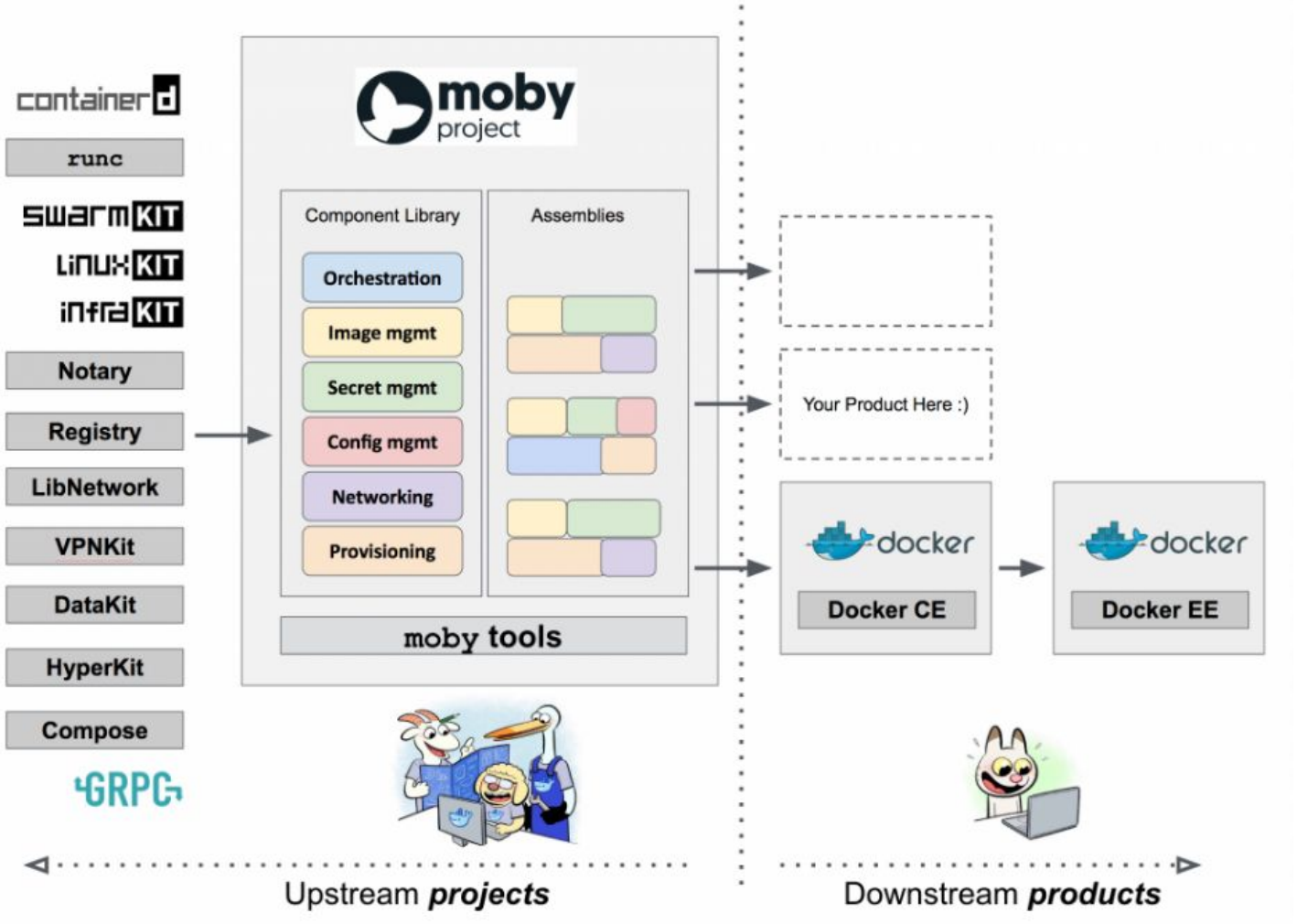
The Moby Project

...and other container related things

Tycho Andersen
tycho@docker.com

Moby project

- Announced at DockerCon US 2017
- Separation of monolith


containerd
runc
swarmKIT
LinuxKIT
infraKIT
Notary
Registry
LibNetwork
VPNKit
DataKit
HyperKit
Compose
GRPC
moby project
Component Library
Orchestration
Image mgmt
Secret mgmt
Config mgmt
Networking
Provisioning
Assemblies
moby tools
Your Product Here :)
docker
Docker CE
docker
Docker EE
Upstream projects
Downstream products

