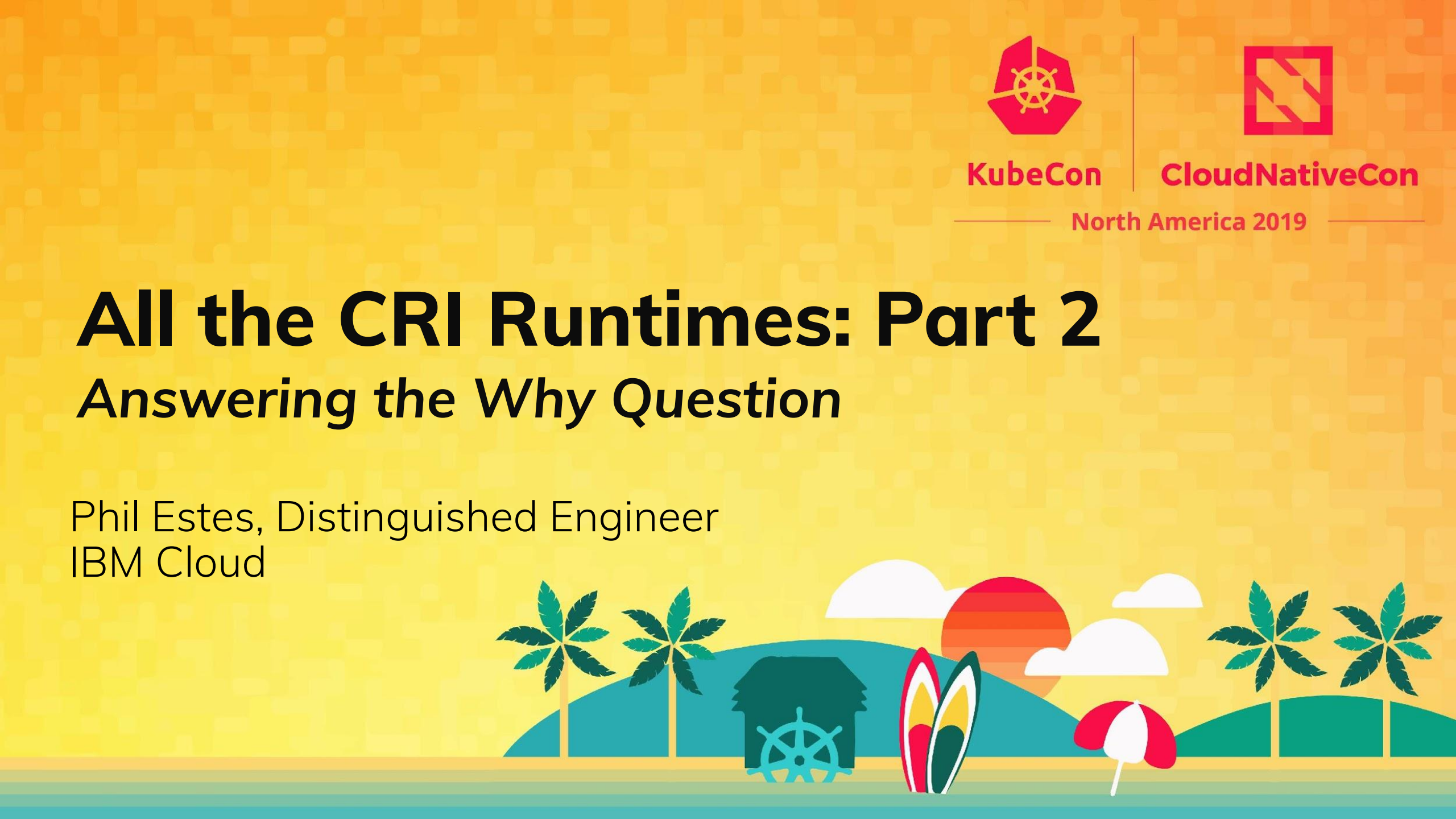


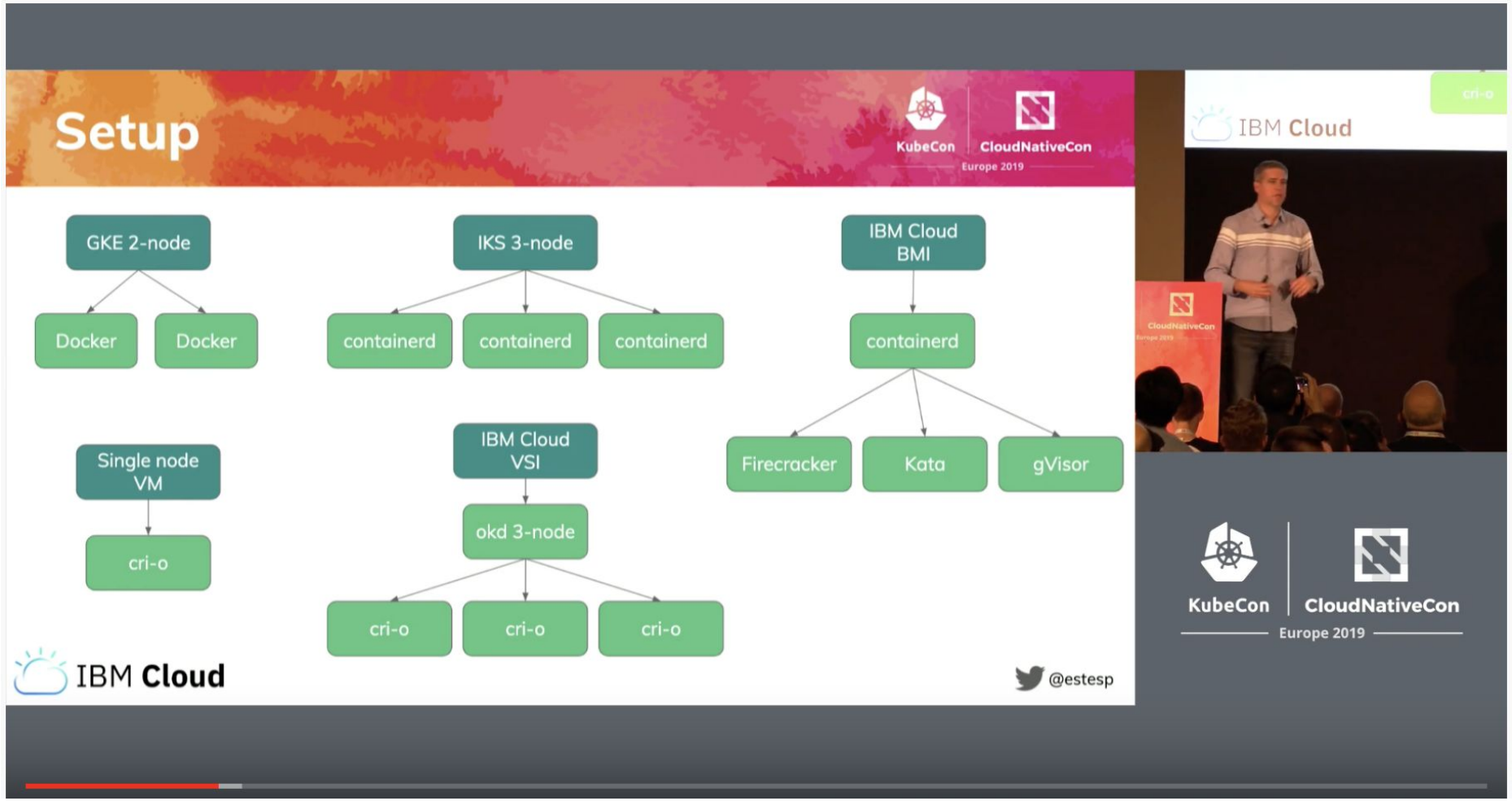

# All the CRI Runtimes: Part 2

## *Answering the Why Question*

Phil Estes, Distinguished Engineer
IBM Cloud

# Again with the runtimes talk!?

KubeCon CloudNativeCon North America 2019




https://www.youtube.com/watch?v=FKoVztEQHss

# Background: OCI



Docker, containerd, cri-o, Kata, Firecracker, gVisor, Nabla, Singularity, ...

DockerHub, OSS distribution project, Harbor, Quay.io, Cloud registries, JFrog, ...

# Background: CRI



**Monday, December 19, 2016**

## Introducing Container Runtime Interface (CRI) in Kubernetes

*Editor's note: this post is part of a series of in-depth articles on what's new in Kubernetes 1.5*

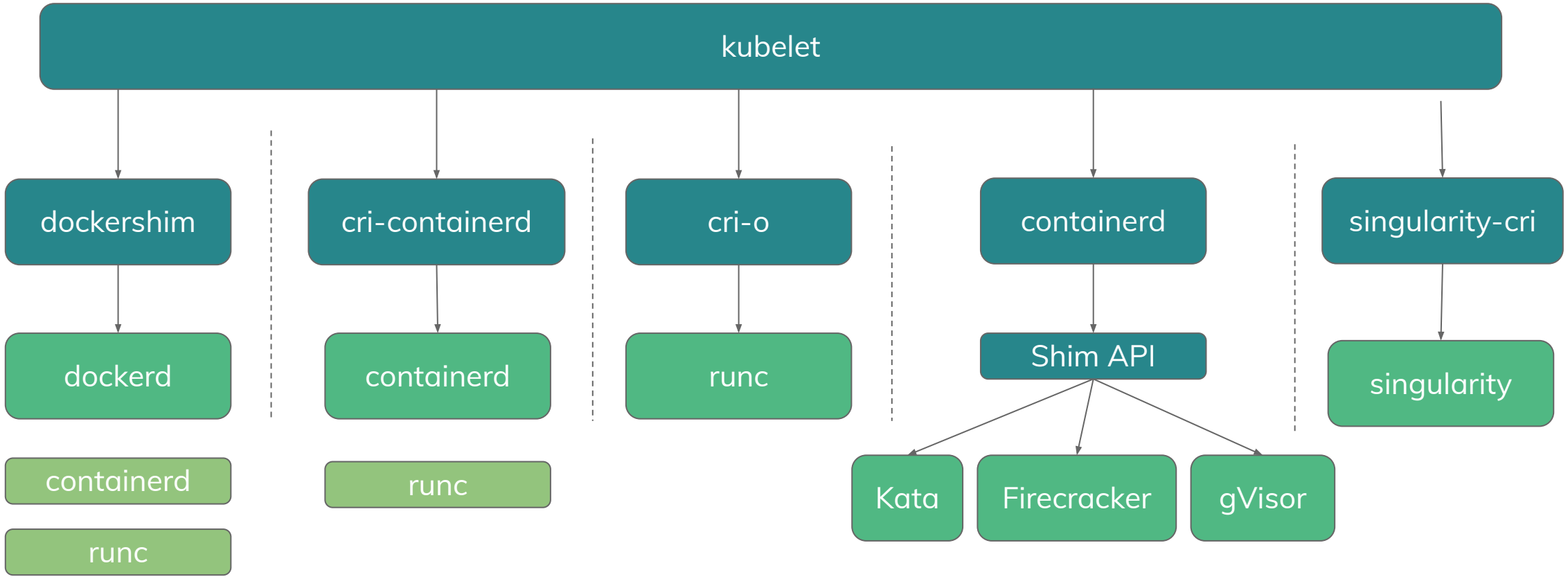
At the lowest layers of a Kubernetes node is the software that, among other things, starts and stops containers. We call this the "Container Runtime". The most widely known container runtime is Docker, but it is not alone in this space. In fact, the container runtime space has been rapidly evolving. As part of the effort to make Kubernetes more extensible, we've been working on a new plugin API for container runtimes in Kubernetes, called "CRI".

**What is the CRI and why does Kubernetes need it?**

Each container runtime has it own strengths, and many users have asked for Kubernetes to support more runtimes. In the Kubernetes 1.5 release, we are proud to introduce the Container Runtime Interface (CRI) -- a plugin interface which enables kubelet to use a wide variety of container runtimes, without the need to recompile. CRI consists of a protocol buffers and gRPC API, and libraries, with additional specifications and tools under active development. CRI is being released as Alpha in Kubernetes 1.5.

