In a Container, No One Can Hear You Scream...

Next Generation Process Isolation

Kubecon EU, September 2020

@sublimino and @controlplaneio


I’m:
- Andy
- Dev-like
- Sec-ish
- Ops-y

Course author:

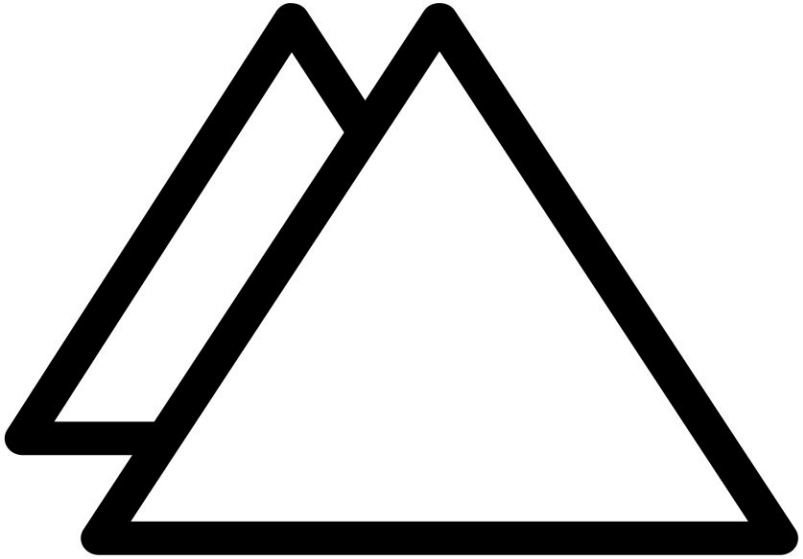
- SANS SEC584
  - Securing Cloud Native: Containers and K8s
- O’Reilly
- ControlPlane

Trainer:

- Hashicorp
- Docker

I’m:

- Andy
- Dev-like
- Sec-ish
- Ops-y



controlplane


In Containers no one can hear you scream.

TL;DR

- We ❤️ Linux
- All software has bugs, C has the worst bugs
- Golang/Rustlang are safer
- MicroVMs are always sandboxes, but not always “VMs”
- Sandboxes combines containers and VMs for a “best of both” approach
- OCI and CRI still relevant for interoperability

Sandboxing Tech

- Virtualisation balances “security” with “usability”
- Sandboxes may use:
  - LSMs and kernel modules
  - containers and capabilities
  - dedicated software
  - hardware-assisted virtualisation
- Sandboxing is better in type safe languages (Rust, Golang etc)

Glossary

- **untrusted workload**: cannot be certified as safe to run
  - public cloud, hosted CI/CD, transcoding services, vulnerable software
  - some container builds?

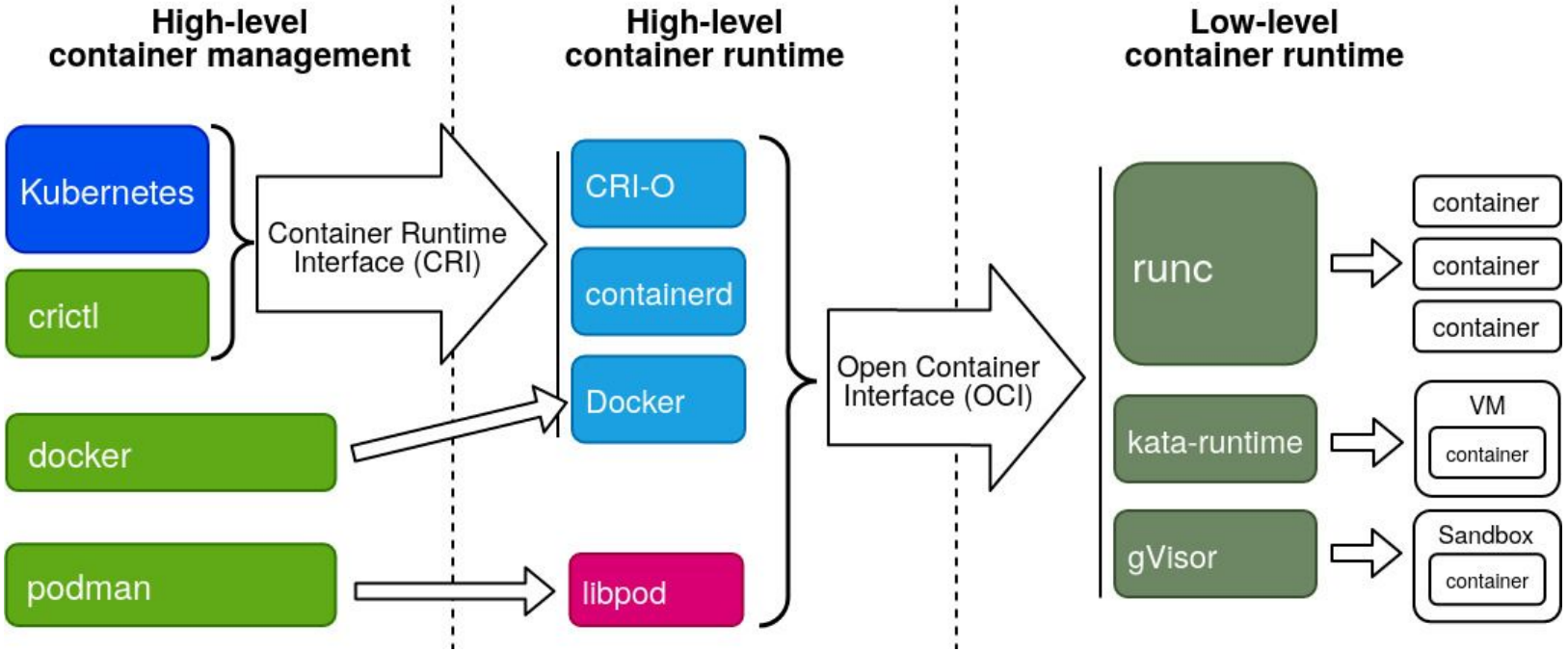
- **sensitive workload** is one who's data or code is too important to permit unauthorised read or write.
  - fraud detection systems, pricing engines, high-frequency trading algorithms.

- A single K8s workload or an entire tenant may be considered untrusted/sensitive
  - Use securityContext, admission control, PSP, and network policy
  - For advanced isolation from unknown vulnerabilities: sensitive and/or untrusted workloads may use sandboxen

controlplane

What’s wrong with containers?

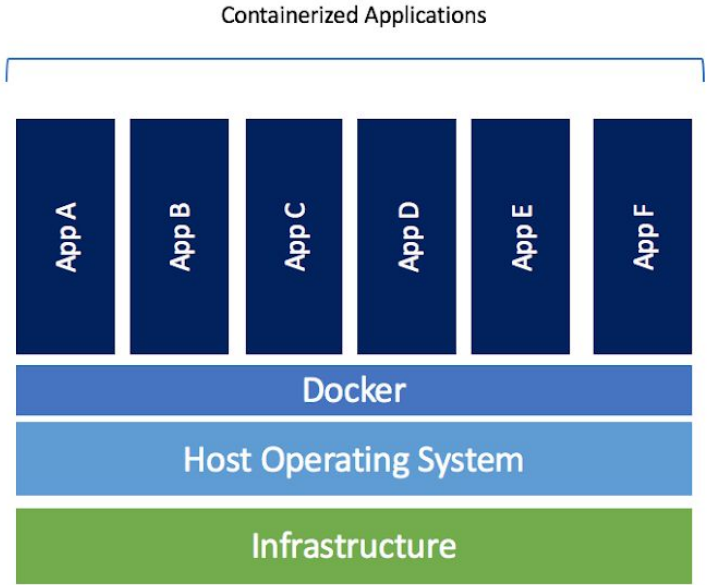
Reminder: CRI ➡️ OCI ➡️ “container”