docker

Project vs Product

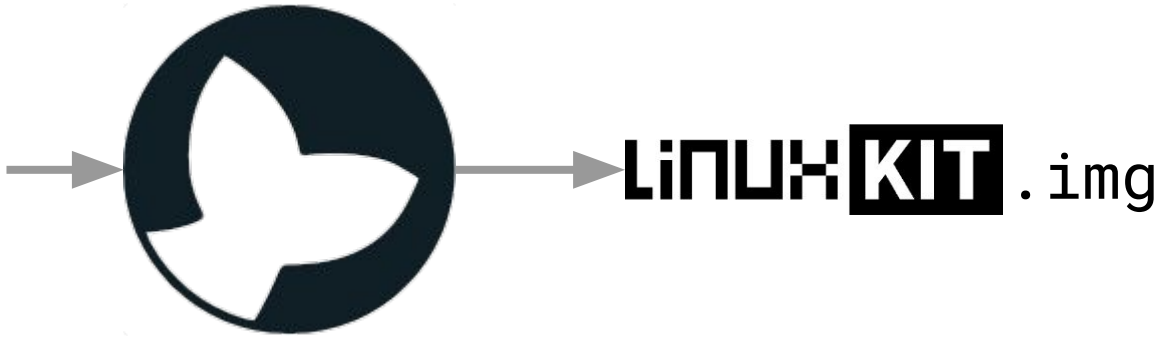
- Moby is the Project used to build Docker the product
- Moby is both the umbrella Project, and the tool to combine projects

Example

```
kernel:
  image: linuxkit/kernel:4.9.54
  cmdline: "console=tty0 console=ttyS0 console=ttyAMA0"
init:
  - linuxkit/init:6b3755e47f00d6027321d3fca99a19af6504be75
  - linuxkit/runc:52f92cb577879ce4cfe4e89be2d63af82523fc92
  - linuxkit/containerd:ed8e8f92e24dd4b94260cf147594ae3fd13a2182
  - linuxkit/ca-certificates:ea3c4c120f929f4f07ac8535d75933365b
...
trust:
  org:
    - linuxkit
    - library
```

Example

```
kernel:
  image: linuxkit/kernel:4.9.54
  cmdline: "console=tty0 console=ttyS0 console=ttyAMA0"
init:
  - linuxkit/init:6b3755e47f00d6027321d3fca99a19af6504be75
  - linuxkit/runc:52f92cb577879ce4cfe4e89be2d63af82523fc92
  - linuxkit/containerd:ed8e8f92e24dd4b94260cf147594ae3fd13a2182
  - linuxkit/ca-certificates:ea3c4c120f929f4f07ac8535d75933365b
...
trust:
  org:
    - linuxkit
    - library
```

