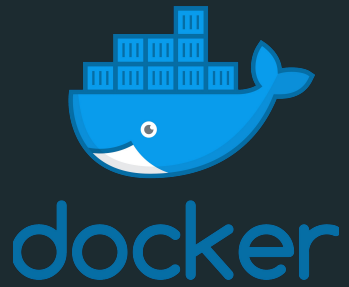
How to Choose a Kubernetes Runtime

Justin Cormack, Docker



Who am I?

Engineer at Docker in Cambridge, UK.

Work on security, LinuxKit, container infrastructure

@justincormack

What is a runtime?

In the beginning there was runc

- evolved from Docker's libcontainer
- Open Container Initiative https://github.com/opencontainers/runc
- spun out to OCI standards body in Linux Foundation in 2015
- Just supports Linux

2018: The Year of Choice

- VM containers: Kata Containers 1.0
- Emulation containers: launch of gVisor
- Windows containers go OCI with runhcs
- Unikernel containers: Nabla
- Systemd containers: Systemd does OCI
- ChromeOS containers: run_oci on ChromeOS
- Four OCI platforms so far: Linux, Windows, Solaris, VM

Why choose a different runtime?

Approximate justification "untrusted" workloads

- Originally proposed by Tim Allclair as the "Sandboxes API"
- This was a boolean for standard or strong isolation
- Rejected by SIG Node
- Replaced by RuntimeClass string
- In alpha, but not really being used
- CRI still exporting an untrusted workload runtime

docker

Untrusted?

What is untrusted code?

- Code that deals with hostile untrusted input that is insecure (eg media)
- Untrusted vendor code eg proprietary code
- If you are a hosting provider, all code is untrusted!

Internal code is not untrusted!

- If you do not trust the code from your developers, fix that
- Code quality should be appropriate to your security requirements
- Security testing, code review should be applied
- There are limited circumstances where it may make sense to treat internal code as untrusted...

Do containers contain?

How contained is a container?

- All code has bugs, including containment
- General kernel bugs can allow escape
- Misconfiguration is also a big issue
- Pods can involve sharing resources with different privileges

All these except the last have been issues. Misconfiguration is by far the largest. Kernel bugs that allow container escape are fairly common. With Docker we blocked about half of these with a seccomp policy but Kubernetes does not yet do this by default. There has been one known runc bug that could lead to container escape.

Mitigations

- seccomp (not in Kubernetes by default, yet!)
- SELinux, Apparmor, other LSMs
- update your kernel regularly to upstream stable kernels
- locked down minimal host OS
- drop all capabilities
- do not run containers as root
- use Pod Security Policy to enforce
- re-pave containers and hosts regularly

For critical use cases this *may* not be sufficient... kernel bugs can bypass

Alternative runtimes

Kata containers

- VM runtime
- reached 1.0 milestone this year
- boot a small stripped down Linux VM for each pod
- uses Qemu/KVM or Qemu-lite/KVM as hypervisor
- requires bare metal hardware for virtualization extensions
- some extra memory overhead from native containers
- used in production at Alibaba, JD.com

docker

gVisor

- emulation based isolation
- emulate Linux system call API with Go code
- not complete yet! But good enough to use for most applications.
- the sandbox is itself sandboxed with seccomp and dropped capabilities
- currently two backends, ptrace and KVM
- KVM does not run a Linux kernel, just stubs using virt. High performance but requires bare metal machine. Recommended.
- Used in production for Google App Engine

Kata and gVisor ready for production

- both really require virtualisation support ie bare metal
- do not use nested virtualisation, not a security boundary
- AWS bare metal, on prem bare metal, Packet.net, other providers
- Use of VMs everywhere for splitting machines means that we can't really use the hardware support for security.
- Running containers on VMs considered harmful.

Nabla: unikernel runtime

- IBM research project
- requires you rewrite your application using a library OS
- therefore do not use much Linux functionality
- eg you implement your own TCP stack for your application, in a similar way to how gVisor implements a userspace TCP stack

Secure?

- If you are using KVM, you are running a hypervisor on Linux
- Even with seccomp you are running a locked down Linux
- gVisor currently requires Linux host
- CVE-2018-17182 only required clone() and mmap() to exploit (although it is difficult with just those)
- there are attempts to measure exploitable use of kernel in these frameworks, eg function usage, common paths
- CVE-2017-5715 Spectre allowed reading hypervisor memory
- speculative execution and side channel attacks are problematic
- rowhammer
- running untrusted code is a hard problem!
- buy more computers for better isolation...

