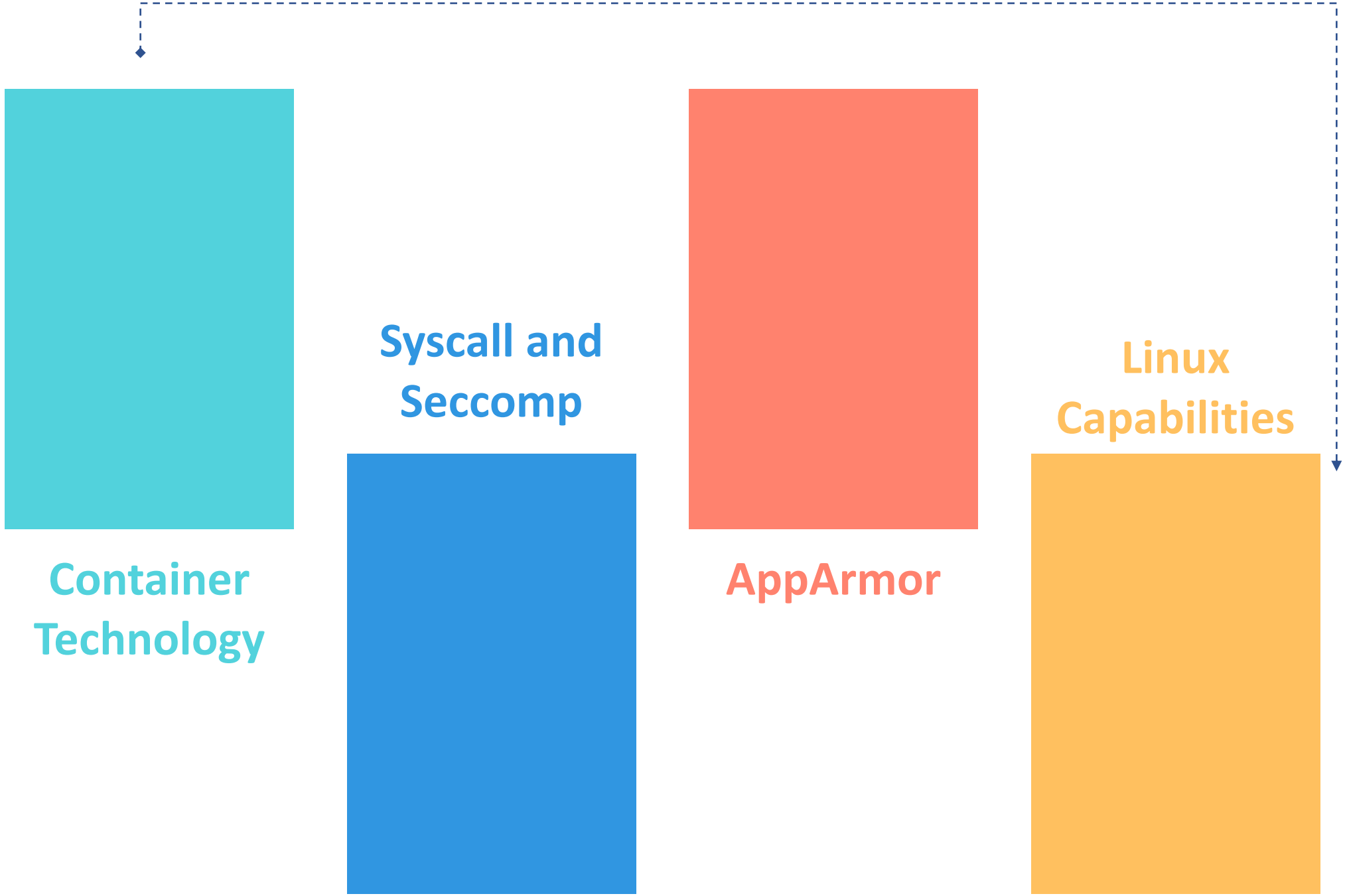
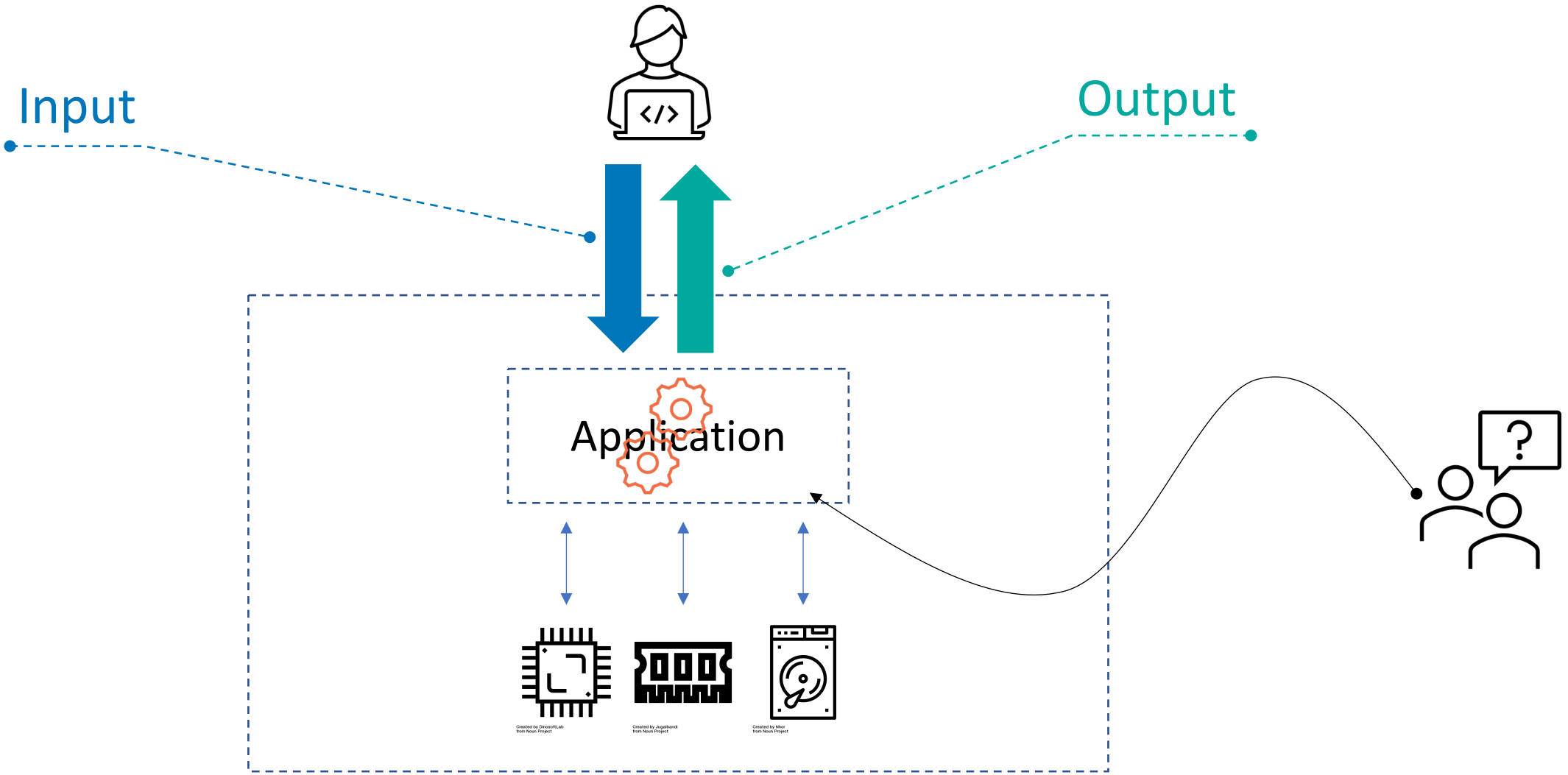
The Technology for Securing Your Containers

A G E N D A


Container Technology
Syscall and Seccomp
AppArmor
Linux Capabilities

System Calls and Seccomp

When you execute the applications?



System call

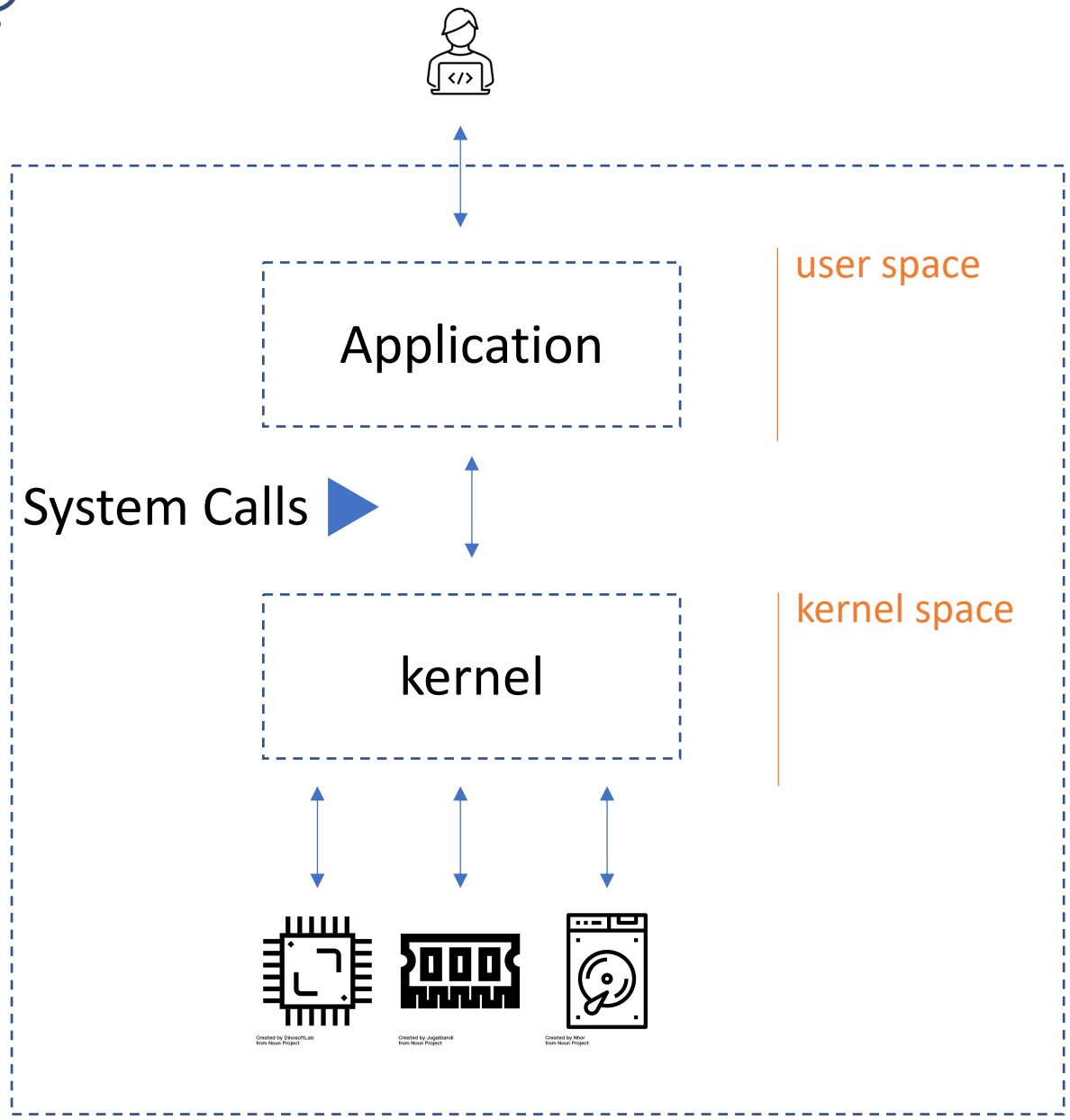
The system call is the **fundamental interface** between an **application and the Linux kernel**.

https://man7.org/linux/man-pages/man2/syscalls.2.html

A system call (commonly abbreviated to **syscall**) is the programmatic way in which a computer program **requests a service from the kernel** of the operating system on which it is executed. This may include hardware-related services (for example, accessing a hard disk drive or accessing the device's camera), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system.