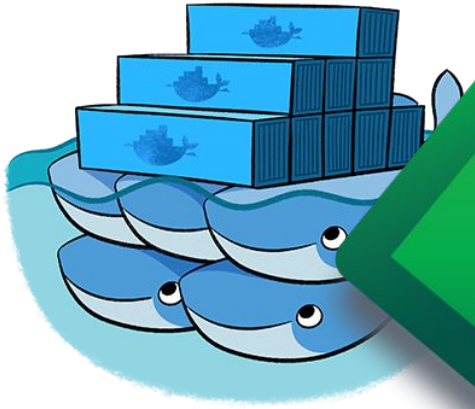


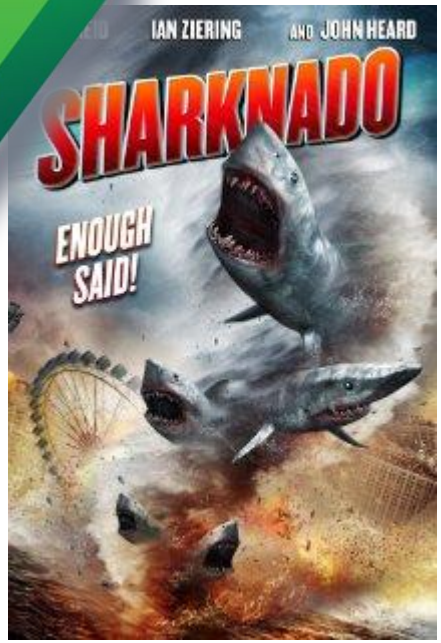
docker

LinuxKit

docker

Who should use Project Mob





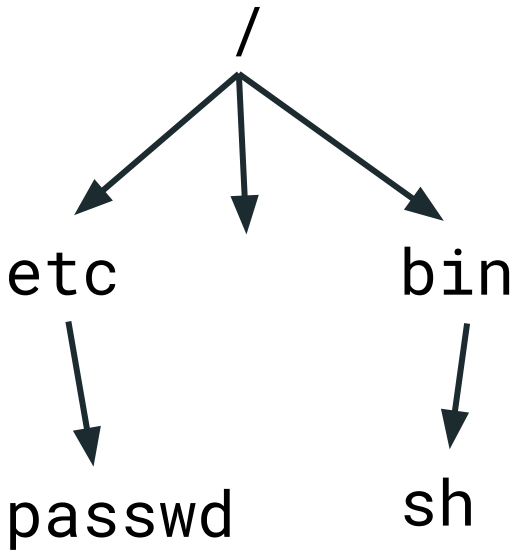
docker

Who should use Kubernetes by?

Who should use Project Moby?

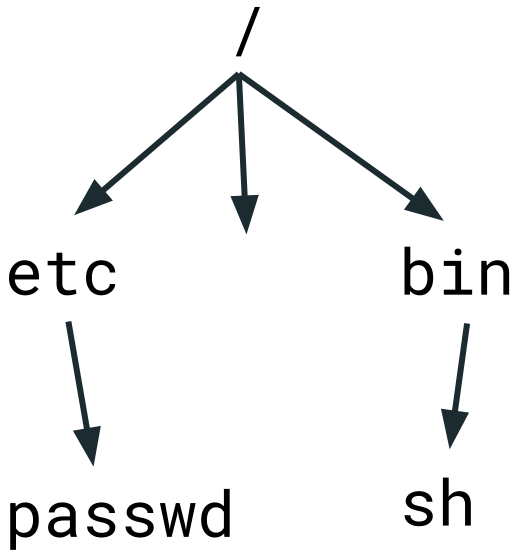
- Is your product a Docker image? No.
- Is your product a container runtime? Yes.

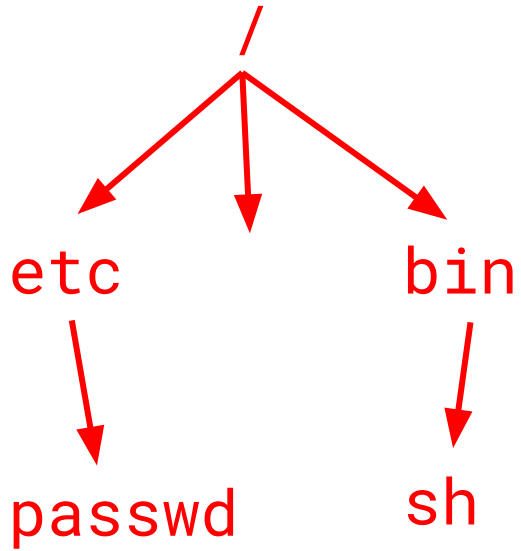
```
int mount(const char *source,
          const char *target,
          const char *filesystemtype,
          unsigned long mountflags,
          const void *data);


int unshare(int flags);
int clone(int (*fn)(void *),
          void *child_stack,
          int flags, void *arg, ...);
```




```
clone(..., CLONE_NEWNS, ...);
unshare(CLONE_NEWNS);
```