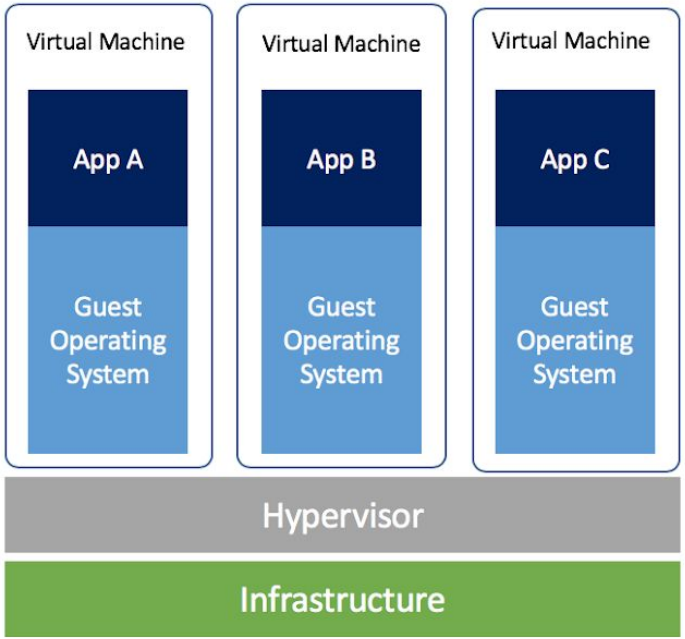


Source https://merlijn.sebrechts.be/blog/2020-01-docker-podman-kata-cri-o/

controlplane

Containers and VMs

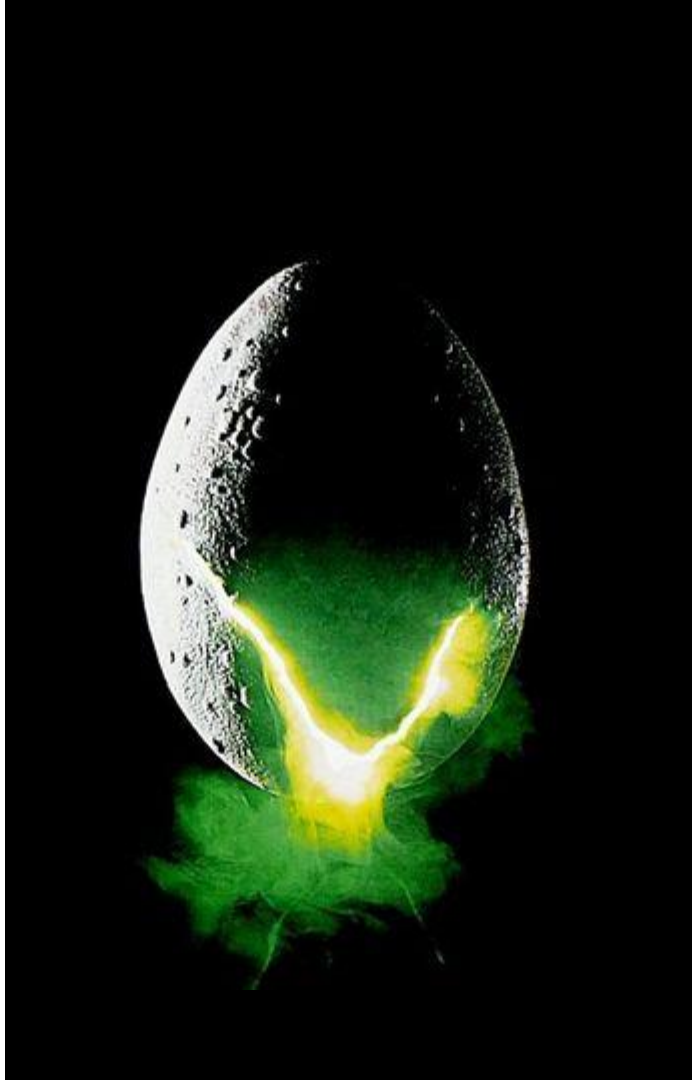




controlplane

What's wrong with containers?

- Low-level container runtimes (e.g. runc) are not inherently insecure, but have some risks
  - Sometimes leaky abstractions
  - Daemon root-fullness
- Unprivileged user namespaces are historically bug-prone
  - CVE 2013-1858: UserNS + CLONE_FS
  - CVE-2014-4014: UserNS + chmod
  - CVE-2015-1328: UserNS + OverlayFS (Ubuntu-only)
  - CVE-2018-18955: UserNS + complex ID mapping
  - See Akihiro Suda's "unresolved rootless issues" slides
- Bugs from kernel code written to assume "root" is in the host namespace
  - Assumptions...



Assumption Maketh the Ass

So:

- Rootfull container daemons: bad ❌
  - /proc/self/exe (CVE-201905736)
  - docker cp attack (CVE-2019-14271)
  - Dirtycow
- Unpriv user namespaces: contentious ⚔️

But:

💀 Lesser of two evils? 💀

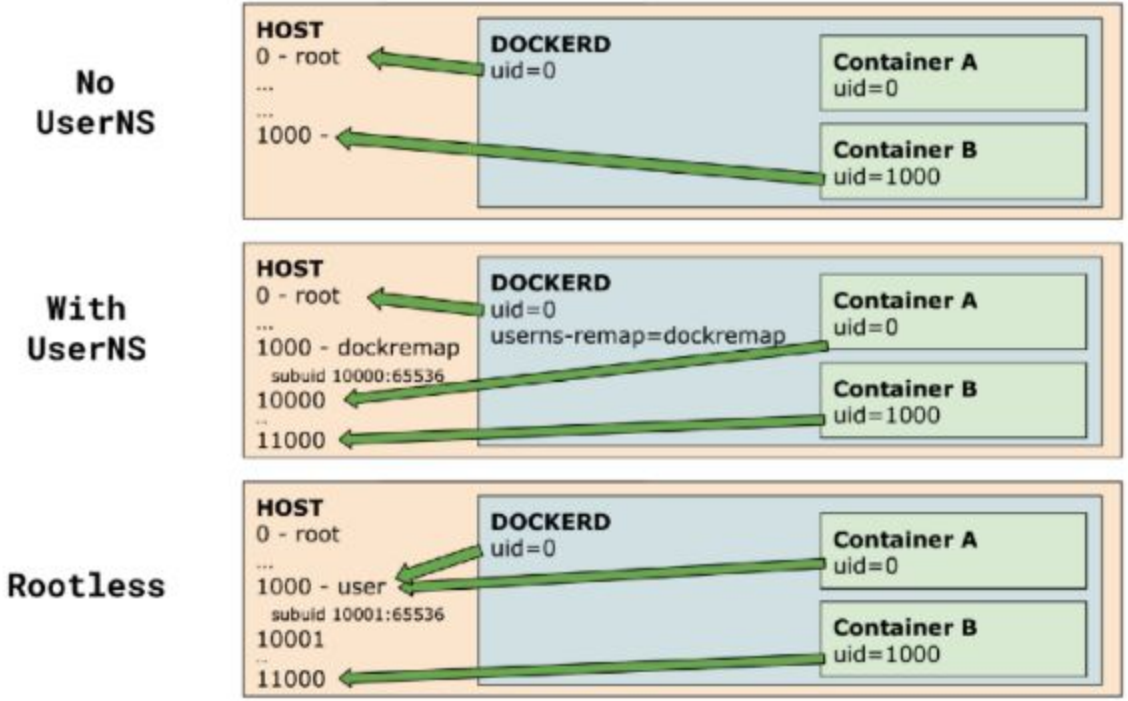
Rootlessness

Container **namespace**:

- User namespaces allow “pretend root” in a child namespace

Container **runtime**:

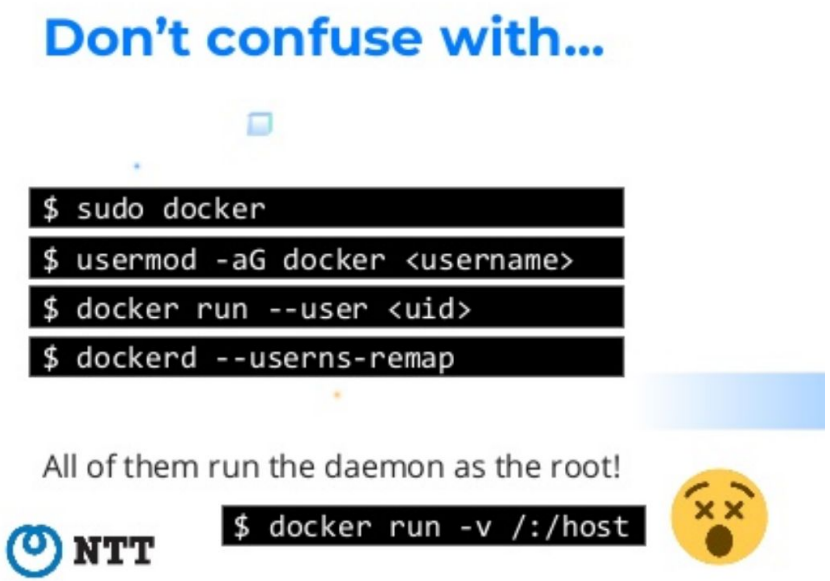
- Unprivileged user namespaces allow creation of containers by non-root users in host namespace



controlplane

Rootless Docker and Podman

- Allows root-in-userns (unprivileged in the host namespace) to create containers
- Bridging to host network namespace denied (owned by host root)
  - “Unprivileged” *slirp4netns* (with seccomp) creates virtual network device
  - Performance is good
- NFS filesystems [tricky]



