
  sudo mv go /usr/local/ &&
  echo 'PATH=$PATH:/usr/local/go/bin' | tee /home/vagrant/.bash_profile

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

Vagrant.configure("2") do |config|

 config.vm.define HOST_NAME do |subconfig|
   subconfig.vm.box = BOX_IMAGE
   subconfig.vm.hostname = HOST_NAME
   subconfig.vm.network :private_network, ip: "192.168.104.2"
   subconfig.vm.provider "virtualbox" do |v|
     v.memory = 1536
     v.cpus = 2
   end
   subconfig.vm.provision "shell", inline: $pre_install
 end

end
```

실습을 위한 사전 준비 사항

Vagrantfile 제공 환경

vagrant + virtual vm
ubuntu 18.04
docker 20.10.5 ** 도커 이미지 다운로드 및 컨테이너 비교를 위한 용도로 사용합니다.*

기타 설치된 툴
~ tree, jq, brctl, … 등 실습을 위한 툴

실습 계정 (root)

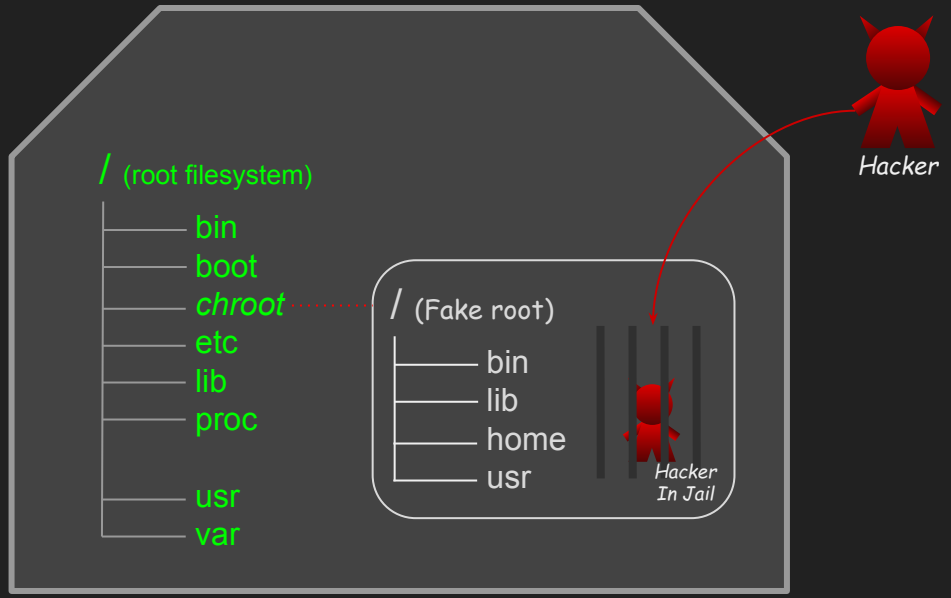
```
# sudo -Es
```

실습 폴더

```
# cd /tmp
```

chroot

change root directory ‘ / ’

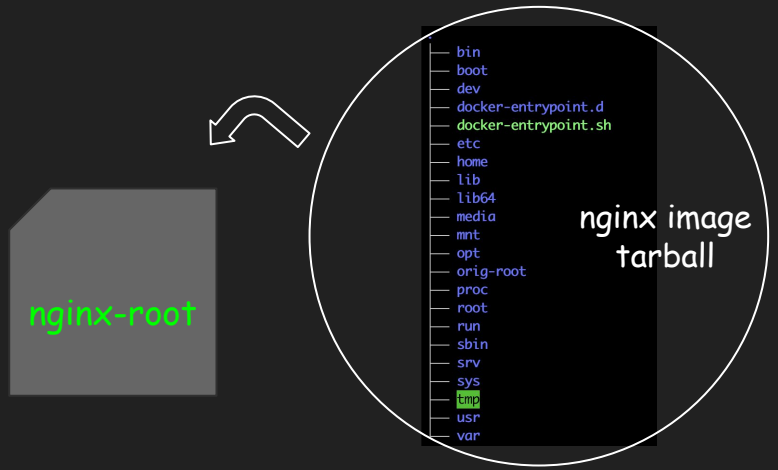


특정 디렉토리 경로를 root로 지정할 수 있으면

해당 경로에 프로세스를 가둘 수 있다는 점에 착안 ~

chroot

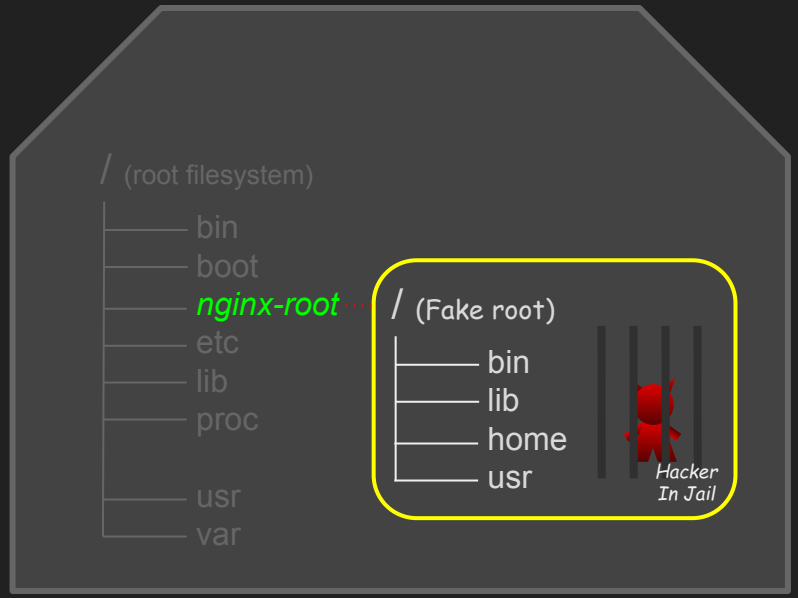
필요한 파일들을 특정 디렉토리에 모아서



컨테이너의 이미지는 일종의 "필요한 파일/라이브러리" tarball 입니다.

chroot

마치 호스트의 root 파일시스템 환경인 것 처럼 가둬놓고 사용할 수 있습니다.




```
# chroot  nginx-root  /bin/sh
# nginx -g “daemon off;”
```

```
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web ser
working. Further configuration is required

<p>For online documentation and support pl
<a href="http://nginx.org/">nginx.org</a>.
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.
```

chroot 문제점

But … “root 디렉토리”를 속여서 경로만 제한했을 뿐

isolation 되지 않음 : 호스트의 filesystems, process tree, network, ipc, … 에 접근 가능

root 권한 사용 : root 권한과 그에 따른 보안 문제 초래 가능

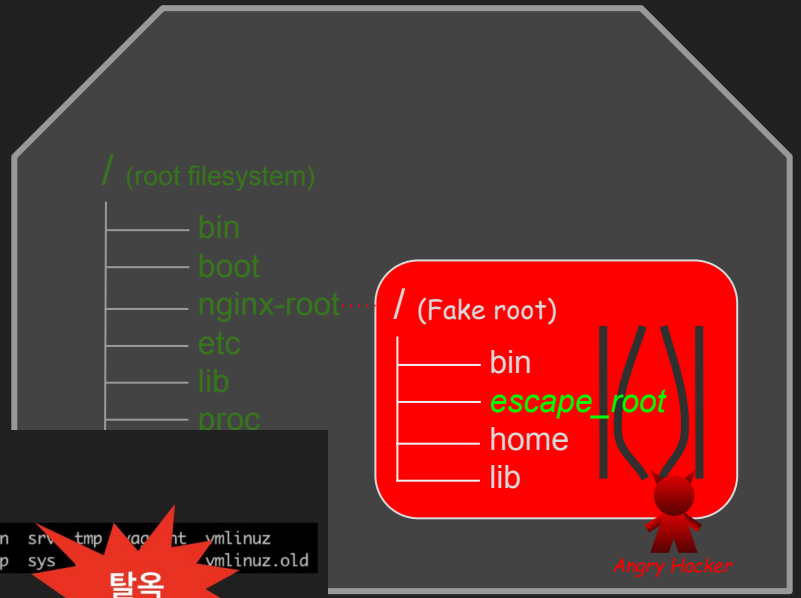
resource 무제한 : cpu, memory, i/o, network, … 호스트의 자원을 제한 없이 사용 가능

chroot 문제점

무엇보다 ...

탈옥이 가능합니다

