KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA

KubeCon CloudNativeCon Europe 2022

CRI-O: Secure, Performant, and Boring as Ever!

Peter Hunt, Urvashi Mohnani, Mrunal Patel, Sascha Grunert

What is CRI-O?

- OCI based Kubernetes runtime
- Supports all OCI based container images, runtimes, and registries
- Balance stability and features
- Focus on Kubernetes
- Focus on security

A Few Updates

- CRI v1alpha2 to v1
  - Support for both implementations
- Reduce CPU overhead of golang GC
- Sysctl CVE
  - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0811

Rethinking container lifecycle management

For many releases, conmon was the desired OCI runtime (runc, crun) monitor:

https://github.com/containers/conmon

- Little helper tool to ensure communication between CRI-O and OCI runtime
- Takes care of container creation and process cleanup upon termination
- Provides endpoints for executing processes and attaching to containers
- Writes container log files for Kubernetes
- Fulfills use cases for Podman

Rethinking container lifecycle management

conmon is designed to have the lowest possible memory footprint

Some drawbacks of conmon:

- It’s written in C and therefore finding maintainers is hard
- The main interface to interact with conmon is the command line (CLI)
- Adding new features increases technical debt
- It still has runtime dependencies like glib, which makes static linking hard

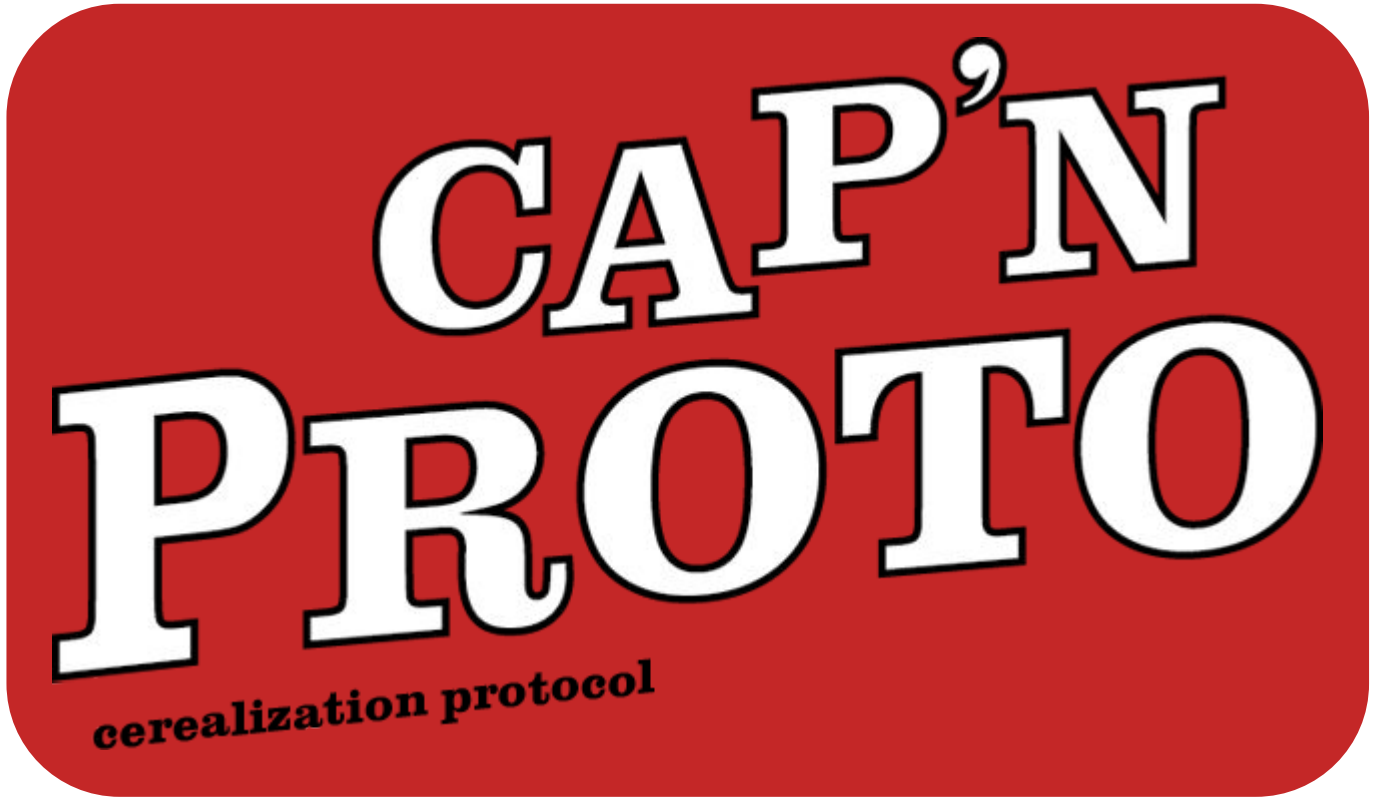
Rethinking container lifecycle management

We’re happy to announce the successor of conmon, **conmon-rs**!