https://en.wikipedia.org/wiki/System_call

Linux System Calls

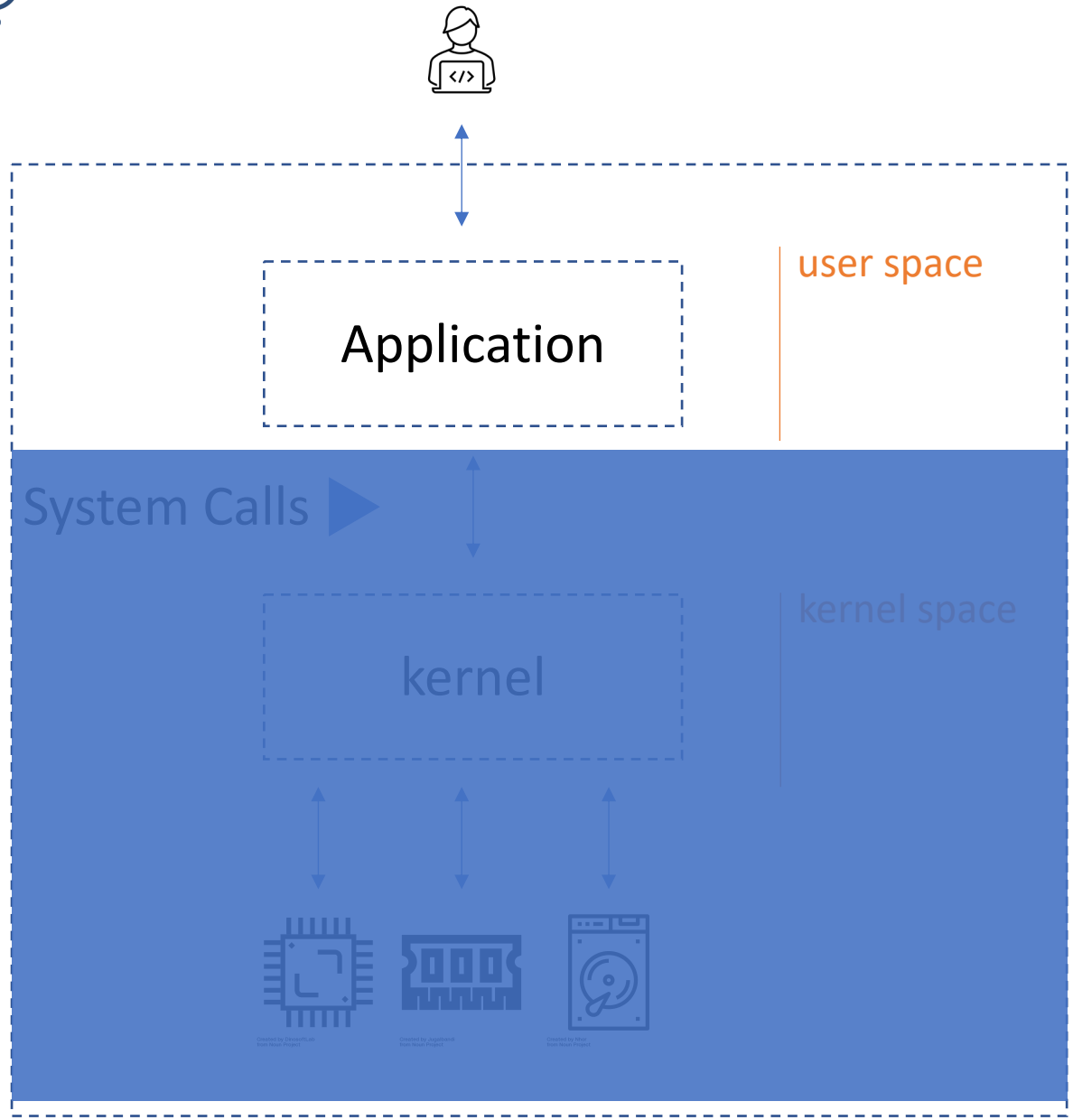


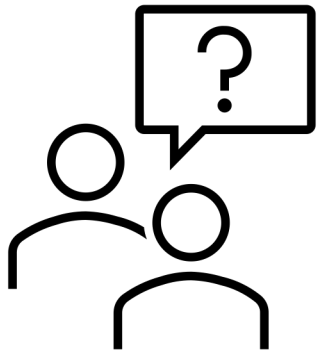
man7.org/linux/man-pages/man2/syscalls.2.html

System call	Kernel	Notes
_llseek(2)	1.2	
_newselect(2)	2.0	
_sysctl(2)	2.0	Removed in 5.5
accept(2)	2.0	See notes on socketcall(2)
accept4(2)	2.6.28	
access(2)	1.0	
acct(2)	1.0	
add_key(2)	2.6.10	
adjtimex(2)	1.0	
alarm(2)	1.0	
alloc_hugepages(2)	2.5.36	Removed in 2.5.44
arc_gettls(2)	3.9	ARC only
arc_settls(2)	3.9	ARC only
arc_usr_cmpxchg(2)	4.9	ARC only
arch_prctl(2)	2.6	x86_64, x86 since 4.12
atomic_barrier(2)	2.6.34	m68k only
atomic_cmpxchg_32(2)	2.6.34	m68k only
bdflush(2)	1.2	Deprecated (does nothing) since 2.6
bind(2)	2.0	See notes on socketcall(2)
bpf(2)	3.18	
brk(2)	1.0	
breakpoint(2)	2.2	ARM OABI only, defined with **__ARM_NR** prefix
cacheflush(2)	1.2	Not on x86
capget(2)	2.2	
capset(2)	2.2	
chdir(2)	1.0	
chmod(2)	1.0	
chown(2)	2.2	See chown(2) for version details
chown32(2)	2.4	
chroot(2)	1.0	
clock_adjtime(2)	2.6.39	
clock_getres(2)	2.6	
clock_gettime(2)	2.6	
clock_nanosleep(2)	2.6	
clock_settime(2)	2.6	
clone2(2)	2.4	IA-64 only
clone(2)	1.0	
clone3(2)	5.3	

[Memory by Jugalbandi, CPU by DinosoftLab, harddisk by Nhor] from the Noun Project

Linux System Calls





Why we need to know?

How to know?

[Memory by Jugalbandi, CPU by DinosoftLab, harddisk by Nhor] from the Noun Project

strace

strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.

https://en.wikipedia.org/wiki/Strace

Original author(s)	Paul Kranenburg
Developer(s)	Dmitry Levin
Stable release	5.15[1] / December 1, 2021; 34 days ago
Repository	github.com/strace/strace
Written in	C[2]
Operating system	Linux
Platform	AArch64, DEC Alpha, ARC, ARM EABI/OABI, AVR32, Blackfin, C-SKY, HP PA-RISC, IA-32, IA-64, Motorola 68k, Imagination META, MicroBlaze, MIPS, Nios II, OpenRISC, Power ISA 32/64 bit, RISC-V, System/390/z/Architecture, SuperH 32/64 bit, SPARC 32/64 bit, TILE, TILEPro, TILE-Gx, x86-64, x32 ABI, Xtensa
Available in	English[note 1]
Type	Debugging
License	LGPL v2.1+[note 2][4]
Website	strace.io

```
$ strace  touch aa.txt
execve("/usr/bin/touch", ["touch", "aa.txt"], 0x7fff86e46808 /* 26 vars */) = 0
brk(NULL)                                       = 0x5647a9241000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc6b4423a0) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=24982, ...}) = 0
[...]
openat(AT_FDCWD, "aa.txt", O_WRONLY|O_CREAT|O_NOCTTY|O_NONBLOCK, 0666) = 3
dup2(3, 0)                                      = 0
close(3)                                        = 0
[...]
exit_group(0)                                   = ?
+++ exited with 0 +++
```

execve() executes the program referred to by *pathname*.

openat() opens the file specified by *pathname*. If the specified file does not exist, it may optionally (if O_CREAT is specified in *flags*) be created

fstat() return information about a file, in the bufferpointed to by statbuf.

https://man7.org/linux/man-pages/man2/syscalls.2.html

```
$ strace -c touch aa.txt
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         3           read
  0.00    0.000000           0        22           close
  0.00    0.000000           0        18           fstat
  0.00    0.000000           0        22           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         6           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           dup2
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0        19           openat
  0.00    0.000000           0         1           utimensat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   104         2 total
```

Tracee: Runtime Security and Forensics using eBPF

Tracee is a **Runtime Security and forensics tool for Linux**. It is using Linux eBPF technology to trace your system and applications at runtime, and analyze collected events to detect suspicious behavioral patterns. It is delivered as a Docker image that monitors the OS and detects suspicious behavior based on a pre-defined set of behavioral patterns.

aqua tracee

only trace events from new processes

```
~$ docker run --name tracee --rm --pid=host --cgroupns=host --privileged -v
/tmp/tracee:/tmp/tracee -v /lib/modules/:/lib/modules/:ro -v /usr/src/:/usr/src/:ro -it
aquasec/tracee:latest trace --trace pid=new
TIME(s)        UID    COMM             PID     TID     RET             EVENT                ARGS
248811.781809  1000   bash             19302   19302   0               execve               pathname: /usr/bin/touch, argv: [touch
/tmp/cc.log]
248811.781897  1000   bash             19302   19302   0               security_bprm_check  pathname: /usr/bin/touch, dev: 8388609,
inode: 1744
248811.786988  1000   touch            19302   19302   -2              access               pathname: /etc/ld.so.preload, mode:
R_OK
248811.787020  1000   touch            19302   19302   0               security_file_open   pathname: /etc/ld.so.cache, flags:
O_RDONLY|O_LARGEFILE, dev: 8388609, inode: 1990
248811.787039  1000   touch            19302   19302   3               openat               dirfd: -100, pathname:
/etc/ld.so.cache, flags: O_RDONLY|O_CLOEXEC, mode: 2023792472
248811.787053  1000   touch            19302   19302   0               fstat                fd: 3, statbuf: 0x7FFF4E9E1700
248811.787073  1000   touch            19302   19302   0               close                fd: 3
248811.787107  1000   touch            19302   19302   0               security_file_open   pathname: /usr/lib/x86_64-linux-
gnu/libc-2.31.so, flags: O_RDONLY|O_LARGEFILE, dev: 8388609, inode: 4663
248811.787129  1000   touch            19302   19302   3               openat               dirfd: -100, pathname: /lib/x86_64-
linux-gnu/libc.so.6, flags: O_RDONLY|O_CLOEXEC, mode: 2023792472
248811.787169  1000   touch            19302   19302   0               fstat                fd: 3, statbuf: 0x7FFF4E9E1750
```

```
#- run this command in the other terminal.
~$ touch aa.txt
```

Criteria will be traced

Example:

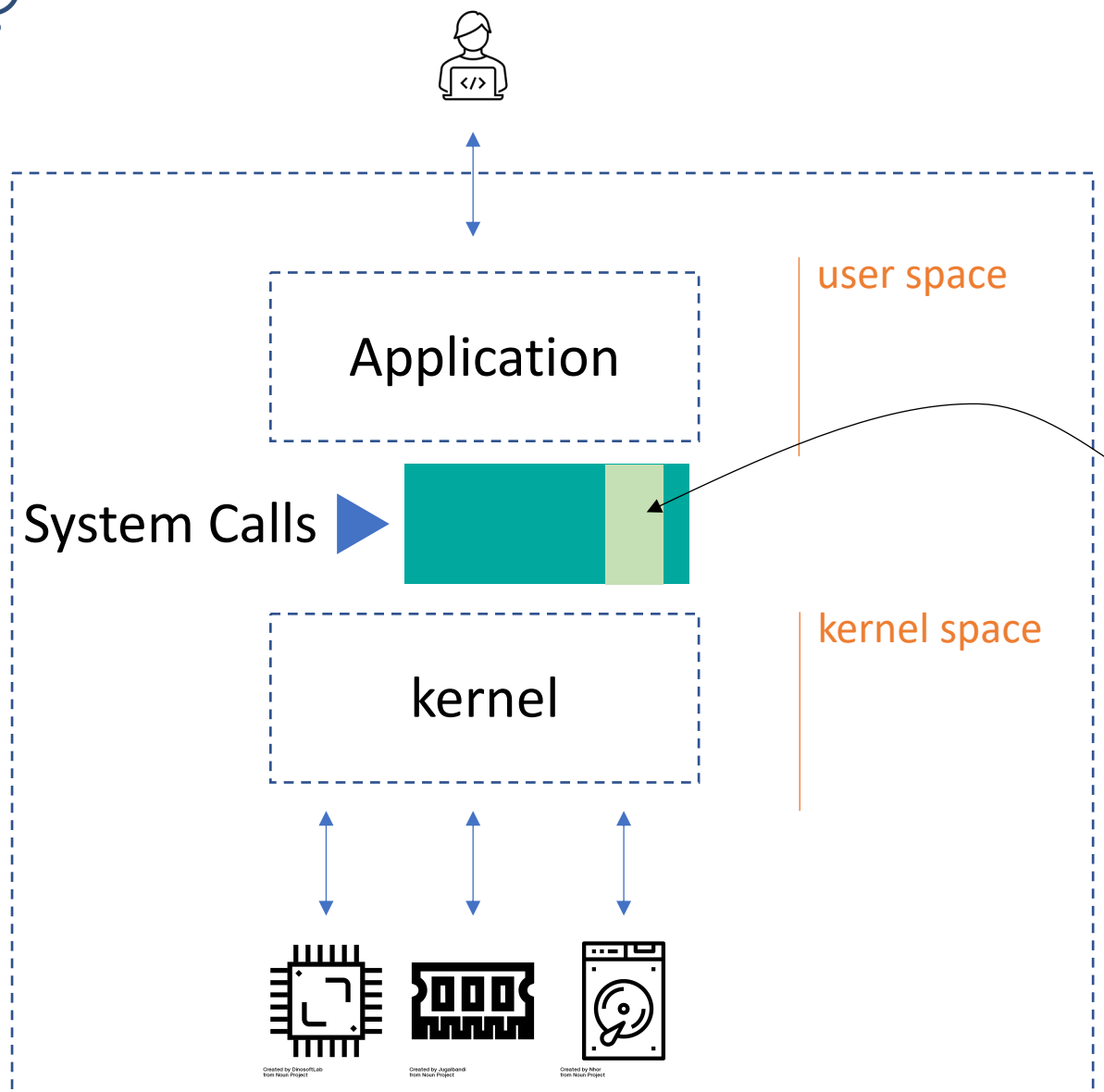
```
  --trace pid=new                                  | only trace events from new processes
  --trace pid=510,1709                             | only trace events from pid 510 or pid 1709
  --trace container=new                            | only trace events from newly created containers
  --trace container                                | only trace events from containers
  --trace '!container'                             | only trace events from the host
  --trace uid=0                                    | only trace events from uid 0
  --trace mntns=4026531840                         | only trace events from mntns id 4026531840
  --trace pidns!=4026531836                        | only trace events from pidns id not equal to 4026531840
  --trace tree=476165                              | only trace events that descend from the process with pid 476165
  --trace tree=3213,5200 --trace tree!=3215        | only trace events if they descend from 3213 or 5200, but not 3215
  --trace 'uid>0'                                  | only trace events from uids greater than 0
  --trace 'pid>0' --trace 'pid<1000'               | only trace events from pids between 0 and 1000
  --trace 'u>0' --trace u!=1000                    | only trace events from uids greater than 0 but not 1000
  --trace event=execve,open                        | only trace execve and open events
  --trace event=open*                              | only trace events prefixed by "open"
  --trace event!=open*,dup*                        | don't trace events prefixed by "open" or "dup"
  --trace set=fs                                   | trace all file-system related events
  --trace s=fs --trace e!=open,openat              | trace all file-system related events, but not open(at)
  --trace uts!=ab356bc4dd554                       | don't trace events from uts name ab356bc4dd554
  --trace comm=ls                                  | only trace events from ls command
  --trace close.fd=5                               | only trace 'close' events that have 'fd' equals 5
  --trace openat.pathname=/tmp*                    | only trace 'openat' events that have 'pathname' prefixed by "/tmp"
  --trace openat.pathname!=/tmp/1,/bin/ls          | don't trace 'openat' events that have 'pathname' equals /tmp/1 or
/bin/ls
  --trace comm=bash --trace follow                 | trace all events that originated from bash or from one of the
processes spawned by bash
```

only trace events from newly created containers

```
~$ docker run --name tracee --rm --pid=host --cgroupns=host --privileged -v
/tmp/tracee:/tmp/tracee -v /lib/modules/:/lib/modules/:ro -v /usr/src/:/usr/src/:ro -it
aquasec/tracee:latest trace --trace container=new
TIME             CONTAINER_ID  UID    COMM             PID/host        TID/host        RET              EVENT                ARGS
16:45:06:268694  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                sched_process_exec   cmdpath:
/docker-entrypoint.sh, pathname: /bin/dash, argv: [/bin/sh /docker-entrypoint.sh nginx -g daemon off;], dev: 57, inode: 258727,
invoked_from_kernel: 0, ctime: 1641275928817981796, stdin_type: S_IFCHR
16:45:06:268818  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   -2               access               pathname:
/etc/ld.so.preload, mode: R_OK
16:45:06:268876  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                cap_capable          cap:
CAP_SYS_ADMIN
16:45:06:268889  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                security_file_open   pathname:
/etc/ld.so.cache, flags: O_RDONLY|O_LARGEFILE, dev: 57, inode: 263633, ctime: 1641275930053985727
16:45:06:268903  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                security_file_open   pathname:
/etc/ld.so.cache, flags: O_RDONLY|O_LARGEFILE|O_NOATIME, dev: 8388609, inode: 263633, ctime: 1641275930053985727
16:45:06:268866  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   3                openat               dirfd: -
100, pathname: /etc/ld.so.cache, flags: O_RDONLY|O_CLOEXEC, mode: 0
16:45:06:268922  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                fstat                fd: 3,
statbuf: 0x7FFC5C237A80
16:45:06:268943  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                close                fd: 3
16:45:06:268971  a147ac5f855d  0      docker-entrypoi  1      /37055   1      /37055   0                cap_capable          cap:
CAP_SYS_ADMIN
```

```
#- run this command in the other terminal.
~$ docker container run -d nginx
9f247fe37131d5edbae4c1cf868cebedd0c066bba1518ab2259d0cd1d4442bdb
```

WHY?

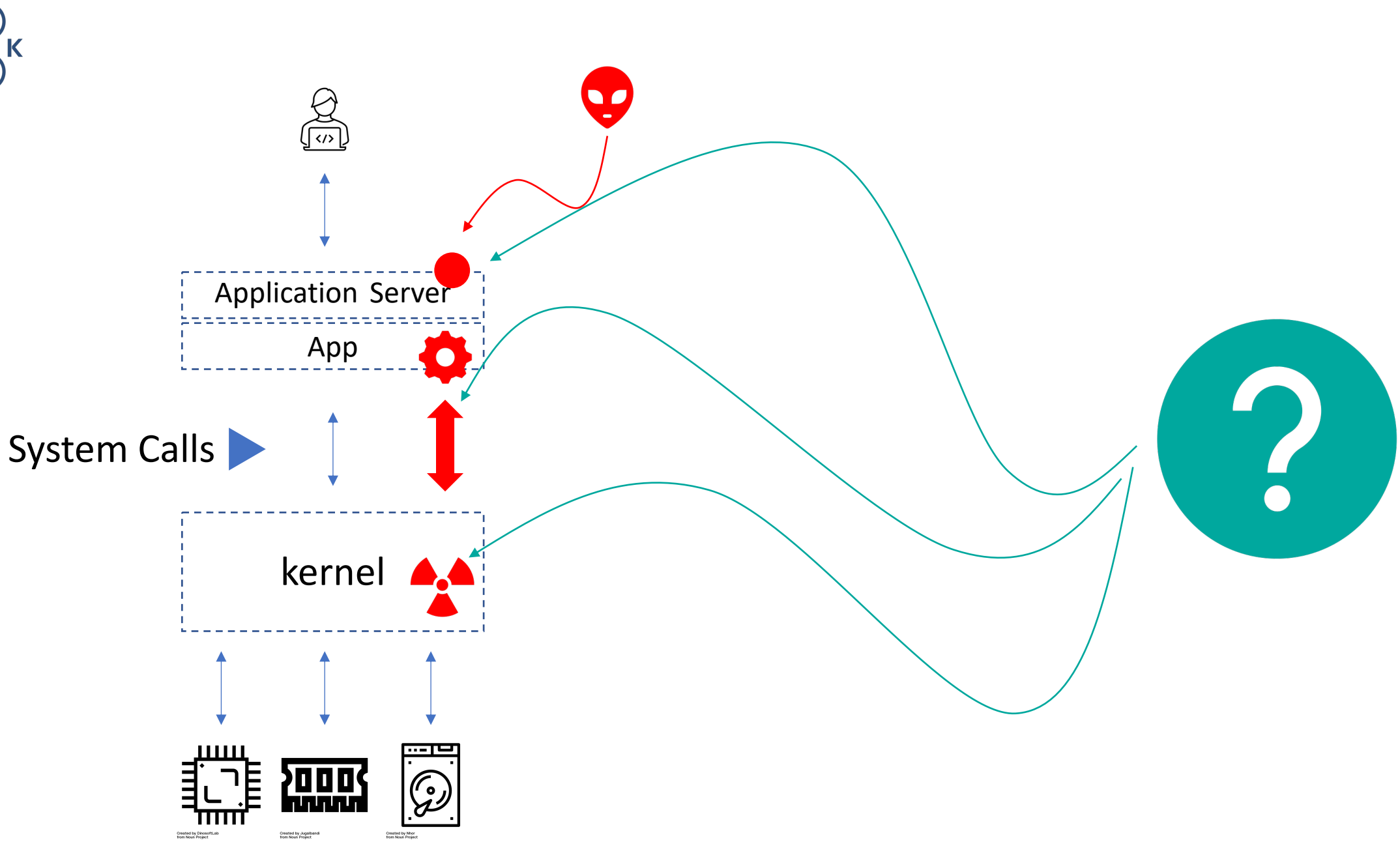


Application may use only 10 system calls

For example, Linux and OpenBSD each have over 300 different calls, NetBSD has close to 500, FreeBSD has over 500, Windows has close to 2000, divided between win32k (graphical) and ntdll (core) system calls.

https://en.wikipedia.org/wiki/System_call

[Memory by Jugalbandi, CPU by DinosoftLab, harddisk by Nhor] from the Noun Project



[Memory by Jugalbandi, CPU by DinosoftLab, harddisk by Nhor] from the Noun Project

secure computing mode

seccomp (short for secure computing mode) is a computer security facility in the Linux kernel. **seccomp allows a process to make a one-way transition into a "secure" state where it cannot make any system calls except exit(), sigreturn(), read() and write() to already-open file descriptors**. Should it attempt any other system calls, the kernel will terminate the process with SIGKILL or SIGSYS. In this sense, it does not virtualize the system's resources but isolates the process from them entirely.

https://en.wikipedia.org/wiki/Seccomp

seccomp stands for **secure computing mode** and has been a feature of the Linux kernel since version 2.6.12. It can be used to sandbox the privileges of a process, **restricting the calls it is able to make from userspace into the kernel.**

https://kubernetes.io/docs/tutorials/clusters/seccomp/

```
~$ docker run -d --name nginx nginx
4bb2e42effc8fc1907f73c453bb4b82b482017355bb9c0900facc79cfbd9938d
$ docker exec -it nginx bash
root@4bb2e42effc8:/# cat /proc/1/status |grep -i seccomp
Seccomp:        2
```