# Background: CRI Runtimes



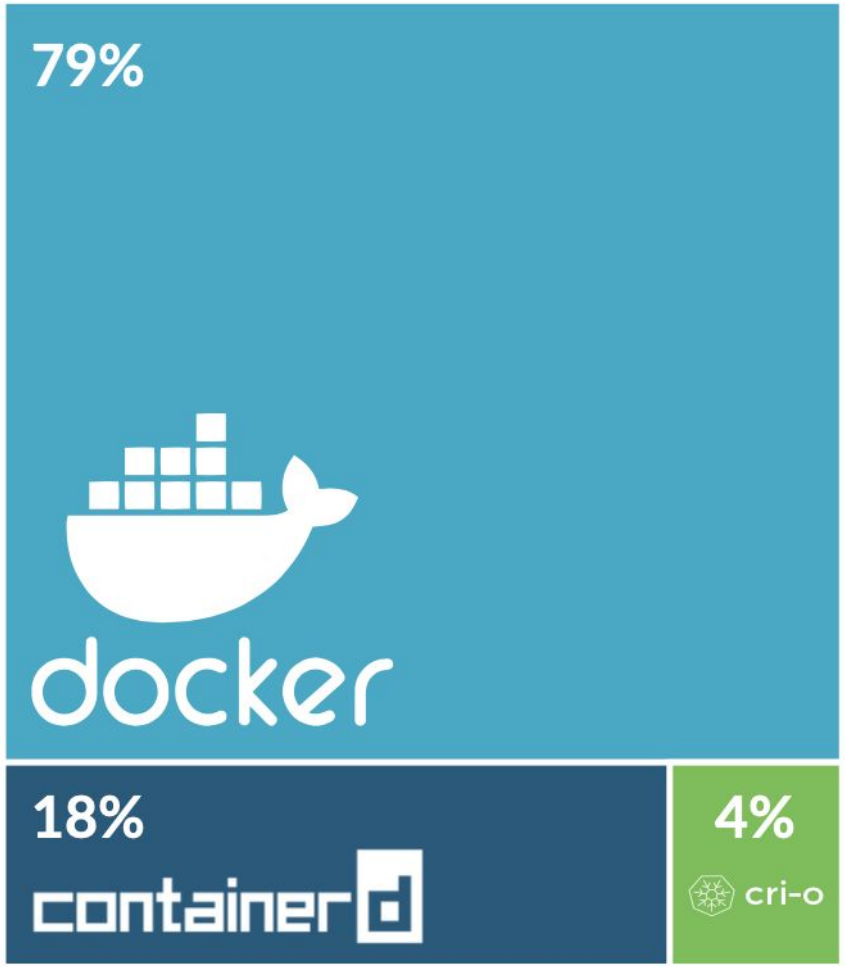



```
kubelet --container-runtime {string}
        --container-runtime-endpoint {string}
```

```
RuntimeClass
and/or annotations
```

# Where are we?

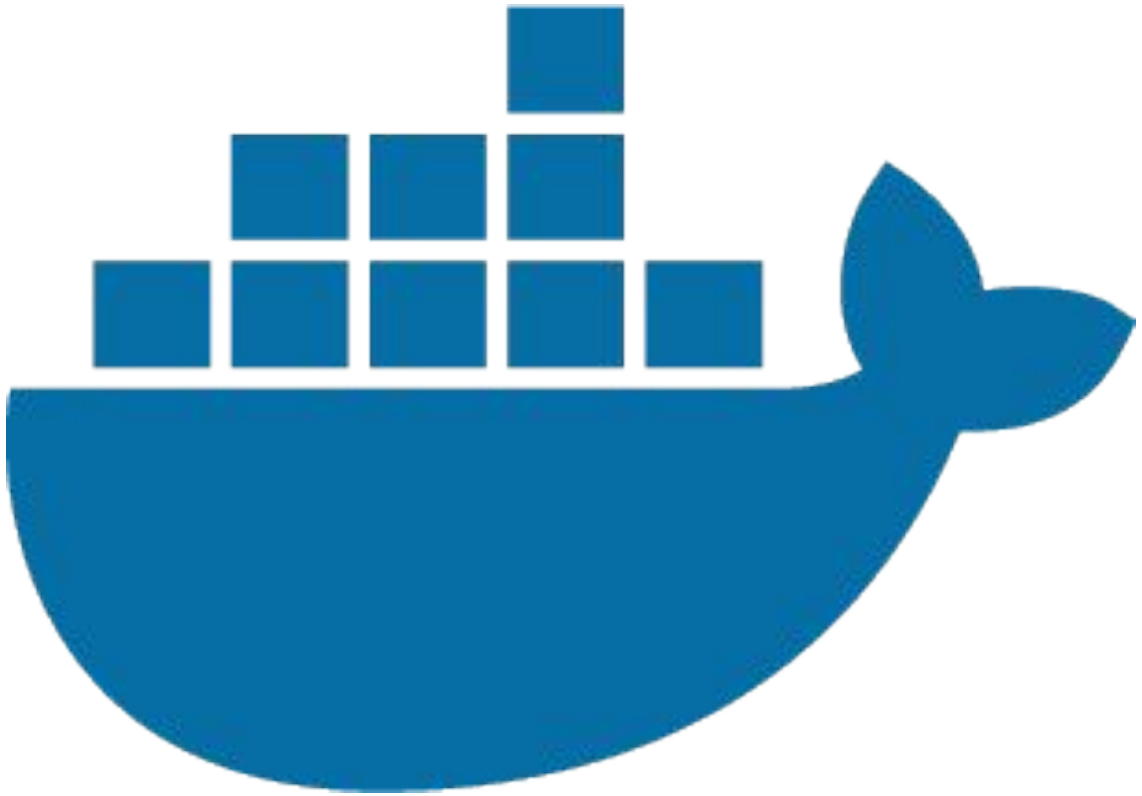





**Source: Sysdig 2019 Container Usage Report**
https://sysdig.com/blog/sysdig-2019-container-usage-report/

# Docker





# Docker: What



- The runtime that started this whole thing.
- **Docker CE**: free community edition built from Moby project and Docker components:
  - CLI, daemon, BuildKit, containerd, runc, notary, etc.
  - defacto “standard” among runtimes used for Kubernetes
- **Docker Enterprise**:  a Kubernetes distribution (including Swarm support) with control plane, dashboard, and registry sold as a packaged offering with a supported engine ~ Docker CE
  - Now owned by Mirantis as of last week
- **Docker Desktop**: single node K8s (and Swarm) on MacOS and Windows