```
##  탈옥코드
# vi escape_chroot.c

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

int main(void)
{
    mkdir(".out", 0755);
    chroot(".out");
    chdir("../../../../../");
    chroot(".");

    return execl("/bin/sh", "-i", NULL);
}
```



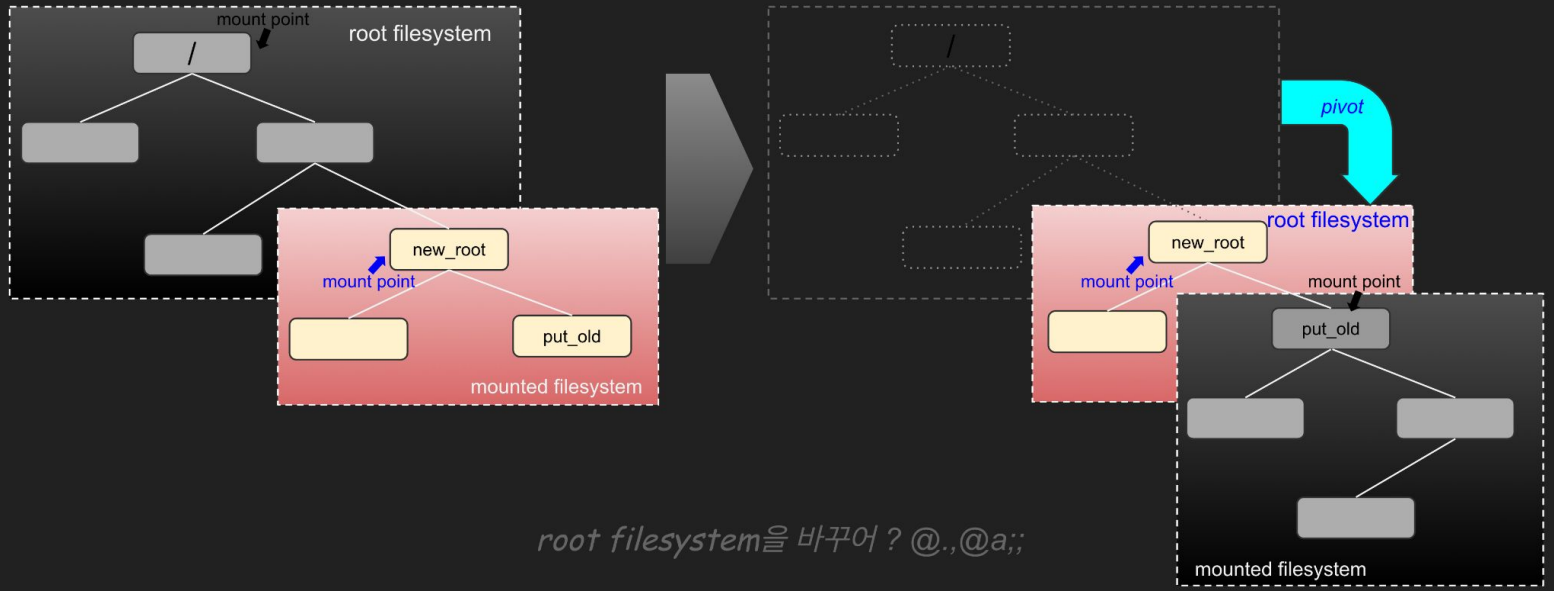

```
bash-4.4# ./escape_chroot

# ls  /
bin   dev  home        initrd.img.old  lib64       media  opt   root  sbin  srv  tmp  vmlinuz
boot  etc  initrd.img  lib             lost+found  mnt    proc  run   snap  sys       vmlinuz.old
```

탈옥

pivot_root

아예 "root filesystem을 바꿔서" 탈옥을 원천 차단

changes the "root filesystem"
* chroot? changes the "root path"



root filesystem을 바꾸어 ? @.,@a;;

- pivot_root [new-root] [old-root]
  - 새로운 root filesystem (**new-root)**
  - 기존 root filesystem → **old-root** (mount)

탈옥을 해결할 주인공입니다 :-)

사용법은 심플합니다 ~ new-root와 old-root 경로를 주면 됩니다

pivot_root는
- 새로운 new-root를 '/'에 붙이고,
- 기존 root filesystem을 old-root에 착 ~ 붙여버립니다.

mount : https://man7.org/linux/man-pages/man8/mount.8.html

- mount -t [filesystem type] [device_name] [directory - mount point]
- root filesystem tree에 다른 파일시스템을 붙이는 명령

-t : filesystem type ex) -t tmpfs (temporary filesystem : 임시로 메모리에 생성됨)
-o : 옵션 ex) -o size=1m (용량 지정 등 …)

참고)
* /proc/filesystems 에서 지원하는 filesystem type 조회 가능

unshare : https://man7.org/linux/man-pages/man1/unshare.1.html

- unshare [options] [program [arguments]]
- creates new namespaces and then
- executes the specified program (default : ${SHELL})

unshare ?
"새로운 네임스페이스를 만들고 나서 프로그램을 실행" 하는 명령어입니다

뒤에 좀 더 다루기로 하고 일단 넘어갑니다. 지금 주인공은 pivot_root 이니까요 ;-)

(실습1) pivot_root

이제 부터 탈옥을 막아 보겠습니다. 잠시 1편을 회고해 볼까요

```
# cd /tmp
# mkdir nginx-root
# docker export $(docker create nginx:latest) | tar -C nginx-root -xvf -
```

chroot + 이미지 실습

1편 링크 클릭

*말 나온김에 **이미지를 가져와 볼까요 ?***

일단, /tmp 에 "nginx-root" 라는 새로운 폴더를 하나 만들어 줍시다.

```
# cd /tmp
# mkdir nginx-root
```

chroot + 이미지 실습

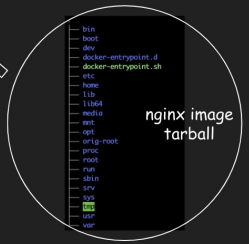
1편 링크 클릭

*말 나온김에 **이미지를 가져와 볼까요 ?***

```
# docker export $(docker create nginx:latest) | tar -C nginx-root -xvf -
```

새로 만든 nginx-root 라는 경로에 nginx 이미지를 풀어줄 건데요
이 명령은 아래의 단계를 수행합니다.

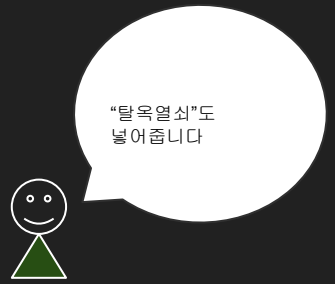
1. 이미지 저장소로 부터 nginx:latest 이미지를 가지고 와서
2. tarball로 export (압축) 한 것을
3. nginx-root 경로에 풀어줍니다.



(실습1) pivot_root

이제 부터 탈옥을 막아 보겠습니다. 잠시 1편을 회고해 볼까요

