실습과 함께 완성해보는

도커 없이 컨테이너 만들기

1편

Sam.0

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
```

컨테이너 ?

도커 ? 회사명 이면서 제품명 → 컨테이너를 다루는 플랫폼

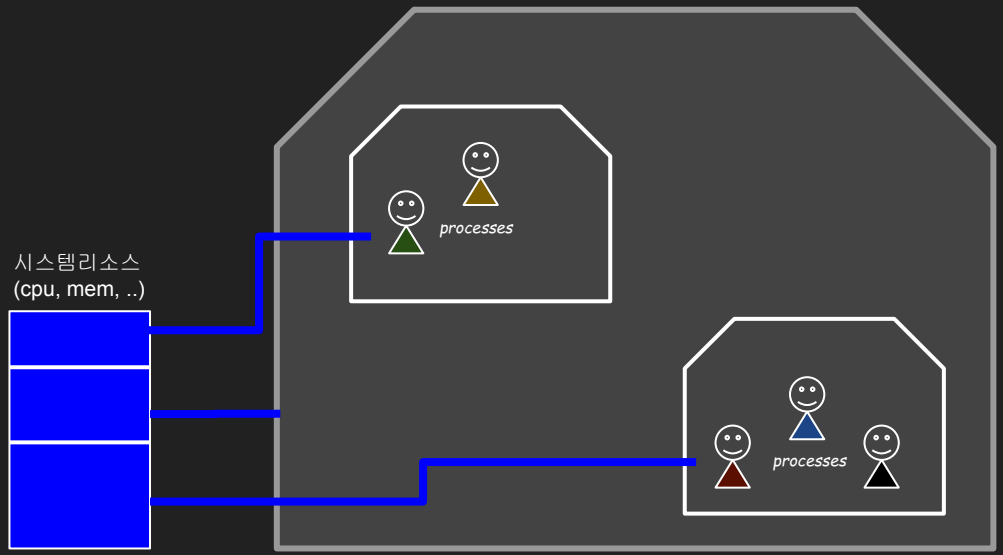
https://docs.docker.com/get-started/overview/

컨테이너 ? 격리된 환경과 통제된 리소스에서 실행되는 프로세스 그룹

차려준달까

호스트 안에 따로 한 살림

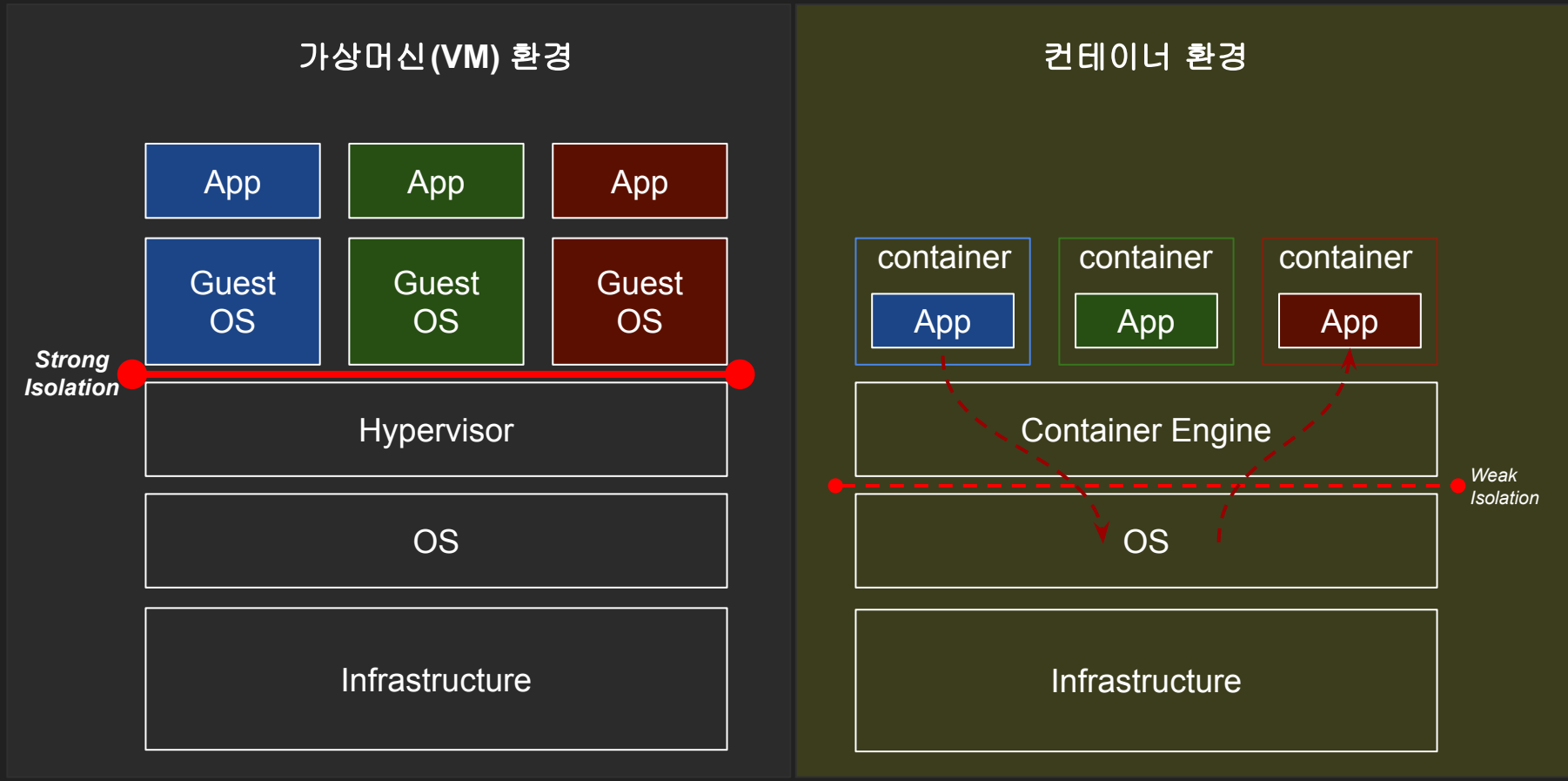
집 안의 집 ~ "컴퓨터 안 의 컴퓨터"