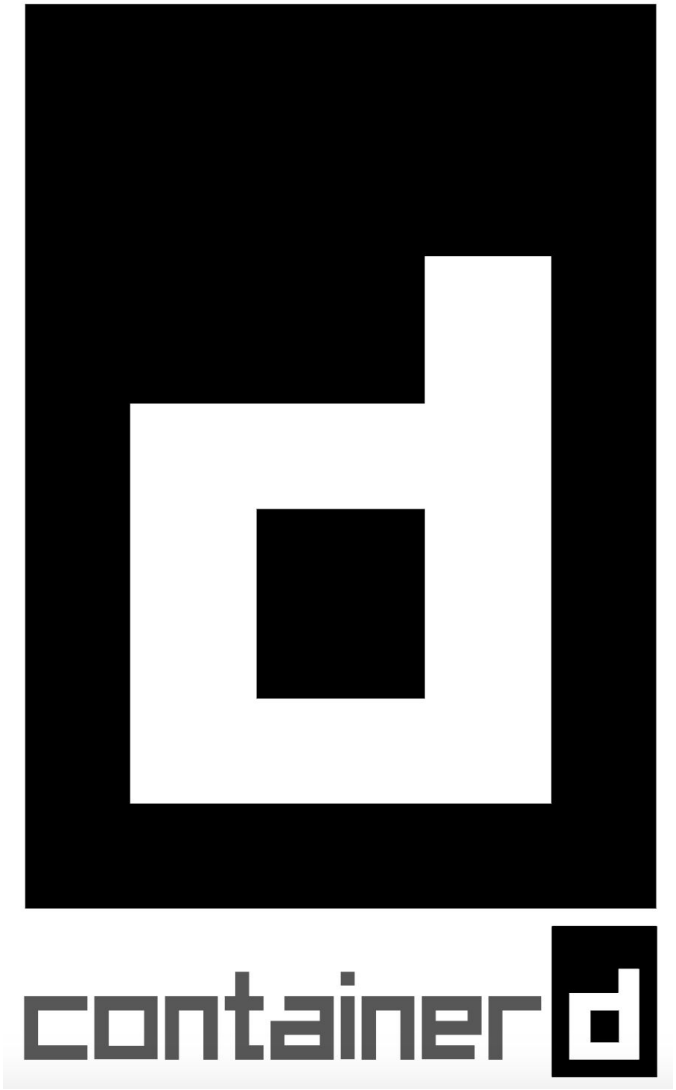
# Docker: Why



- Most commonly used, original runtime for Kubernetes clusters.
  - Many tools, installers, deployers automatically default to Docker
- Simplifies tooling for mixed use nodes
  - e.g. applications relying on `docker …` commands “just work”
  - hacks (that people shouldn’t rely on in 2019!) like `docker build` from the node runtime’s API endpoint “just work”
- Docker Enterprise customers get a supported engine and multi-orchestrator support (swarm + K8s in same cluster)
- Although there are challenges, the Docker engine is time-tested and production deployed in significant use cases

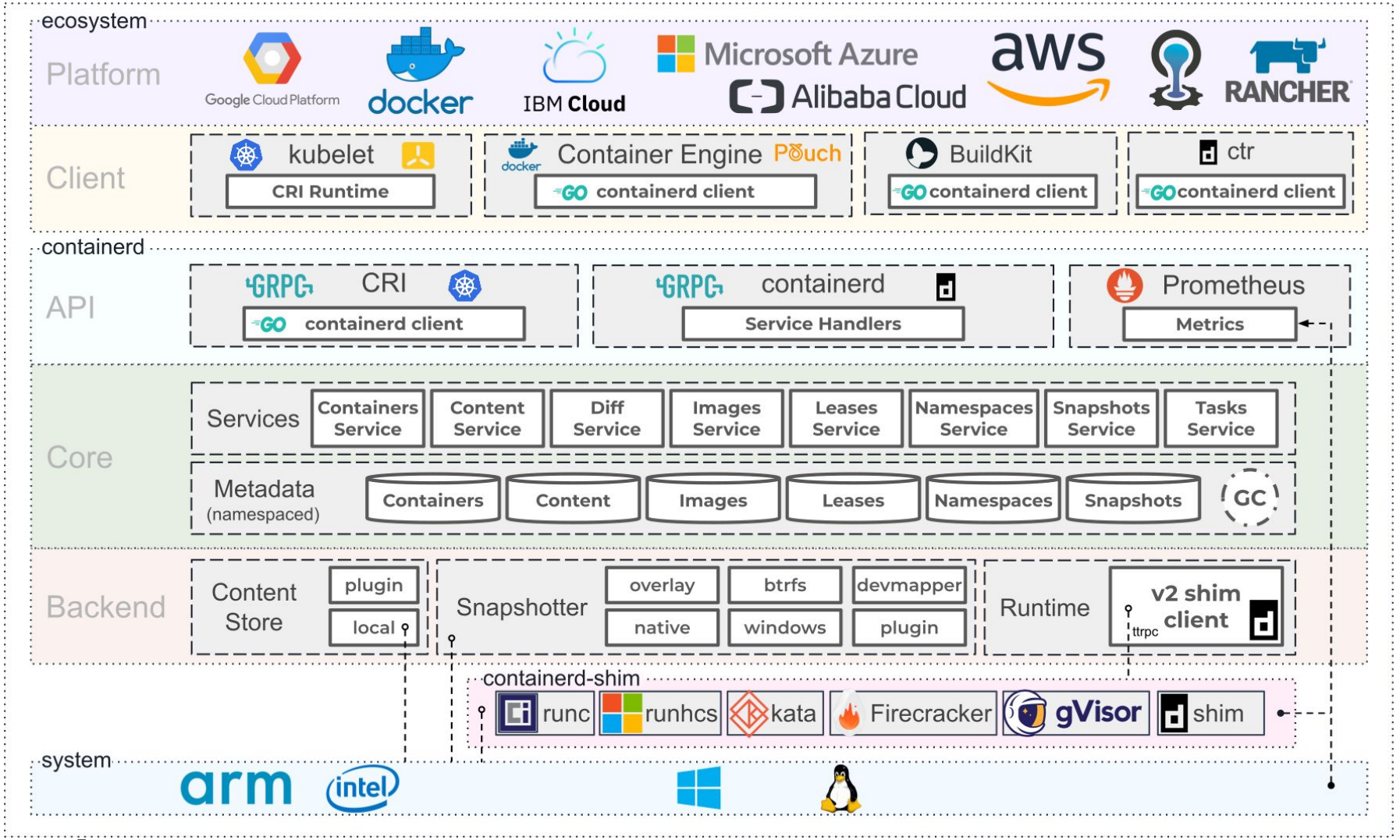
# Containerd





# Containerd: What





# Containerd: What



- Graduated CNCF project with broad base of contributors, maintainers, and adopters
- Small, stable, and clearly scoped container runtime built on OCI standards and default lower-level OCI runtime, **runc**
- Used in GKE (Google), IKS (IBM), & Alibaba public clouds
- Extensible, clean API that has led to easy embedding and usage by KinD, AWS ECR, AWS Firecracker, Kata, Weaveworks Ignite/Firekube, Alibaba Pouch, k3s, ECI, Azure Teleport, among many other tools & projects
- Lower memory/CPU footprint; focus on stability & performance