docker



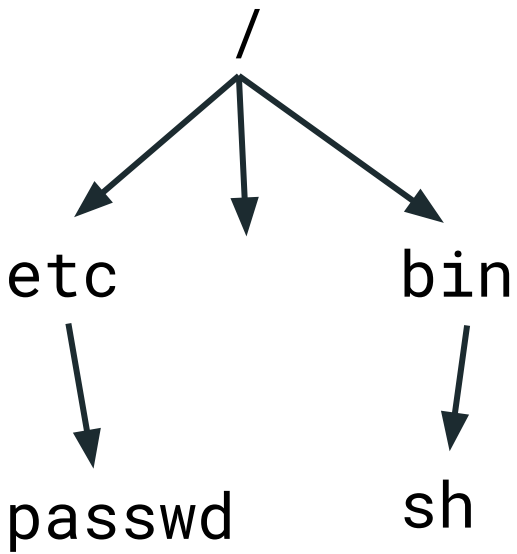


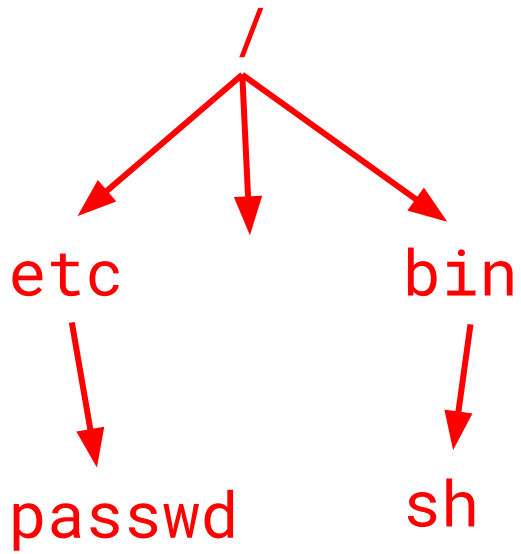
docker

```
int mount(const char *source,
          const char *target,
          const char *filesystemtype,
          unsigned long mountflags,
          const void *data);
```