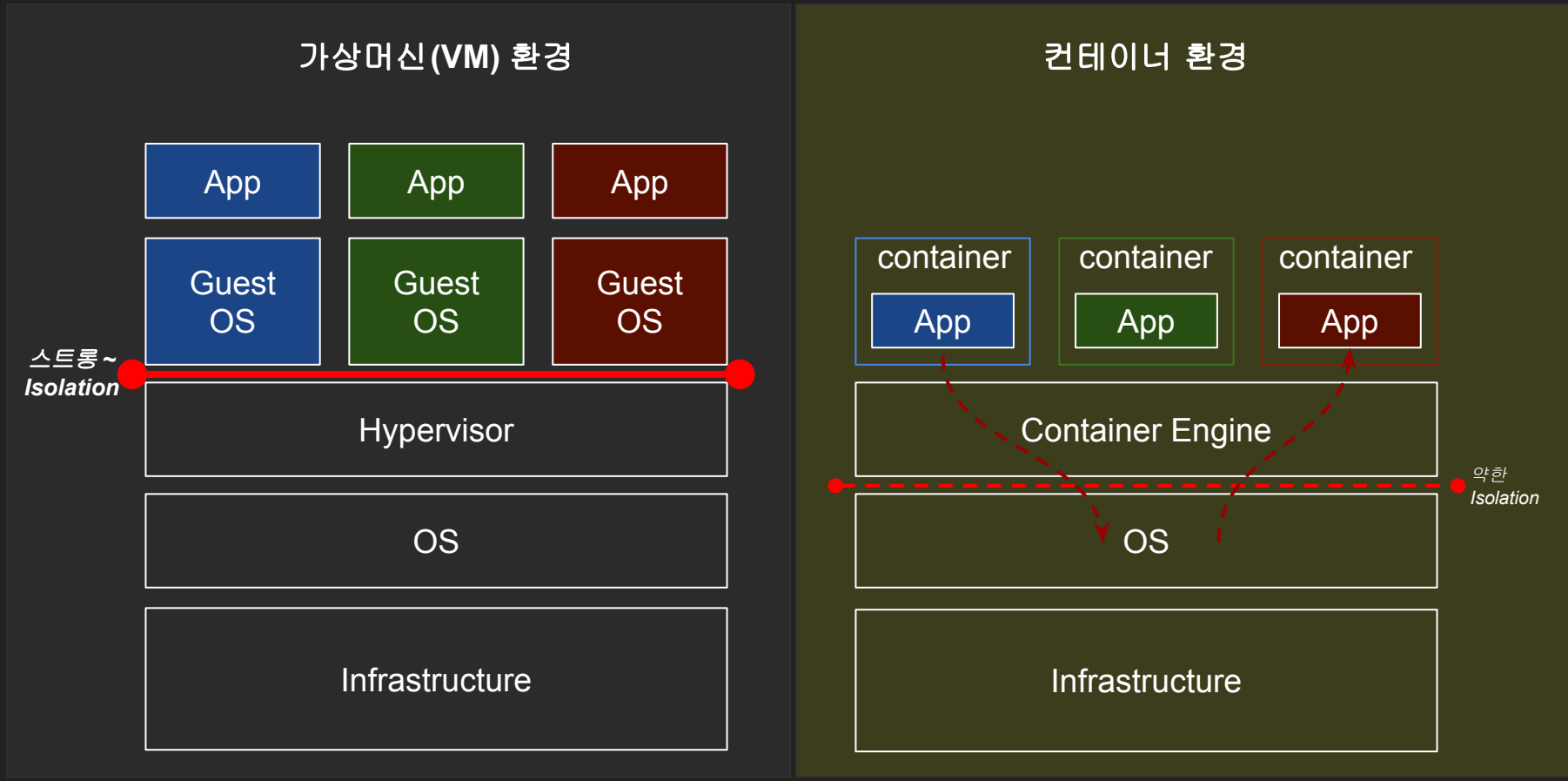
https://jessicagreben.medium.com/what-is-the-difference-between-a-process-a-container-and-a-vm-f36ba0f8a8f7

컨테이너?

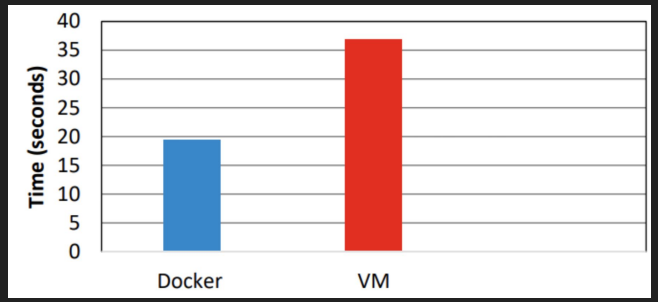
VM 보다 가볍다



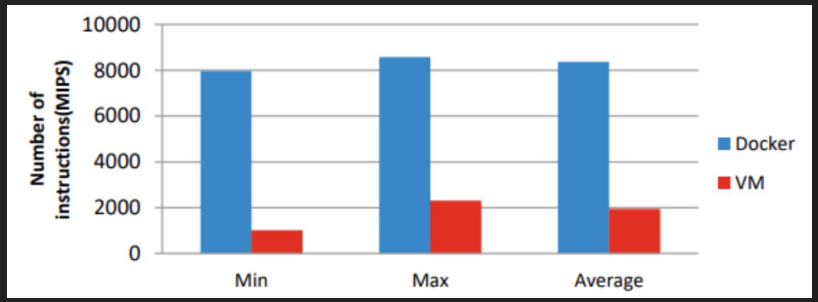
컨테이너? VM 보다 가볍다 왜 ? Guest OS가 없다 (호스트 OS를 공유)



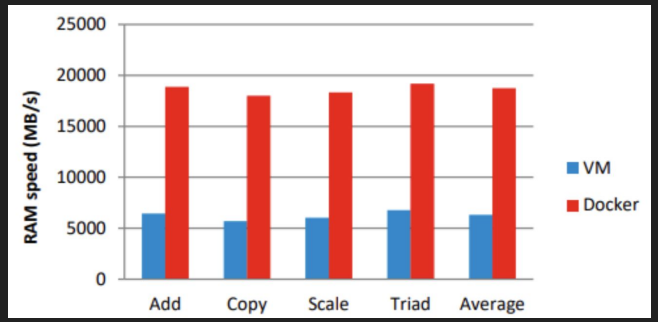
컨테이너 vs VM 성능비교 ~ 컨테이너가 빠름 빠름 ~