Two talks later with more on this

- Kata and gVisor: A Quantitative Comparison - Xu Wang 3:40pm
- Container Security and Multi-Tenancy Tales from Kata and Nabla - Ricardo Aravena, Branch Metrics & James Bottomley 4:30pm

More runtimes...

runc alternatives

- Linux OCI runtime in Rust https://github.com/oracle/railcar
- Linux C++ OCI runtime for ChromeOS https://chromium.googlesource.com/chromiumos/platform2/+/HEAD/run_oci
- Systemd-nspawn OCI runtime support

docker

Firecracker

- Amazon's new Firecracker VMM is KVM without Qemu, based on CrosVM from ChromeOS
- Qemu drivers have been source of many security issues that allow escape for a VM
- Firecracker is seccomp sandboxed
- written in Rust not C
- minimal device drivers
- much safer than Qemu/KVM
- containerd integration in progress
  https://github.com/firecracker-microvm/firecracker-containerd

docker

Other operating systems

- Windows runhcs is runc compatible
  - Windows process isolation (requires same kernel version)
  - Windows Hyper-V isolation
  - LCOW for Linux containers with Hyper-V
- Solaris runtime has gone silent this year
- FreeBSD had a Docker integration just before OCI, some interest in reviving but nothing concrete
- OSX, OpenBSD?

So you want to write a runtime?

- understand the platform and system calls
- read the OCI spec
- understand the runc invocations
- hack!

Summary

First de-privilege your applications

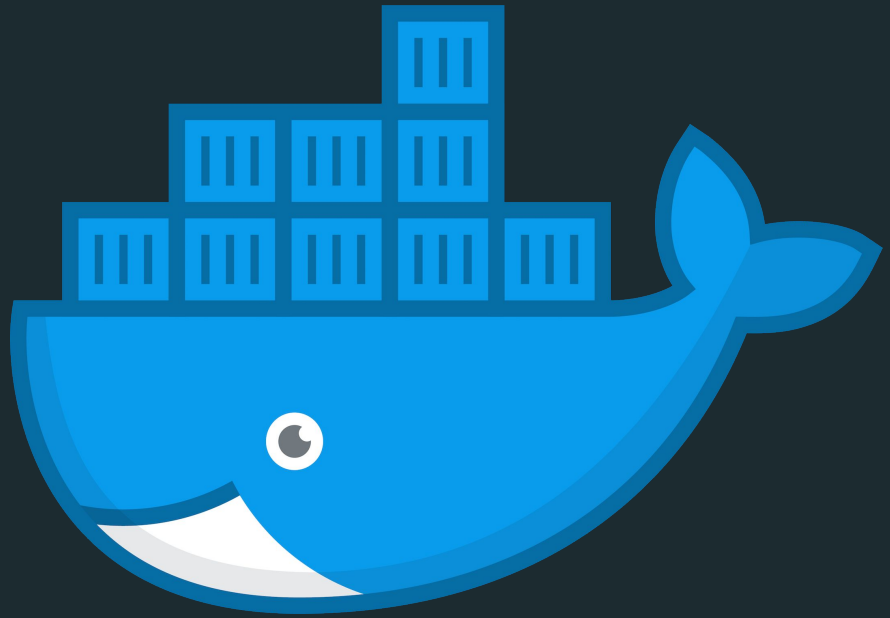
- most of the reasons applications used to run as root do not apply in containers
- the container runtime does most of the work for you
- things like privileged ports are obsolete
- applications should run with no capabilities, as non root, read only filesystem
- this applies whether you trust the code or not, eg how AWS Lambda runs
- Even if you choose a different runtime do this first

If you want to run untrusted code

- do not if at all possible
- this is a huge undertaking
- build a security team
- understand the runtimes, and the configuration
- you need to lock many things down in many places
- configuration risks are very complex
- the standard runtimes are unlikely to be suitable
- restrict configurations
- run on bare metal so you can use VT APIs
- the price of security is eternal vigilance

If you just want extra security

- once you have done all the hardening steps
- static analysis on your applications
- intrusion detection
- isolation of applications with similar exposure onto same hosts
- control of launch configuration eg ingress controller
- you may want to consider further isolation via runtimes



THANK YOU