seccomp mode

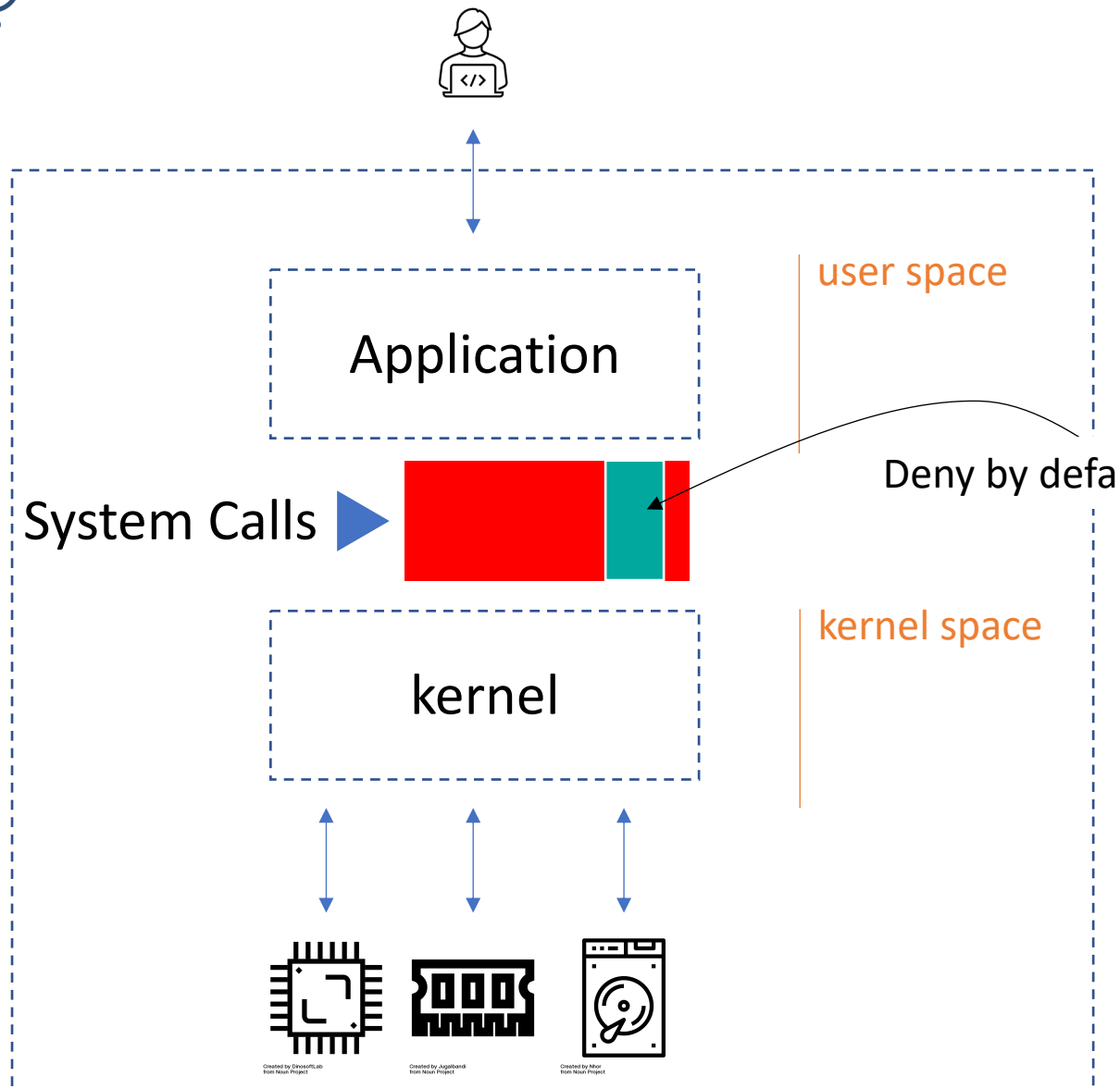
0 - Disable
1 - Restrict
2 - Filtered

Mode 1 is the original seccomp that is extremely restrictive and only allows four syscalls:

- read()
- write()
- exit()
- rt_sigreturn

Mode 2 is the newer one that involves a **userspace-created policy** being sent to the kernel. This policy defines the permitted syscalls and arguments along with the action to take in the case of a violation.

WHY?



seccomp profile

Deny by default and allow some syscalls

Allow by default and deny some syscalls

[Memory by Jugalbandi, CPU by DinosoftLab, harddisk by Nhor] from the Noun Project

Action in seccomp profile

SCMP_ACT_KILL The thread will be terminated by the kernel with SIGSYS when it calls a syscall that does not match any of the configured seccomp filter rules. The thread will not be able to catch the signal.

SCMP_ACT_KILL_PROCESS The entire process will be terminated by the kernel with SIGSYS when it calls a syscall that does not match any of the configured seccomp filter rules.

SCMP_ACT_TRAP The thread will be sent a SIGSYS signal when it calls a syscall that does not match any of the configured seccomp filter rules. It may catch this and change its behavior accordingly. When using SA_SIGINFO with sigaction(2), si_code will be set to SYS_SECCOMP, si_syscall will be set to the syscall that failed the rules, and si_arch will be set to the AUDIT_ARCH for the active ABI.

SCMP_ACT_ERRNO The thread will receive a return value of errno when it calls a syscall that does not match any of the configured seccomp filter rules.

SCMP_ACT_TRACE If the thread is being traced and the tracing process specified the PTRACE_O_TRACESECCOMP option in the call to ptrace(2), the tracing process will be notified, via PTRACE_EVENT_SECCOMP, and the value provided in msg_num can be retrieved using the PTRACE_GETEVENTMSG option.

SCMP_ACT_LOG The seccomp filter will have no effect on the thread calling the syscall if it does not match any of the configured seccomp filter rules but the syscall will be logged.

SCMP_ACT_ALLOW The seccomp filter will have no effect on the thread calling the syscall if it does not match any of the configured seccomp filter rules.

seccomp profile example

```
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "architectures": [
        "SCMP_ARCH_X86_64"
    ],
    "syscalls": [
        {
            "names": [
               "chmod"
            ],
            "action": "SCMP_ACT_ERRNO"
        }
    ]
}
```

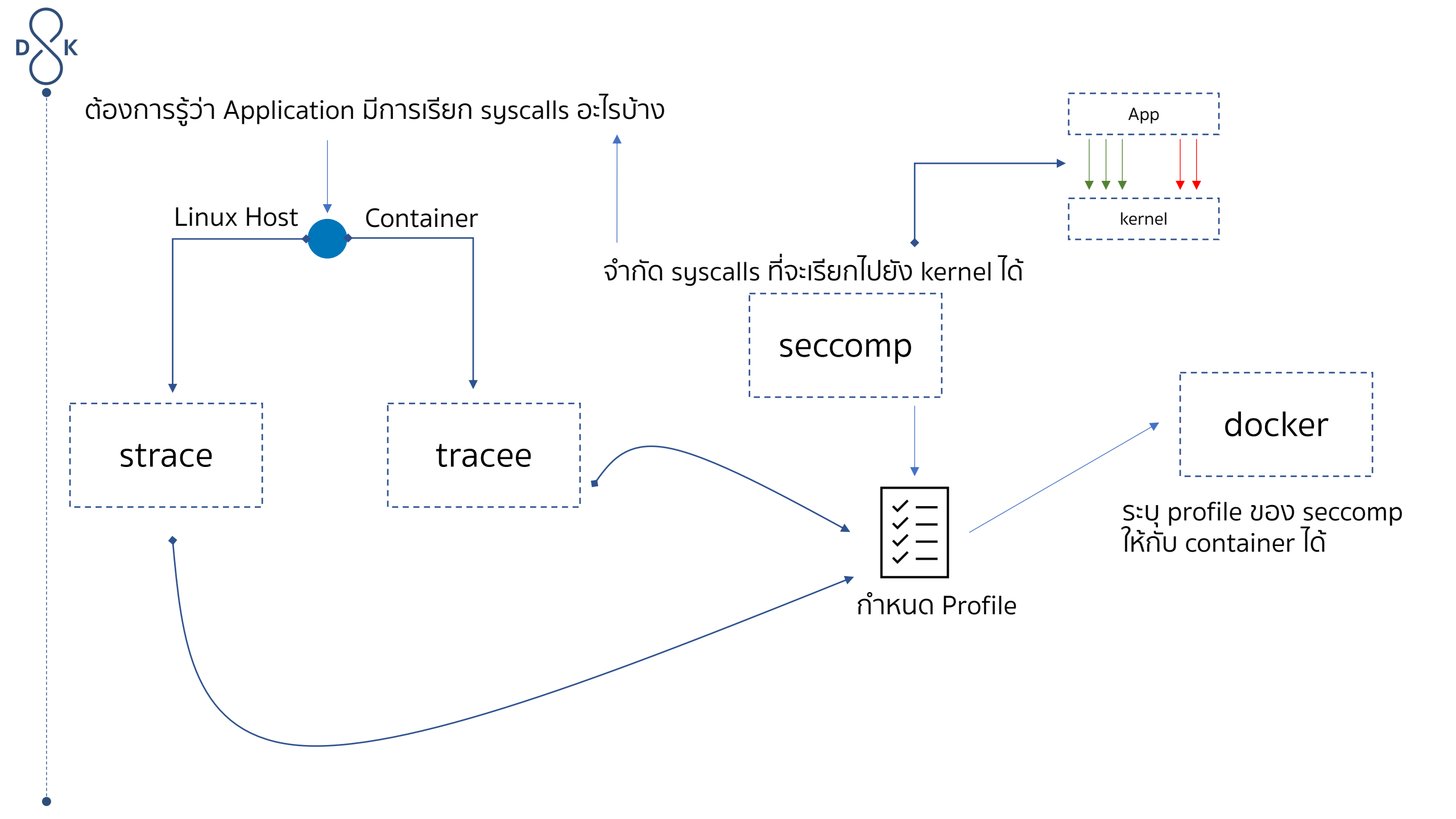
```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64"
    ],
    "syscalls": [
        {
            "names": [
                "chmod"
            ],
            "action": "SCMP_ACT_ALLOW"
        }
    ]
}
```

```
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "architectures": [
        "SCMP_ARCH_X86_64"
    ],
    "syscalls": [
        {
            "names": [
                "chmod"
            ],
            "action": "SCMP_ACT_ERRNO"
        }
    ]
}
```

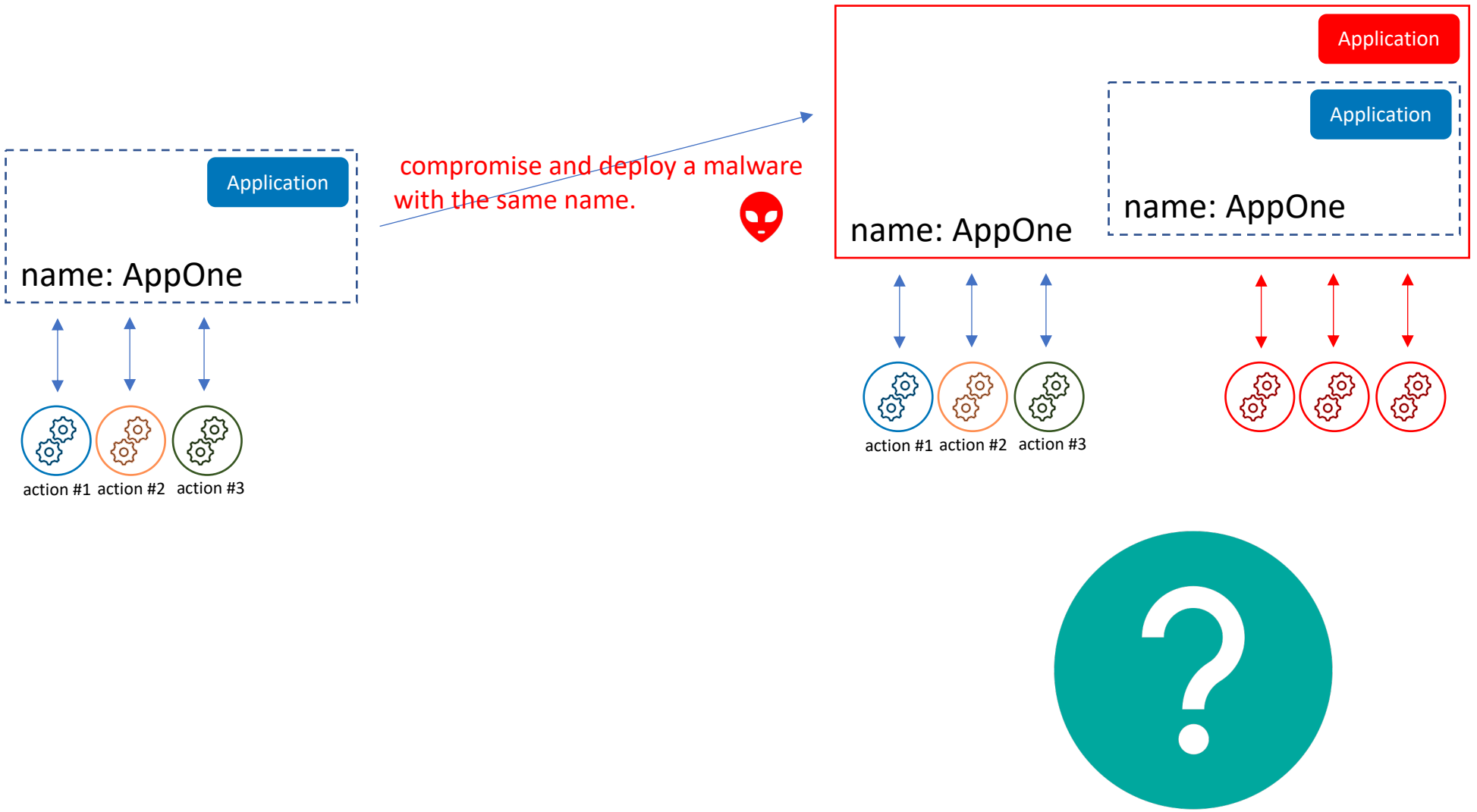
sc.json

```
#- running with default seccomp from Docker
~$ docker container run -it  damrongsak/curl sh -c "chmod 777 /etc/passwd;ls -la /etc/passwd"
-rwxrwxrwx    1 root     root          1172 Jun 19  2020 /etc/passwd
```