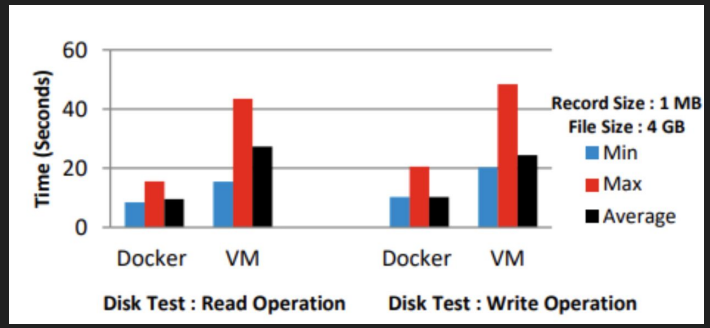
최대소수연산(sysbench) 소요시간 비교



압축 성능 비교 (LZMA benchmark) for CPU performance



RAM Speed 비교

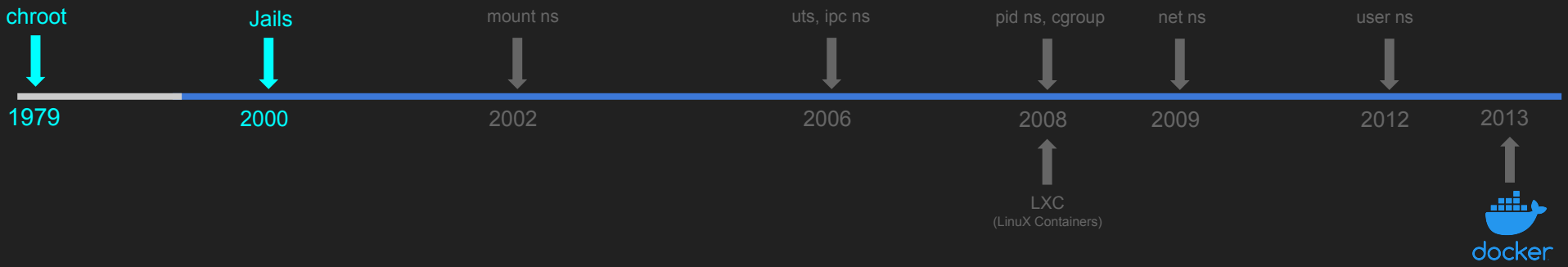


디스크 I/O 성능비교 (IOzone benchmark)

https://artistdata.tistory.com/5

컨테이너의 역사

컨테이너의 시작은 1979년 chroot 로 부터 ~

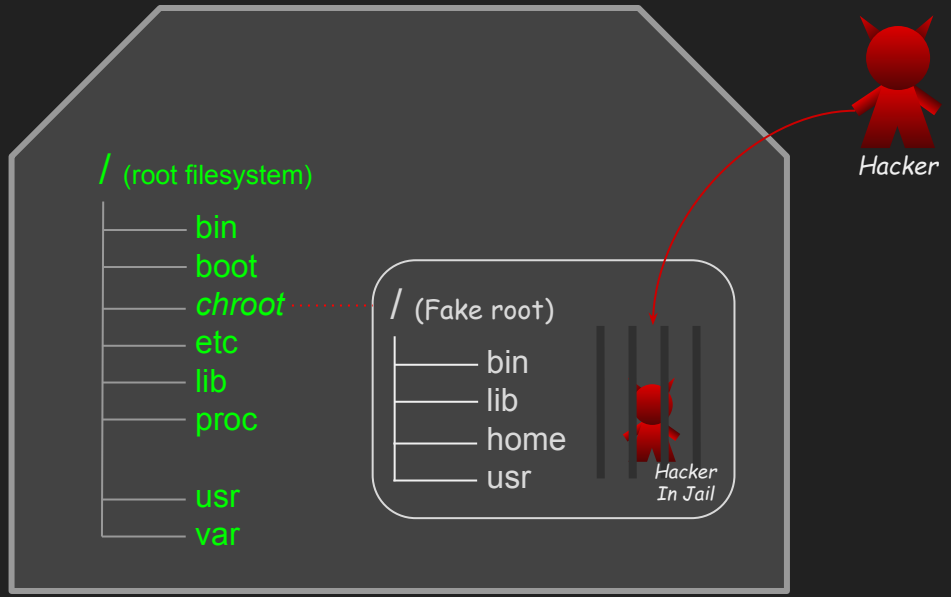


chroot ? change root directory ‘ / ’