```
# gcc -o nginx-root/escape_chroot escape_chroot.c
```

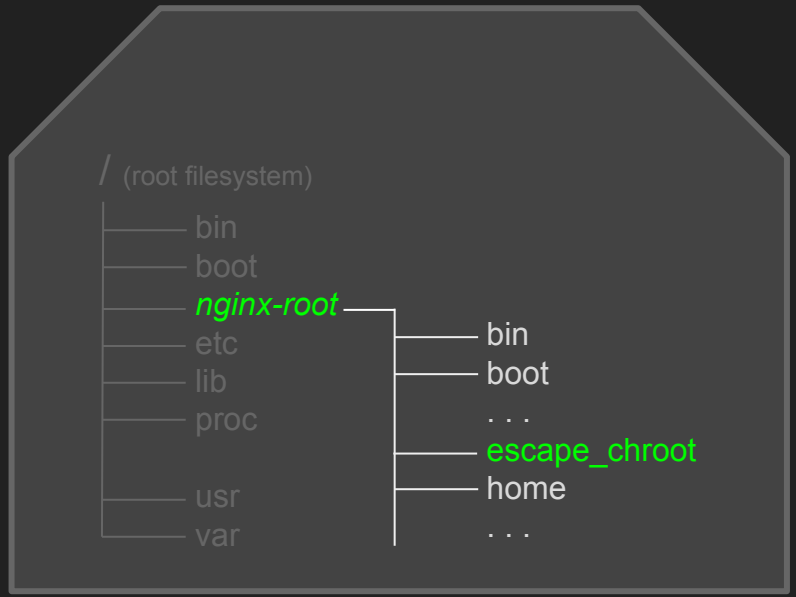


chroot 에서 탈출해보자

1편 읽고 풀어보기

탈옥 코드

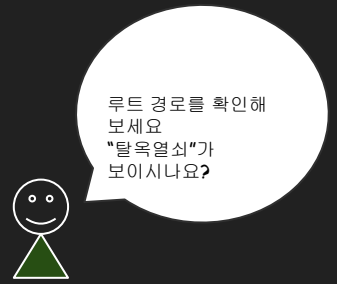
```
# vi escape_chroot.c

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

int main(void)
{
    mkdir(".out", 0755);
    chroot(".out");
    chdir("../../../../../");
    chroot(".");

    return execl("/bin/sh", "-i", NULL);
}
```

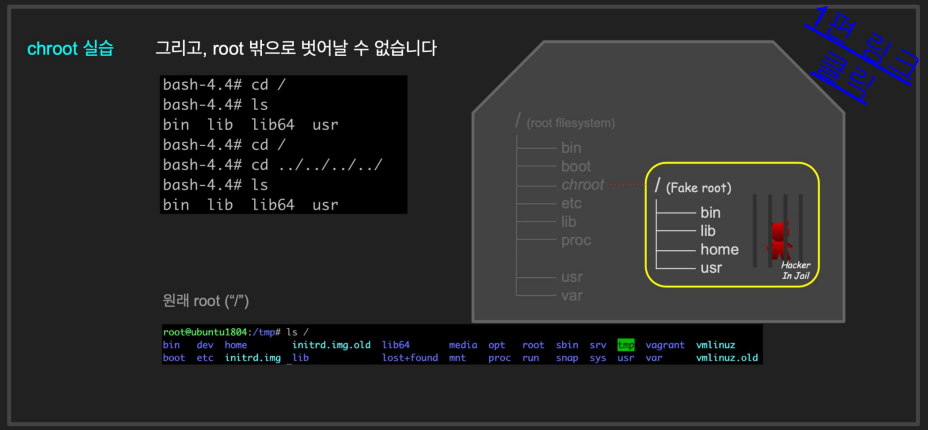


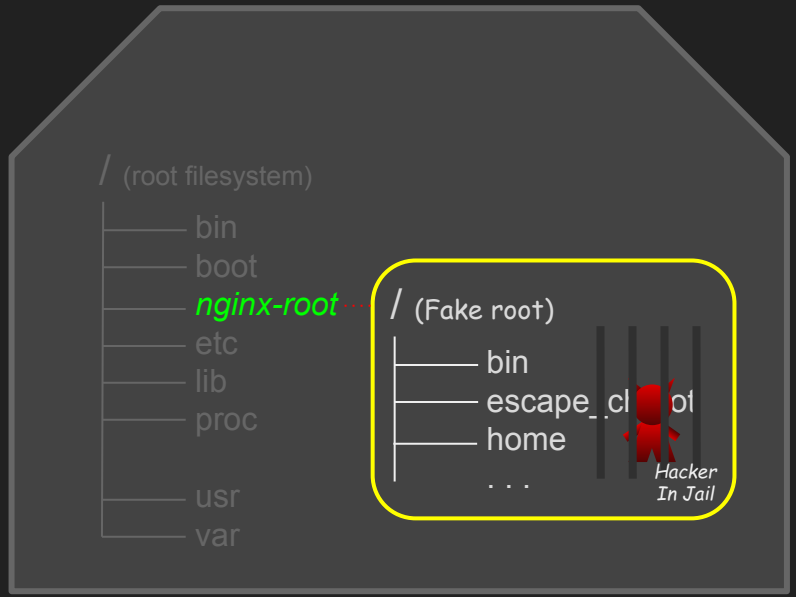
(실습1) pivot_root

이제 부터 탈옥을 막아 보겠습니다. 잠시 1편을 회고해 볼까요

```
# chroot nginx-root /bin/sh
# ls /
bin   dev                 docker-entrypoint.sh  etc   lib    media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d escape_chroot         home  lib64  mnt    proc  run   srv   tmp  var
# cd / && cd ../../../../../
# ls
```





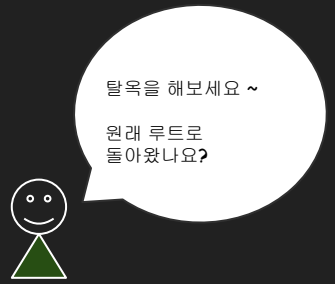


(실습1) pivot_root

이제 부터 탈옥을 막아 보겠습니다. 잠시 1편을 회고해 볼까요

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

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



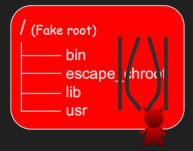
chroot 에서 탈출해보자

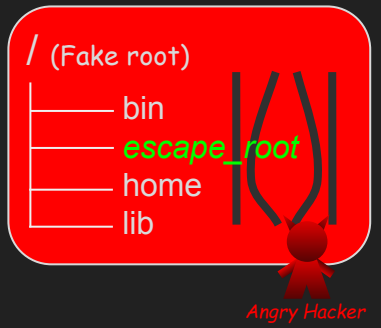
escape_chroot 를 실행하면 ...

1편 실습 클릭

```
# gcc -o new-root/escape_chroot escape_chroot.c
# chroot new-root
bash-4.4# ls /
bin  escape_chroot  lib  lib64  usr
bash-4.4# ./escape_chroot
# ls /
bin   dev  home        initrd.img.old  lib64       media  opt   root  sbin  srv  tmp  vagrant  vmlinuz
boot  etc  initrd.img  lib             lost+found  mnt    proc  run   snap  sys  usr  var      vmlinuz.old
```

탈옥





(실습1) pivot_root

이제 부터 "진짜" 탈옥을 막아 보겠습니다.

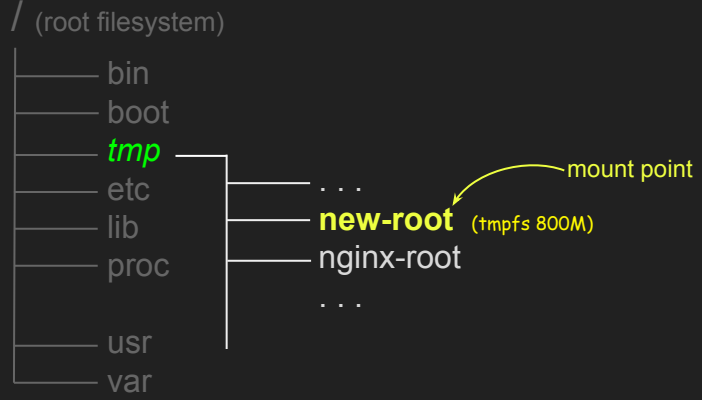

```
# exit; exit;

# pwd
/tmp

# ls ./nginx-root
```

```
bin   docker-entrypoint.d   etc   lib64   opt   run   sys   var
boot  docker-entrypoint.sh  home  media   proc  sbin  tmp
dev   escape_chroot         lib   mnt     root  srv   usr
```

(실습1) pivot_root

이제 부터 "진짜" 탈옥을 막아 보겠습니다.