Source [DockerCon 2020] Hardening Docker daemon with Rootless Mode

controlplane

Limitation of Rootless Docker/Podman

- Podman: supports SELinux
- Docker: no AppArmor
- Both:
  - CAP_NET_BIND_SERVICE capability
  - no CRIU (snapshotting)
- Defaults that need changing:
  - ping is not supported for users with high UIDs
  - Cgroups v2 only runs under systemd
  - Cgroups v1 not supported

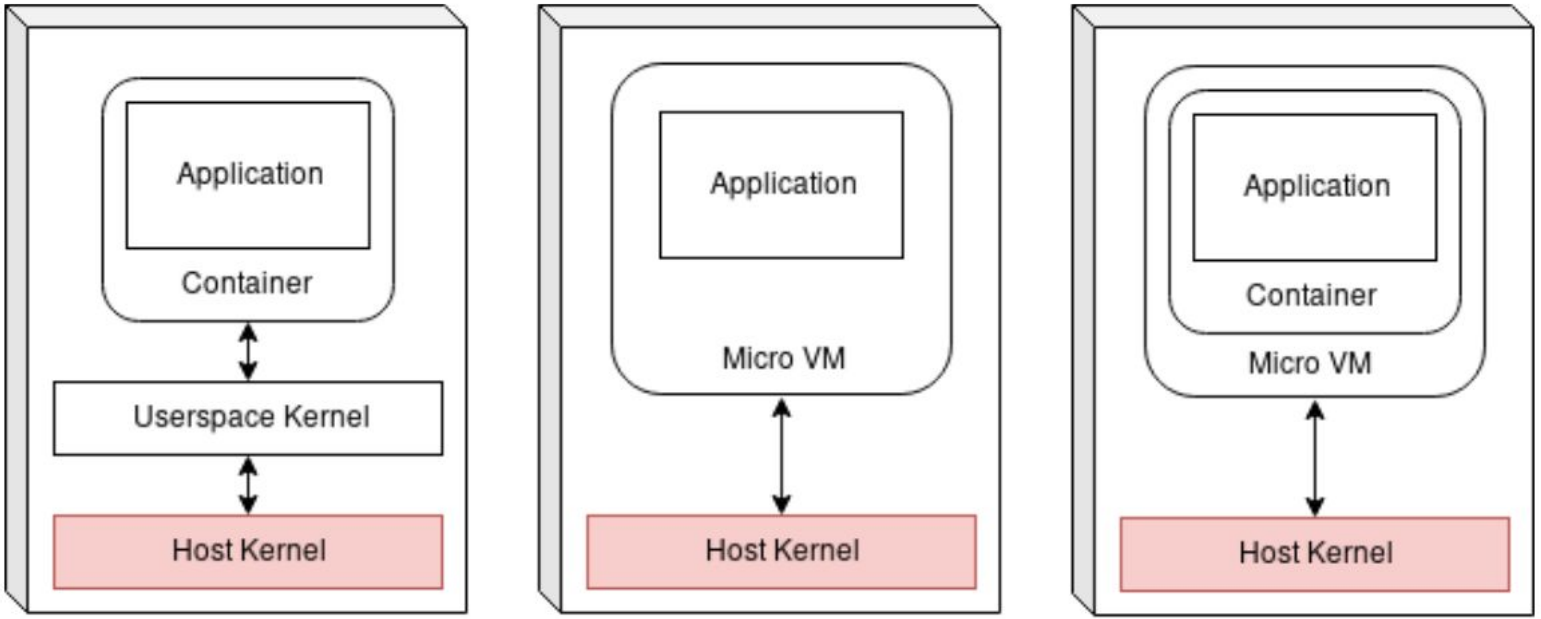
controlplane

I can haz rootless runtime?

- Docker/Moby and Podman share a lot of rootless code
- Container abstractions still sometimes leaky (/proc, devices, Kernel)
- The Linux system call interface still source of risk
- Unikernels have been tried, yet to see a success story
- Containers successful as no guest modification required

Virtual Machines & Sandboxes

Sandboxes: Mixing Containers and VMs



Source http://ceur-ws.org/Vol-2508/paper-bar.pdf

controlplane

Virtualisation tradeoffs

- “Security” vs “Performance”
- Abstractions may be costly
- Observability can be challenging
- Open source contribution a good indicators of adoption and support

History of Virtualisation

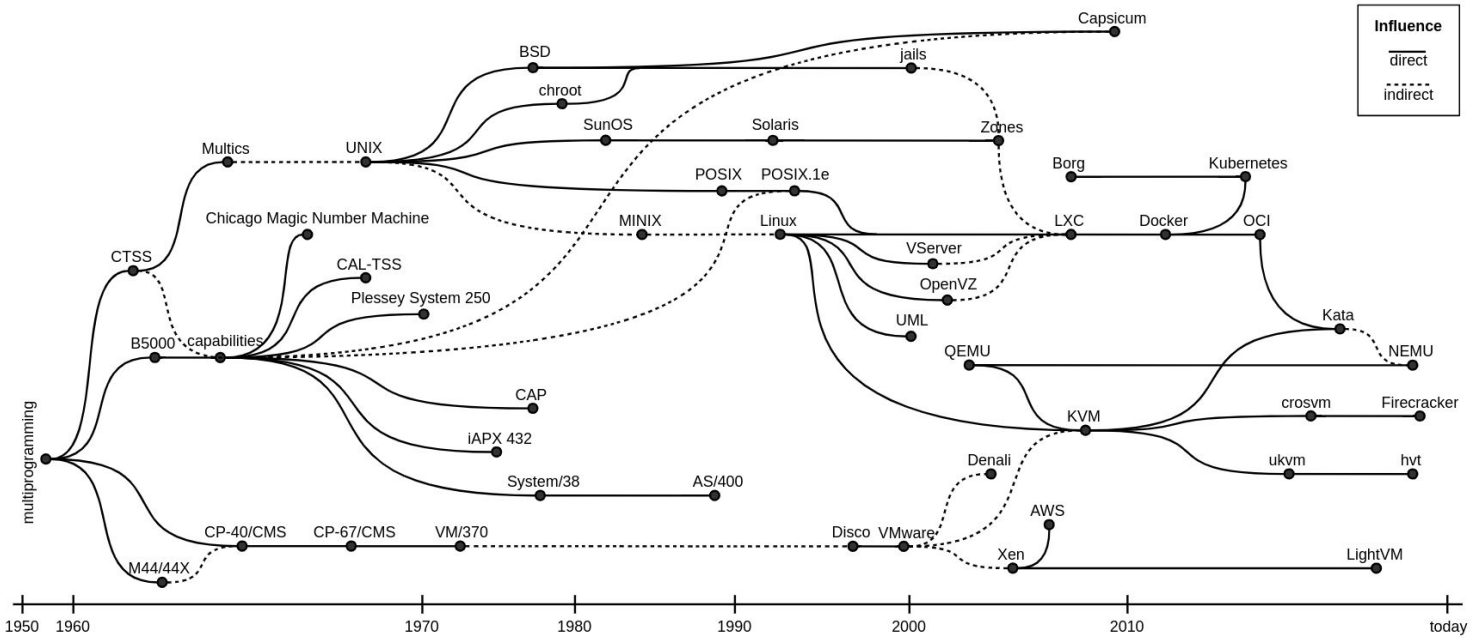


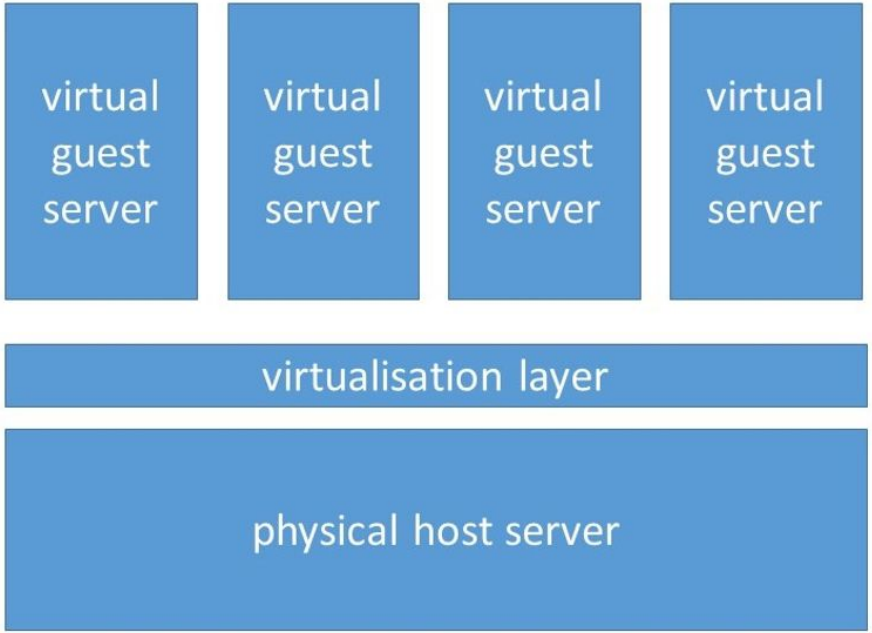
Figure 1: The evolution of virtual machines and containers.