리눅스 파일시스템은

모든 파일 및 디렉토리가 “root (/)” 로 부터 시작된다.

chroot ? change root directory ‘ / ’

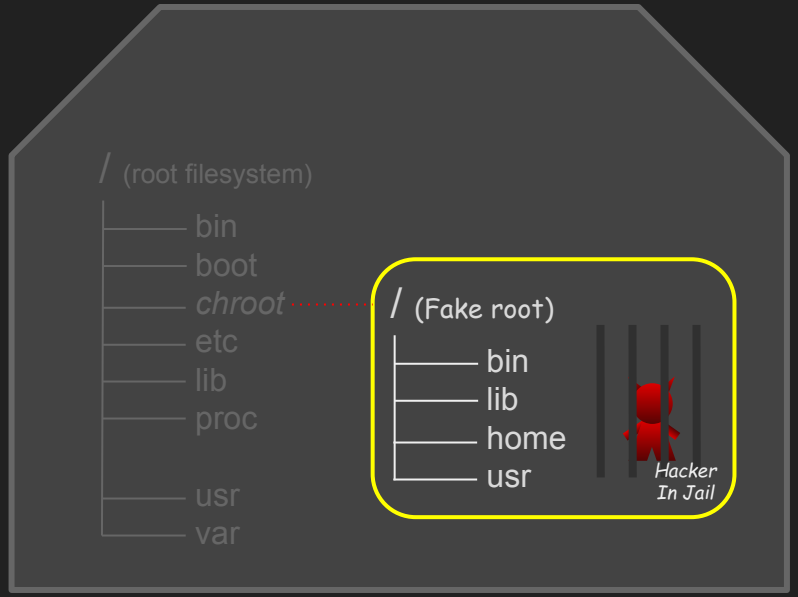


따라서.. 특정 디렉토리 경로를 root로 지정할 수 있으면
해당 경로에 프로세스를 가둘 수 있다는 점에 착안 ~

원격 유저(FTP 등)를 특정 디렉토리
경로에 가두기 위한 용도 등으로 사용됨

chroot 실습

*chroot*로 컨테이너를 만들어 보자



chroot 실습

chroot는 새로운 경로(NEWROOT)와 실행할 커맨드를 인자로 받습니다.

man chroot