# Containerd: Why



- Broadly adopted, clear extended support terms and alignment with Kubernetes CRI and official releases
- Significant hardening/testing by nature of use in every Docker installation (tens of millions of engines)
- Need an extendable/embeddable runtime? Clean/clear API and extension points.
- Maturing Windows support
- Shim v2 API (gVisor, Kata, Firecracker, etc.)
- Remote/proxy plugins for snapshotters & content store (see Microsoft Teleport, Google CRFS, work with CERN on CVMFS)

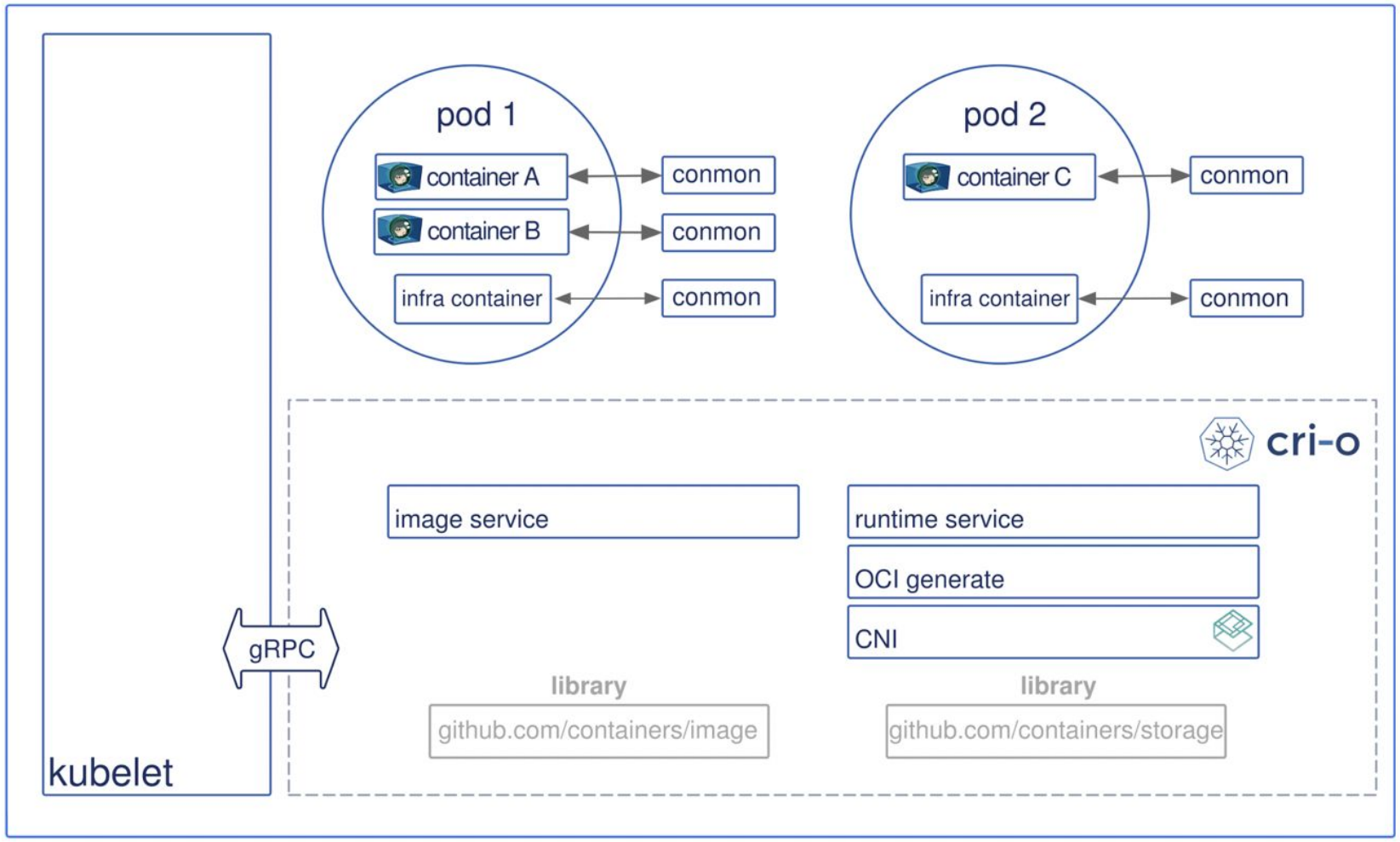
# CRI-O



# CRI-O: What



https://sched.co/Uai5 **Introduction to CRI-O - Mrunal Patel & Peter Hunt, Red Hat, Inc.**



# CRI-O: What



- Direct support of OCI specifications via implemented storage and image libraries shared among Red Hat tooling for containers; depends on **runc** as default runtime
- Used in RH OpenShift; SuSE CaaS; among other customers & use cases; provided via RHEL (no more docker rpms in 8.x)
- “all the runtime Kubernetes needs and nothing more”
- Suite of tools work together on the same on-disk representation of image/storage


