Exploring container mechanisms through the story of a syscall

SELinux, seccomp-bpf, capabilities, overlayfs, path lookups

WE BUILD LINUX TECHNOLOGIES

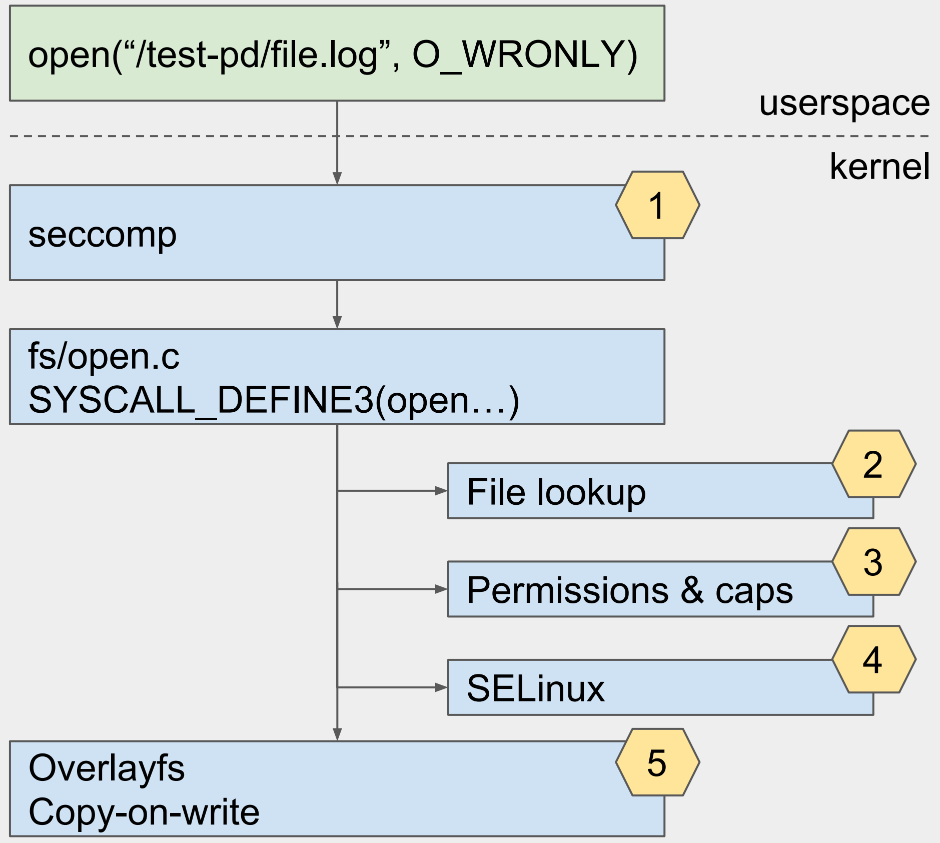
Hi, I'm Alban

Alban Crequy
CTO @ Kinvolk

alban@kinvolk.io

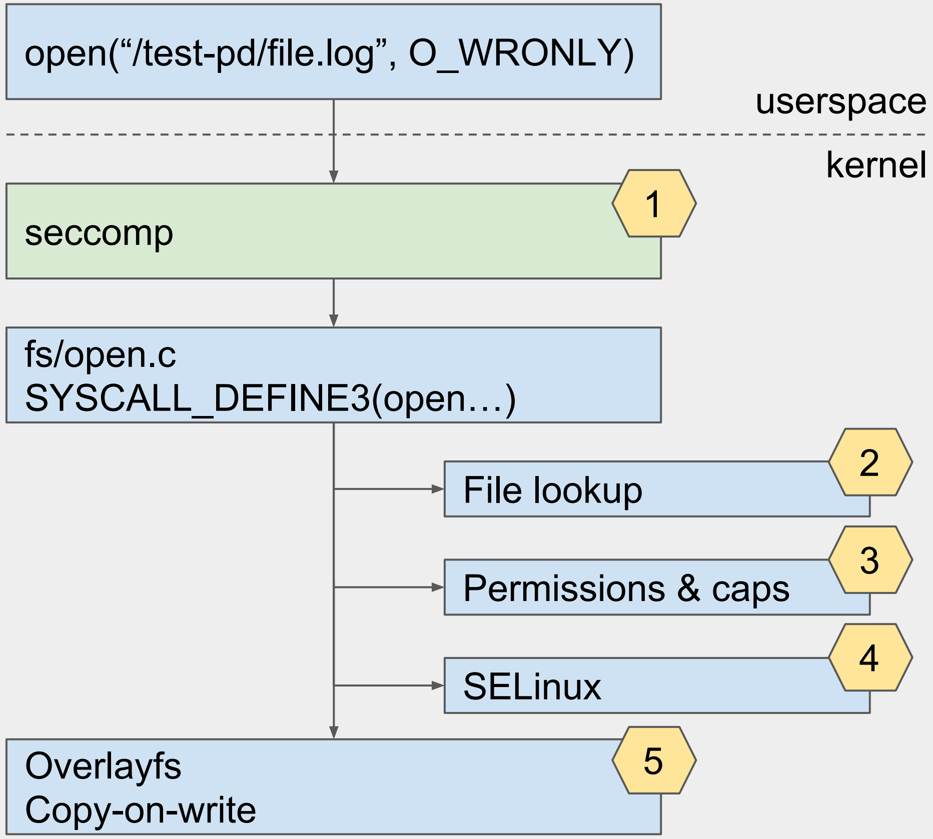
Plan

Story of a syscall and
how it works with Kubernetes



1 seccomp

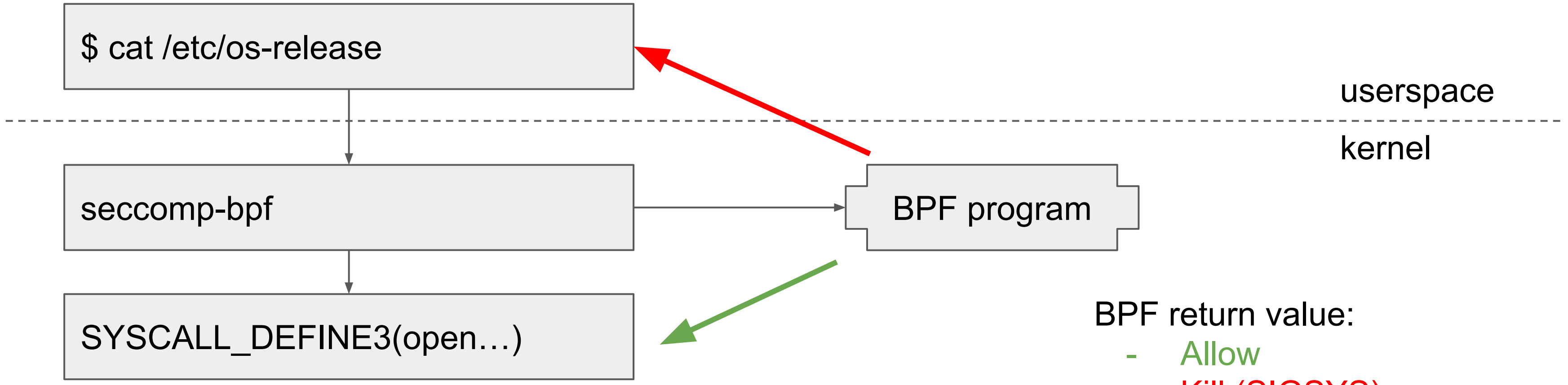
BPF



seccomp, 2 modes

- SECCOMP_SET_MODE_STRICT
    - Syscalls allowed: read(), write(), exit()
    - Other syscalls: SIGKILL
    - Not useful in the context of Kubernetes
- SECCOMP_SET_MODE_FILTER
    - Execute a custom BPF program
    - Actions: allow, kill (SIGSYS), errno