```
Usage: chroot [OPTION] NEWROOT [COMMAND [ARG]...]
  or:  chroot OPTION
Run COMMAND with root directory set to NEWROOT.

  --groups=G_LIST        specify supplementary groups as g1,g2,..,gN
  --userspec=USER:GROUP  specify user and group (ID or name) to use
  --skip-chdir           do not change working directory to '/'
      --help     display this help and exit
      --version  output version information and exit

If no command is given, run '"$SHELL" -i' (default: '/bin/sh -i').
```

*~ 커맨드를 따로 지정하지 않으면 **default** 커맨드 (/bin/sh) 로 동작합니다.*

chroot 실습 *new-root 폴더를 만들고, chroot를 해봅시다*

```
# mkdir new-root

# chroot new-root /bin/bash
```

chroot 실습

```
# chroot new-root /bin/bash
chroot: failed to run command ‘/bin/bash’: No such file or directory
```

~ chroot의 커맨드(/bin/bash)는 new-root 경로를 기준으로 합니다
즉, 실행할 커맨드가 경로에 없다고 에러 빽 ~

chroot 실습 */bin/bash 파일을 new-root/bin 으로 복사해주세요*

```
# which bash
/bin/bash

# mkdir -p new-root/bin

# cp /bin/bash new-root/bin
```

chroot 실습

다시 실행해 보지만 …

```
# chroot new-root /bin/bash
chroot: failed to run command ‘/bin/bash’: No such file or directory
```

왜 또 ~ ㅠ

chroot 실습

/bin/bash에서 실행 시 참조하는 라이브러리들이 있었군요

ldd /bin/bash

```
linux-vdso.so.1 (0x00007ffd0e722000)
libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007f60309fd000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f60307f9000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6030408000)
/lib64/ld-linux-x86-64.so.2 (0x00007f6030f41000)
```

ldd prints the shared libraries required
참고) man ldd (list dynamic dependencies)

chroot 실습 *bash 실행을 위한 라이브러리도 복사해주세요* 한땀한땀~

```
linux-vdso.so.1 (0x00007ffd0e722000)
libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007f60309fd000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f60307f9000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6030408000)
/lib64/ld-linux-x86-64.so.2 (0x00007f6030f41000)
```

```
# mkdir new-root/lib
# cp /lib/x86_64-linux-gnu/libtinfo.so.5 new-root/lib/
# cp /lib/x86_64-linux-gnu/libdl.so.2 new-root/lib/
# cp /lib/x86_64-linux-gnu/libc.so.6 new-root/lib/

# mkdir new-root/lib64
# cp /lib64/ld-linux-x86-64.so.2 new-root/lib64/
```

Q) linux-vdso.so.1 은 왜 복사안해요?
A) *-vdso 는 커널레벨에서 제공되는 공유 라이브러리입니다.

참고) man vdso (virtual dynamic shared object)

chroot 실습

```
# chroot new-root /bin/bash
bash-4.4#
```

성공 ~

chroot 실습

But …

```
bash-4.4# ls
bash: ls: command not found
```

세상일이라는게 호락호락 하지 않네요 ...

ls 명령어도 복사해 넣어봅시다

*먼저, **exit** 를 입력하여 **chroot**로 실행된 프로세스를 종료합니다*

```
bash-4.4# exit
#
```

ls 명령어도 복사해 넣어봅시다

*앞서 **bash** 복사와 동일합니다 :-) 한땀한땀 ~ **cp** 해주세요*

```
root@ubuntu1804:~# which ls
/bin/ls
root@ubuntu1804:~# ldd /bin/ls
        linux-vdso.so.1 (0x00007ffd715e3000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fe124c8d000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe12489c000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fe12462a000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fe124426000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fe1250d7000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fe124207000)
```

```
# cp /bin/ls new-root/bin
# cp /lib/x86_64-linux-gnu/libselinux.so.1
```

. . . 이하 생략 . . .

직접 하실 수 있겠지요 ?

chroot 실습

다시 chroot 로 new-root의 bash를 실행해 보세요

```
# chroot new-root /bin/bash
bash-4.4#
```

이번에는 ls 도 동작합니다

```
bash-4.4# ls /
bin  lib  lib64  usr
```

chroot 실습 ls 로 확인해보니 “/” (root) 가 달라졌습니다

```
bash-4.4# ls /
bin  lib  lib64  usr
```

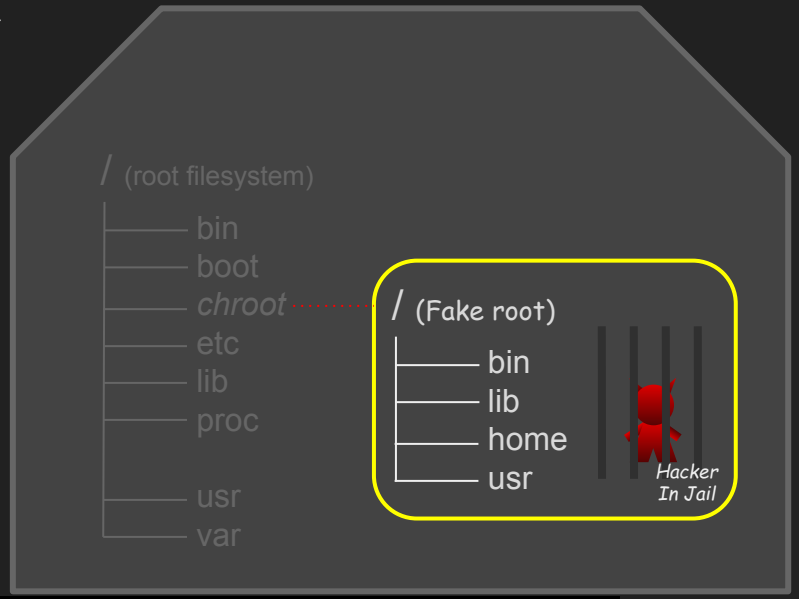
원래 root (“/”)

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

chroot 실습

그리고, root 밖으로 벗어날 수 없습니다