- MS_PRIVATE
- MS_SLAVE
- MS_SHARED

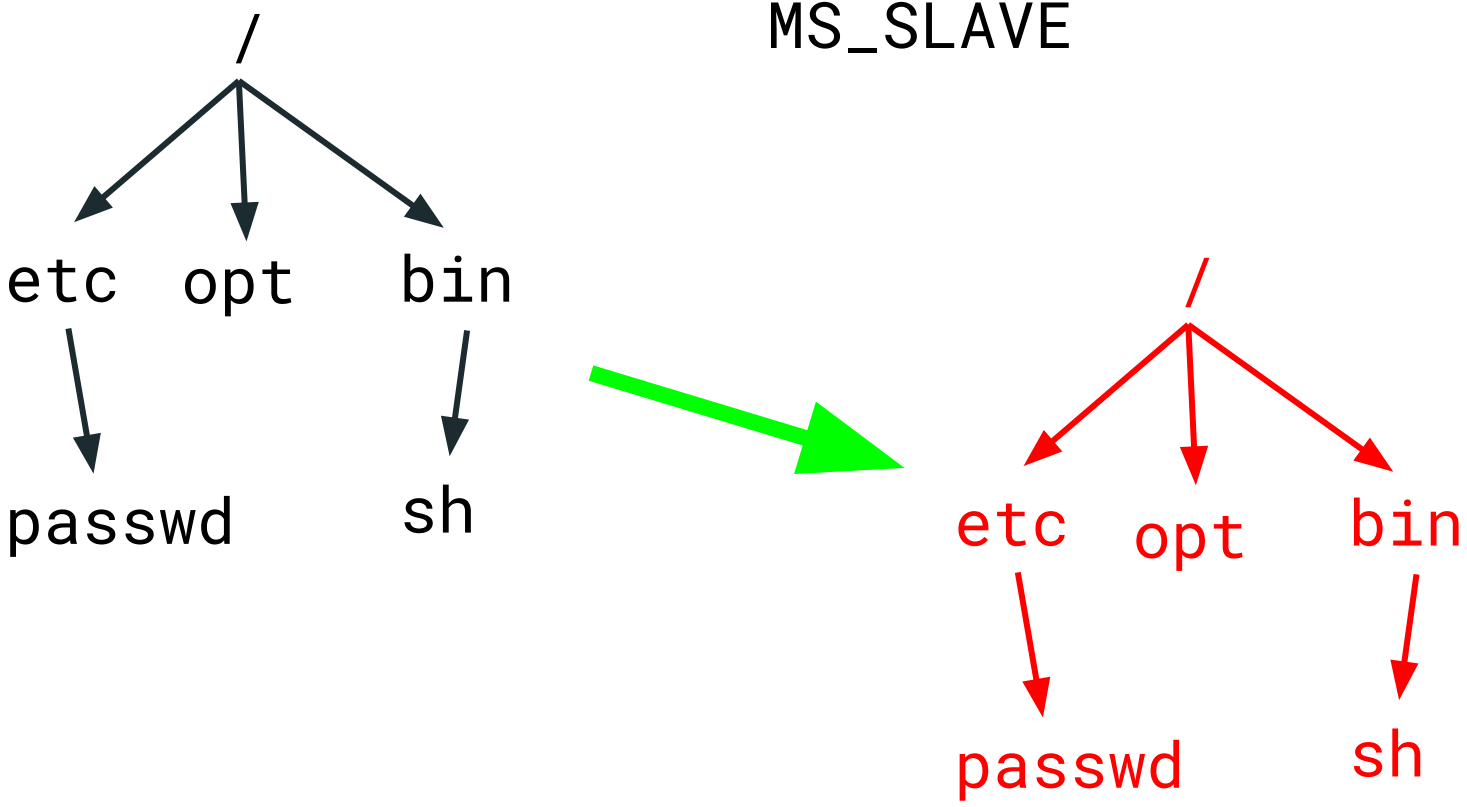
MS_PRIVATE





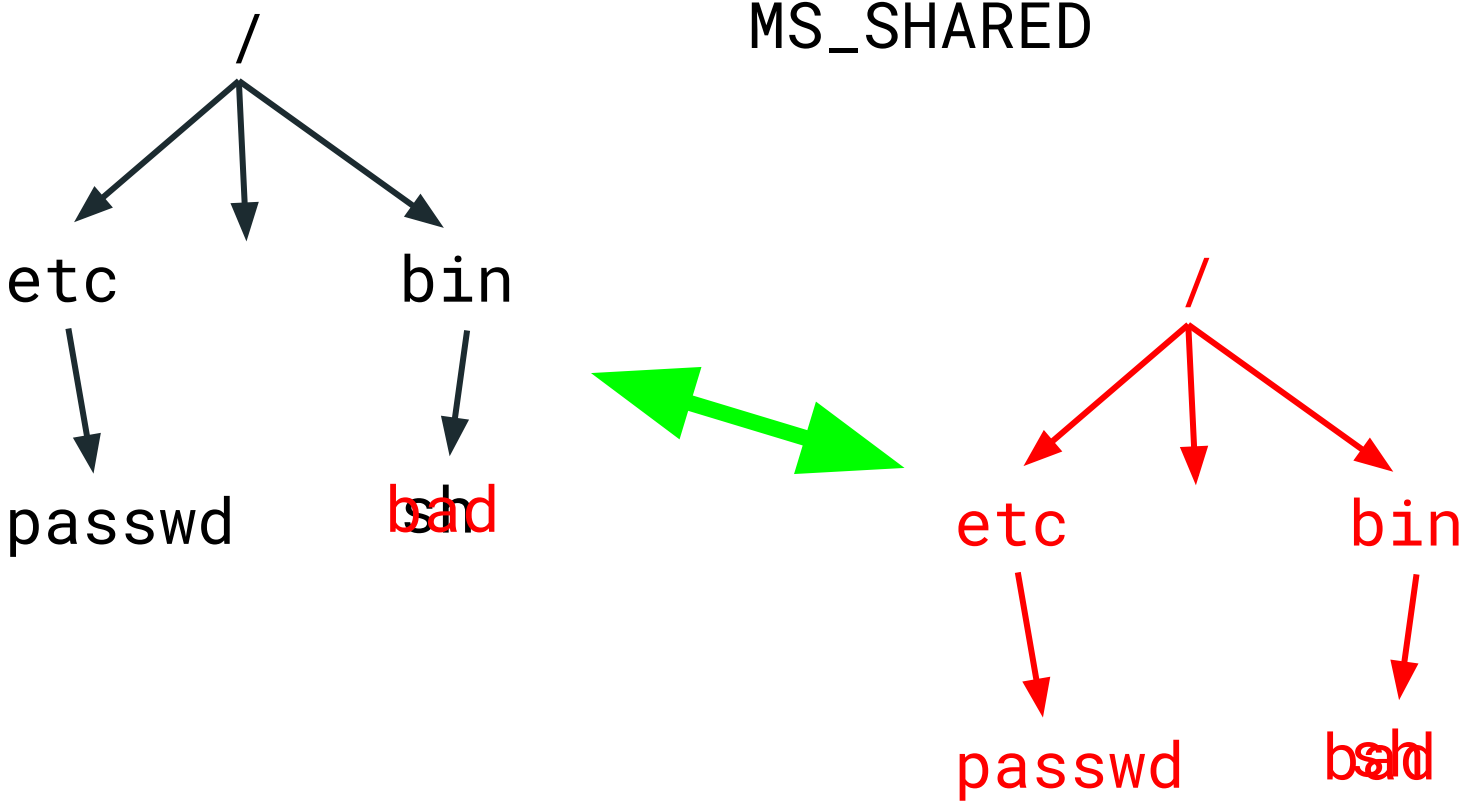
docker

MS_SLAVE



docker

MS_SHARED

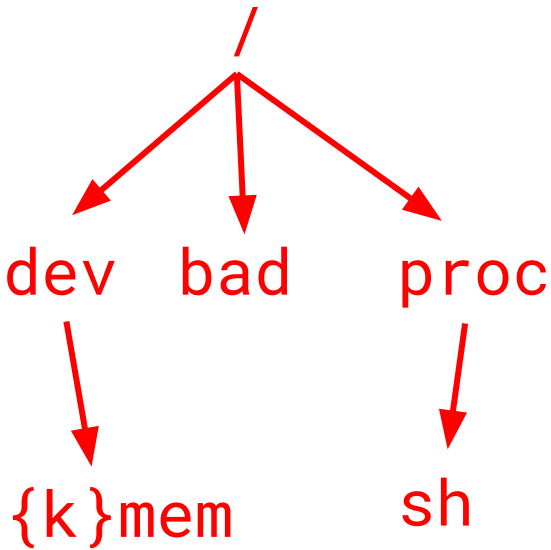