```
# mkdir ./new-root
# mount -n -t tmpfs -o size=800M none ./new-root
```



new-root 폴더를 만들고 (800메가 짜리) tmpfs 파일시스템을 new-root에 붙였습니다.

아래⬇ 의 명령어로 잘 붙었는지 확인해 보세요

```
root@ubuntu1804:/tmp# mount | grep new-root
none on /tmp/new-root type tmpfs (rw,relatime,size=819200k)
```

(실습1) pivot_root

이제 부터 "진짜" 탈옥을 막아 보겠습니다.

```
# cp -r nginx-root/* ./new-root
# mkdir ./new-root/old-root
# tree -L 1 ./new-root
```

앞서 mount한 new-root 경로에 nginx-root의 파일들을 복사해 줍니다

pivot_root 시에 기존의 root filesystem을 mount 할 mount point 로

old-root 를 new-root 밑에 만들어 줍니다

```
root@ubuntu1804:/tmp# tree -L 1 new-root
new-root
├── bin
├── boot
├── dev
├── docker-entrypoint.d
├── docker-entrypoint.sh
├── escape_chroot
├── etc
├── home
├── lib
├── lib64
├── media
├── mnt
├── old-root
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

(실습1) pivot_root

이제 부터 "진짜" 탈옥을 막아 보겠습니다.

```
# cd ./new_root
# unshare -m
```

unshare 명령의 -m 옵션은 *(앞으로 다룰)* mount namespace를 분리하여 프로세스를 실행해 줍니다

pivot_root 는 앞에서도 언급했지만 root filesystem의 mount point를 변경하기 때문에

호스트에 영향을 주지 않기 위하여 unshare -m을 실행하여 호스트와 mount namespace를

분리하였습니다

(실습1) pivot_root

이제 부터 "진짜" 탈옥을 막아 보겠습니다.

```
# pivot_root . old-root
# cd /
# ls
```

```
bin   dev                 docker-entrypoint.sh  etc   lib    media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d escape_chroot         home  lib64  mnt    proc  run   srv   tmp  var
```

```
# cd / && cd ../../../../
# ls
```

드디어 pivot_root을 실행하였습니다.

"탈옥열쇠" 가 보이는 군요.

cd 로 root 경로가 바뀌는지 확인해 보세요

(실습1) pivot_root

탈옥시도

```
# ls
bin   dev                 docker-entrypoint.sh  etc   lib    media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d escape_chroot         home  lib64  mnt    proc  run   srv   tmp  var
# ./escape_chroot
# ls
bin   dev                 docker-entrypoint.sh  etc   lib    media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d escape_chroot         home  lib64  mnt    proc  run   srv   tmp  var
```

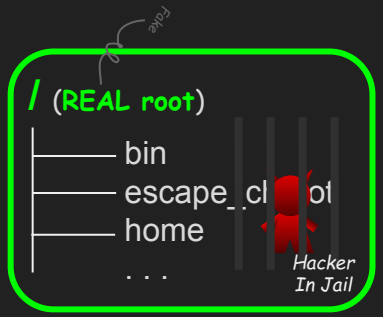
탈옥이 되지 않습니다

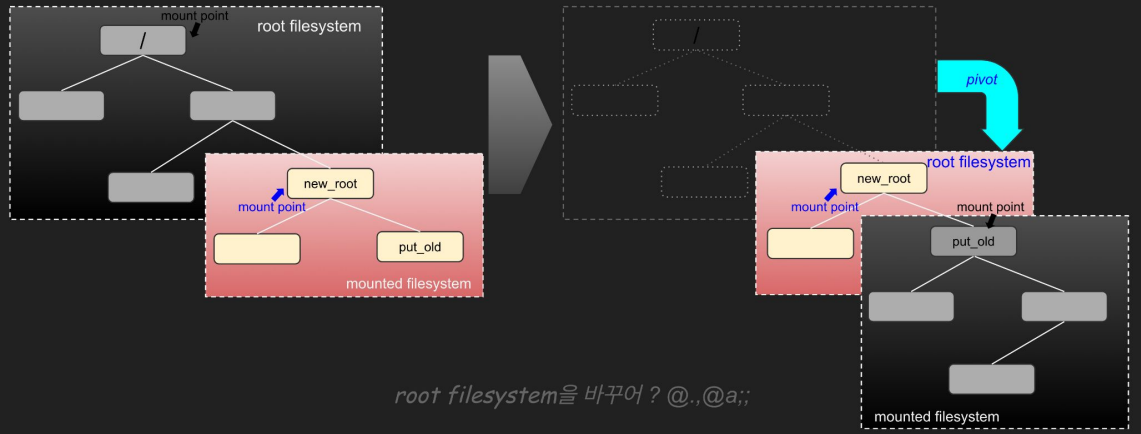
프로세스의 root filesystem 자체를 바꿔버렸기 때문이죠

(실습1) pivot_root

pivot_root로 root filesystem의 mount point를 바꿔서 탈옥문제를 해결하였습니다

탈옥문제 Solved (pivot_root)





root filesystem을 바꾸어 ? @.,@a;;

하지만 ...