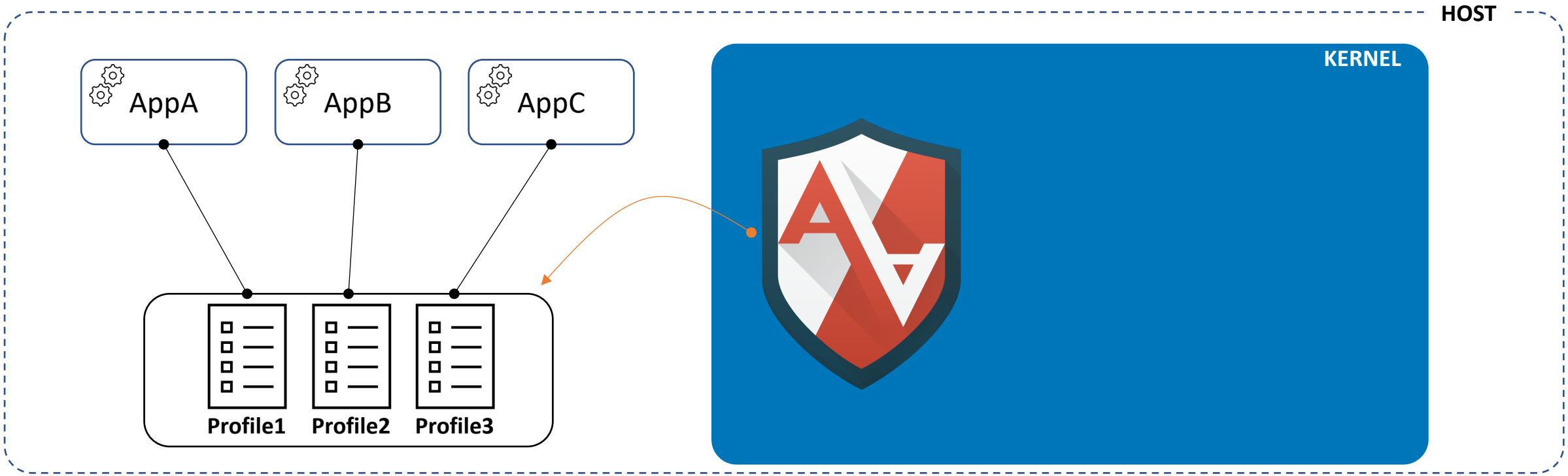
```
#- running with specific seccomp profile
~$ docker container run -it --security-opt seccomp=sc.json damrongsak/curl  \
   sh -c "chmod 777 /etc/passwd;ls -la /etc/passwd"
chmod: /etc/passwd: Operation not permitted
-rw-r--r--    1 root     root          1172 Jun 19  2020 /etc/passwd
```

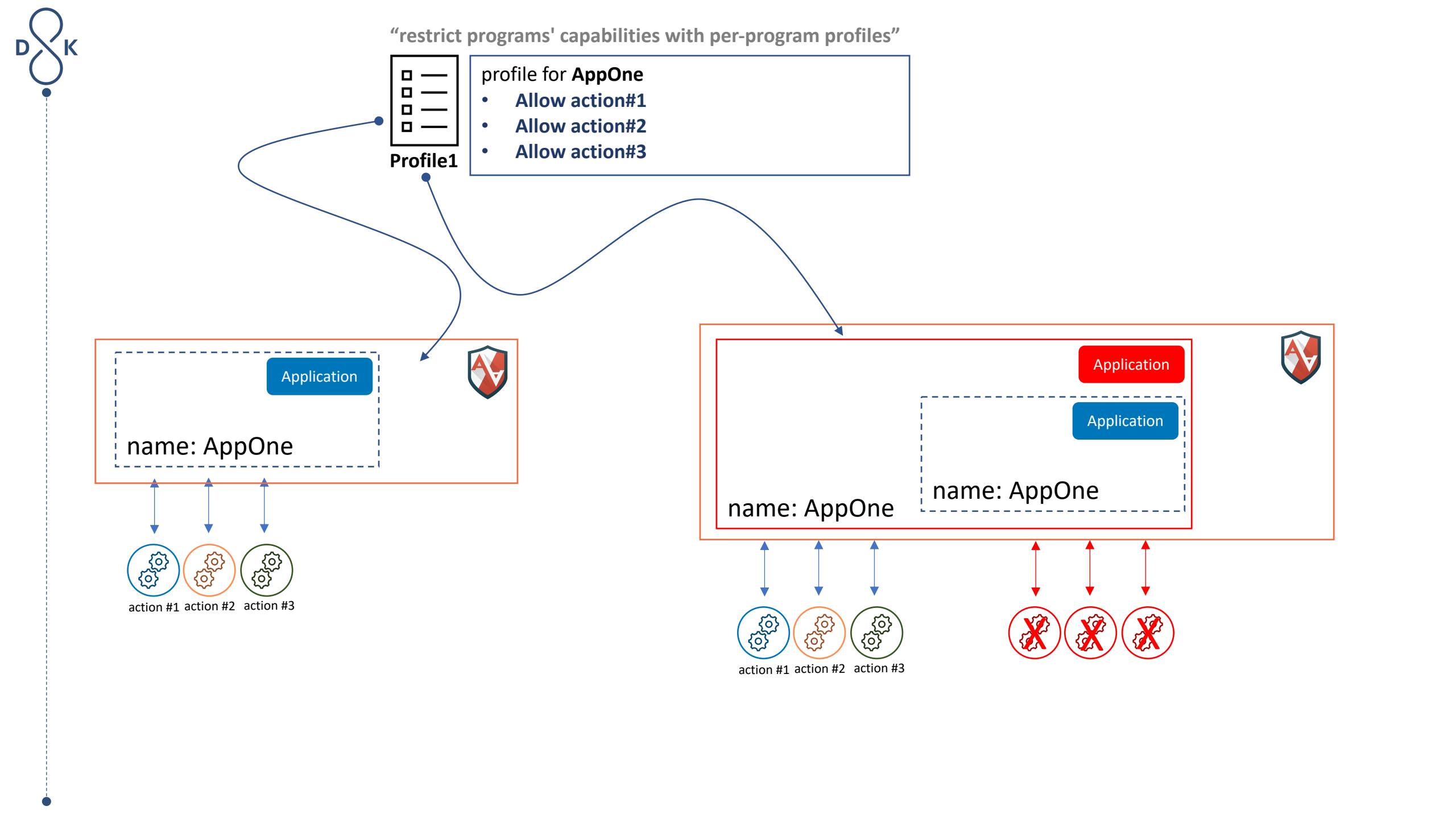

ต้องการรู้ว่า Application มีการเรียก syscalls อะไรบ้าง
Linux Host
Container
App
kernel
จำกัด syscalls ที่จะเรียกไปยัง kernel ได้
seccomp
strace
tracee
docker
ระบุ profile ของ seccomp
ให้กับ container ได้
กำหนด Profile

AppArmor


Application
name: AppOne
action #1
action #2
action #3
compromise and deploy a malware
with the same name.
Application
Application
name: AppOne
name: AppOne
action #1
action #2
action #3

AppArmor ("Application Armor") is a **Linux kernel security module** that allows the system administrator to **restrict programs' capabilities** **with per-program profiles**. Profiles can allow capabilities like network access, raw socket access, and the permission to read, write, or execute files on matching paths.




"restrict programs' capabilities with per-program profiles"
profile for AppOne
Allow action#1
Allow action#2
Allow action#3
Profile1
Application
name: AppOne
action #1
action #2
action #3
Application
Application
name: AppOne
name: AppOne
action #1
action #2
action #3

AppArmor Profile Example

```
:~$ sudo cat /etc/apparmor.d/usr.local.bin.appone
#include <tunables/global>

/usr/local/bin/appone {
  #include <abstractions/base>
  #include <abstractions/bash>

  /dev/tty rw,
  /usr/bin/bash ix,
  /usr/bin/touch mrix,
  /usr/local/bin/appone r,
  owner /tmp/d8k.io w,

}
```

Execute Modes : Inherit Execute Mode (ix) prevents the normal AppArmor domain transition on execve(2) when the profiled program executes the named program. Instead, the executed resource inherits the current profile.

Execute Modes : Allow Executable Mapping (m) This mode allows a file to be mapped into memory using mmap(2)'s PROT_EXEC flag. This flag marks the pages executable. It is used on some architectures to provide non executable data pages, which can complicate exploit attempts. AppArmor uses this mode to limit which files a well-behaved program.

https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-apparmor-profiles.html

```
$ cat /usr/local/bin/appone
#!/bin/bash

touch /tmp/d8k.io
```

/etc/apparmor.d/usr.local.bin.appone	/usr/local/bin/appone	execute /usr/local/bin/appone

`#include <tunables/global>`
 	
`/usr/local/bin/appone {`
`  #include <abstractions/base>`
`  #include <abstractions/bash>`
 	
`  /dev/tty rw,`
`  /usr/bin/bash ix,`
`  /usr/bin/touch mrix,`
`  /usr/local/bin/appone r,`
`  owner /tmp/d8k.io w,`
 	
`}`
	`#!/bin/bash`
	`touch /tmp/d8k.io`
		`~$ /usr/local/bin/appone`
		`~$ echo $?`
		`0`

	`#!/bin/bash`
	`touch /tmp/d8k.io`
	`ls /tmp/d8k.io`
		`$ /usr/local/bin/appone`
		`/tmp/d8k.io`

	`#!/bin/bash`
	`touch /tmp/d8k.io`
	`ls /tmp/*.io`
		`$ /usr/local/bin/appone`
		`ls: cannot access '/tmp/*.io': No such file or directory`

	`#!/bin/bash`
	`touch /tmp/d8k.io`
	`touch /tmp/home.io`
		`$ /usr/local/bin/appone`
		`touch: cannot touch '/tmp/home.io': Permission denied`

`#include <tunables/global>`
`/usr/local/bin/appone {`
`  #include <abstractions/base>`
`  #include <abstractions/bash>`
 	
`  /dev/tty rw,`
`  /usr/bin/bash ix,`
`  /usr/bin/touch mrix,`
`  /usr/local/bin/appone r,`
`  owner /tmp/d8k.io w,`
`  owner /tmp/home.io w,`
`}`
	`#!/bin/bash`
	`touch /tmp/d8k.io`
	`touch /tmp/home.io`
		`~$ /usr/local/bin/appone`
		`~$ echo $?`
		`0`

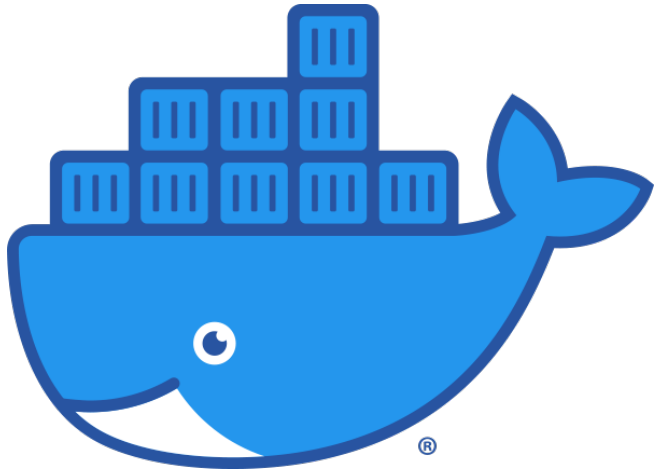
How to apply AppArmor Profile ?

```
/etc/apparmor.d/
├── abstractions
│   ├── X
│   ├── apache2-common
│   ├── apparmor_api
│   │   ├── change_profile
│   │   ├── examine
│   │   ├── find_mountpoint
│   │   ├── introspect
│   │   └── is_enabled
[...]
├── lsb_release
├── nvidia_modprobe
├── sbin.dhclient
├── usr.bin.man
├── usr.lib.snapd.snap-confine.real
├── usr.sbin.rsyslogd
├── home.drs.bin.myprog
└── usr.sbin.tcpdump
[...]
```

1. Copy an AppArmor profile to /etc/apparmor.d
2. Enforce Profile

```
# apparmor_parser /etc/apparmor.d/home.drs.bin.myprog
```

```
$ sudo aa-status
apparmor module is loaded.
29 profiles are loaded.
29 profiles are in enforce mode.
   /snap/snapd/12883/usr/lib/snapd/snap-confine
   /snap/snapd/12883/usr/lib/snapd/snap-confine//mount-
namespace-capture-helper
   /usr/bin/man
   /usr/lib/NetworkManager/nm-dhcp-client.action
   /usr/lib/NetworkManager/nm-dhcp-helper
   /usr/lib/connman/scripts/dhclient-script
   /usr/lib/snapd/snap-confine
   /home/drs/bin/myprog
[…]
```

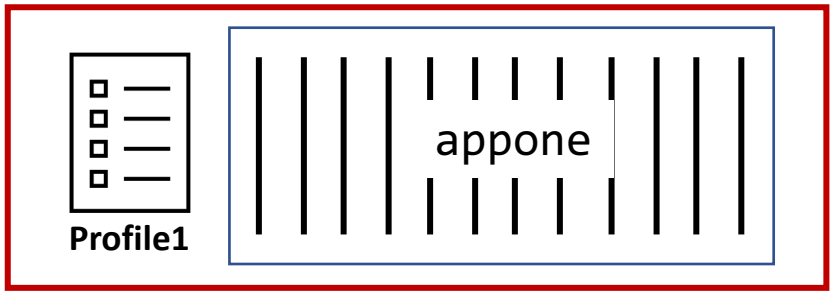


Docker automatically generates and loads a **default profile for containers named *docker-default***. The Docker **binary generates this profile in tmpfs** and then loads it into the kernel.