Source [Making Containers More Isolated: An Overview of Sandboxed Container Technologies]

controlplane

Virtualisation

- Subdivide a host into logical “guests”
- Simulate a full machine to each guest
- Each guest runs its own kernel
- Each kernel can also run cgroups, namespaces, LSMs, capabilities etc
- Maintain full isolation between guests



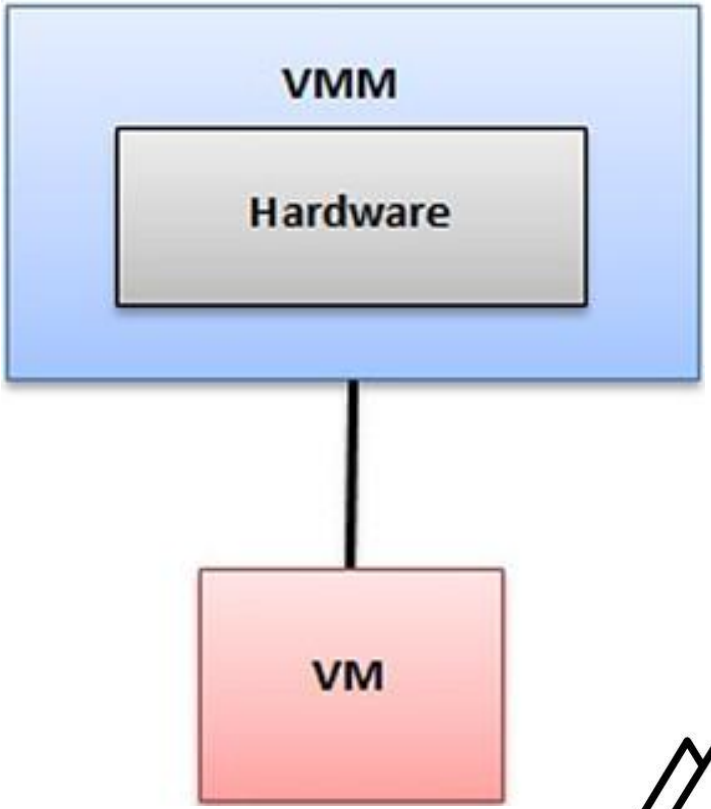
controlplane

Isolation

- To run a process
  - CPU power on
  - BIOS
  - Bootloader
  - Kernel
  - Init system
  - *<ambiguous machine code>*
  - Process

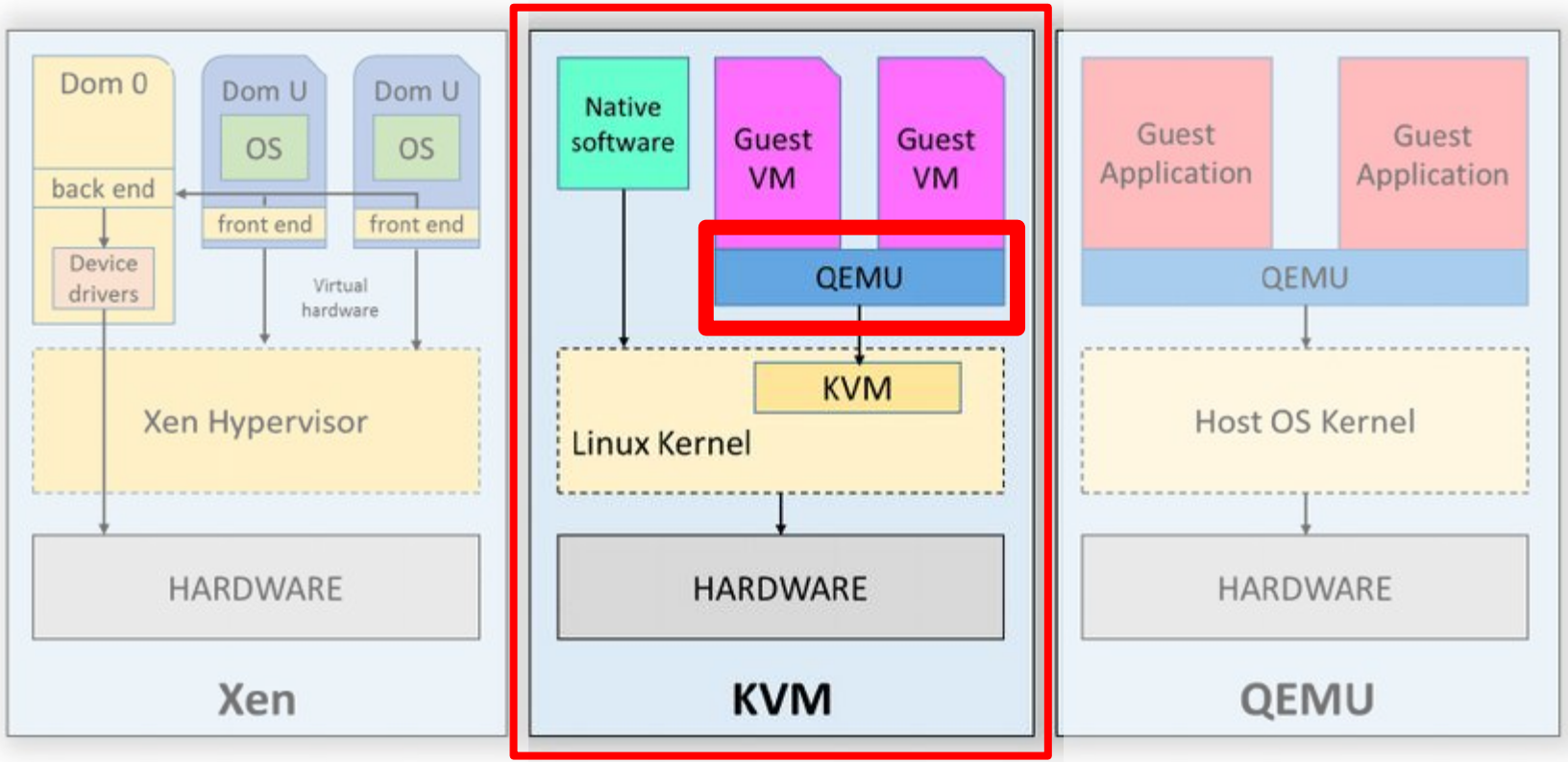
Virtual Machine Monitor

- To run a process
  - CPU power on
  - BIOS
  - Bootloader
  - Kernel
  - Init system
  - *<ambiguous machine code>*
  - Process
- To run a VM
  - VMM starts VM
  - Follow above list (ish)



controlplane

KVM vs Xen vs QEMU



controlplane

Spectrum of Isolation

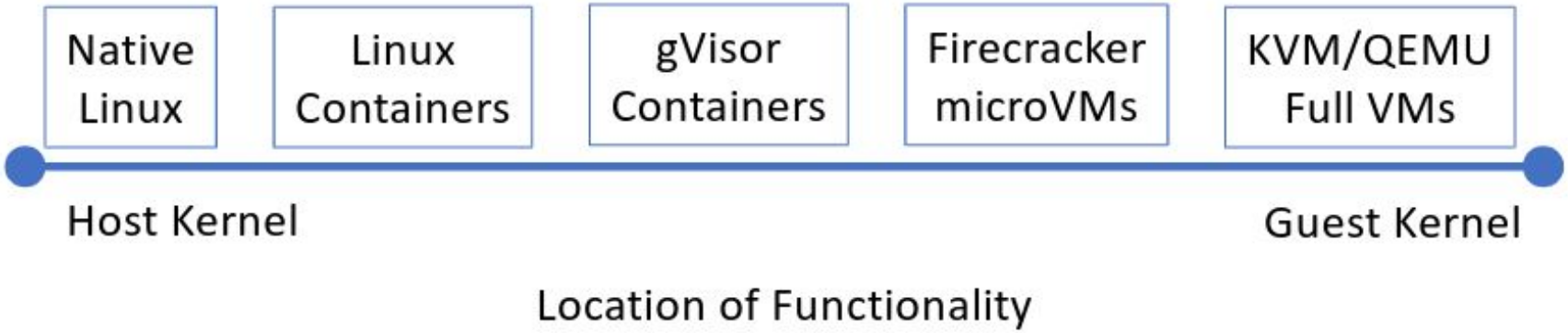


Figure 1. Spectrum of OS functionality location in isolation platforms.