```
mount("/tmp/bad", "/bin/sh", ...)
execl("/bin/sh", "-c", ...)
```

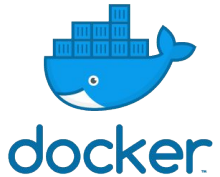
docker

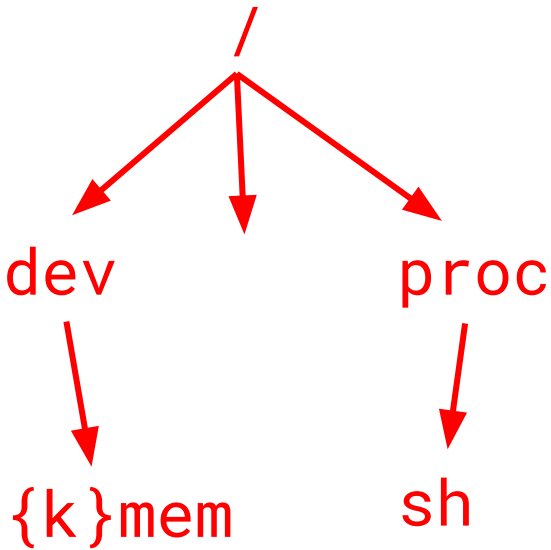
Ok, but what about chroot?