The profile **is used on containers**, not on the Docker Daemon.

HOST

```
$ docker conatiner run --security-opt “apparmor=Profile1” -d myappone
```



```
$ sudo cat /etc/apparmor.d/usr.local.bin.appone


#include <tunables/global>

/usr/local/bin/appone {
  #include <abstractions/base>
  #include <abstractions/bash>

  /dev/tty rw,
  /usr/bin/bash ix,
  /usr/bin/sleep mrix,
  /usr/bin/touch mrix,
  /usr/local/bin/appone r,
  owner /tmp/d8k.io w,

}
```

```
$ cat appone
#!/bin/bash
echo "Creating /tmp/d8k.io"
touch /tmp/d8k.io
sleep 10000
```

```
$ cat Dockerfile
FROM ubuntu

copy appone /usr/local/bin/appone
ENTRYPOINT ["/usr/local/bin/appone"]
```

build container image

```
$ docker image build -t myappone .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM ubuntu
[...]
Successfully tagged myappone:latest
```

load profile into kernel

```
$ sudo apparmor_parser /etc/apparmor.d/usr.local.bin.appone
$ sudo aa-status |grep appone
   /usr/local/bin/appone
```

The profile **is used on containers**, not on the Docker Daemon.

```
$ docker container run --security-opt "apparmor=/usr/local/bin/appone" \
  --name w-apparmor -d myappone
f03b3870bbbc0c7c6ba0a3e27fbbf300e87fb156d268e3bae176030035f423ec

$ docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS     NAMES
f03b3870bbbc   myappone   "/usr/local/bin/appo…"   55 seconds ago   Up 54 seconds             w-apparmor

$ docker container exec w-apparmor ls -la /tmp/d8k.io
-rw-r--r-- 1 0 0 0 Jan  8 20:53 /tmp/d8k.io
$ docker container exec -it w-apparmor bash
bash: /usr/bin/groups: Permission denied
I have no name!@f03b3870bbbc:/# date
bash: /usr/bin/date: Permission denied
I have no name!@f03b3870bbbc:/# touch drs.txt
touch: cannot touch 'drs.txt': Permission denied
I have no name!@f03b3870bbbc:/# ls
ls: cannot open directory '.': Permission denied
```

Linux Capabilities

Linux Capabilities

For the purpose of performing permission checks, **traditional UNIX implementations** distinguish two categories of processes

- **Privileged processes** (whose effective user ID is 0, referred to as superuser or root) **bypass all kernel permission checks**.
- **Unprivileged processes** (whose effective UID is nonzero) **are subject to full permission checking** based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux **divides the privileges** traditionally associated with superuser **into distinct units, known as capabilities**, which can be independently enabled and disabled. Capabilities are a per-thread attribute.

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

Capabilities list :: example

CAP_AUDIT_CONTROL (since Linux 2.6.11) Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.
CAP_AUDIT_READ (since Linux 3.16) Allow reading the audit log via a multicast netlink socket.
CAP_AUDIT_WRITE (since Linux 2.6.11) Write records to kernel auditing log.
CAP_CHOWN Make arbitrary changes to file UIDs and GIDs
CAP_DAC_OVERRIDE Bypass file read, write, and execute permission checks.
CAP_DAC_READ_SEARCH Bypass file read permission checks and directory read and execute permission checks
CAP_KILL Bypass permission checks for sending signals
CAP_NET_ADMIN Perform various network-related operations
CAP_NET_BIND_SERVICE Bind a socket to Internet domain privileged ports
CAP_SYS_ADMIN Perform a range of system administration operations.
CAP_SYS_TIME Set system clock; set real-time (hardware) clock.
CAP_SYSLOG (since Linux 2.6.37) Perform privileged syslog operations.
CAP_SYS_MODULE Load and unload kernel modules
CAP_SETGID Make arbitrary manipulations of process GIDs and supplementary GID list;
CAP_SETUID Make arbitrary manipulations of process UIDs

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

```
$ getcap /usr/bin/ping
/usr/bin/ping = cap_net_raw+ep
$ getcap /usr/bin/tar
$
```

```
$ ls -la /etc/shadow
-rw-r----- 1 root shadow 1051 Jan  4 12:48 /etc/shadow
$ id
uid=1000(drs) gid=1000(drs)
groups=1000(drs),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plug
dev),117(netdev),118(lxd),119(docker)
$ tar -cvf shadow.tar /etc/shadow
tar: Removing leading `/' from member names
tar: /etc/shadow: Cannot open: Permission denied
tar: Exiting with failure status due to previous errors
```

```
$ sudo setcap cap_dac_read_search+ep /usr/bin/tar
$ getcap /usr/bin/tar
/usr/bin/tar = cap_dac_read_search+ep
$ tar -cvf shadow.tar /etc/shadow
tar: Removing leading `/' from member names
/etc/shadow
$ ls -la shadow.tar
-rw-rw-r-- 1 drs drs 10240 Jan  9 06:22 shadow.tar
```

Docker has a default list of capabilities that are kept.

The lists the Linux capability options which are allowed by default.
AUDIT_WRITE, CHOWN, DAC_OVERRIDE, FOWNER, FSETID, KILL, MKNOD, NET_BIND_SERVICE, NET_RAW, SETFCAP, SETGID, SETPCAP, SETUID, SYS_CHROOT

The capabilities which are not granted by default.
AUDIT_CONTROL, AUDIT_READ, BLOCK_SUSPEND, BPF, CHECKPOINT_RESTORE, DAC_READ_SEARCH, IPC_LOCK, IPC_OWNER, LEASE, LINUX_IMMUTABLE, MAC_ADMIN, MAC_OVERRIDE, NET_ADMIN, NET_BROADCAST, PERFMON, SYS_ADMIN, SYS_BOOT, SYS_MODULE, SYS_NICE, SYS_PACCT, SYS_PTRACE, SYS_RAWIO, SYS_RESOURCE, SYS_TIME, SYS_TTY_CONFIG, SYSLOG, WAKE_ALARM

https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

docker container run option: Linux Capabilities

Option	Description
--cap-add	Add Linux capabilities
--cap-drop	Drop Linux capabilities

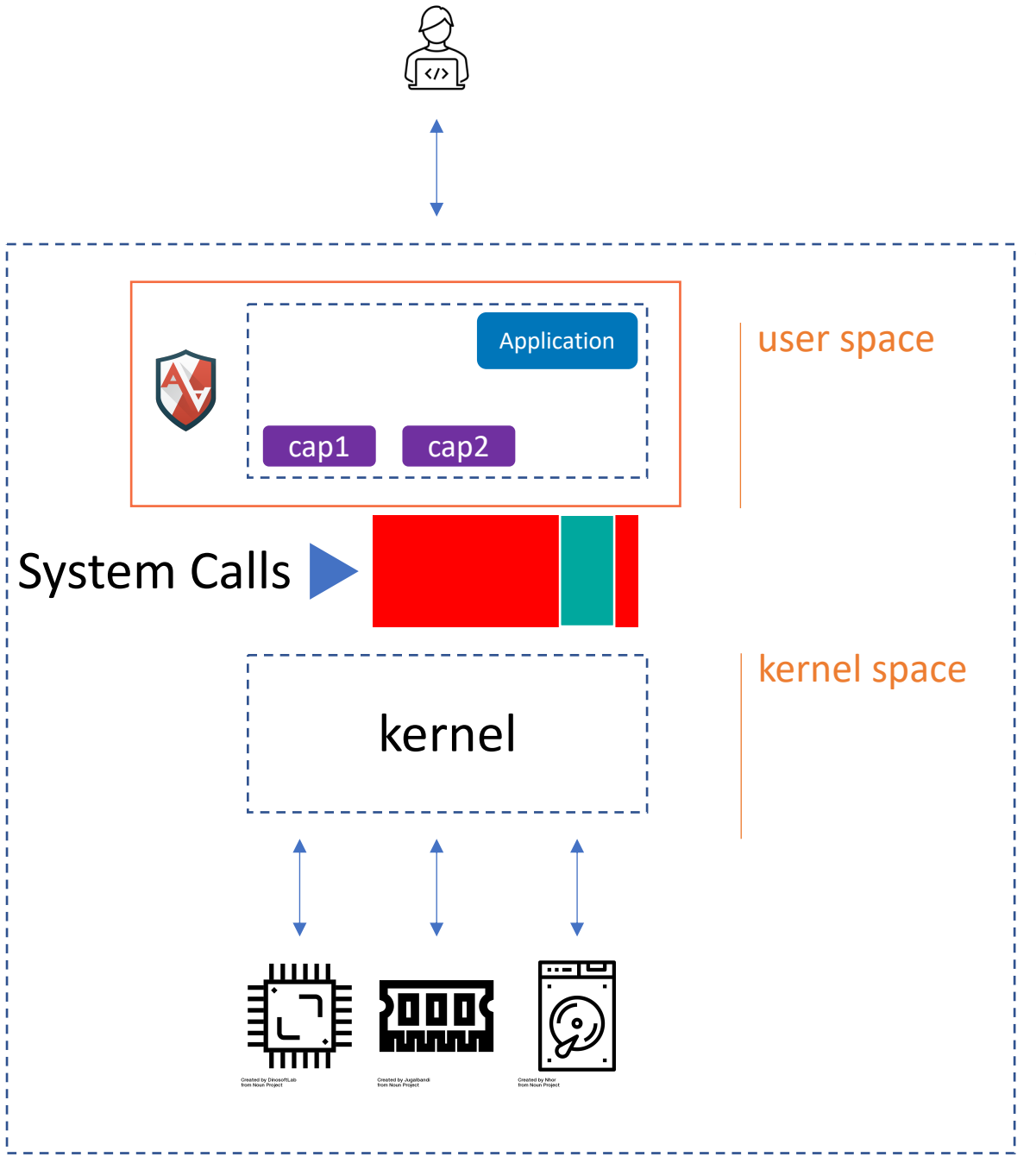
```
$ docker run -it --rm  centos ip link add dummy0 type dummy
RTNETLINK answers: Operation not permitted

$ docker run -it --rm --cap-add=NET_ADMIN   centos ip link add dummy0 type dummy
$
```

```
$ docker container run -it  ubuntu  chown nobody /
$
$ docker container run -it --cap-drop CHOWN ubuntu  chown nobody /
chown: changing ownership of '/': Operation not permitted

$ docker container run -it --cap-drop ALL ubuntu  chown nobody /
chown: changing ownership of '/': Operation not permitted

$ docker container run -it --cap-drop ALL --cap-add CHOWN ubuntu  chown nobody /
$
```


Application
cap1
cap2
user space
System Calls
kernel
kernel space

เพื่อ ?