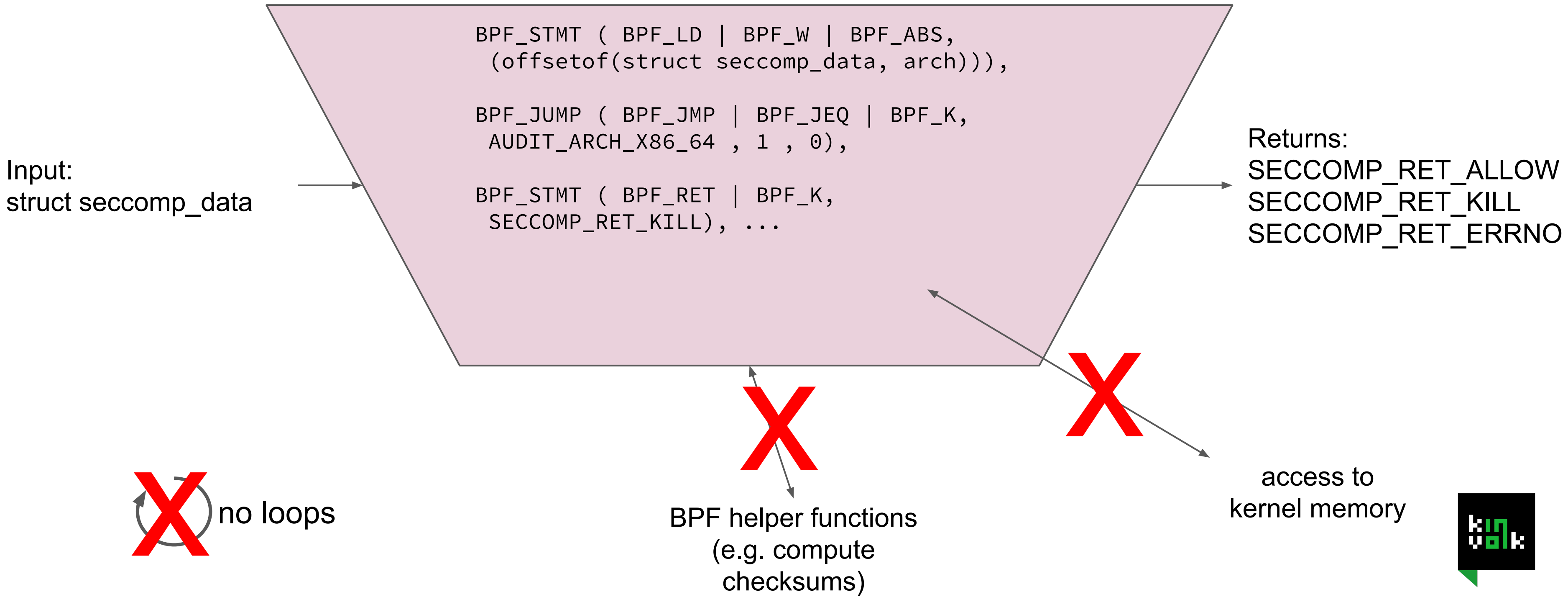
seccomp-bpf



BPF return value:

- Allow
- Kill (SIGSYS)
- Errno

BPF program

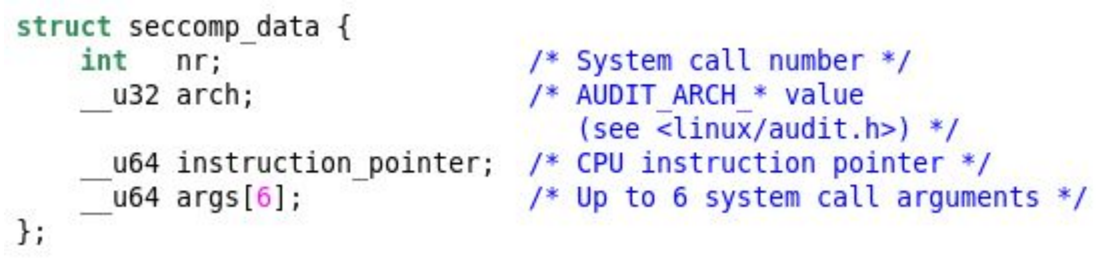


```
BPF_STMT ( BPF_LD | BPF_W | BPF_ABS,
 (offsetof(struct seccomp_data, arch))),

BPF_JUMP ( BPF_JMP | BPF_JEQ | BPF_K,
 AUDIT_ARCH_X86_64 , 1 , 0),

BPF_STMT ( BPF_RET | BPF_K,
 SECCOMP_RET_KILL), ...
```

Seccomp-bpf limitations



```
struct seccomp_data {
    int   nr;                    /* System call number */
    __u32 arch;                  /* AUDIT_ARCH_* value
                                    (see <linux/audit.h>) */
    __u64 instruction_pointer;   /* CPU instruction pointer */
    __u64 args[6];               /* Up to 6 system call arguments */
};
```

```
open("/test-pd/file.log", O_WRONLY)

open(0x558d8fcf2000, 0x0001)
```

Seccomp-bpf limitations

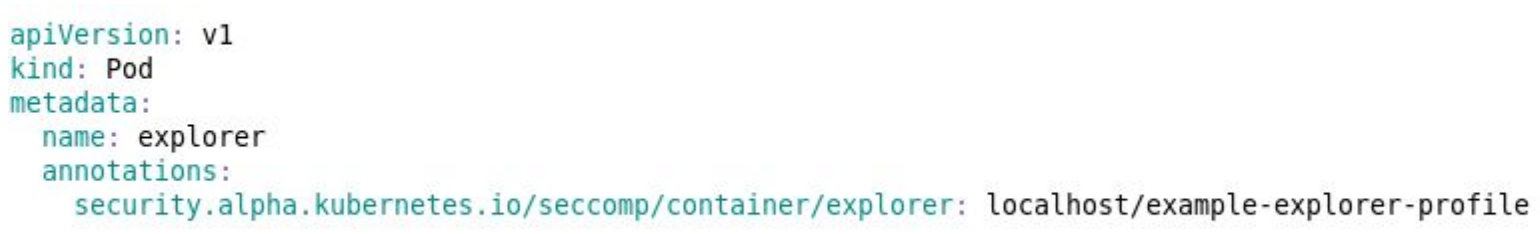
- ★ Once installed, cannot update a BPF program
- ★ Classic BPF, no maps to store context
- ★ No loops (no strcmp or similar)
- ★ Cannot read kernel or process memory. Cannot dereference pointers.
- ★ Cannot interpret paths
  - ○ Time of check to time of use (TOCTOU)

Seccomp-bpf in Docker

- Docker has a default seccomp profile
  - Returns Errno by default
  - Whitelist, parametrized by capabilities
  - Blocking some syscalls, e.g. ‘add_key’
- Can be changed:
  - docker run --security-opt seccomp=/path/to/seccomp/profile.json …
  - See format:
    - https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

Seccomp-bpf in Kubernetes

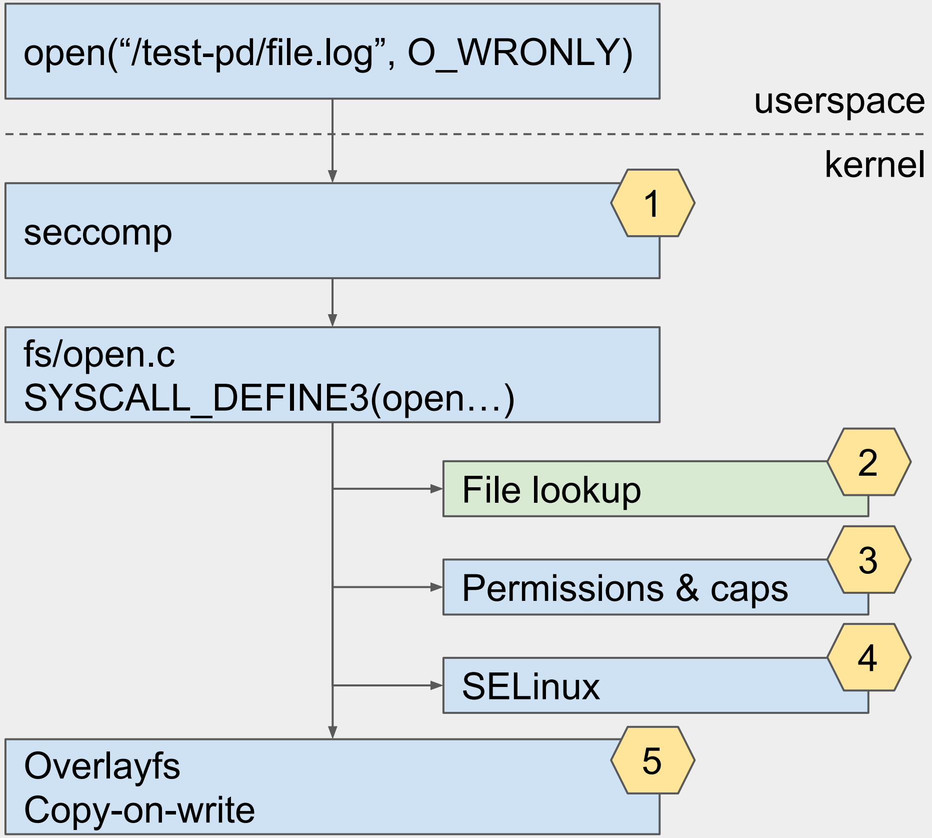
- ★ Work in progress…
  - Issue: https://github.com/kubernetes/features/issues/135
  - Proposal merged: https://github.com/kubernetes/kubernetes/pull/24602