buildah


podman

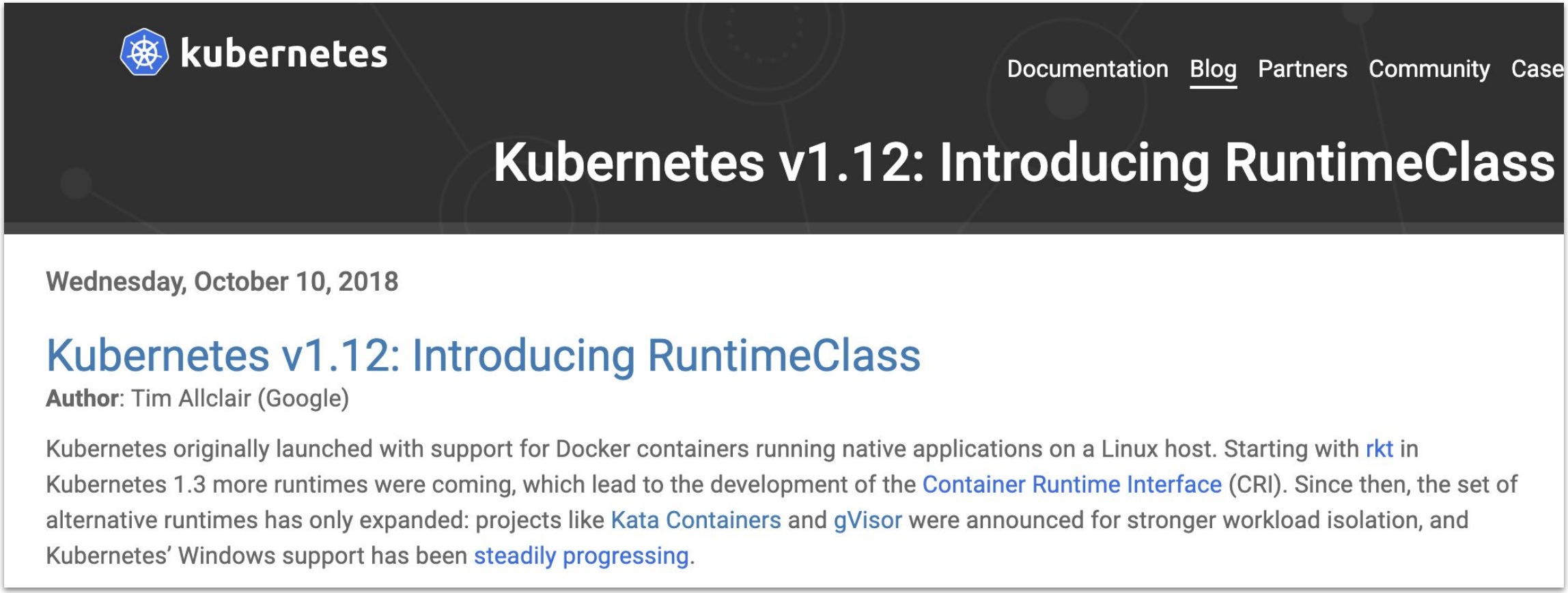
# CRI-O: Why



- OpenShift fully supports (and now defaults) to cri-o as the CRI runtime for Kubernetes delivered via OKD
- Red Hat tool suite alignment, delivered and supported in Fedora/RHEL with fully enabled/tested SELinux support
  - synergy with the skopeo, buildah, podman family of tools
- Simple, supported, and tested CRI implementation aligned tightly with Kubernetes release cycle

# Sandboxes + RuntimeClass





kubernetes

Documentation Blog Partners Community Case

## Kubernetes v1.12: Introducing RuntimeClass

Wednesday, October 10, 2018

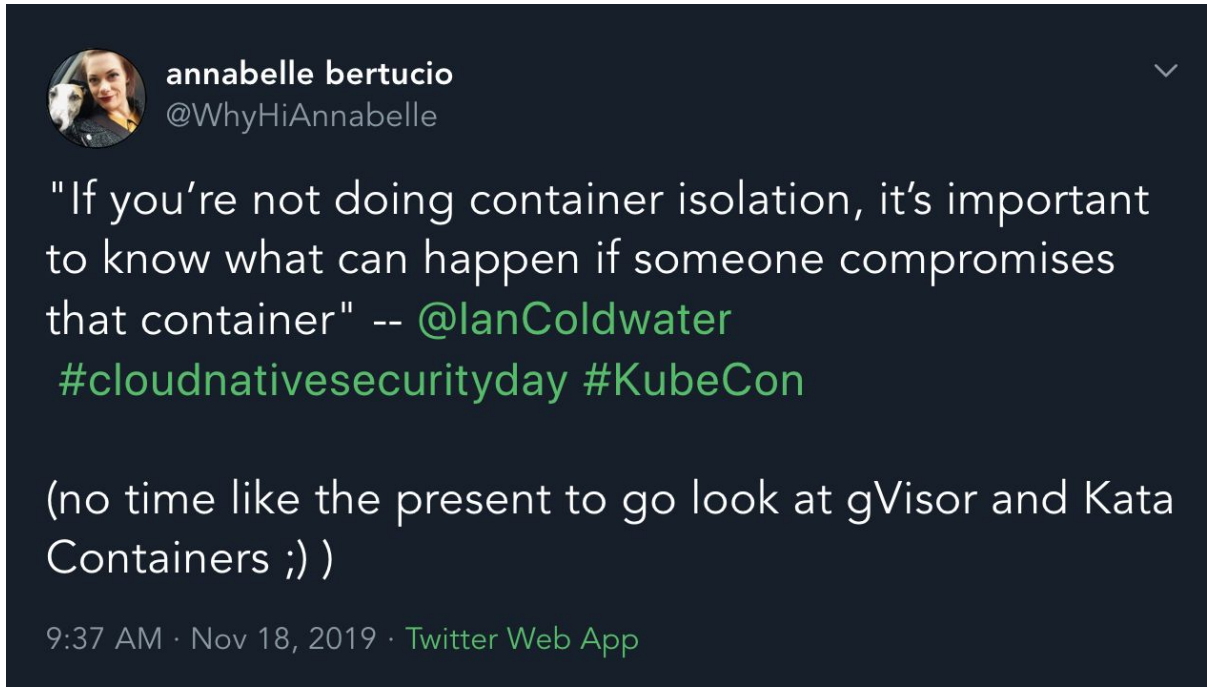
### Kubernetes v1.12: Introducing RuntimeClass