- Completely new software architecture
- Utilizing Rust and the asynchronous tokio runtime: https://tokio.rs
- Targeting to keep the memory usage (vmrss) as low as possible
- Supporting multiple containers (Pods) in one instance
- Re-engineering in-container process execution (exec)
- Providing an extensible API and golang client

Rethinking container lifecycle management

An API between Rust and golang?



Rethinking container lifecycle management

Cap’n Proto is faster than protobuf and smaller than a gRPC runtime (https://grpc.io)

```
19      struct CreateContainerRequest {
20          id @0 :Text;
21          bundlePath @1 :Text;
22          terminal @2 :Bool;
23          exitPaths @3 :List(Text);
24          logDrivers @4 :List(LogDriver);
25      }
26
27      struct LogDriver {
28          type @0 :Type;
29          path @1 :Text;
30
31          enum Type {
32              # The CRI logger, requires `path` to be set.
33              containerRuntimeInterface @0;
34          }
35      }
36
37      struct CreateContainerResponse {
38          containerPid @0 :UInt32;
39      }
40
41      createContainer @1 (request: CreateContainerRequest) -> (response: CreateContainerResponse);
```

Rethinking container lifecycle management

- Cap’n Proto allows us to provide clients for multiple languages
- It trims the CLI to only require a runtime (like runc) and directory for the state handling
- Allows us to use its streaming capabilities rather than using sockets for attach

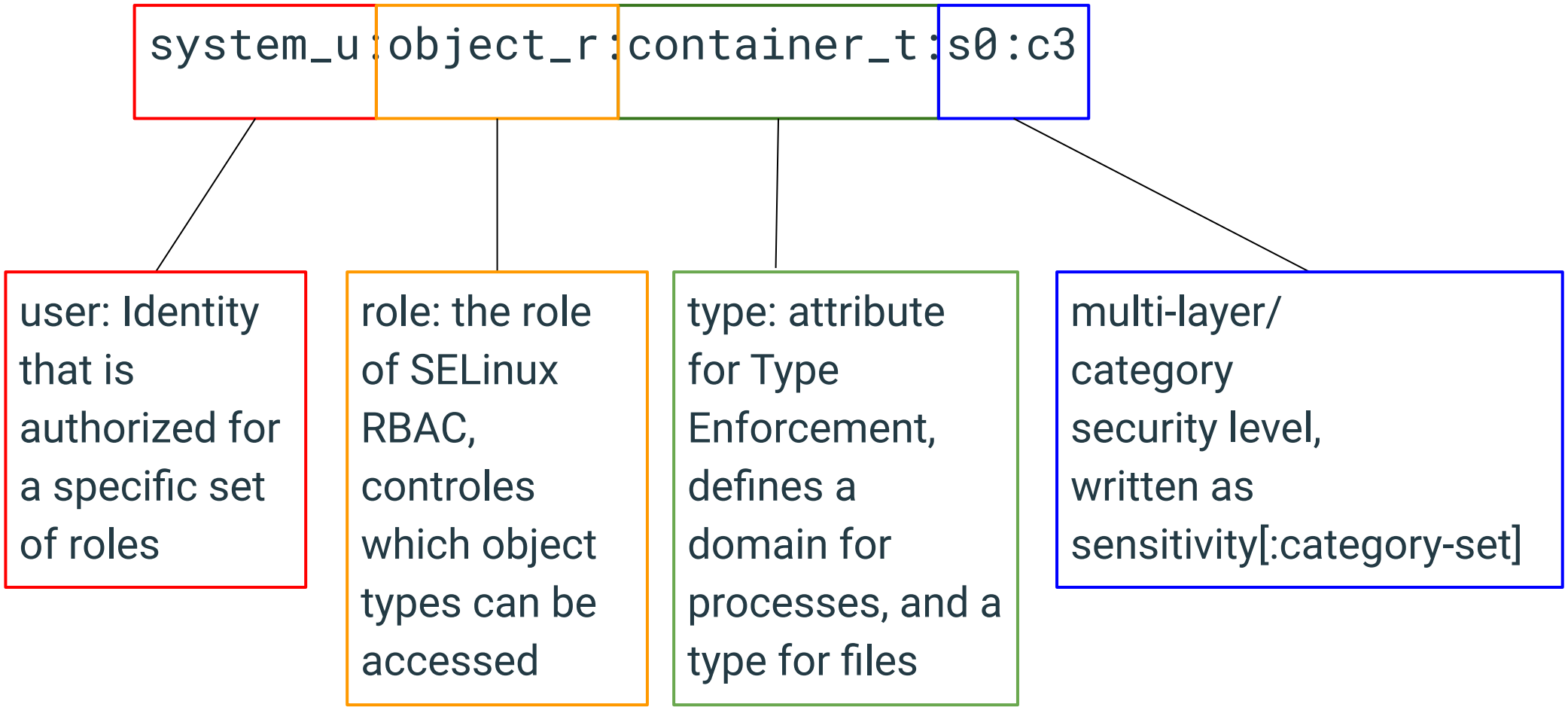
conmon-rs is not ready for production usage yet