아직 해결해야 할 문제들이 남아있습니다

fake root path : 탈출이 가능함 **Solved (pivot_root)**

isolation 되지 않음 *: 호스트의 filesystems, process tree, network, ipc, … 에 접근 가능*

root 권한 사용 *: root 권한과 그에 따른 보안 문제 초래 가능*

resource 무제한 *: cpu, memory, i/o, network, … 호스트의 자원을 제한 없이 사용 가능*

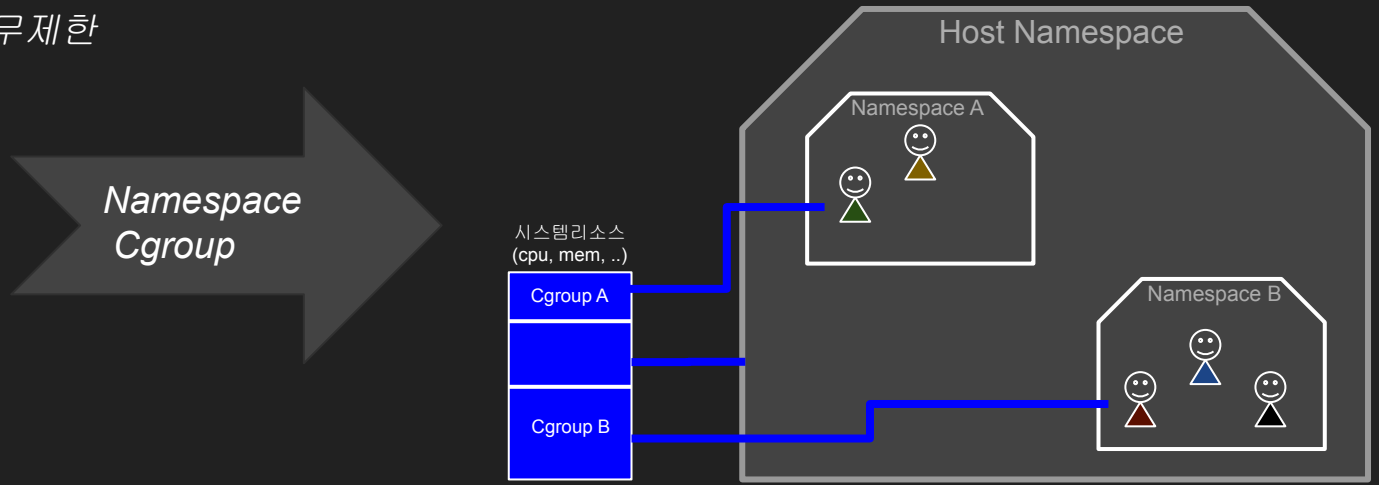
chroot 문제를 해결해 보자

fake root path *Solved (pivot_root)*

isolation 되지 않음

root 권한 사용

resource 무제한



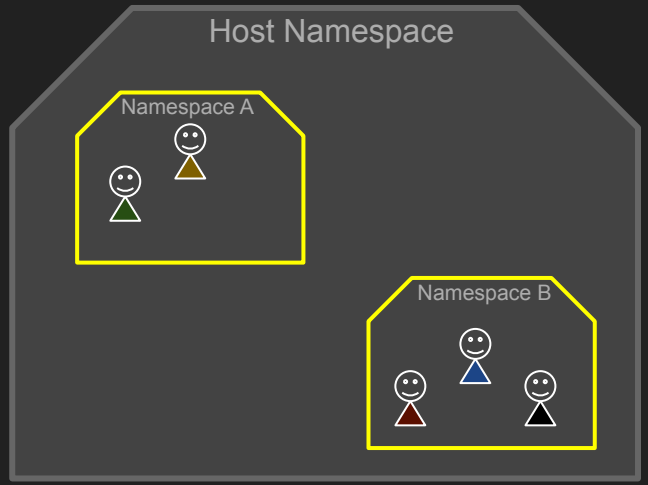
Namespaces

네임스페이스는 프로세스에 격리된 환경과 리소스를 제공합니다

네임스페이스 안에서의 변경은 내부의 프로세스들에게만 보여지고

다른 프로세스들에게는 보이지 않습니다

https://man7.org/linux/man-pages/man7/namespaces.7.html



Namespaces

Linux kernel feature (2002, Linux 2.4.19)

"container ready" (2013, kernel version 3.8)

리눅스 커널 피쳐로 네임스페이스가 처음 소개가 된 것은 2002년
이지만
실제 사용할 수 있는 수준인 지금의 컨테이너 형태를 갖춘 것은 2013년
입니다.

https://man7.org/linux/man-pages/man7/namespaces.7.html

Namespaces

Namespace와 관련된 process의 특징

~ 모든 process 들은 namespace type별로 특정 네임스페이스에 속합니다

~ Child Process는 Parent process의 namespace를 상속받습니다

~ process는 네임스페이스 종류별로 일부는 host(root) namespace에
일부는 container의 namespace에 포함돼 있을 수 있습니다

예) pid, network, mount namespace는 컨테이너의 네임스페이스로 격리되고,
나머지는 호스트의 네임스페이스를 그대로 이용할 수 있습니다

https://man7.org/linux/man-pages/man7/namespaces.7.html

Namespace 종류

7 namespaces

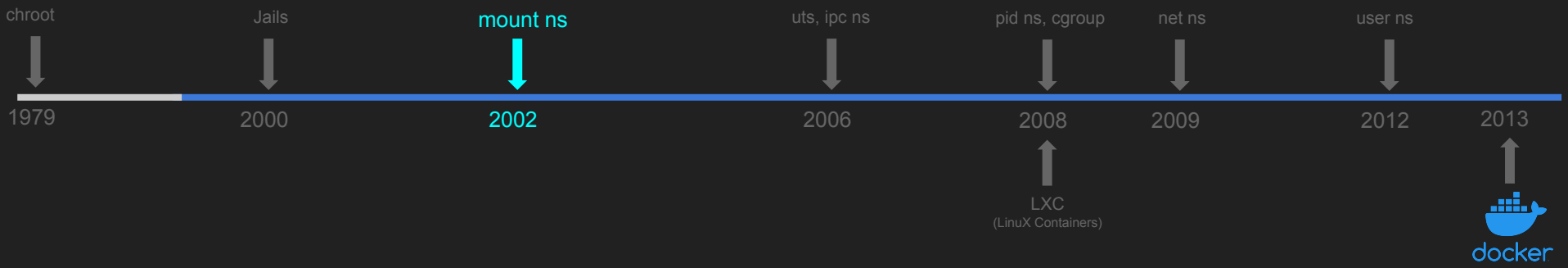
- mnt (CLONE_NEWNS)
- pid (CLONE_NEWPID)
- net (CLONE_NEWNET)
- ipc (CLONE_NEWIPC)
- uts (CLONE_NEWUTS)
- user (CLONE_NEWUSER)
- cgroup (CLONE_NEWCGROUP)
- time, syslog (2016, not fully implemented)

https://man7.org/linux/man-pages/man7/namespaces.7.html

Mount Namespace

2002년 최초의 네임스페이스

Mount Namespace : isolates mount points



Mount Namespace

mount namespace를 생성해 봅시다

터미널 #1

```
# unshare -m /bin/bash
```

unshare -m [명령어]

-m 옵션을 주면 [명령어]를 mount namespace를 isolation 하여 실행합니다

Mount Namespace

*mount namespace*를 생성해 봅시다

터미널 #1

```
# mkdir /tmp/mount_ns

# mount -n -t tmpfs tmpfs /tmp/mount_ns
```

새로운 폴더를 만들고

해당 폴더를 ***mount point****로 하여*

tmpfs *(임시 가상파일시스템)를 마운트합니다*

앞에서 (pivot_root) 잠깐 다뤘었는데요 :-) 기억하시나요?

mount namespace로 프로세스를 “격리”하게 되면

마운트와 관련된 변경 내용이 호스트에 영향을 주지 않습니다.

mount namespace를 생성해 봅시다

터미널 #1

```
# df -h | grep mount_ns

# mount | grep mount_ns
```

마운트가 잘 됐는지 확인해 봅니다

```
root@ubuntu1804:/tmp# df -h | grep mount_ns
tmpfs                       745M     0  745M   0% /tmp/mount_ns
root@ubuntu1804:/tmp# mount | grep mount_ns
tmpfs on /tmp/mount_ns type tmpfs (rw,relatime)
```

Mount Namespace

터미널(#2)을 하나 더 열어서 호스트와 비교해 봅니다

터미널 #1 (mount namespace)

```
# df -h | grep mount_ns

# mount | grep mount_ns
```

```
root@ubuntu1804:/tmp# df -h | grep mount_ns
tmpfs                   745M     0  745M   0% /tmp/mount_ns
root@ubuntu1804:/tmp# mount | grep mount_ns
tmpfs on /tmp/mount_ns type tmpfs (rw,relatime)
```

터미널 #2 (호스트)

```
# df -h | grep mount_ns

# mount | grep mount_ns
```

```
root@ubuntu1804:~# df -h | grep mount_ns
root@ubuntu1804:~# mount | grep mount_ns
```