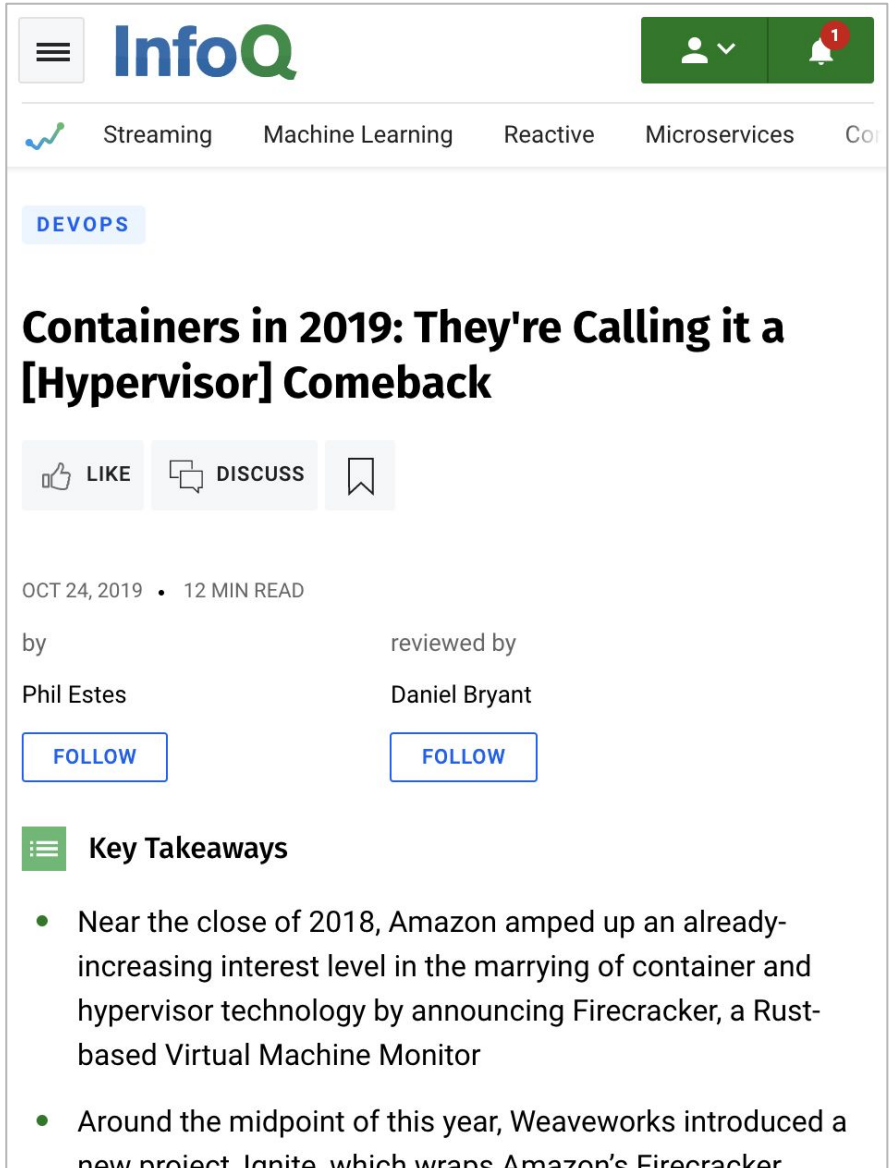
**Author**: Tim Allclair (Google)

Kubernetes originally launched with support for Docker containers running native applications on a Linux host. Starting with rkt in Kubernetes 1.3 more runtimes were coming, which lead to the development of the Container Runtime Interface (CRI). Since then, the set of alternative runtimes has only expanded: projects like Kata Containers and gVisor were announced for stronger workload isolation, and Kubernetes' Windows support has been steadily progressing.