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: explorer
  annotations:
    security.alpha.kubernetes.io/seccomp/container/explorer: localhost/example-explorer-profile
```

2 File lookups

In the mount namespace
of the container



File lookup

open("/test-pd/file.log", O_WRONLY)

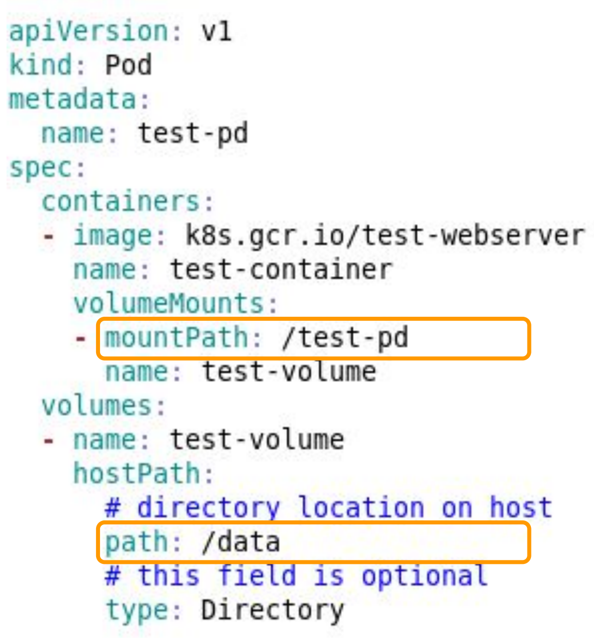
Each process can potentially have a different root ("/") with chroot()

See /proc/$pid/root/
(with CAP_SYS_PTRACE)

Path resolved following mountpoints in the mount namespace of the process

See /proc/$pid/ns/mnt

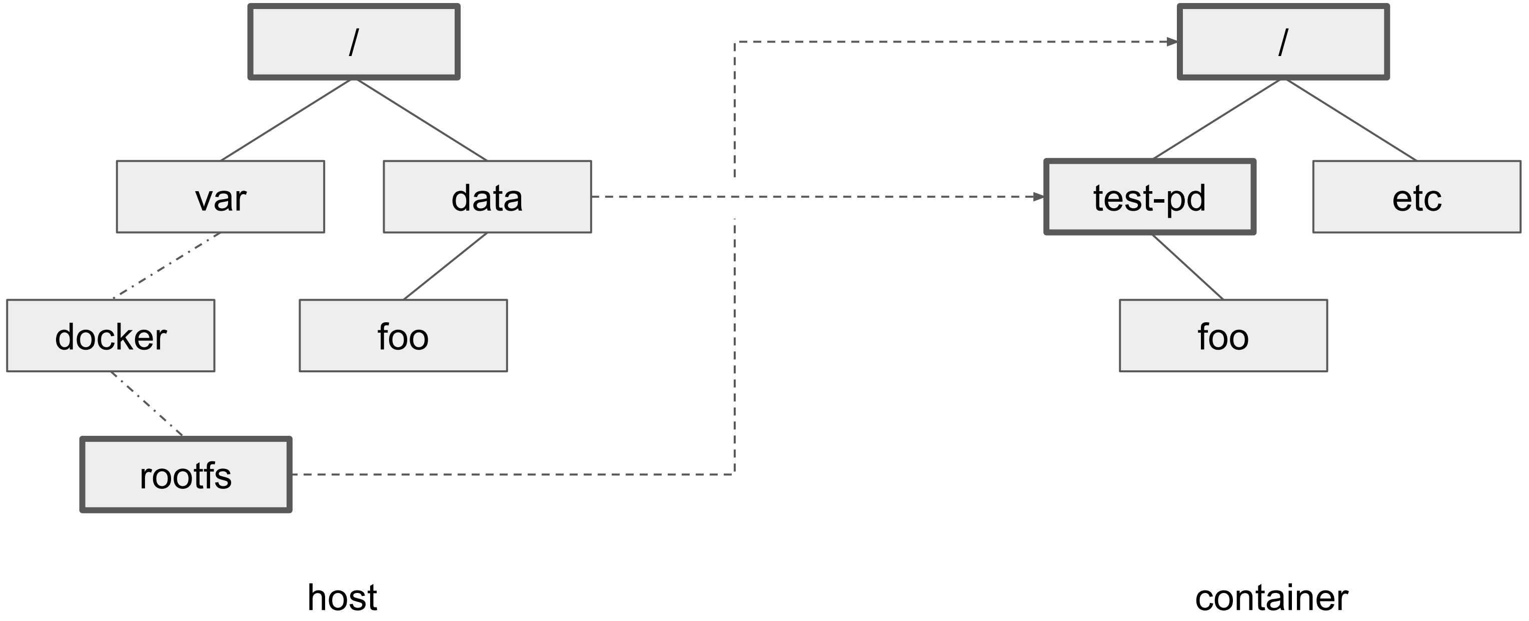
Volumes in Kubernetes



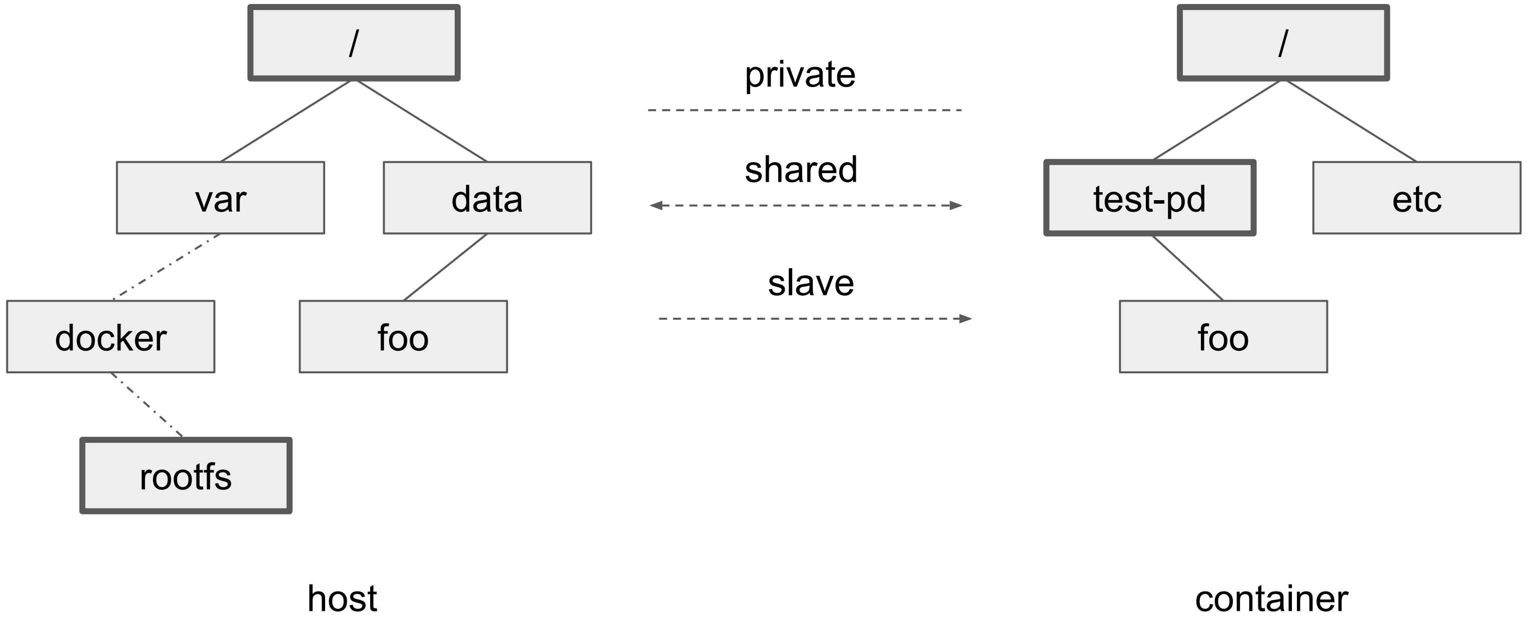
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

Example from https://kubernetes.io/docs/concepts/storage/volumes/

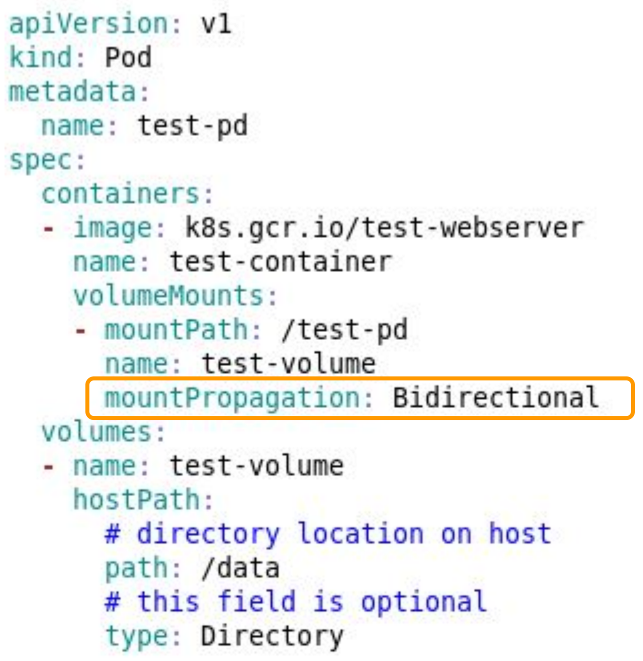
Volumes in Kubernetes



Mount propagation in Linux



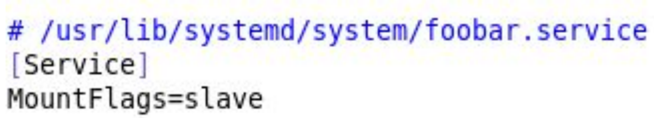
Mount propagation in Kubernetes



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
      mountPropagation: Bidirectional
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

MountPropagation = [HostToContainer | Bidirectional]

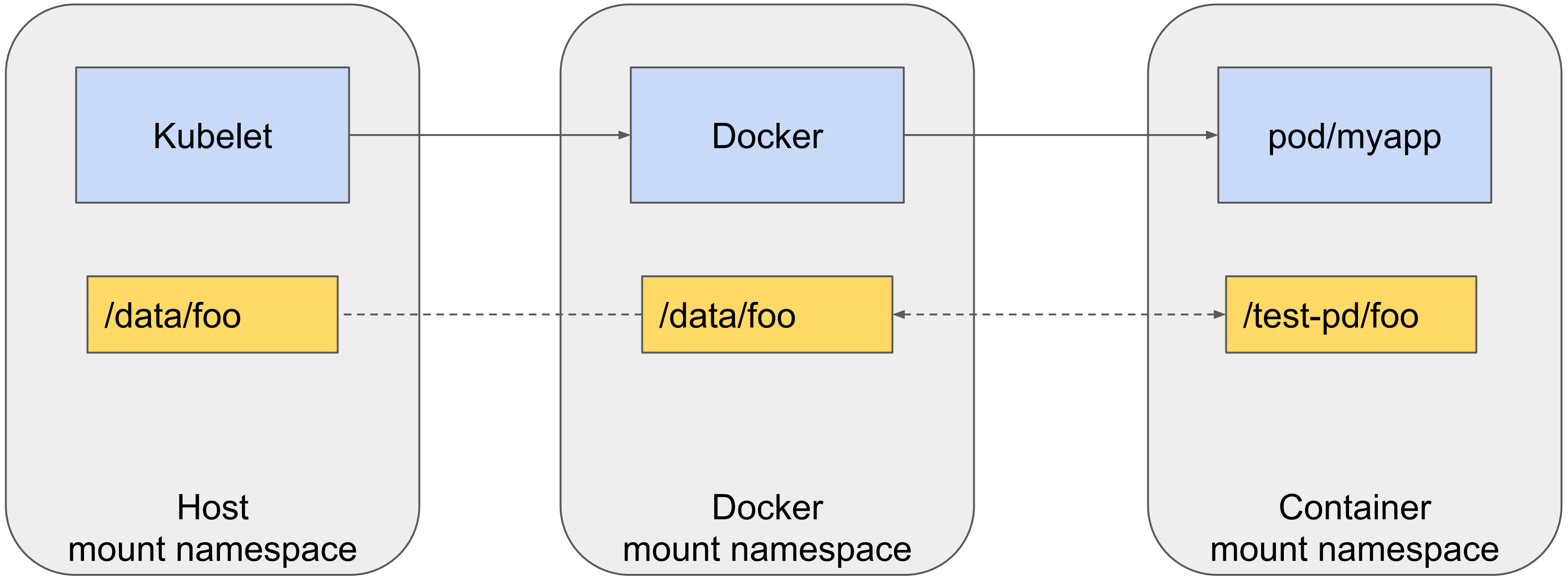
Mount propagation with systemd