- Working on the integration into CRI-O, because for that it’s already feature-complete
- Following up to make it ready for Podman (https://podman.io)

https://github.com/containers/conmon-rs

SELinux and Kubernetes: A Primer

seLinuxOptions have the following sub fields:



SELinux and Kubernetes: A Primer

- The provided SELinux label will be passed 1:1 from the kubelet to the container runtime (CRI-O)
- CRI-O will use the data on sandbox and container creation to pass it down to the underlying storage library: (https://github.com/containers/storage or c/storage)
  - c/storage will generate a new process and file (mount) label used within that sandbox
  - The process label will be used for the container process
  - The mount label will be used for the rootfs and volume mounts (if supported)
  - Note: the mount label will only be added to a volume in the volume plugin requests it. hostPath volumes **do not** request the path be relabeled.

```
securityContext:
  seLinuxOptions:
    level: s0:c3
```

```
> k exec -it test-pod -- ls -Z /etc/os-release
system_u:object_r:container_file_t:s0:c3
```

SELinux and Kubernetes: A Primer

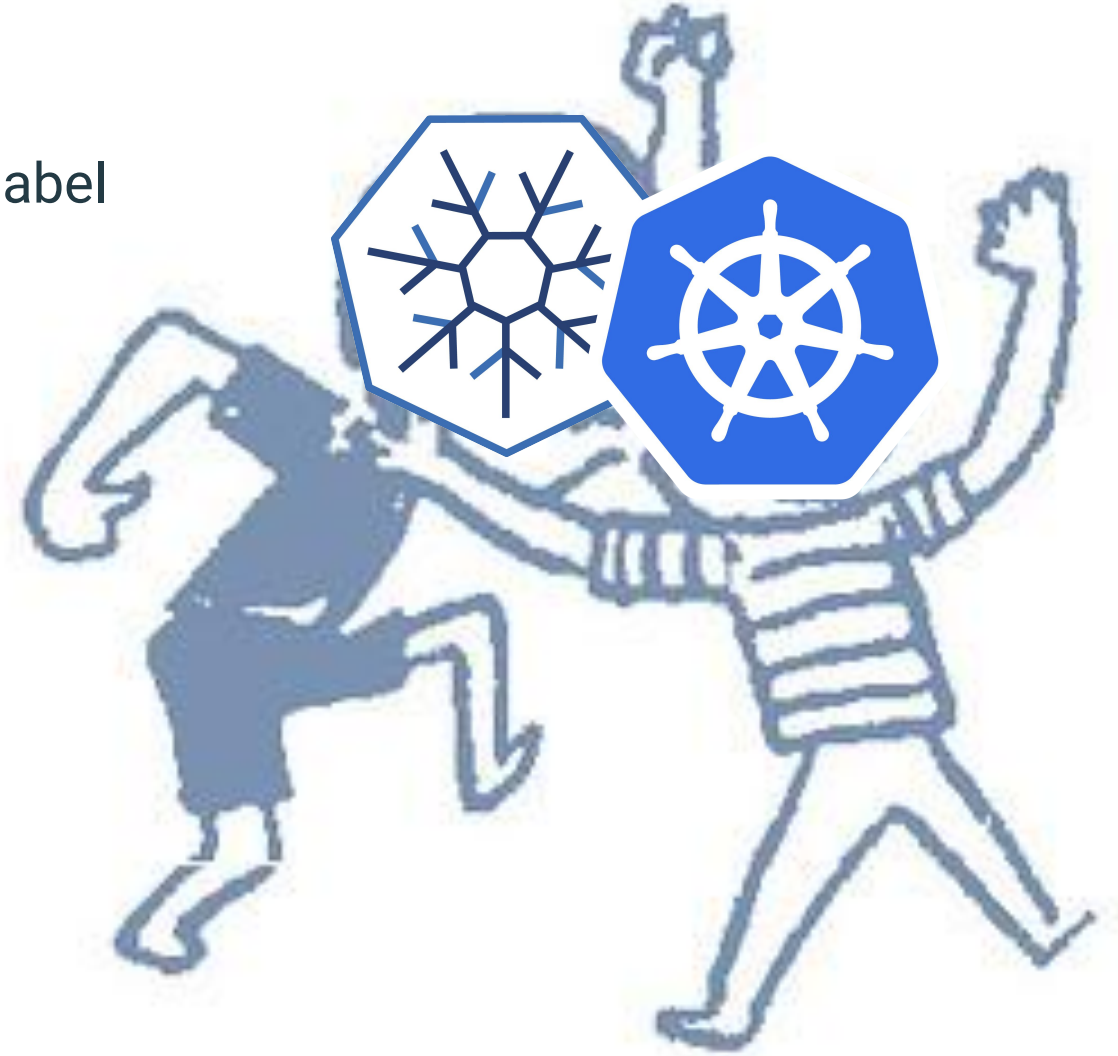
- There is a special type available for Super Privileged Containers called spc_t
  - Basically disables SELinux for the container or pod
- spc_t is almost similar to unconfined_t, except:
  - container runtimes are allowed to transition to spc_t
  - confined processes can communicate with sockets using spc_t