```
bash-4.4# cd /
bash-4.4# ls
bin  lib  lib64  usr
bash-4.4# cd /
bash-4.4# cd ../../../../
bash-4.4# ls
bin  lib  lib64  usr
```



원래 root (“/”)

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

chroot 실습 cat, mkdir, ps, … 원하는 프로그램을 복사하고 chroot 에서 실행해 보세요

그렇습니다 .

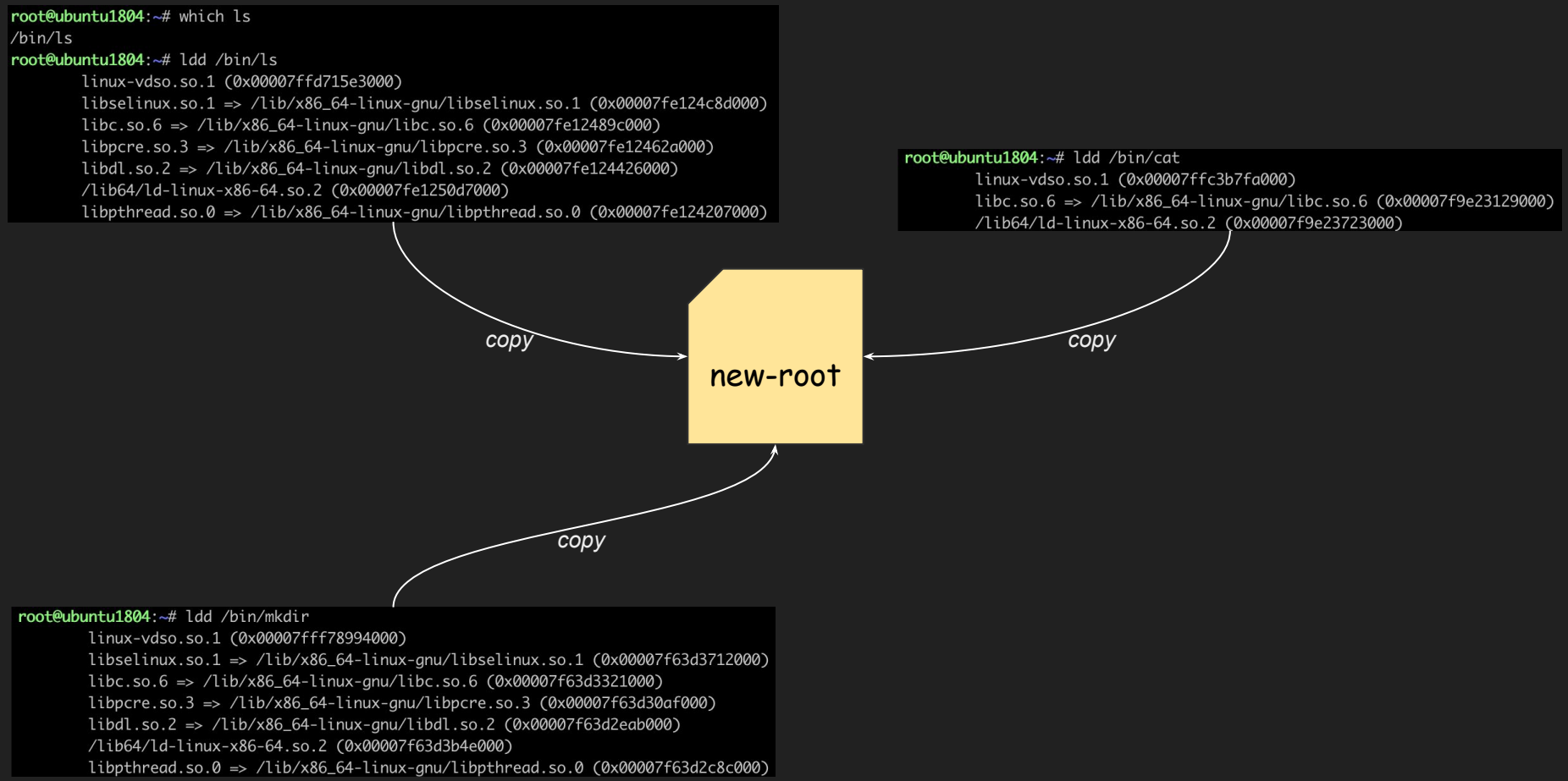
사용하고 싶은 명령어는 한땀한땀 넣어주어야 합니다.

bash와 라이브러리들을 복사해온 것 처럼 말이죠

chroot 실습 cat, mkdir, ps, … 원하는 프로그램을 복사하고 chroot 에서 실행해 보세요

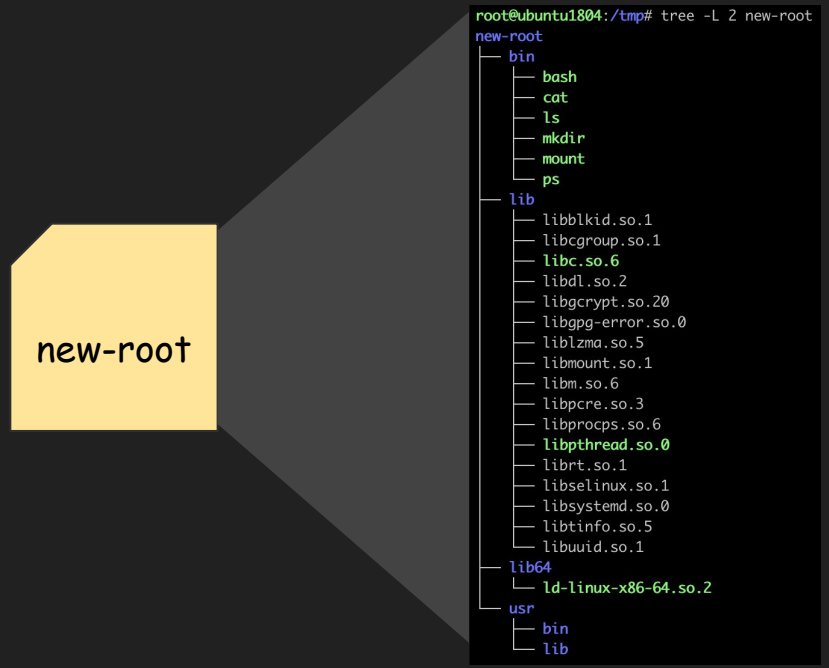
```
root@ubuntu1804:~# which ls
/bin/ls
root@ubuntu1804:~# ldd /bin/ls
        linux-vdso.so.1 (0x00007ffd715e3000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fe124c8d000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe12489c000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fe12462a000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fe124426000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fe1250d7000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fe124207000)
```