RISK

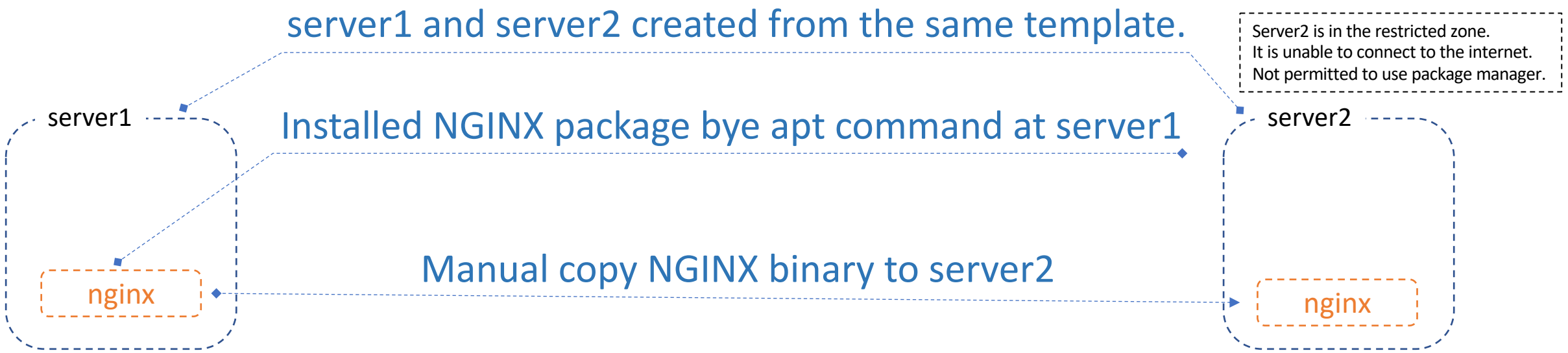
Container Technology

Servers icons created by juicy_fish - Flaticon| Container icons created by Freepik - Flaticon

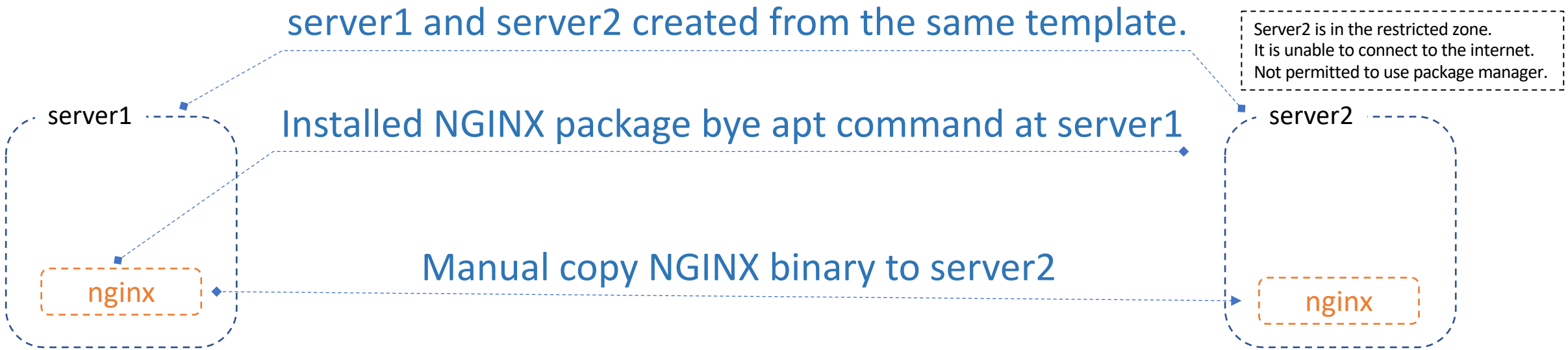
```
drs@server200:~$ docker container run -d nginx
e3c2a803e4e3a1d86687aeb253fdc5fa433f7bf4af179579f094aed5ce3e9475
```

Create a Container == Running the processes in a host.

```
4374 ?        Sl     0:00 /usr/bin/containerd-shim-runc-v2 -namespace moby ....
4397 ?        Ss     0:00  \_ nginx: master process nginx -g daemon off;
4468 ?        S      0:00      \_ nginx: worker process
4469 ?        S      0:00      \_ nginx: worker process
4470 ?        S      0:00      \_ nginx: worker process
4471 ?        S      0:00      \_ nginx: worker process
```

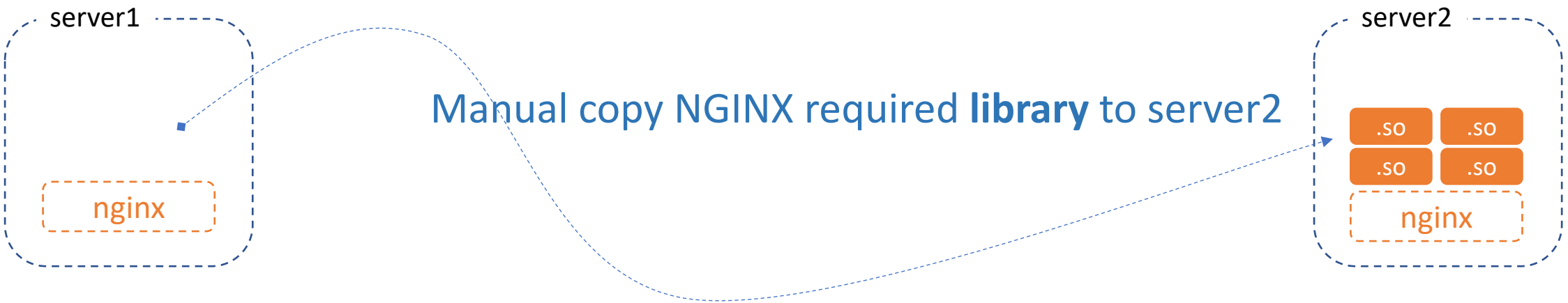


? Can NGINX at server2 work?

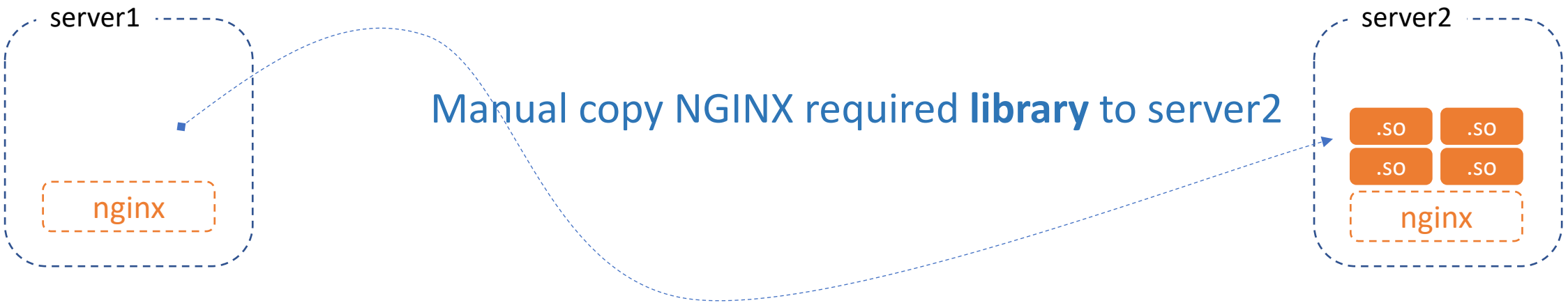


/usr/sbin/nginx required some libraries to execute.

```
]# ldd /usr/sbin/nginx
        libdl.so.2 => /lib64/libdl.so.2 (0x00007f3acb008000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f3acadec000)
        libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007f3acabb5000)
        libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f3aca953000)
        libssl.so.10 => /lib64/libssl.so.10 (0x00007f3aca6e1000)
        libcrypto.so.10 => /lib64/libcrypto.so.10 (0x00007f3aca27e000)
[...]
```

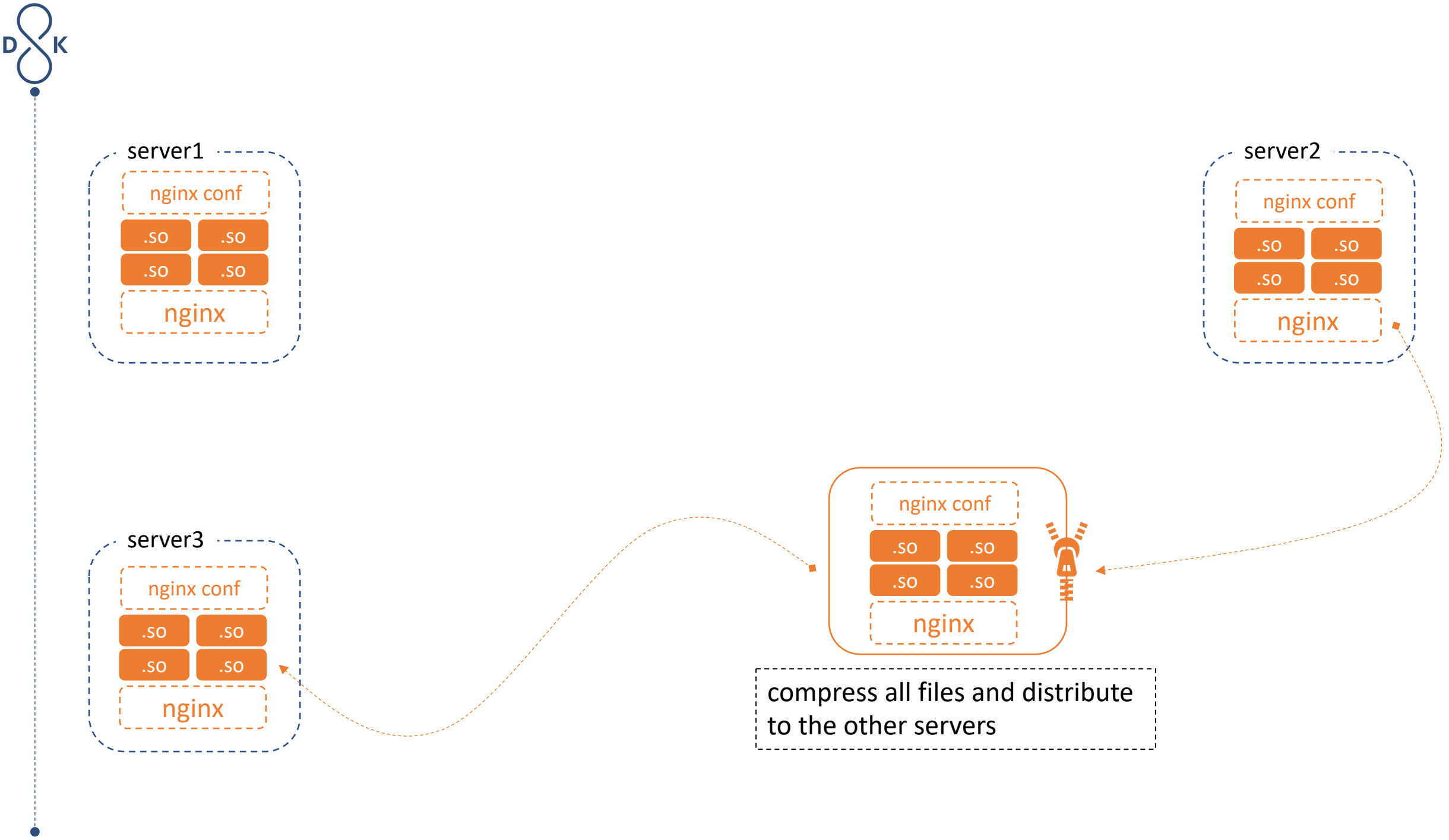


Can NGINX at server2 work?



```
]# /usr/sbin/nginx
nginx: [emerg] open() "/etc/nginx/nginx.conf" failed (2: No
such file or directory)
```

Can NGINX at server2 work?


server1
nginx conf
.so
.so
.so
.so
nginx
server2
nginx conf
.so
.so
.so
.so
nginx
server3
nginx conf
.so
.so
.so
.so
nginx
nginx conf
.so
.so
.so
.so
nginx
compress all files and distribute to the other servers