```
# /usr/lib/systemd/system/foobar.service
[Service]
MountFlags=slave
```

MountFlags = [shared | slave | private]

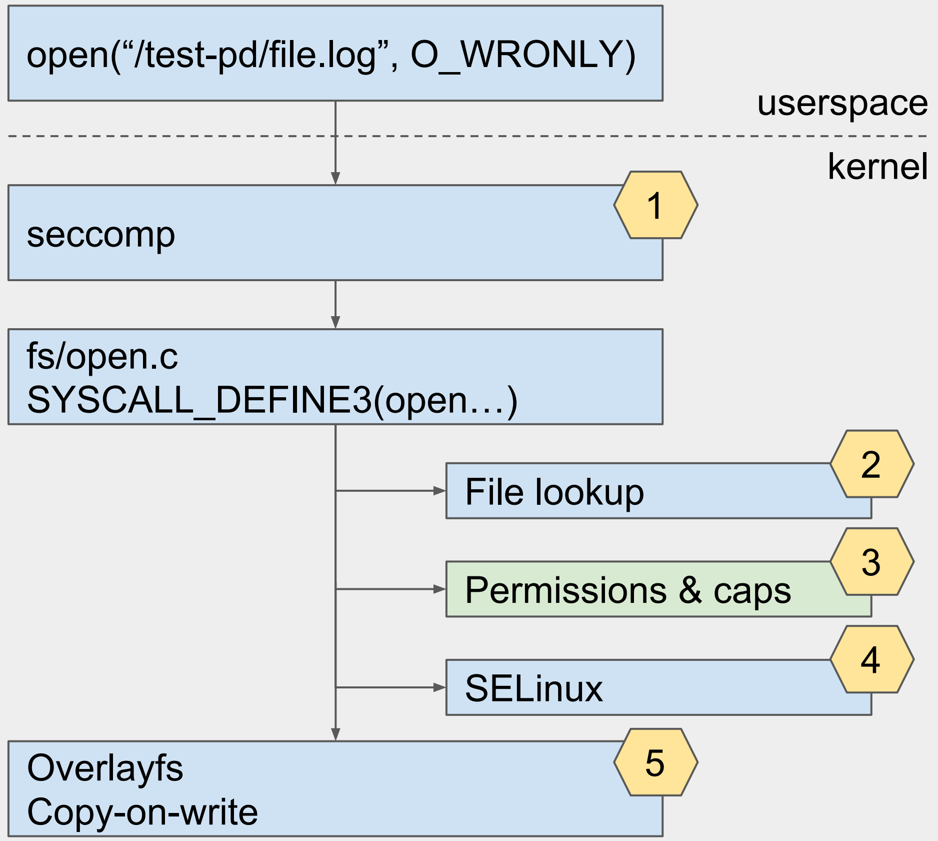
systemd's MountFlags= in the context of K8s



MountFlags=private

MountPropagation=
Bidirectional

3 Permissions & caps



Capabilities

★ After seccomp-bpf and the file lookup

★ Check for permissions (-rwxrwxrwx)

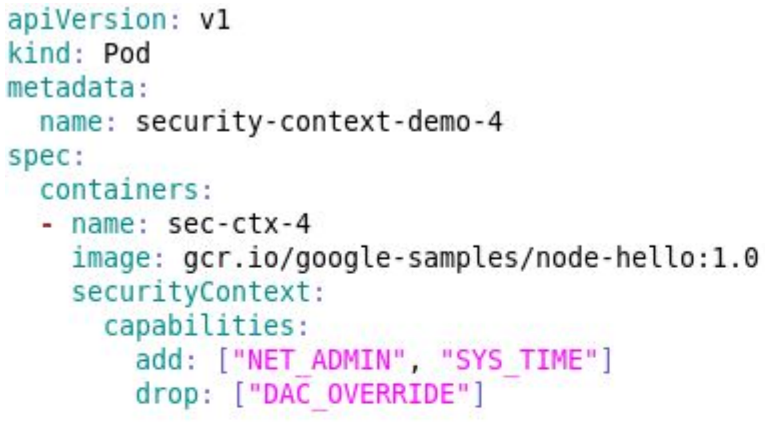
★ Check for CAP_DAC_OVERRIDE

  ○ I.e. root can access files even if permissions don’t allow it

Capabilities with Docker & Kubernetes

- In Docker:

  -
    ```
    docker run --cap-add=NET_ADMIN --cap-add=SYS_TIME \
               --cap-drop=DAC_OVERRIDE
    ```
- In Kubernetes:

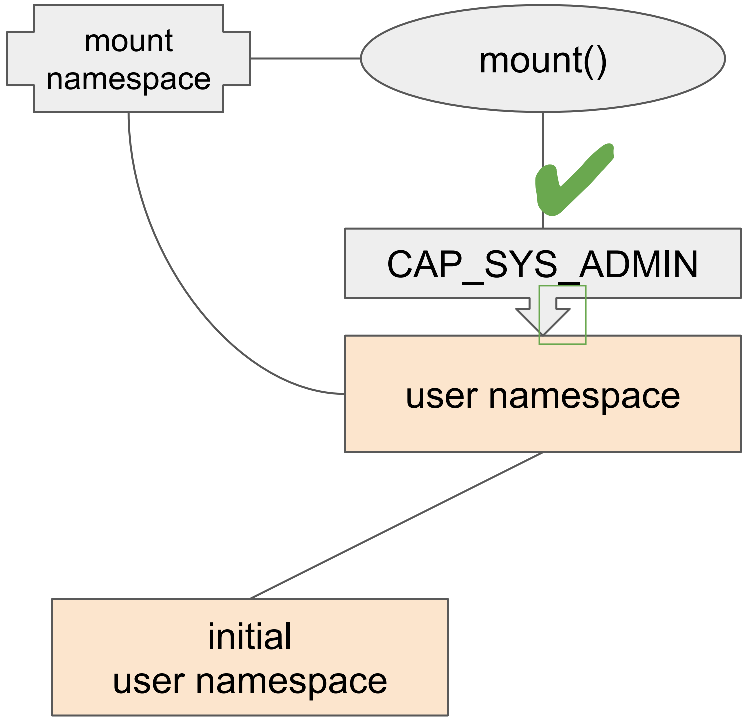


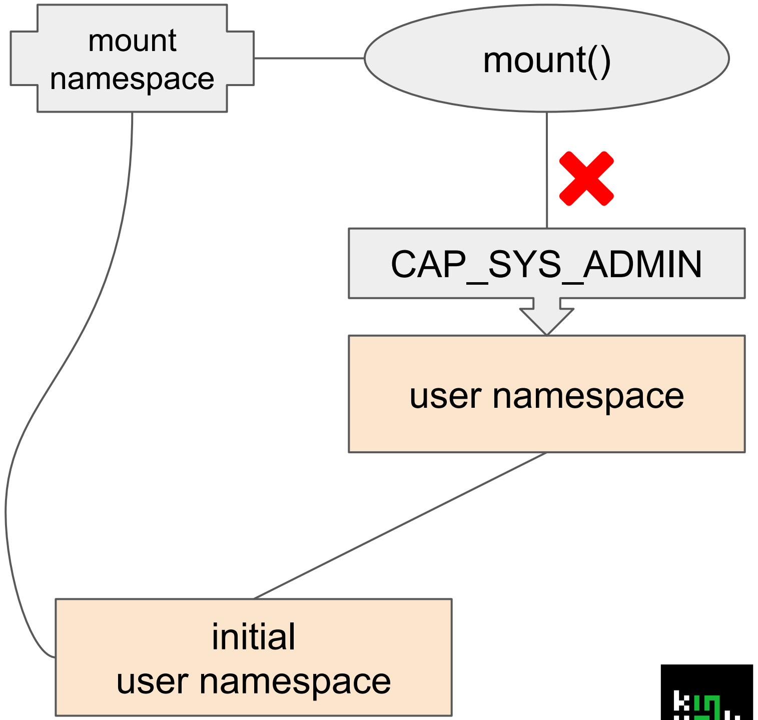
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
  - name: sec-ctx-4
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
        drop: ["DAC_OVERRIDE"]
```

Capabilities and user namespaces

- ★ Capabilities are relative to a user namespace
- ★ Capabilities can be checked with regards to the user namespace owning a {mount|network} namespace

Userns & capabilities example: mount()



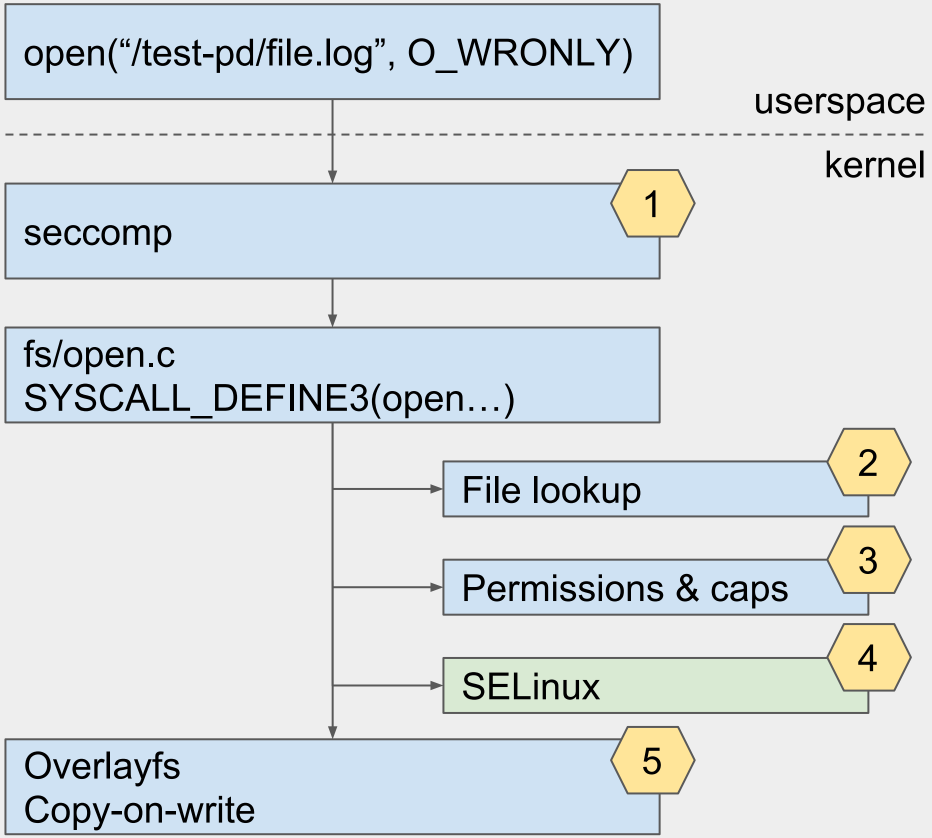


User namespaces in Kubernetes

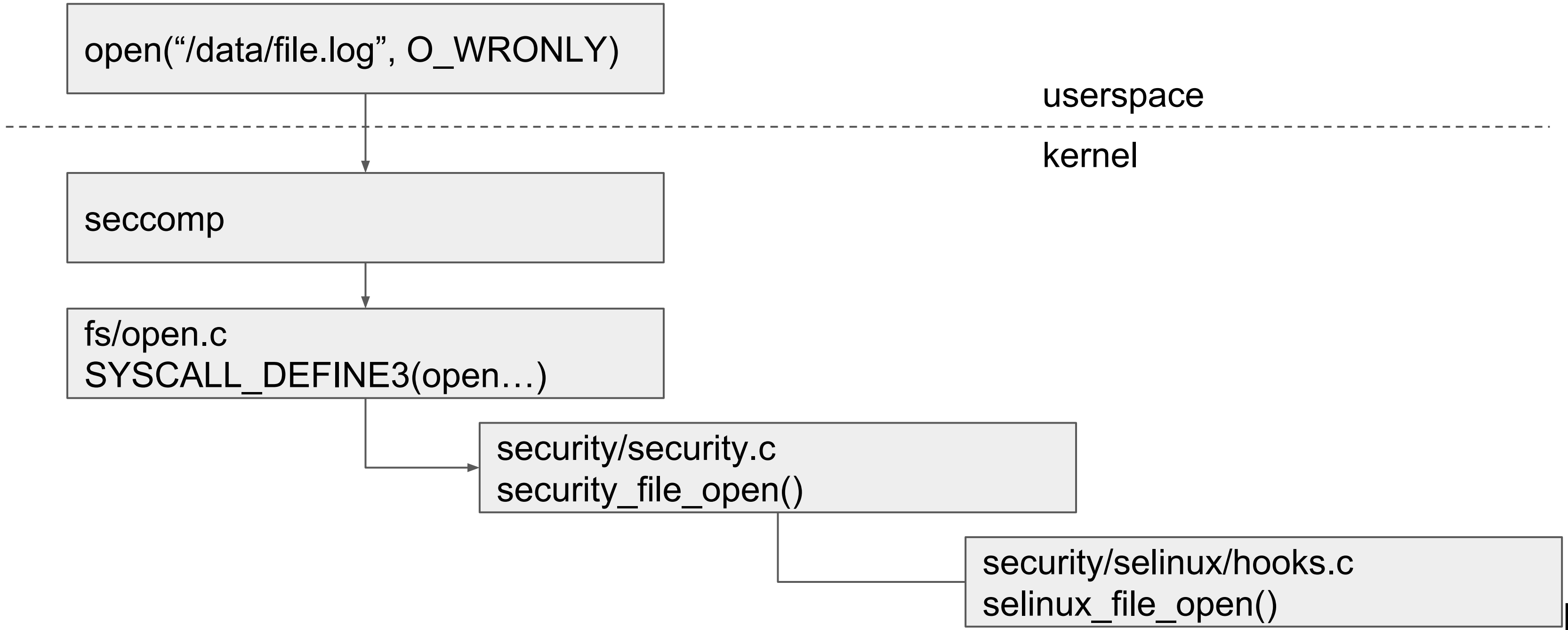
- ★ Kubernetes does not use user namespaces (yet):
  - https://github.com/kubernetes/community/pull/2042
  - https://github.com/kubernetes/community/pull/2067
- ★ However, understanding user namespaces is still relevant for Kubernetes:
  - Unprivileged user namespaces used for building container images
    https://github.com/genuinetools/img

4 SELinux

LSM, labels, enforcement, integration in Kubernetes



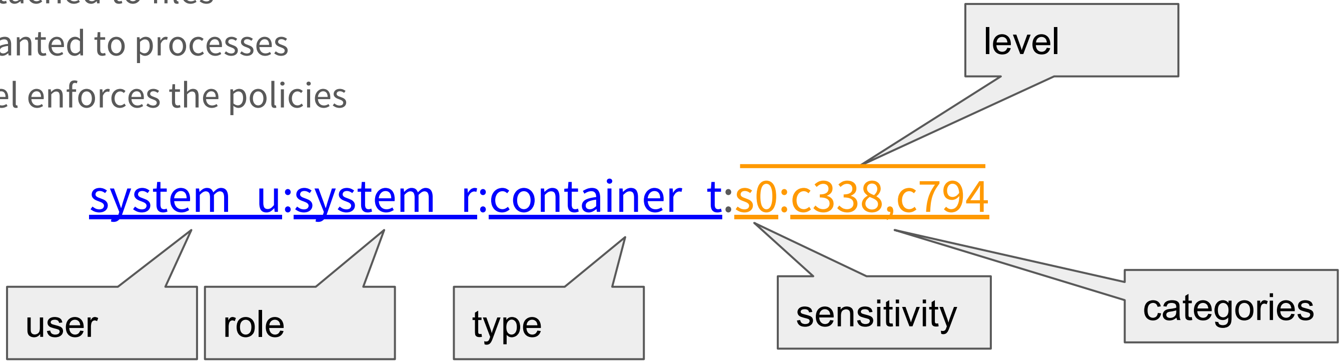
Hooks with Linux Security Module (LSM)



SELinux: labelling

★ Labelling system

- Labels attached to files
- Labels granted to processes
- The kernel enforces the policies



SELinux: enforcement

system_u:system_r:container_t:s0:c338,c794

Type Enforcement (TE)

Protecting the host from the container

Multi categories security (MCS)

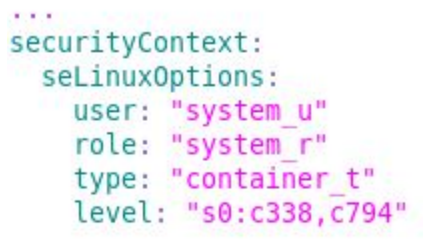
Protecting the containers from each others

SELinux and Kubernetes

★ SELinux policies defined by the Linux distro (/etc/selinux/)

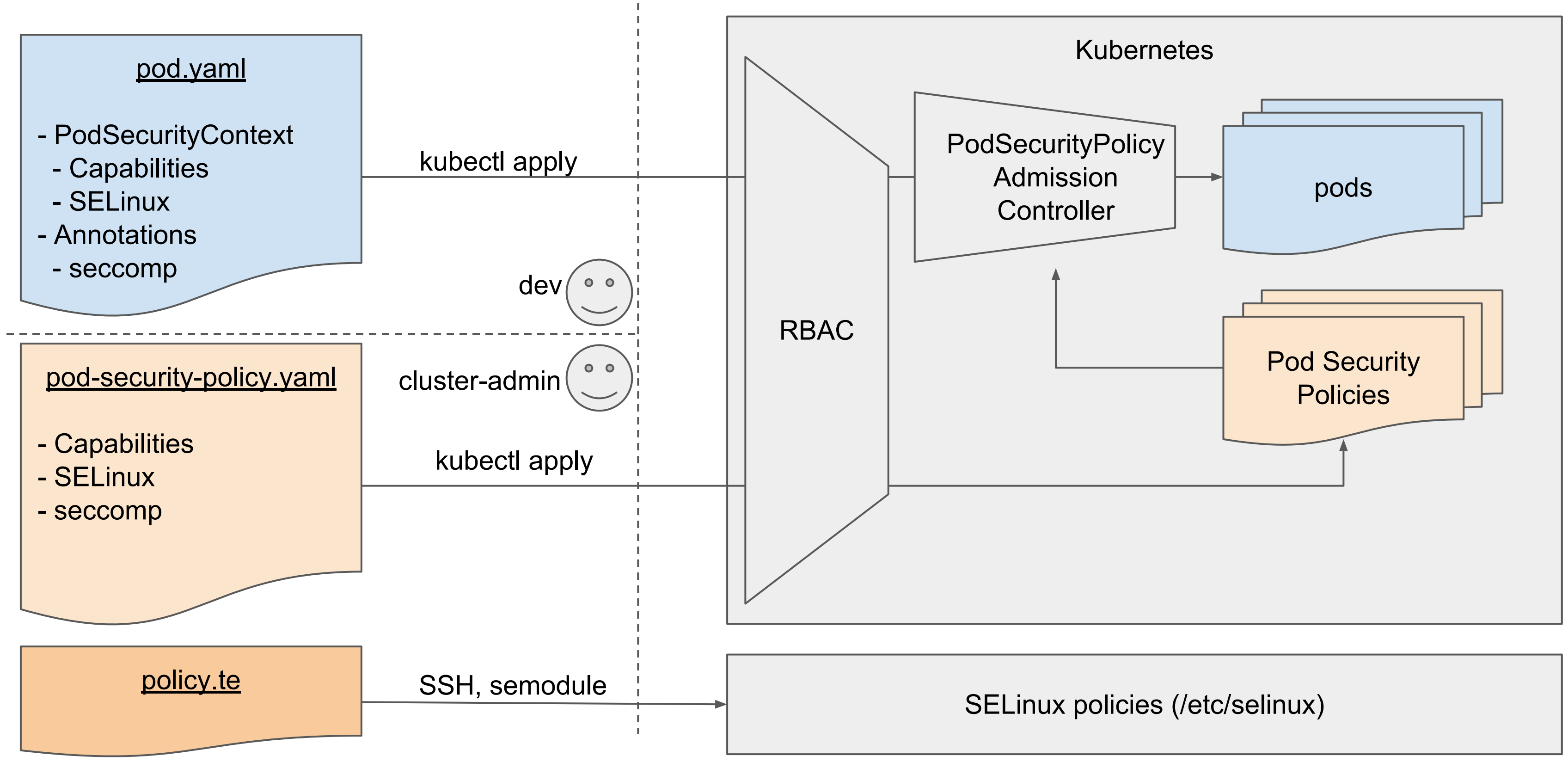
★ Pods can be configured to

- Attach a label to files in the container
- Grant that label to processes in the container