터미널#1은 마운트 정보가 존재하고, 호스트(터미널#2)에는 mount_ns가 보이지 않습니다

Mount Namespace

터미널(#2)을 하나 더 열어서 호스트와 비교해 봅니다

터미널 #1 (mount namespace)

```
# readlink /proc/$$/ns/mnt
mnt:[4026532202]
```

터미널 #2 (호스트)

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

“각 프로세스의 네임스페이스 정보”는 /proc/{pid}/ns 에서 확인할 수 있습니다

$$ 는 현재 프로세스 id (pid)이며 readlink /proc/$$/ns/mnt 를 통해서

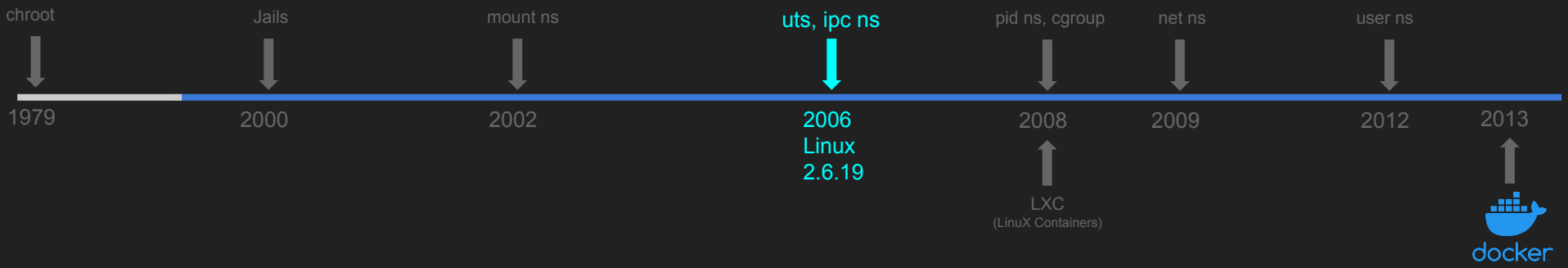
현재 프로세스의 mount namespace inode 값을 확인할 수 있고

이 값을 비교하여 동일한 네임스페이스 여부를 확인합니다

UTS Namespace, IPC Namespace

UTS Namespace : isolates Hostname and domain name

IPC Namespace : isolates System V IPC, Posix message queues



UTS Namespace

UTS ? UNIX Time-sharing System , 시분할 시스템

여러 사용자 작업 환경 제공하고자 ...

```
# unshare -u

# hostname
ubuntu1804

# hostname <your-name>

# hostname
<your-name>
```

unshare -u [명령어]

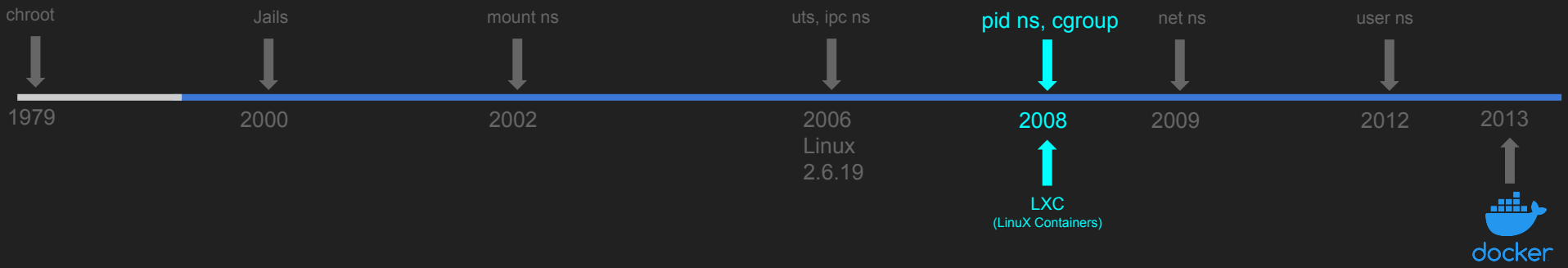
-u 옵션을 주면 [명령어]를 UTS namespace를 isolation 하여 실행합니다

** [명령어]를 지정하지 않으면 환경변수 $SHELL 을 실행합니다*

PID Namespace, Cgroup Namespace

PID Namespace : isolates Process IDs

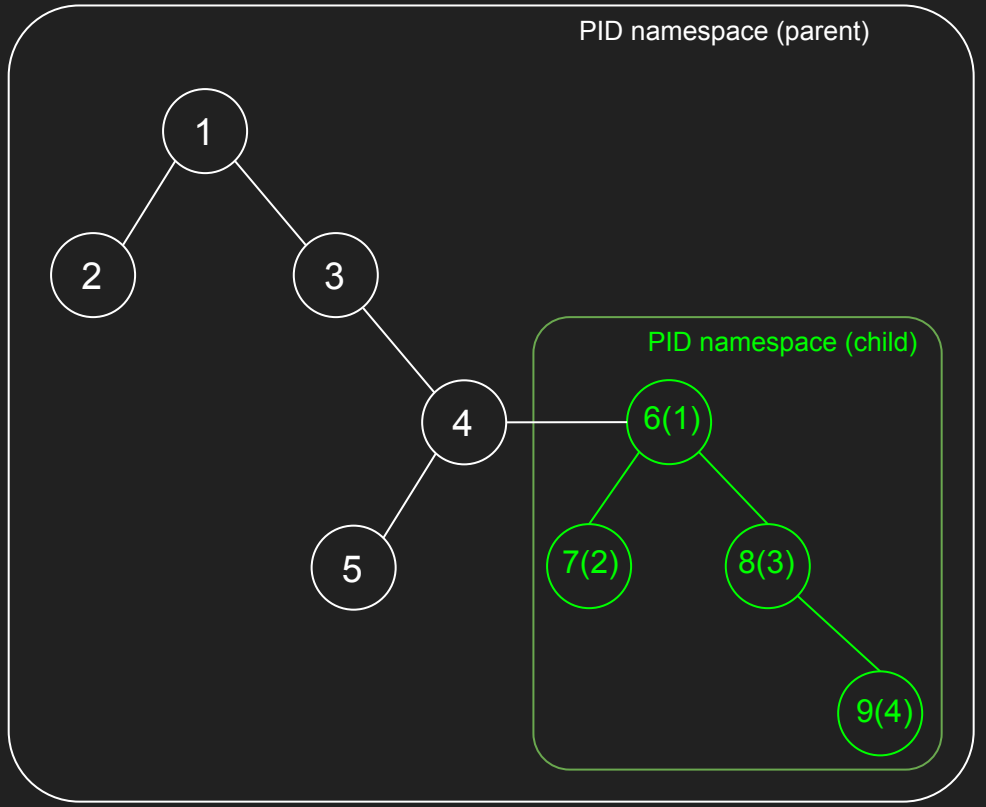
Cgroup Namespace : isolates Cgroup root directory



PID Namespace

PID ? Process ID

- parent-child의 nested 구조
- parent tree의 id와 subtree의 id 두 개를 가짐
- child process가 pid namespace의 pid1
- pid1 (init)이 종료되면 pid namespace도 종료



PID Namespace

터미널 #1

```
# echo $$

# unshare -fp --mount-proc

# echo $$
```

unshare -p [명령어]

-p 옵션을 주면 [명령어]를 PID namespace를 isolation 하여 실행합니다

** PID namespace는 child process를 새로운 네임스페이스로 격리하기 때문에 -f (fork) 옵션을 사용하였고 ps 명령어를 사용하려면 /proc 를 mount 해야 하기 때문에 --mount-proc 옵션을 주었습니다*