```
root@ubuntu1804:~# ldd /bin/cat
        linux-vdso.so.1 (0x00007ffc3b7fa000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9e23129000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9e23723000)
```




```
root@ubuntu1804:~# ldd /bin/mkdir
        linux-vdso.so.1 (0x00007fff78994000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f63d3712000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f63d3321000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f63d30af000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f63d2eab000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f63d3b4e000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f63d2c8c000)
```

chroot 실습 필요한 프로그램들은 이처럼 한땀한땀 복사해 넣어야 합니다.



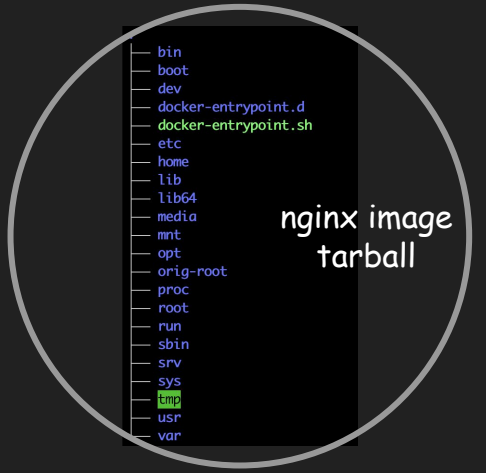
저는 욕심을 좀 부려보았습니다. ps 를 해보기 위해 mount 까지 넣어보았습니다.

누군가 미리 필요한 것들을 모아 둔 것을 가져다 쓰면 편하겠죠 ?

그것이 바로 이미지(image) 입니다 :-)

흔히들, “도커 이미지”라고 부르는 그것 말이죠.

일종의 tarball 이라고 생각하면 됩니다.



chroot + 이미지 실습

말 나온김에 이미지를 가져와 볼까요 ?

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

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

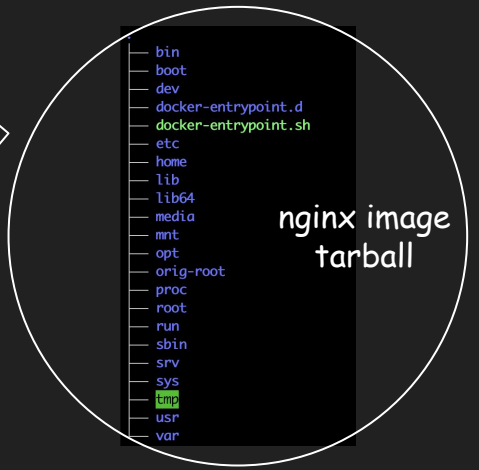
chroot + 이미지 실습

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.



chroot + 이미지 실습

*이처럼 필요한 이미지를 풀어놓은 경로를 **chroot** 를 해주면 ~*