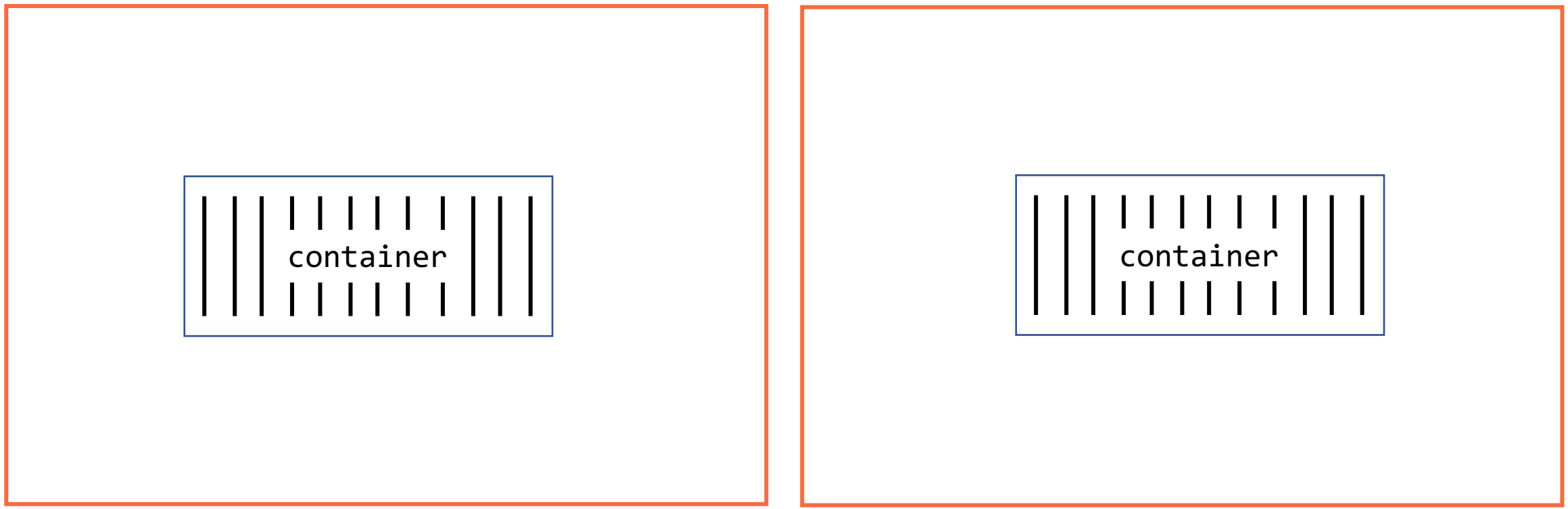
```
# cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.2 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.2 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

execute in host

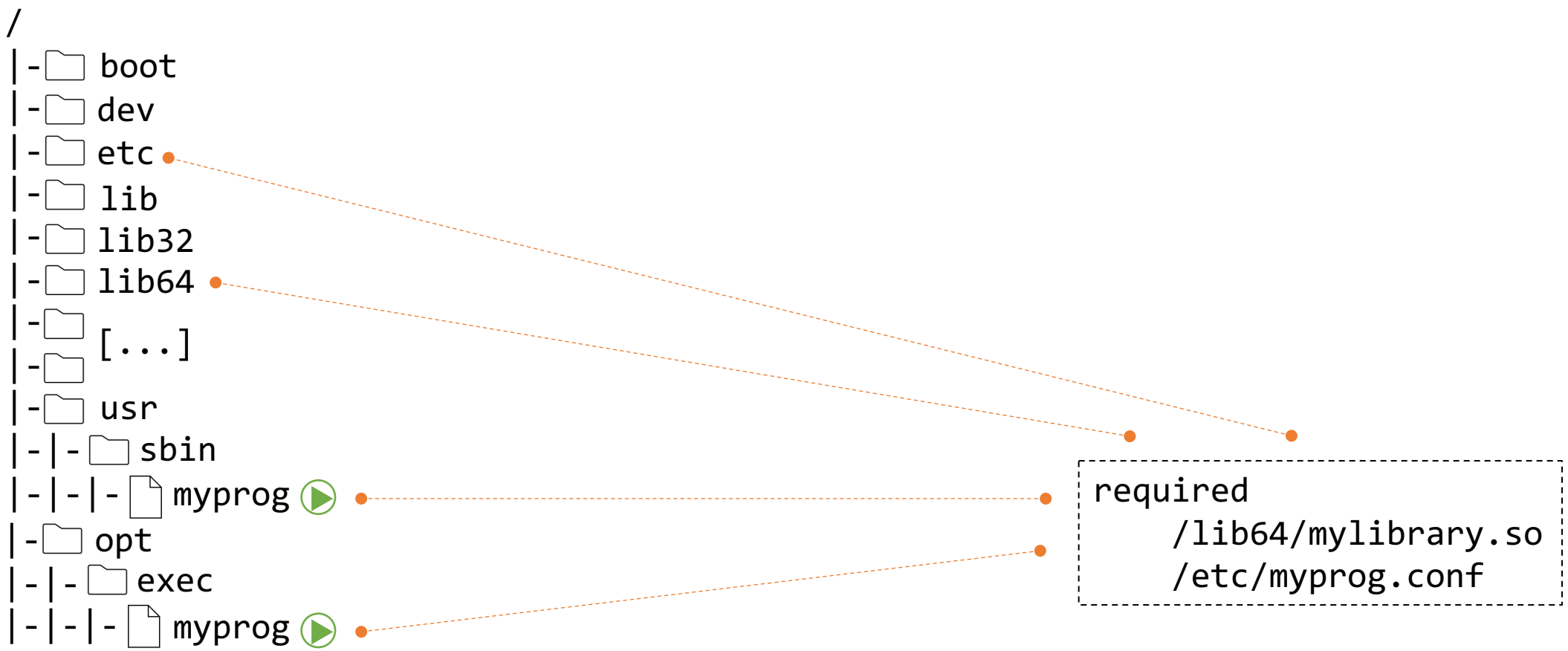
```
# docker container run centos:8 cat /etc/os-release
NAME="CentOS Linux"
VERSION="8"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="8"
PLATFORM_ID="platform:el8"
PRETTY_NAME="CentOS Linux 8"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:8"
HOME_URL="https://centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-8"
CENTOS_MANTISBT_PROJECT_VERSION="8"
```

execute inside container

I S O L A T I O N

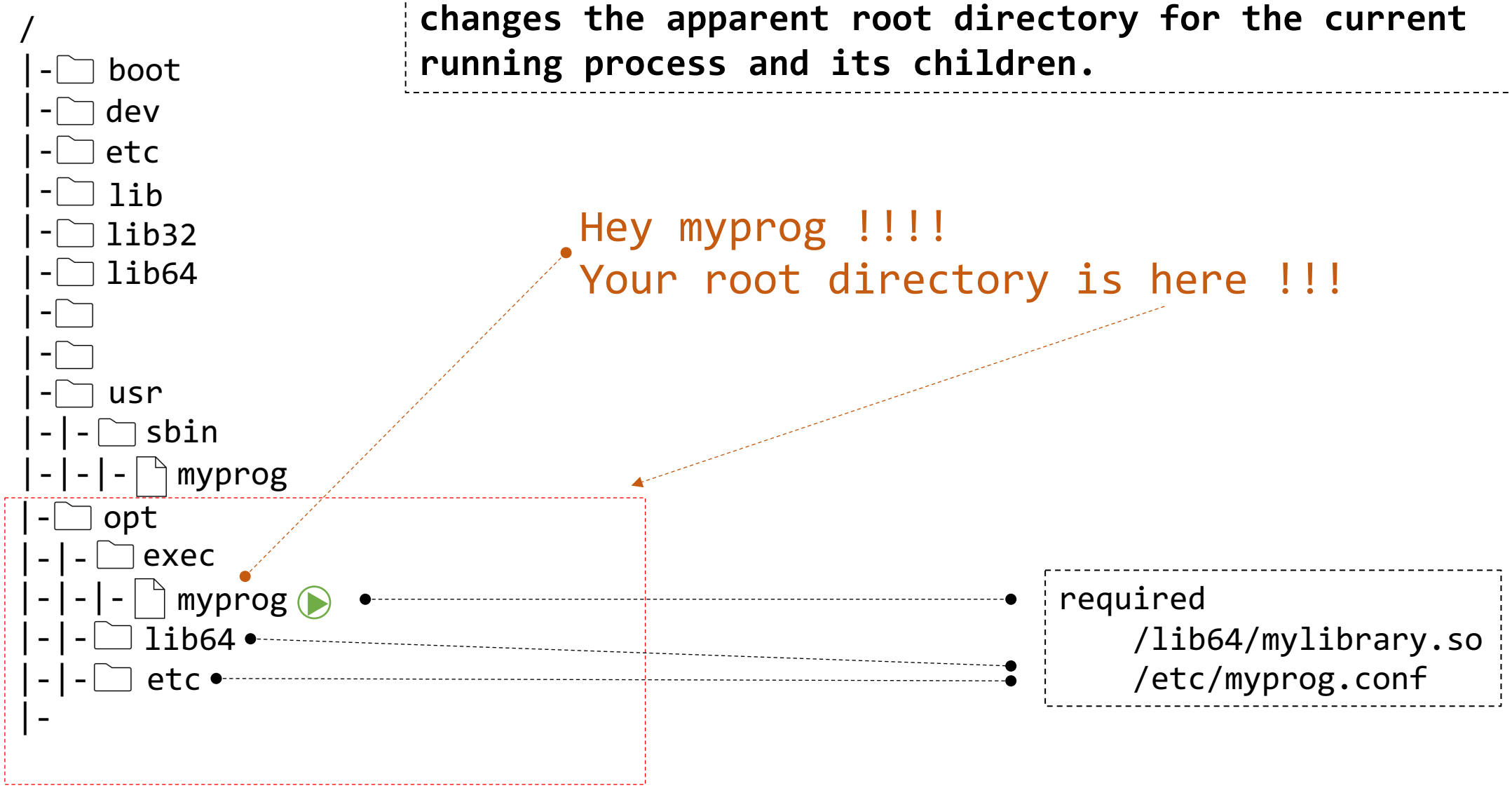


copy ./myprog ไปเรียกอีก Directory



chroot - change root

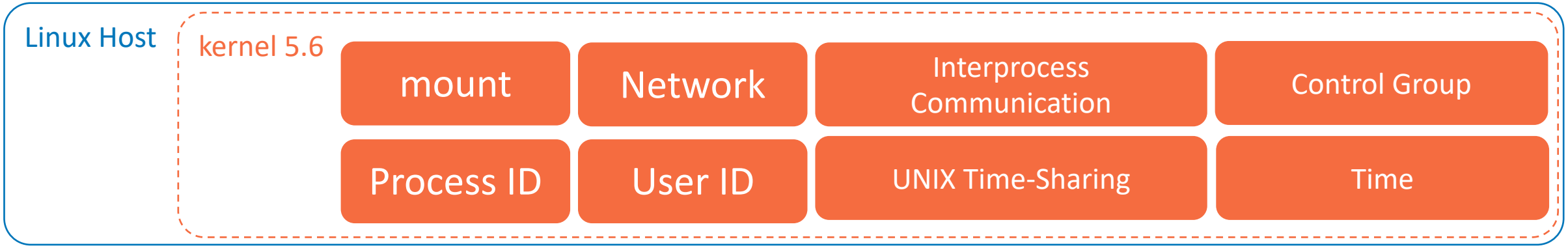
A chroot on Unix operating systems is an operation that **changes the apparent root directory for the current running process and its children.**



“Docker **uses a technology** called namespaces **to provide the isolated workspace** called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.”

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



```
$ docker container run -d nginx
```

```
$ sudo lsns
```

```
$ sudo lsns
        NS TYPE   NPROCS   PID USER             COMMAND
4026531835 cgroup    200     1 root             /sbin/init
4026531836 pid       195     1 root             /sbin/init
[...]
4026532554 mnt         5  2070 root             nginx: master process nginx -g daemon off;
4026532555 uts         5  2070 root             nginx: master process nginx -g daemon off;
4026532556 ipc         5  2070 root             nginx: master process nginx -g daemon off;
4026532557 pid         5  2070 root             nginx: master process nginx -g daemon off;
4026532559 net         5  2070 root             nginx: master process nginx -g daemon off;
4026532659 mnt         1   674 root             /usr/sbin/irqbalance --foreground
4026532660 mnt         1   679 root             /lib/systemd/systemd-logind
4026532661 uts         1   679 root             /lib/systemd/systemd-logind
```

<DEV>
MOUNTAIN
TECH FESTIVAL

[Once in A Lifetime]