자세한 내용은 뒷 편에서 다루도록 하겠습니다

PID Namespace

터미널 #1

ps aux

```
root@ubuntu1804:/tmp# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.2  21480  3836 pts/0    S    12:18   0:00 -bash
root          5  0.0  0.2  37376  3240 pts/0    R+   12:24   0:00 ps aux
```

터미널 #2 (호스트)

ps aux

```
root@ubuntu1804:~# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.5  77604  8728 ?        Ss   Apr25   0:01 /sbin/init
root          2  0.0  0.0      0     0 ?        S    Apr25   0:00 [kthreadd]
root          4  0.0  0.0      0     0 ?        I<   Apr25   0:00 [kworker/0:0H]
root          6  0.0  0.0      0     0 ?        I<   Apr25   0:00 [mm_percpu_wq]
root          7  0.0  0.0      0     0 ?        S    Apr25   0:00 [ksoftirqd/0]
root          8  0.0  0.0      0     0 ?        I    Apr25   0:08 [rcu_sched]
root          9  0.0  0.0      0     0 ?        I    Apr25   0:00 [rcu_bh]
root         10  0.0  0.0      0     0 ?        S    Apr25   0:00 [migration/0]
root         11  0.0  0.0      0     0 ?        S    Apr25   0:00 [watchdog/0]
root         12  0.0  0.0      0     0 ?        S    Apr25   0:00 [cpuhp/0]
root         13  0.0  0.0      0     0 ?        S    Apr25   0:00 [cpuhp/1]
root         14  0.0  0.0      0     0 ?        S    Apr25   0:00 [watchdog/1]
root         15  0.0  0.0      0     0 ?        S    Apr25   0:00 [migration/1]
root         16  0.0  0.0      0     0 ?        S    Apr25   0:00 [ksoftirqd/1]
```

PID Namespace

PID namespace에서의 pid와 호스트 네임스페이스에서의 pid 2개를 가집니다

터미널 #1

```
# readlink /proc/$$/ns/pid
```

```
root@ubuntu1804:/tmp# readlink /proc/1/ns/pid
pid:[4026532203]
```

터미널 #2 (호스트)

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

```
root@ubuntu1804:~# readlink /proc/7062/ns/pid
pid:[4026532203]
```

<target pid>는 PID namespace의 process (pid=1)의 호스트 상에서의 pid를 의미합니다.

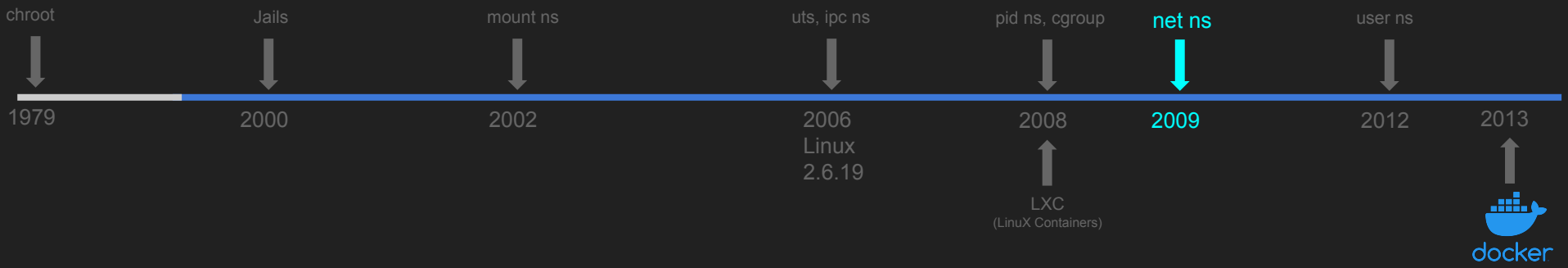
아래와 같이 unshare 명령어의 child를 찾으면 됩니다

 # ps -ef

```
root      7061  6875  0 12:18 pts/0    00:00:00 unshare -fp --mount-proc
root      7062  7061  0 12:18 pts/0    00:00:00 -bash
```

NET Namespace

NET Namespace : isolates Network (devices, stacks, ports, …)



NET Namespace

터미널 #1

```
# unshare -n

# ip a

# lsns -p $$

# lsns -p 1
```

NET Namespace

터미널 #1

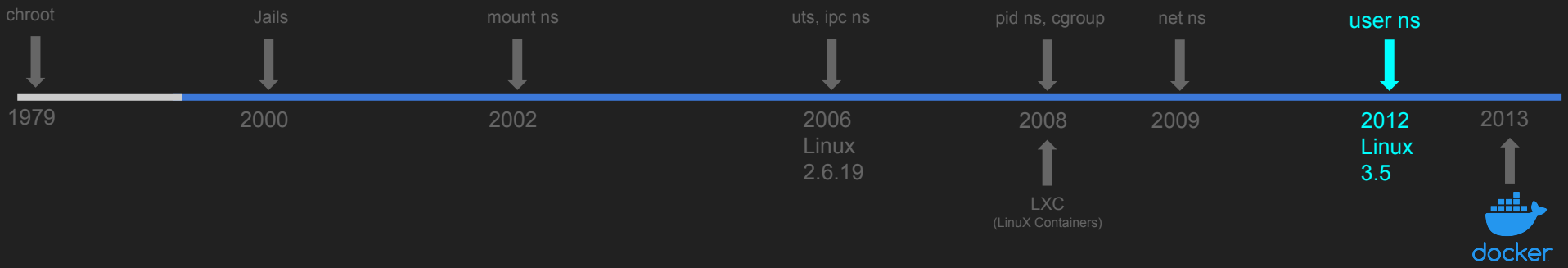
```
# unshare -n

# ip a

# lsns -p $$

# lsns -p 1
```

터미널 #2

```
# ip netns add mynet

# ip netns list

# ls /var/run/netns

# ip a

# nsenter --net=/var/run/netns/mynet

# ip a

# lsns -p $$

# lsns -p 1
```

USER Namespace

USER Namespace : isolates User and group IDs



USER Namespace

- 보안 상 중요
- 컨테이너의 “root 권한" 문제를 해결
- 네임스페이스 안과 밖의 UID/GID를 다르게 설정할 수 있음
    - 호스트 상에서는 권한이 없는 일반 유저를 컨테이너 안에서는 모든 권한을 가지게 할 수 있음

USER Namespace

터미널 #1

```
# unshare -U

# whoami

# id

# ls -al /proc/$$/ns

# lsns -p $$
```

터미널 #2 (호스트)

```
# whoami

# id

# ls -al /proc/$$/ns

# lsns -p $$
```

여러 Namespaces를 통하여 isolation 하는 것을 살펴보았습니다

fake root path *Solved*

isolation 되지 않음 *Solved*

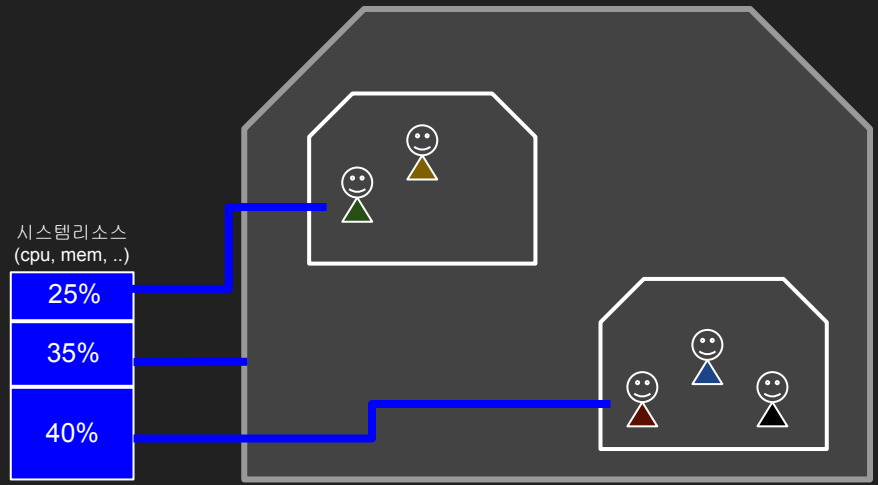
root 권한 사용 *Solved*

resource 무제한

Namespace	Flag	Isolates
Mount	CLONE_NEWNS	Mount points
Network	CLONE_NEWNET	Network devices, stacks, ports, . . .
Pid	CLONE_NEWPID	Process IDs Hierarchy
User	CLONE_NEWUSER	User and group IDs
IPC	CLONE_NEWIPC	System V IPC, POSIX message queues
UTS	CLONE_NEWUTS	Hostname and NIS domain name