```
# chroot  nginx-root  /bin/sh
#
```

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

ls 명령도 동작을 하네요 ~
누군가 필요한 것들을 잘 모아
놓았습니다.

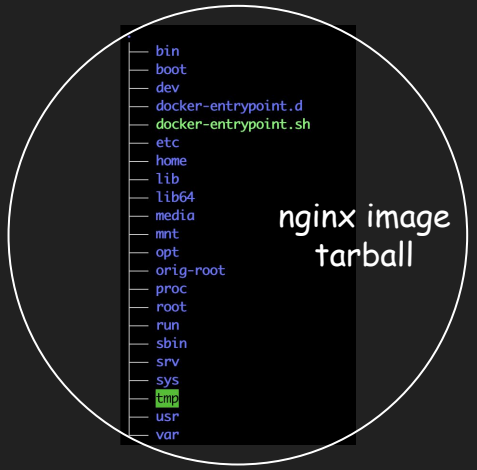
chroot + 이미지 실습

nginx image tarball 의 내용과 동일합니다 ... 압축을 풀어놓은 그대로죠 ㅎㅎ

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

=

동일



chroot + 이미지 실습 nginx 도 한번 실행해 볼까요?

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

chroot + 이미지 실습 터미널창을 하나 더 열어서 실제 접속이 되는지 확인해 보세요

터미널 #2

curl localhost

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

접속이 잘 되었다면 성공 ~

chroot + 이미지 실습 지금까지 nginx 이미지와 chroot를 이용하여 nginx 웹서버를 실행해 보았습니다

"실제 컨테이너 처럼" 프로그램 실행에 필요한 파일들을 모아놓고

*해당 경로를 **root**로 하여 프로세스를 실행하는 것을 재현해 보았는데요*

chroot 문제점

그런데 말입니다 …

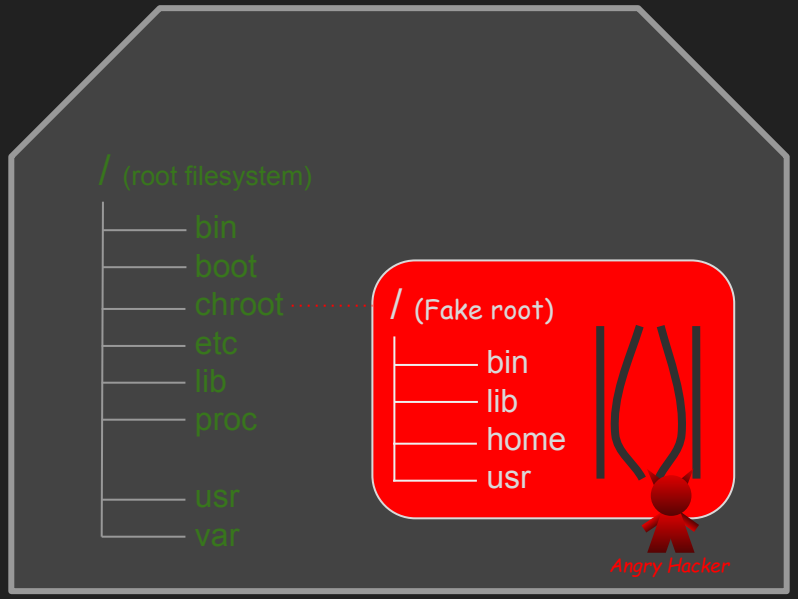
*chroot*에는 치명적인 문제가 있습니다


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

chroot 문제점

그것은 바로 ~~~

탈옥이 가능하다는 점입니다 ...



chroot 에서 탈출해보자

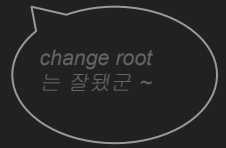
탈옥 코드

```
# 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);
}
```

chroot 에서 탈출해보자

*탈옥 코드를 컴파일하고 **new-root**에 복사합니다*

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

# chroot new-root /bin/bash

bash-4.4# ls  /
bin  escape_chroot  lib  lib64  usr
```

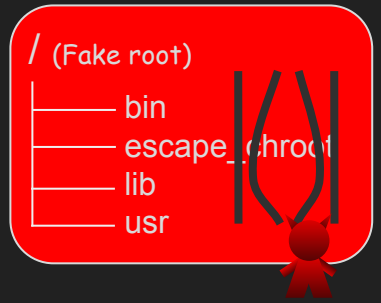


chroot 에서 탈출해보자

escape_chroot 를 실행하면 ...

```
# 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

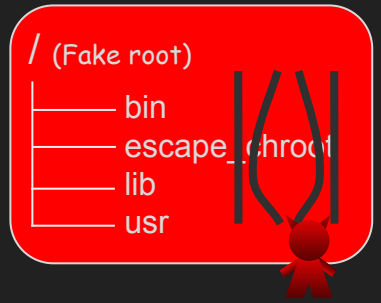


chroot 에서 탈출해보자

```
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
```

탈옥에 성공했다는 것은 ...

실제 루트 (/)를 취득했다는 얘기가 되겠죠 ㅠ.ㅠ



chroot 문제점

탈옥 말고도 ...

***chroot** 가 가지고 있는 문제점은 많습니다.*

~ 호스트와의 격리 문제, 리소스 제한, 보안 ...

앞으로 이러한 문제들 하나하나 해결해 가면서

함께 완전한 컨테이너를 만들어 보시죠 :-)

2편 예고

다음 편에서는 chroot 의 탈옥문제를 해결하는 방법으로써 pivot_root에 대해 다루도록 하겠습니다

END