Source https://research.cs.wisc.edu/multifacet/papers/vee20_blending.pdf

controlplane

gVisor vs Firecracker vs Kata vs Nabla

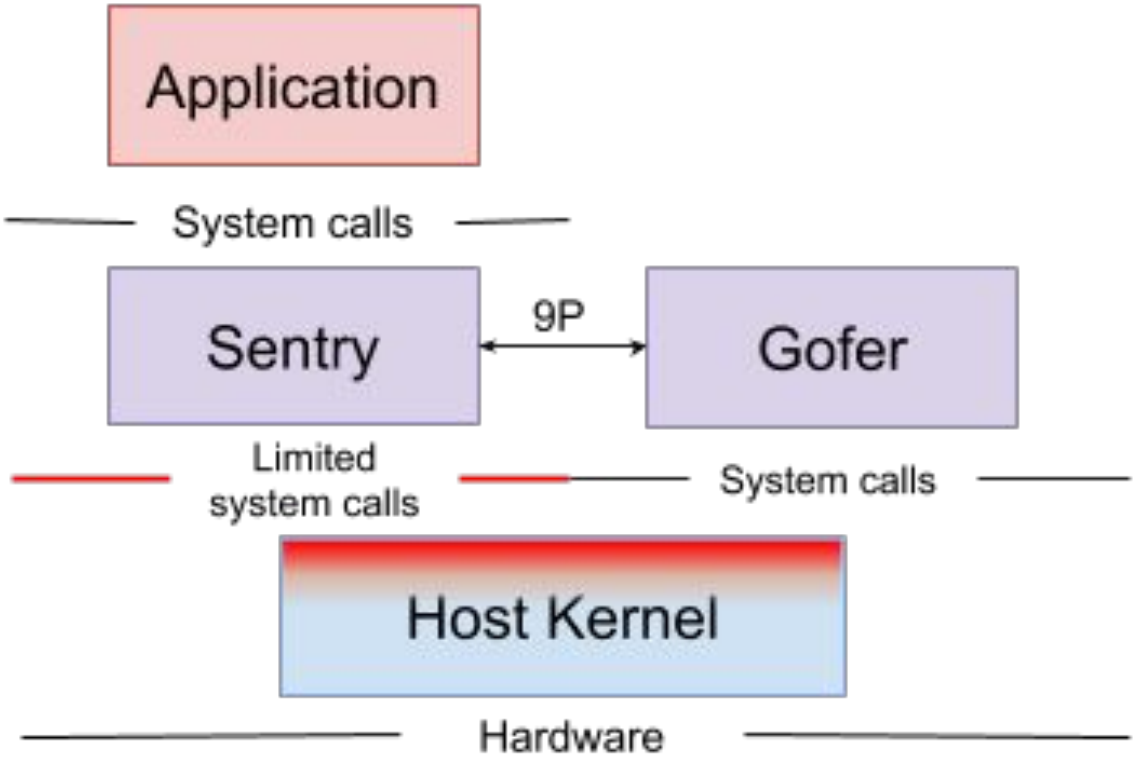
	Supported container platforms	Dedicated guest kernel	Support different guest kernels	Open source	Hot-plug	Direct access to HW	Required hypervisors	Backed by
Nabla	Docker, K8s	Yes	Yes	Yes	No	No	None	IBM
gVisor	Docker, K8s	Yes	No	Yes	No	No	None	Google
Firecracker	Not yet	Yes	Yes	Yes	No	No	KVM	Amazon
Kata	Docker, K8s	Yes	Yes	Yes	Yes	Yes	KVM or Xen	OpenStack

Source [Making Containers More Isolated: An Overview of Sandboxed Container Technologies]

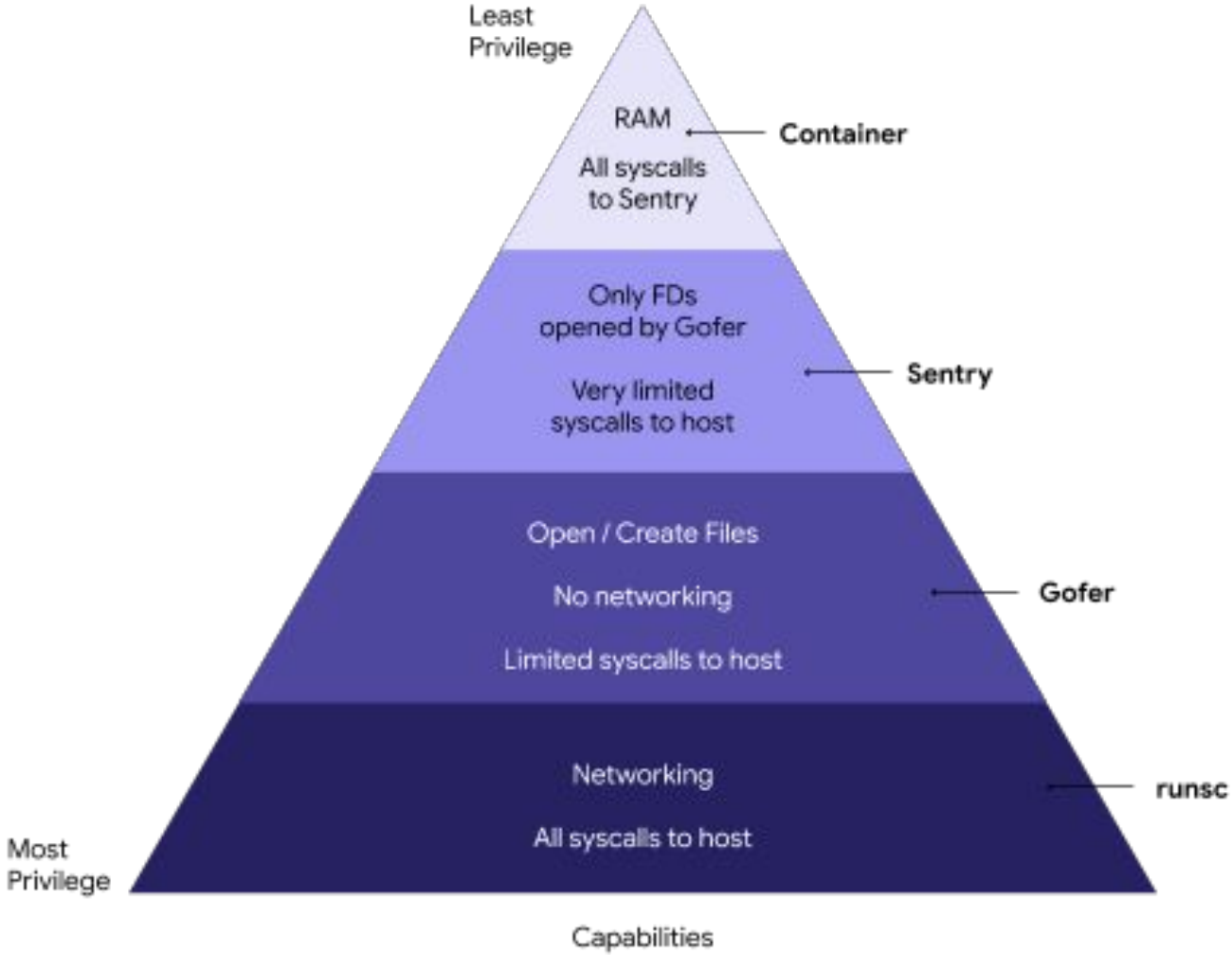
controlplane

gVisor

gVisor may be thought of as either a **merged guest kernel and VMM**, or as **seccomp on steroids**