- Nobody shares roots
- `pivot_root()`

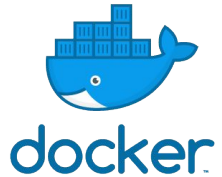



```
int mount("/dev/vda1",
          "/rootfs",
          "ext4", 0, NULL);
write("/bad/bin/sh", ...);
```



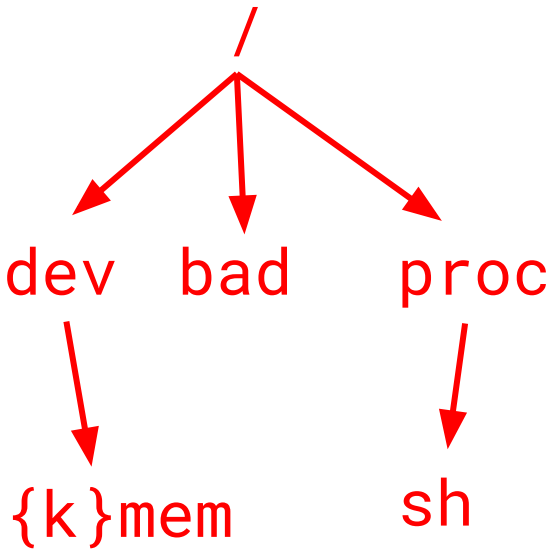



```
write("/dev/{k}mem", ...)
```

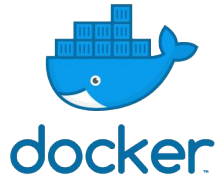


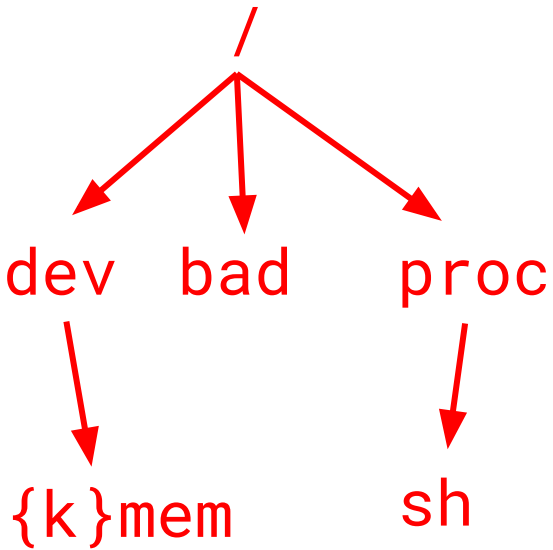
Start building the policy

- http://github.com/tych0/presentations/tree/master/osseu2017

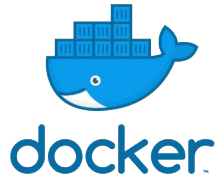



```
int mount("/dev",
          "/bad",
          NULL, MS_BIND, NULL);
write("/bad/{k}mem", ...);
```






```
int mount("/dev",
          "/bad",
          NULL, MS_MOVE, NULL);
write("/bad/{k}mem", ...);
```



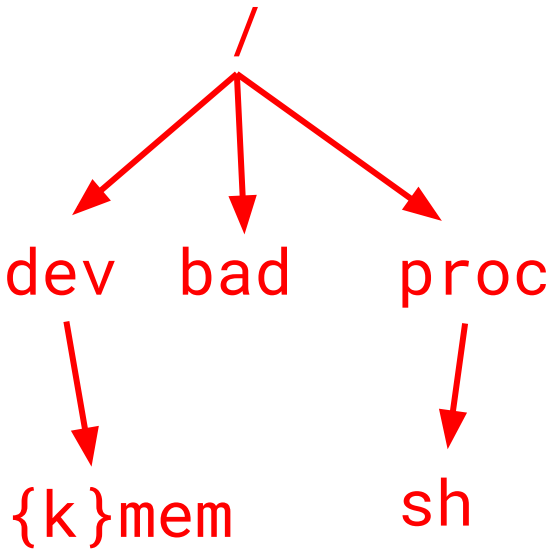
Allow block device mounts?

```
write("/dev/rdb/foo", "bad superblock")
```