- **Please don’t run unconfined containers if not absolutely necessary.**

- Distributing SELinux policies / profiles can be done using the security-profiles-operator:
  - https://sigs.k8s.io/security-profiles-operator

The Problem: Volume Relabeling

- CRI-O sometimes can’t relabel volume in time.
- Kubelet must cap request with a time limit.
- Kubelet and CRI-O bicker trying to create the specified container.



(No daemons were hurt in the making of this presentation)

More Concretely…

- Container must have a process label that can access the volume.
- Volume must be labeled to prevent other containers from accessing.
- Relabel as few times **as can be trusted**
  - Allow timely starts/restarts of containers.

Solution #1: Conditionally Skip if Correct

If the label is already correct, and the container is explicitly allowed, skip the relabel

- Pros:
  - Secure: Container is still confined.
  - Workload-defined: Can be enabled for specific containers that need it.
  - Restart-friendly: If the container is restarted, the label doesn't need to be reapplied.
  - Pod-friendly: If multiple containers in a pod try to access the volume, the relabel only needs to happen once.
  - Optimizable: Volume can be labeled ahead of time
- Cons:
  - Label has to happen once. If done by CRI-O, timeouts can be incurred.
  - If a file in the volume, but not the top level, is relabeled, the container won't have access.

Solution #2: Always Skip if ‘spc_t’

Never attempt the relabel if the container’s label is ‘spc_t’

- Pros:
  - Fastest: never label, never timeout!
  - Easy: No configuration required.
  - Portable: Add any volume with no overhead.
- Cons:
  - Not secure: the container is privileged in the eyes of SELinux

Efficiency vs Security

Imagine a world in which…

There is no relabel, only an initial label on mount, that the kubelet does itself.

(see github.com/kubernetes/enhancements/issues/1710 for more information)

Enabling Seccomp by Default

- Container security is multi-layered
  - The more layers in the onion, the more secure you are
  - Seccomp restricts the syscalls that a container can make

- State of seccomp in k8s
  - Default is Unconfined (no seccomp!!!)
  - SeccompDefault Feature gate is alpha
  - Blog https://kubernetes.io/blog/2021/08/25/seccomp-default/

Enabling Seccomp by Default

- CRI-O defaulting to seccomp in 1.24
  - seccomp_use_default_when_empty = true
  - Working with upstream k8s to graduate SeccompDefault to beta and GA

Looking Forward

- Multi storage support
  - Selection based on runtime classes
- Cosign verification
- Checkpoint & Restore
  - https://github.com/kubernetes/enhancements/pull/1990
- OpenTelemetry traces

CRI-O: Secure, Performant, and Boring as Ever!

Thank you for listening to our talk!