controlplane



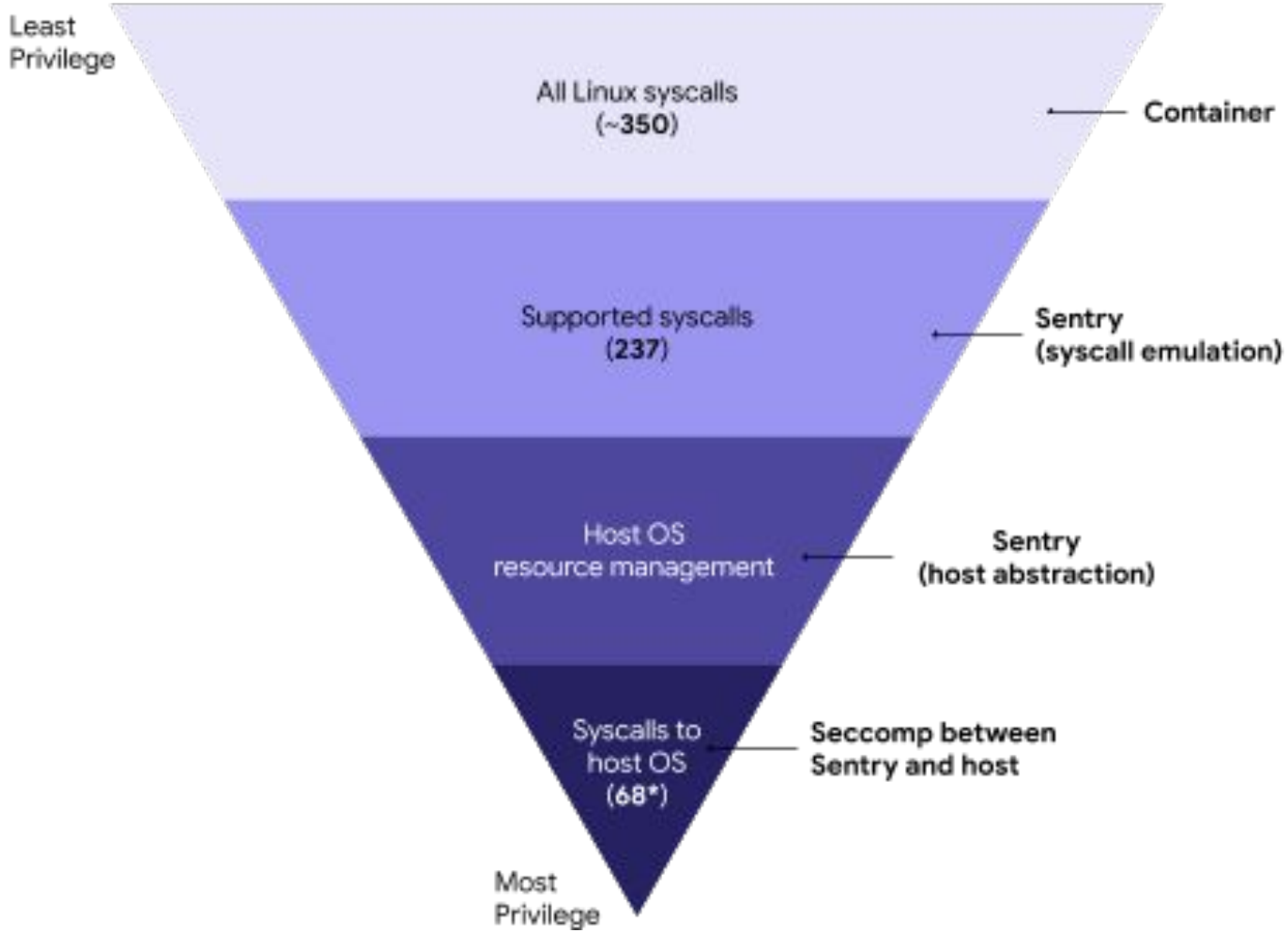
controlplane

gVisor Sentry

```
for (;;) {
  ptrace(PTRACE_SYSEMU, pid, 0, 0);
  waitpid(pid, 0, 0);

  struct user_regs_struct regs;
  ptrace(PTRACE_GETREGS, pid, 0, &regs);

  switch (regs.orig_rax) {
    case OS_read:
      /* ... */

    case OS_write:
      /* ... */

    case OS_open:
      /* ... */

    case OS_exit:
      /* ... */

    /* ... and so on ... */
  }
}
```

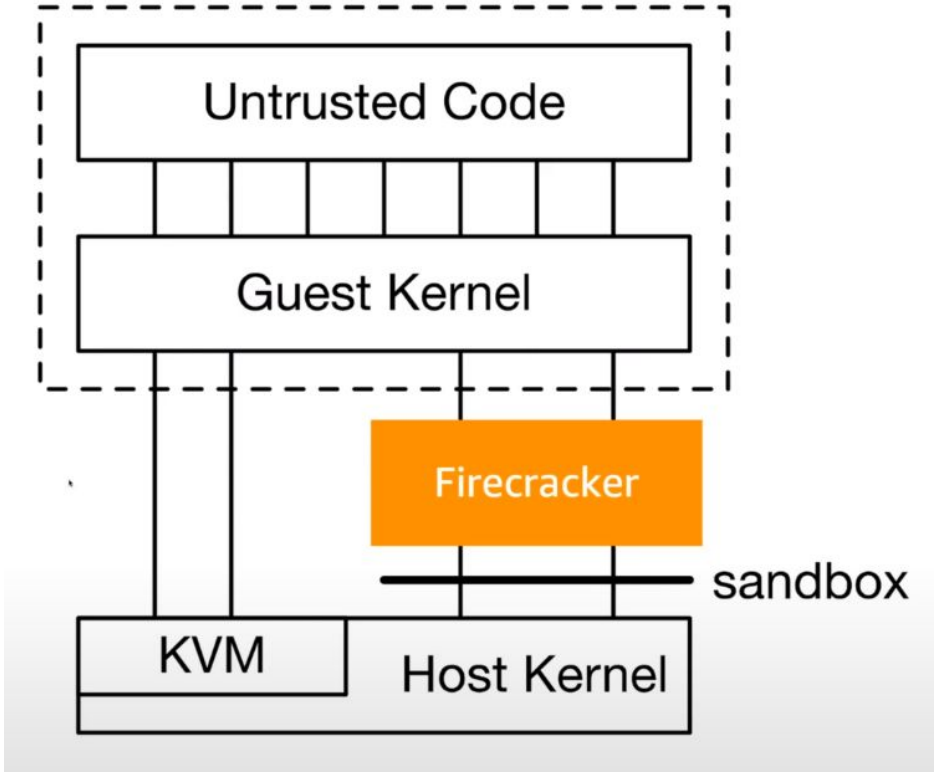
Source [Resource Sharing]