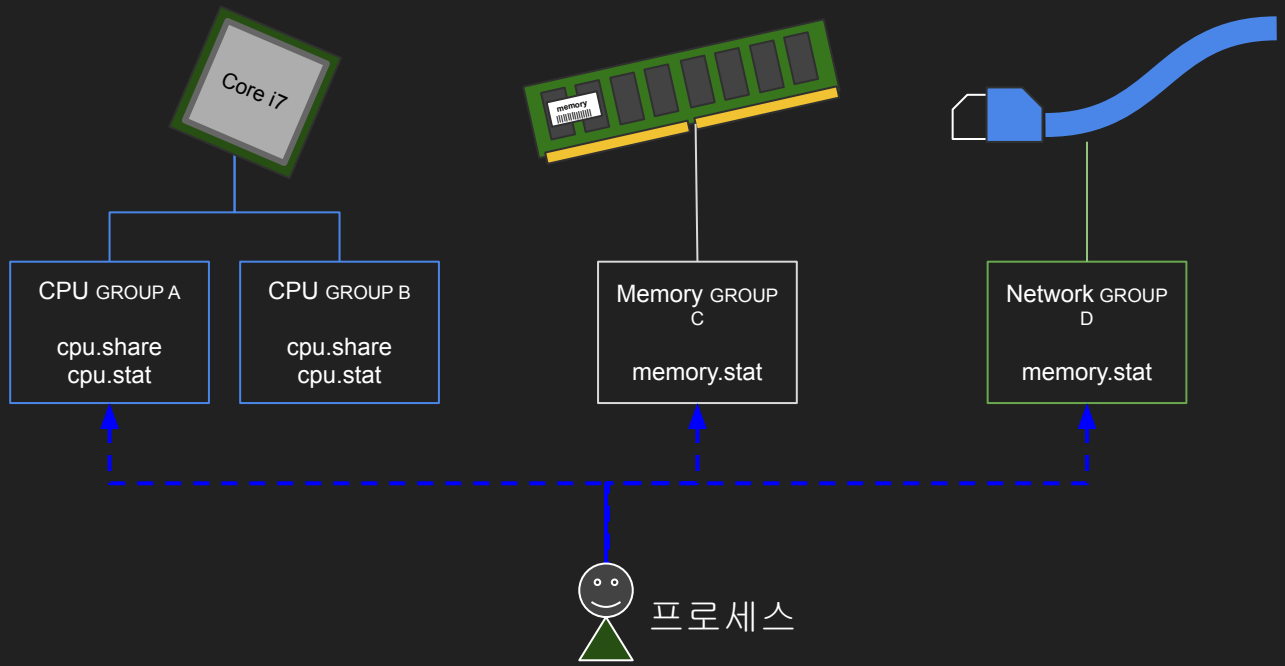
Cgroup

HW자원을 "그룹"별로 관리할 수 있는 리눅스 모듈
CPU, MEMORY, NETWORK, DISK IO . . .



Cgroup

하나 또는 복수의 장치를 묶어서 그룹을 만들 수도 있고
프로세스가 사용하는 리소스의 총량은 cgroup의 통제를 받음



Cgroup

cgroup은 파일시스템으로 관리됨

```
root@ubuntu1804:/vagrant# tree -L 1 /sys/fs/cgroup
/sys/fs/cgroup
├── blkio
├── cpu -> cpu,cpuacct
├── cpuacct -> cpu,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
```

```
root@ubuntu1804:/vagrant# cat /proc/$$/cgroup
12:freezer:/
11:perf_event:/
10:cpuset:/
9:pids:/user.slice/user-1000.slice/session-16.scope
8:memory:/user.slice
7:blkio:/user.slice
6:rdma:/
5:hugetlb:/
4:net_cls,net_prio:/
3:cpu,cpuacct:/user.slice
2:devices:/user.slice
1:name=systemd:/user.slice/user-1000.slice/session-16.scope
0::/user.slice/user-1000.slice/session-16.scope
```

Cgroup

실습 준비

```
# apt install -y cgroup-tools

# apt install -y stress
```

Cgroup

터미널 #1

```
# stress -c 1
```

터미널 #2

```
# top
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
28303 root      20   0    8248     96      0 R  99.7  0.0   0:14.09 stress
```

Cgroup

터미널 #1

```
# cgcreate -a root -g cpu:mycgroup

# ls -al /sys/fs/cgroup/cpu/ | grep mycgroup

# cgset -r cpu.cfs_quota_us=30000 mycgroup

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

터미널 #2

```
# top
```

cpu 사용률 (%CPU)

cpu.cfs_quota_us / cpu.cfs_period_us * 100

** 1ms = 1000us*

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

Cgroup

*리소스를 제한(**cpu 30%**) 하여 **chroot** 에서 **stress -c 1** 을 실행해 보세요*

지금까지 ...

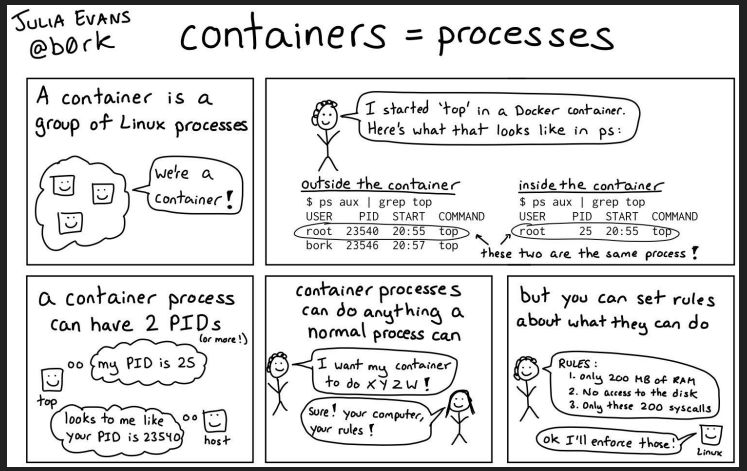
chroot 로 시작해서 컨테이너의 역사를 따라가면서
pivot_root, namespaces, cgroup 등 문제를 어떻게 해결해 가는지 살펴보았습니다

fake root path	*Solved*
isolation 되지 않음	*Solved*
root 권한 사용	*Solved*
resource 무제한	*Solved*

정리해보면 ...

> "Containers are processes" ,
> born from tarballs,
> anchored to namespaces,
> controlled by cgroups"

출처: https://twitter.com/jpetazzo/status/1047179436959956992



https://twitter.com/b0rk

3편 예고

1편과 2편을 통하여 컨테이너의 역사와 문제를 해결하는 과정을 개괄적으로 다루었습니다

이제부터는 좀 더 deep ~ 하게 하나씩 다뤄보려고 합니다.

3편에서는 “네트워크 네임스페이스”에 대해서 얘기하도록 하겠습니다.

END