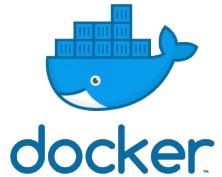
```
ext4_mount()
```

What about mknod?

```
mknod c 1 1
```

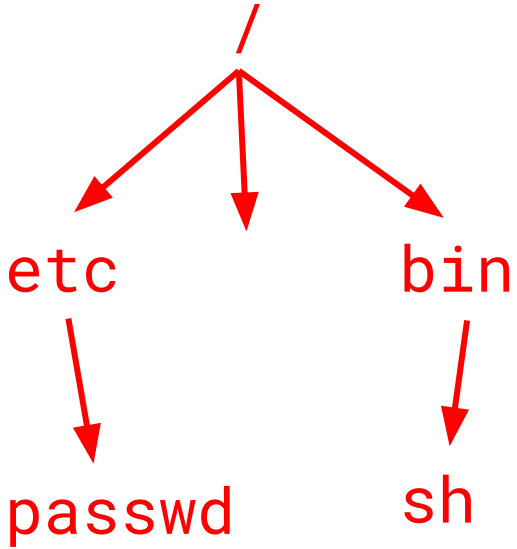



```
int mount(NULL,
          “/bad”,
          “devtmpfs”, 0, NULL);
write(“/bad/{k}mem”, ...);
```

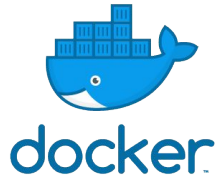


```
int mount(const char *source,
          const char *target,
          const char *filesystemtype,
          unsigned long mountflags,
          const void *data);
```

- MS_NOSUID
- MS_NOEXEC
- MS_READONLY
- MS_*ATIME
- MS_REMOUNT




```
mount(MS_REMOUNT) = 0
unshare(CLONE_NEWUSER);
mount(MS_REMOUNT) = -EPERM
```



Screw it, let's use a runtime policy

- `seccomp()` BPF program to decide which `mount()`s are allowed
- `mount("/dev…")` or `mount(..., "/dev…")`
- Time Of Check Time Of Use! (TOCTOU)
- syscalls are not atomic

Screw it, let's use a runtime policy

```
ptr = "/tmp/foo";
mount(ptr, "/tmp/bar", ...)
                        __secure_computing(...) = 0

ptr = "/dev/sda1"
                        do_mount(...)
```

Syscalls are not atomic: CVE-2016-9962

```
fd = open(“/sensitive”, O_CLOEXEC | O_PATH)
prctl(PR_SET_DUMPABLE, SUID_DUMP_DISABLE)
setns(container, ...)
execl(“/bin/sh”)
                                {re}set_dumpable()

open(“/proc/pid/fd/3”,...)

                                do_close_on_exec()
```

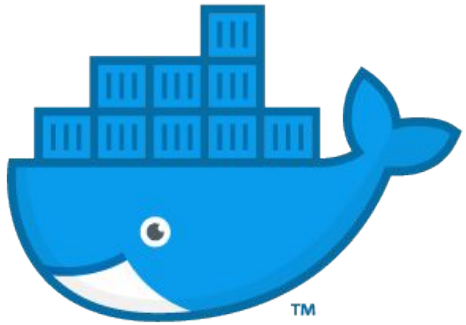
Other interesting bits

- `/proc/pid/root -> /`
- `/proc/pid/cwd -> ...`
- `/proc/pid/exe -> ...`
- yama

But what if we want to mount?

- SECCOMP_USER_NOTIF

```
mount()
        __secure_computing()
                                struct seccomp_notif {
                                    u32 id;
                                    pid_t pid;
                                    struct data *sd;
                                }
```



THANK YOU