```
...
securityContext:
  seLinuxOptions:
    user: "system_u"
    role: "system_r"
    type: "container_t"
    level: "s0:c338,c794"
```

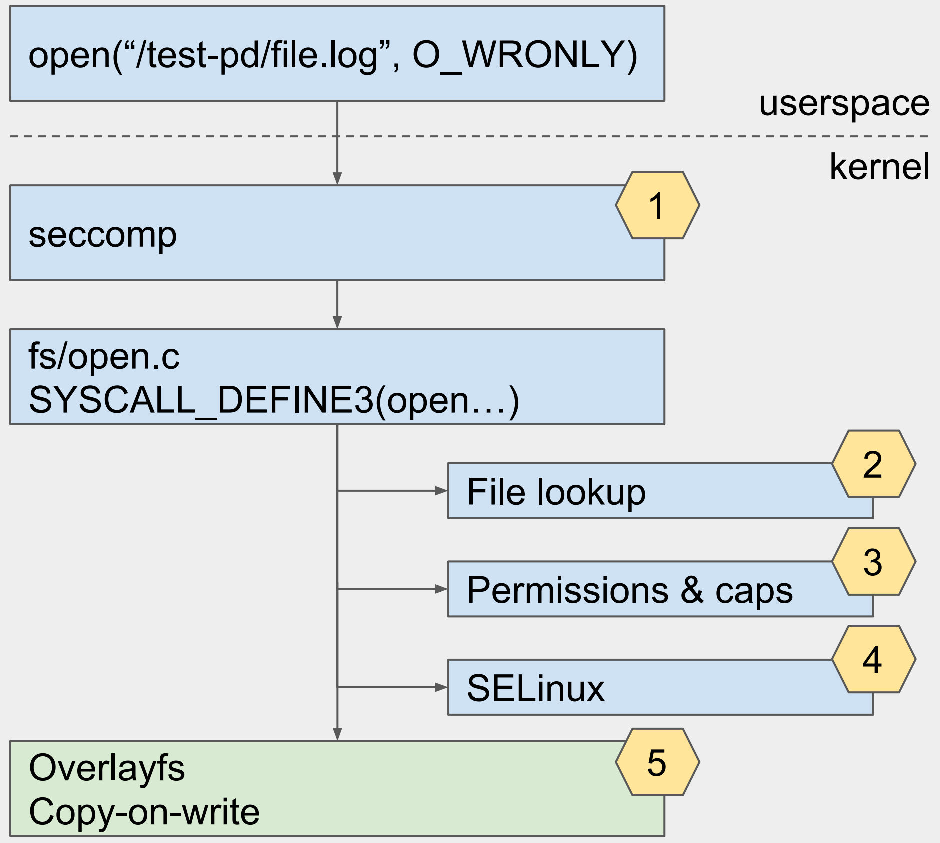
PodSecurityPolicy and PodSecurityContext



5

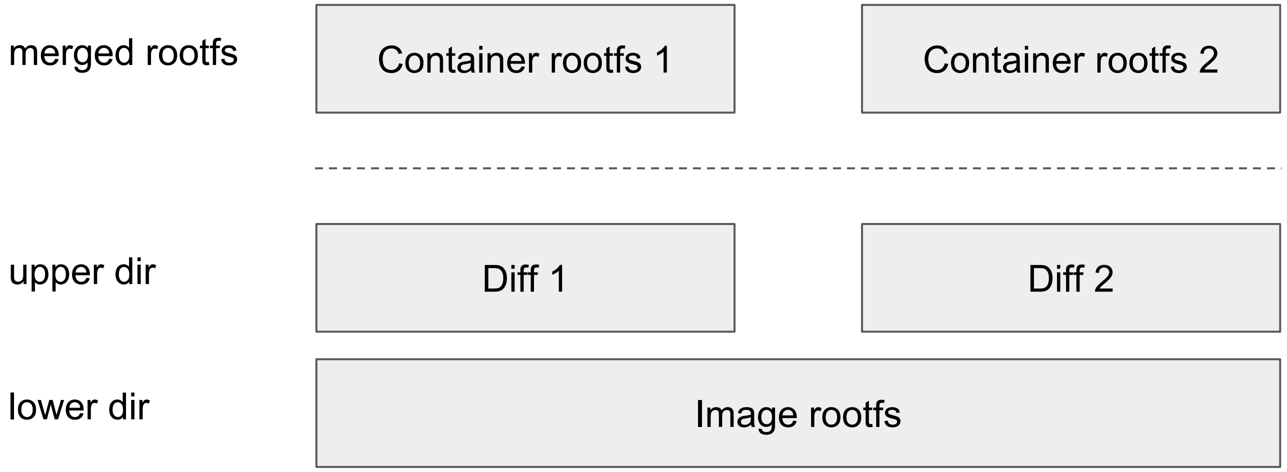
Overlay filesystem

Copy On Write (COW)

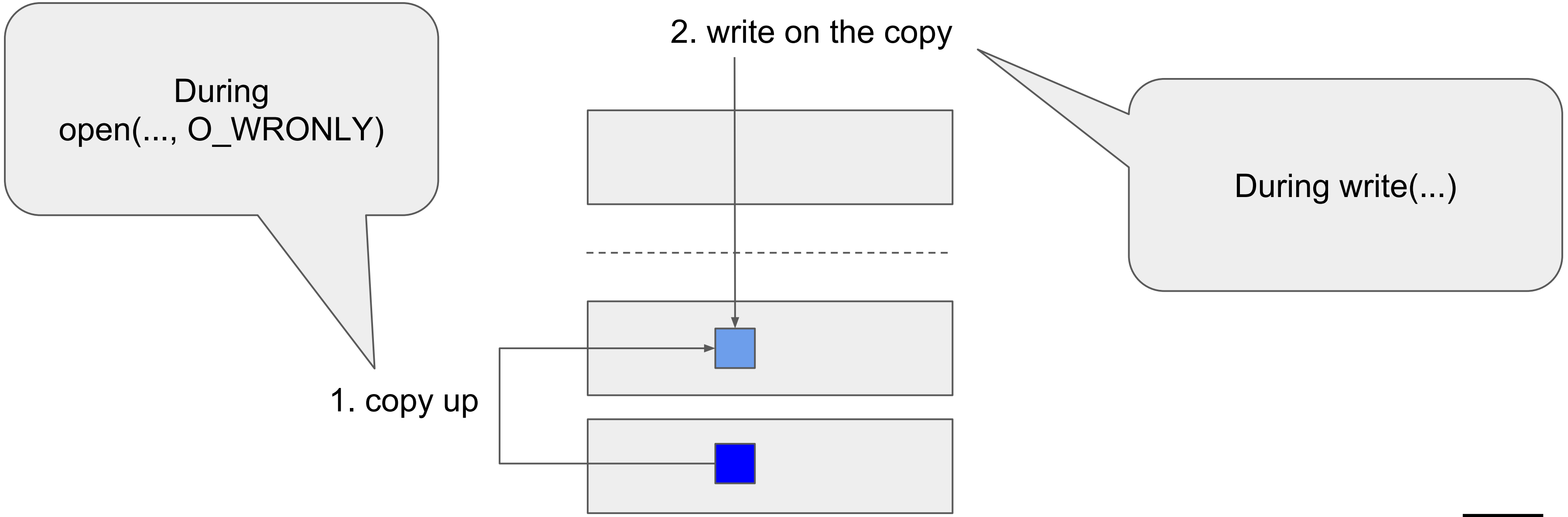


Overlayfs mounts

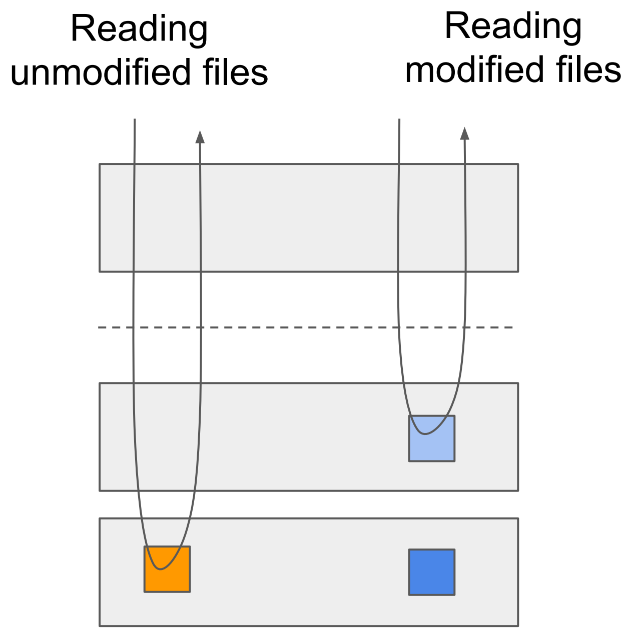
```
$ sudo mount -t overlay overlay \