controlplane



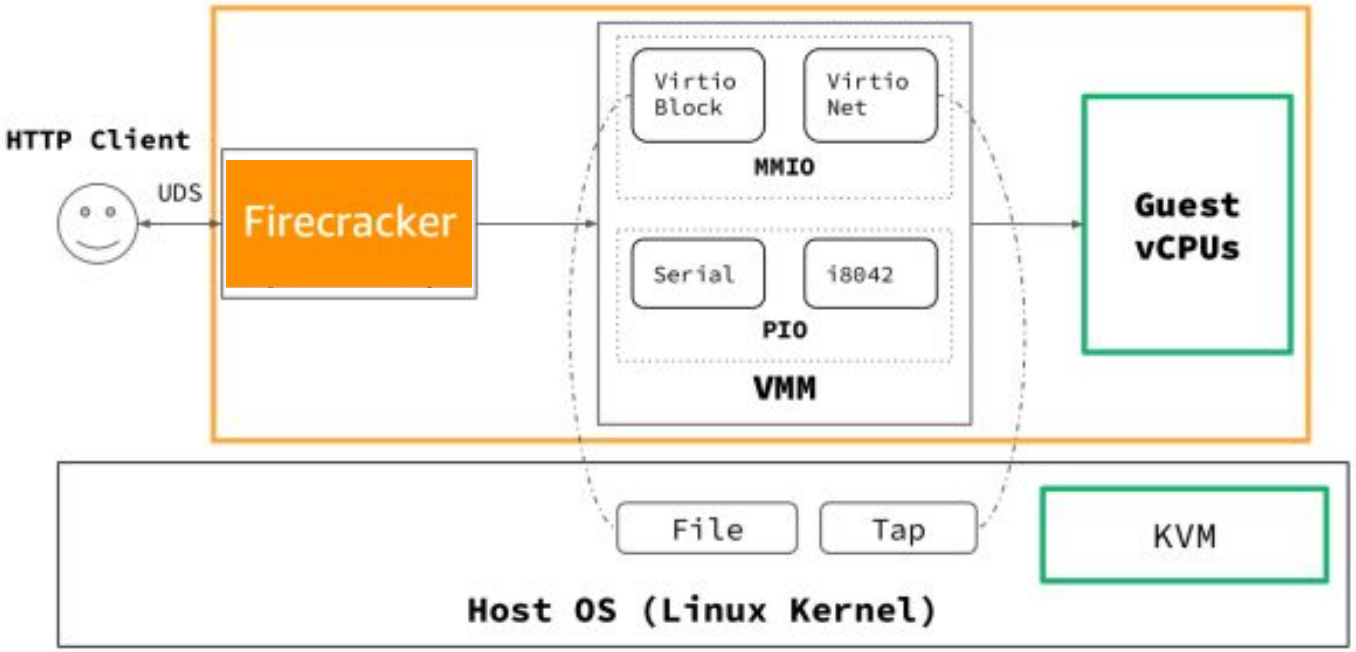
controlplane

Firecracker



controlplane

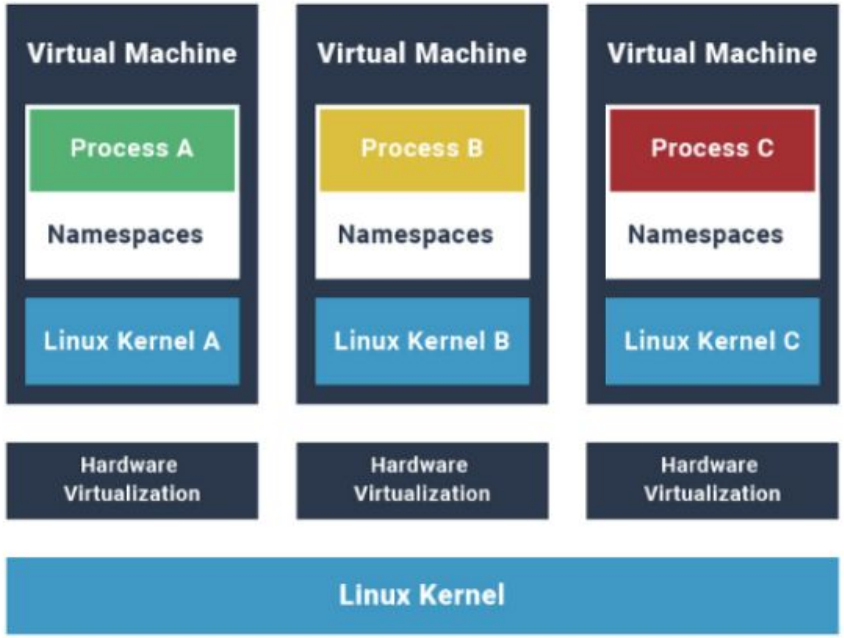
Firecracker Device Model



Source [chromiumos/platform/crosvm - Git at Google]