    -olowerdir=$IMAGE,upperdir=$CONTAINER_DIFF,workdir=$WORK merged
```



Writing on overlayfs, copy up

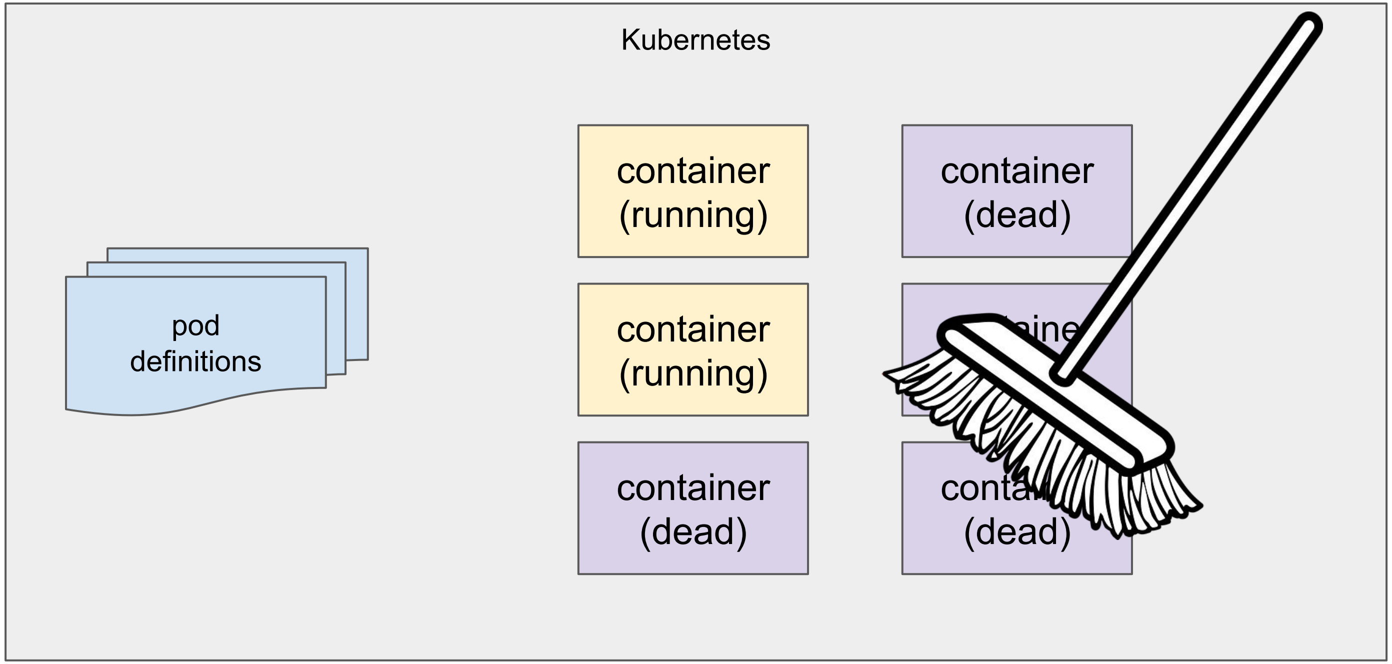


Reading on overlayfs



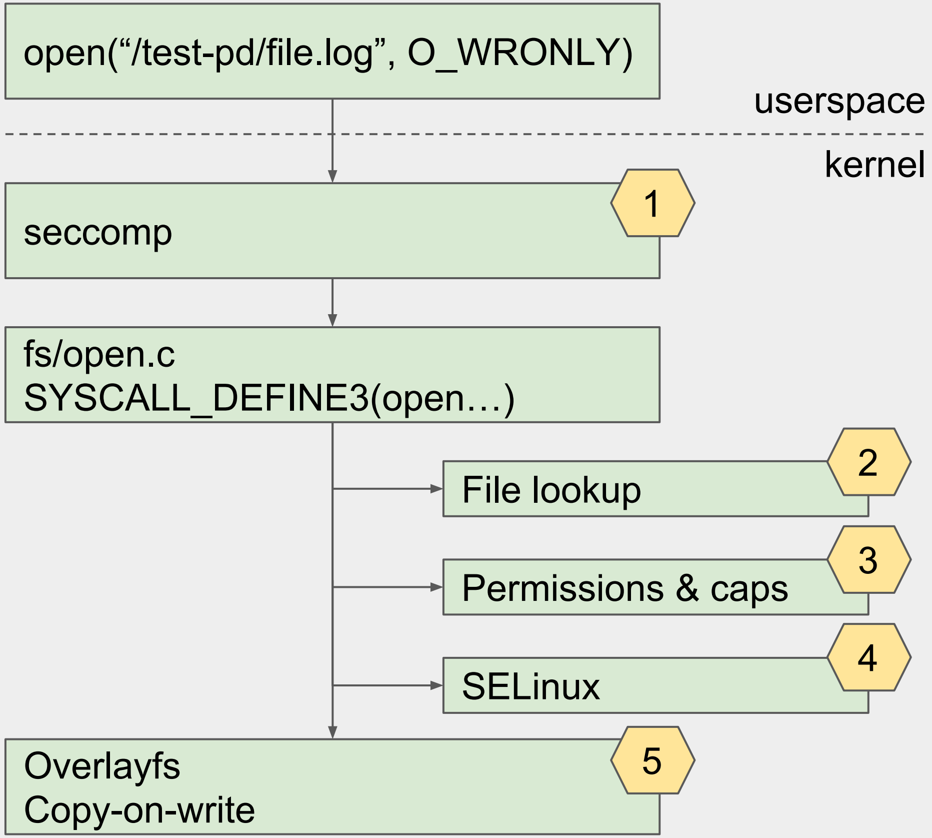
Kubelet container collection

★ Every minute, the Kubelet checks for old dead containers to remove



Summary

Questions?


open("/test-pd/file.log", O_WRONLY)
userspace
kernel
seccomp
1
fs/open.c
SYSCALL_DEFINE3(open…)
File lookup
2
Permissions & caps
3
SELinux
4
Overlayfs
Copy-on-write
5