controlplane

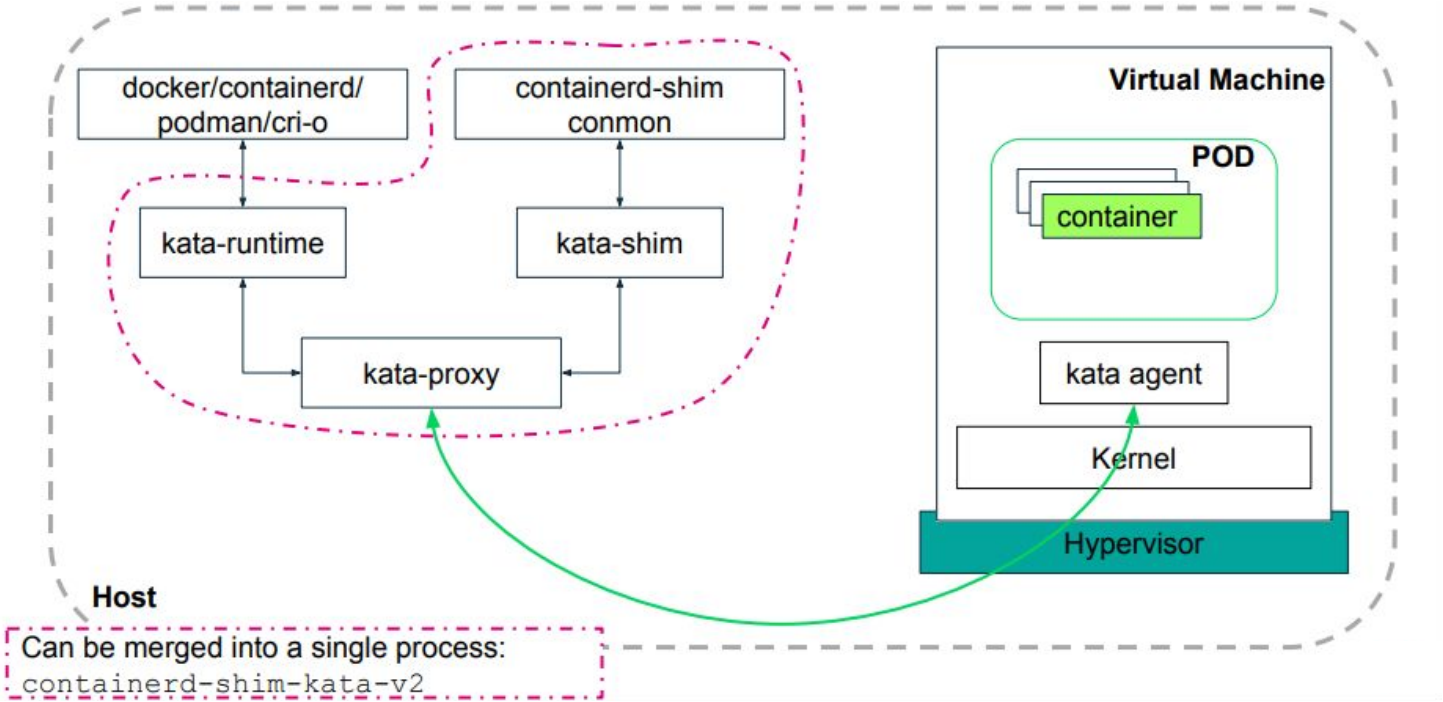
Kata Containers



Additional isolation with a lightweight VM and individual kernels

controlplane

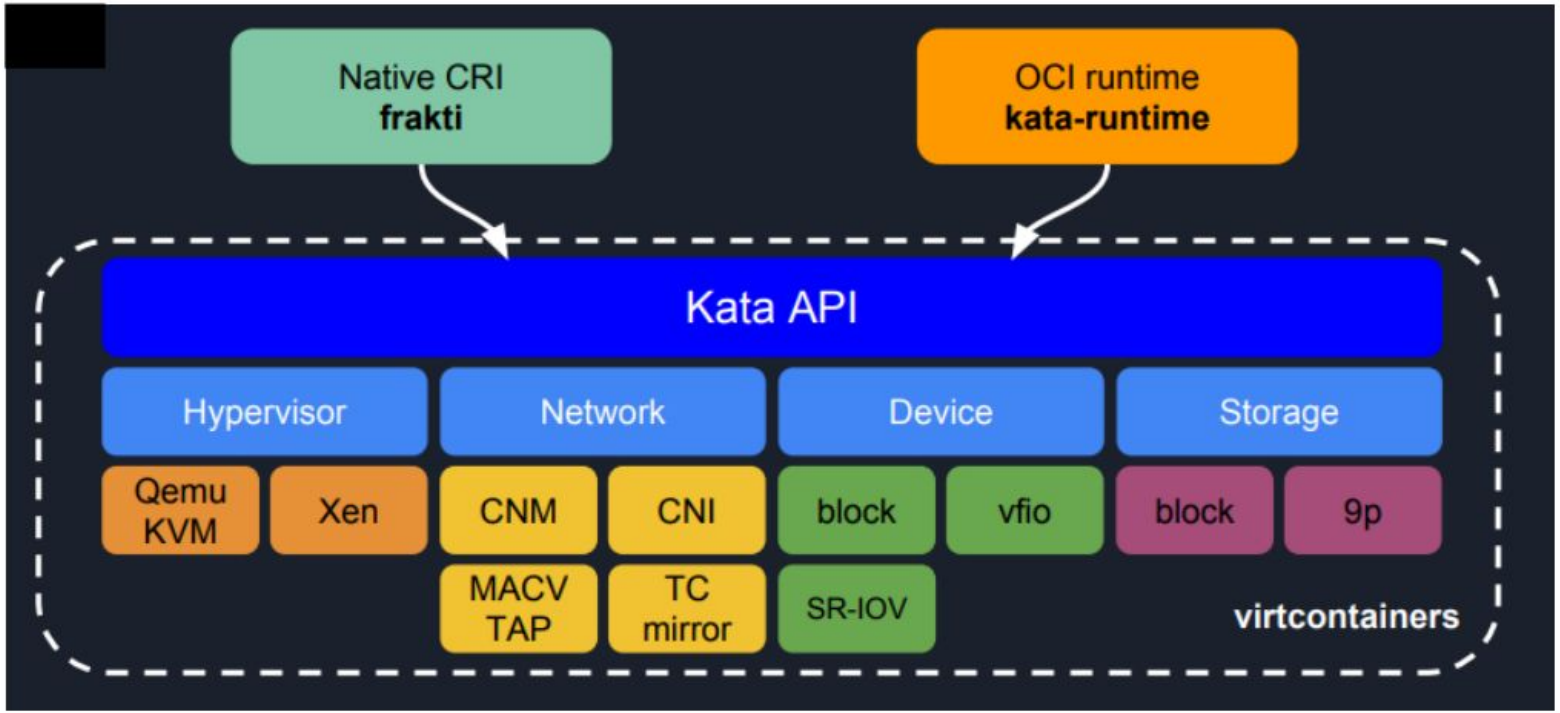
Kata Containers



Source [About Kata Containers | Kata Containers]

controlplane

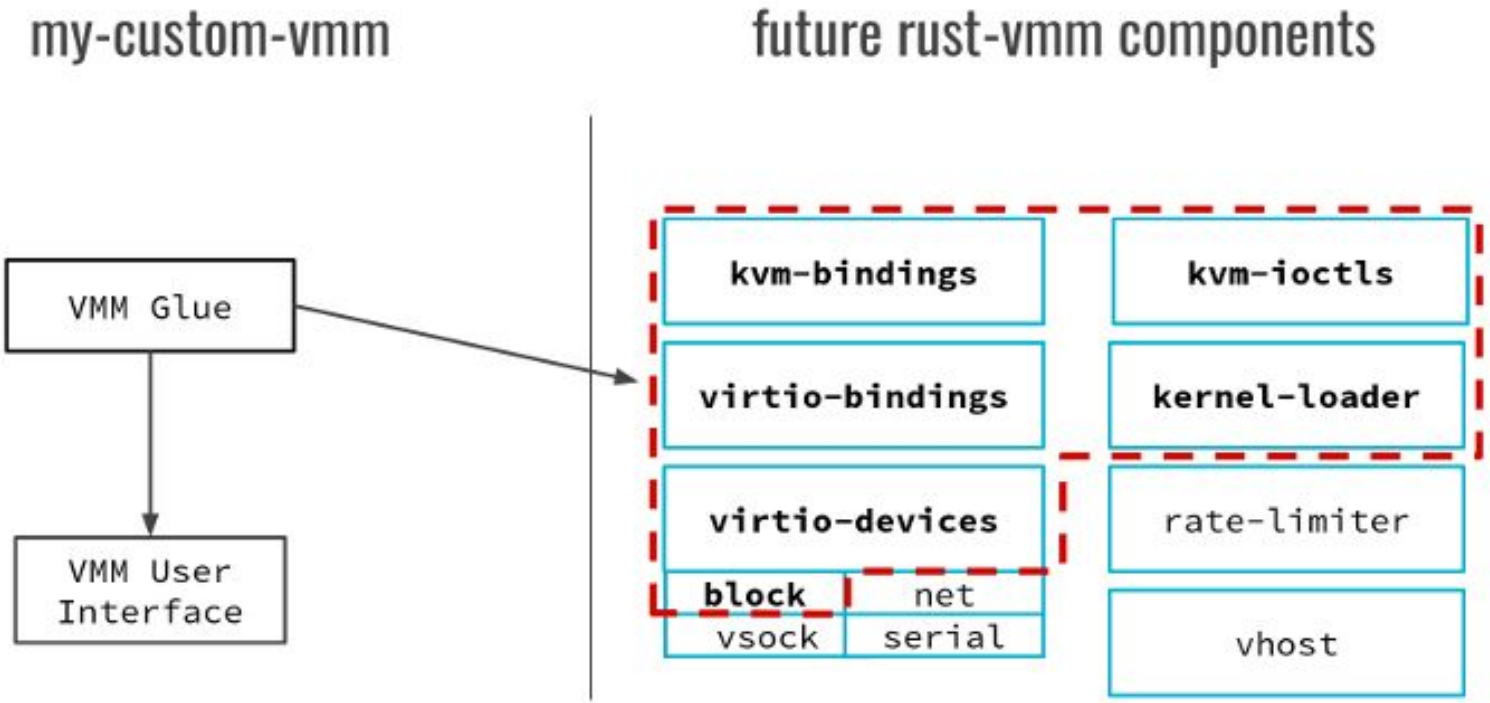
Kata Containers



Source [About Kata Containers | Kata Containers]

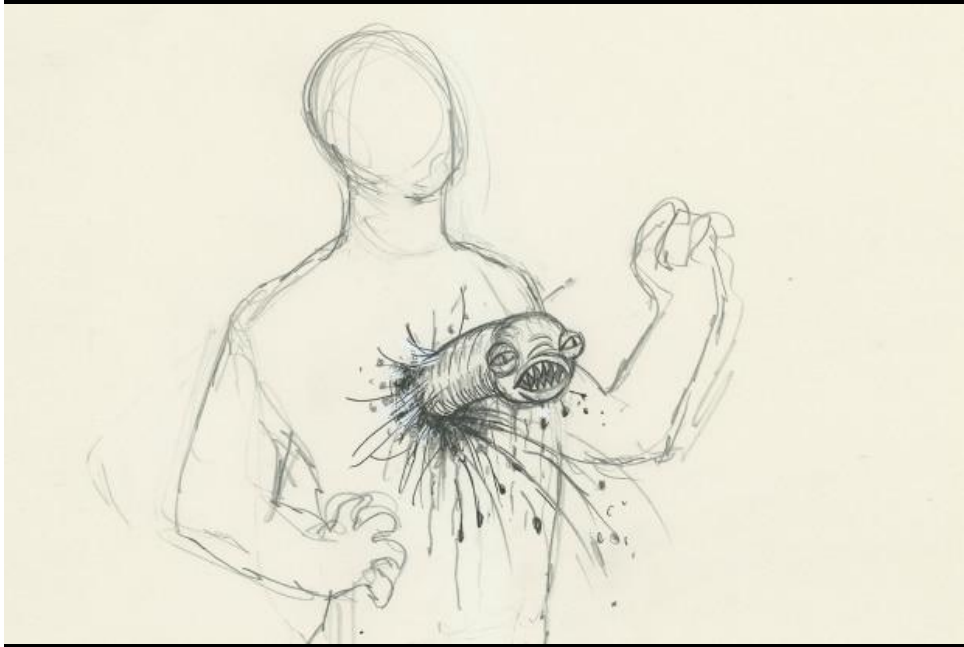
controlplane

Honourable mention: rust-vmm



controlplane

Escaping Sandboxes



Run it

Docker & Kubernetes RuntimeClass

- **`docker run -it --runtime=runsc`** starts a sandboxed OCI container
- Different container runtimes can co-tenant on same cluster/node etc
- **`spec.template.spec.runtimeClassName`** can target a sandbox for a K8s workload via CRI
- K8s doesn't distinguish between sandboxes yet
- Node affinity and toleration can be set for the K8s scheduler

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  runtimeClassName: crio-kata-qemu-lite
  # ...
```

```yaml
apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: crio-kata-qemu-lite
  # RuntimeClass is a non-namespaced resource
handler: qemu-lite  # The name of the corresponding CRI configuration
```

What are the risks of next gen proc iso?

controlplane

What should I use?

IT DEPENDS - threat model your workloads first

- Debugging rogue processes may be difficult
- General, minor performance overhead (~50-200ms startup)
  - https://gvisor.dev/docs/architecture_guide/performance/
  - https://www.usenix.org/conference/nsdi20/presentation/agache
- May be limited by platform and/or nested virtualisation options
- Abstraction from Linux Syscall surface e.g. how much of the kernel is utilised?

Conclusion

- Unless you have problems: containers are just fine!
- High sensitivity workloads and data may need more isolating
- Pick a supported cloud provider offering:
  - gVisor runs in GKE
  - Firecracker runs in Fargate
- Or Kata runs anywhere virtualisation is supported
- rust-vmm means there will be many more runtimes