# Why Sandboxes?

KubeCon CloudNativeCon North America 2019



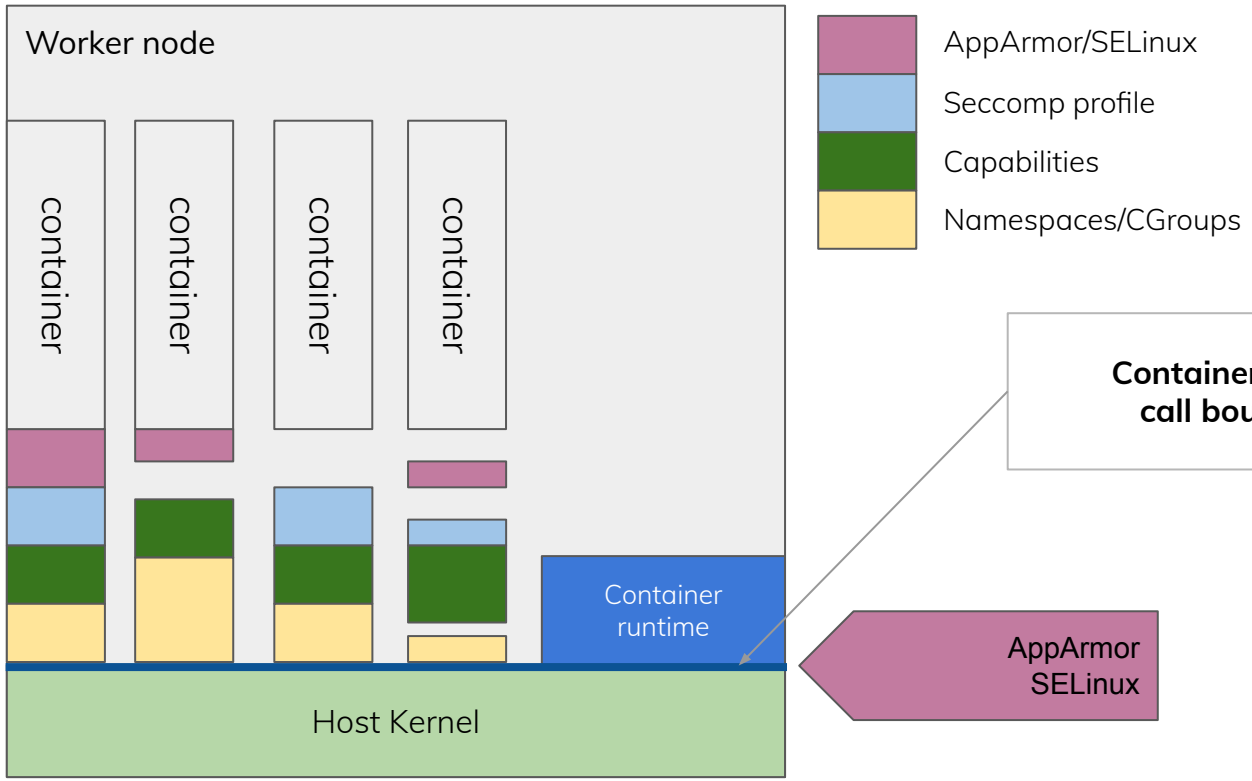

IBM Cloud

@estesp

# Isolators



**Traditional container runtime**



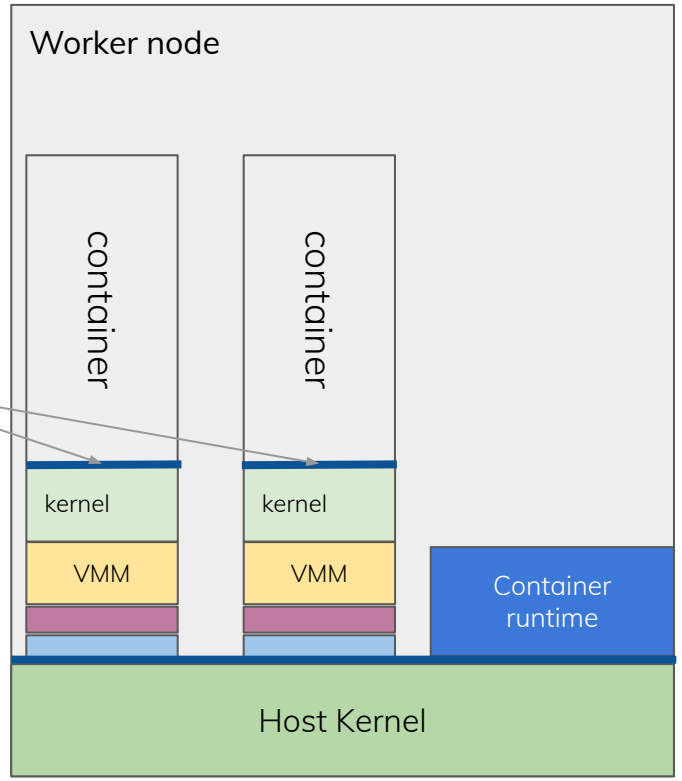




**Container system call boundary**

AppArmor
SELinux

**Lightweight hypervisor-based runtime**

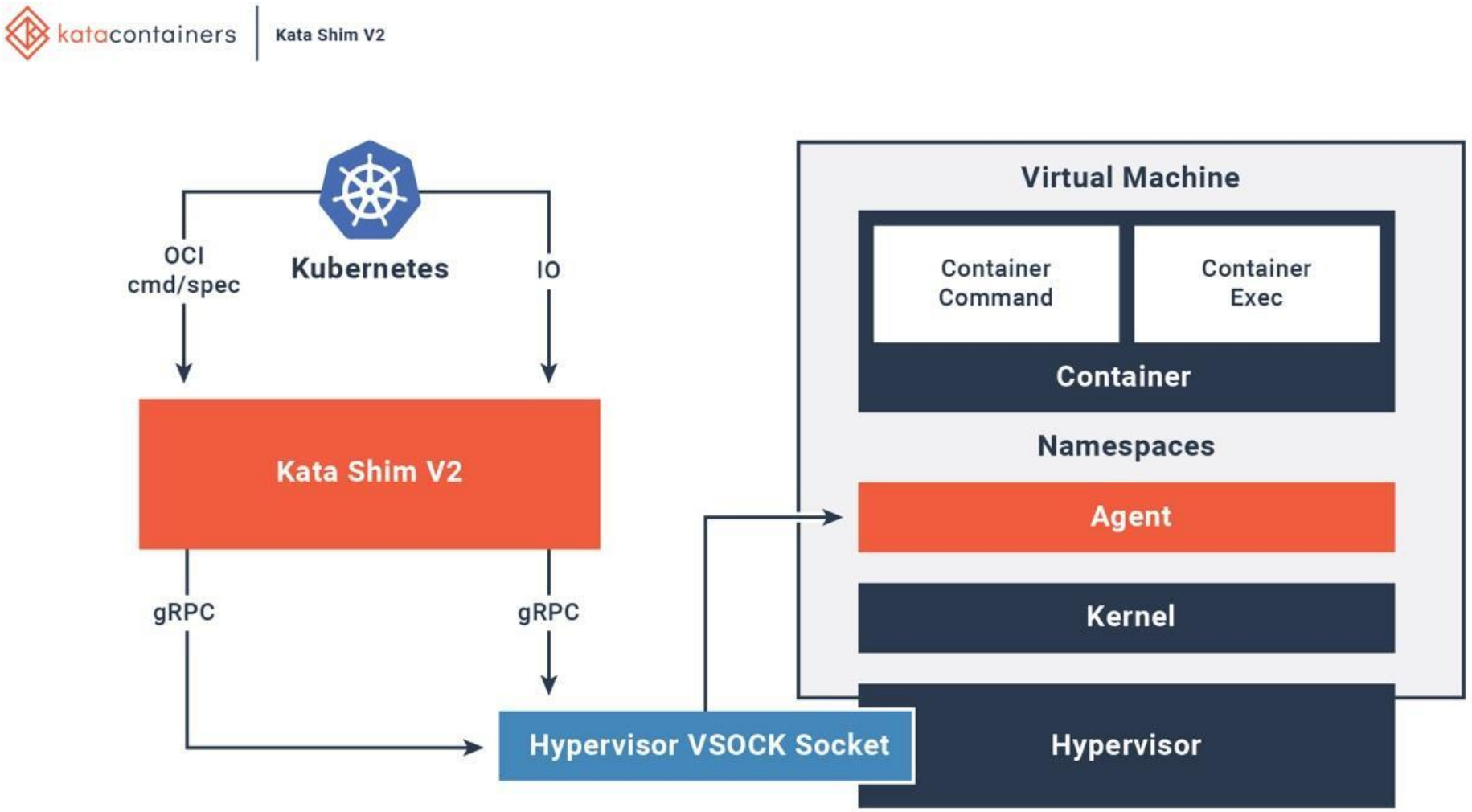


# Kata Containers



# Kata Containers: What





# Kata Containers: What



- Lightweight virtualization implementation via Intel Clear Containers + Hyper.sh predecessors
- Implemented via qemu-based KVM hypervisor, but supports Firecracker (Rust-based VMM) as well
- Works with Docker, cri-o, & containerd; supports Kubernetes use case
- Solid and maturing project with Intel and others leading; governance under OpenStack Foundation
- Supports ARM, x86_64, AMD64, and IBM p and zSeries
- Baidu AI Cloud whitepaper reveals use of Kata @ Baidu
  - https://medium.com/kata-containers/kata-baidu-whitepaper-16ad04a5302

# Kata Containers: Why



- Desire for additional isolation than Linux container primitives
- Prefer hypervisor-based, with choice of Firecracker (rust-vmm) or qemu/KVM-based backend
- Want broad multi-architecture support
- Fewer restrictions (more containerized workloads will work out of the box than Firecracker and gVisor)
- Better integration and support for Kubernetes today given Firecracker's narrower focus for a Lamba engine

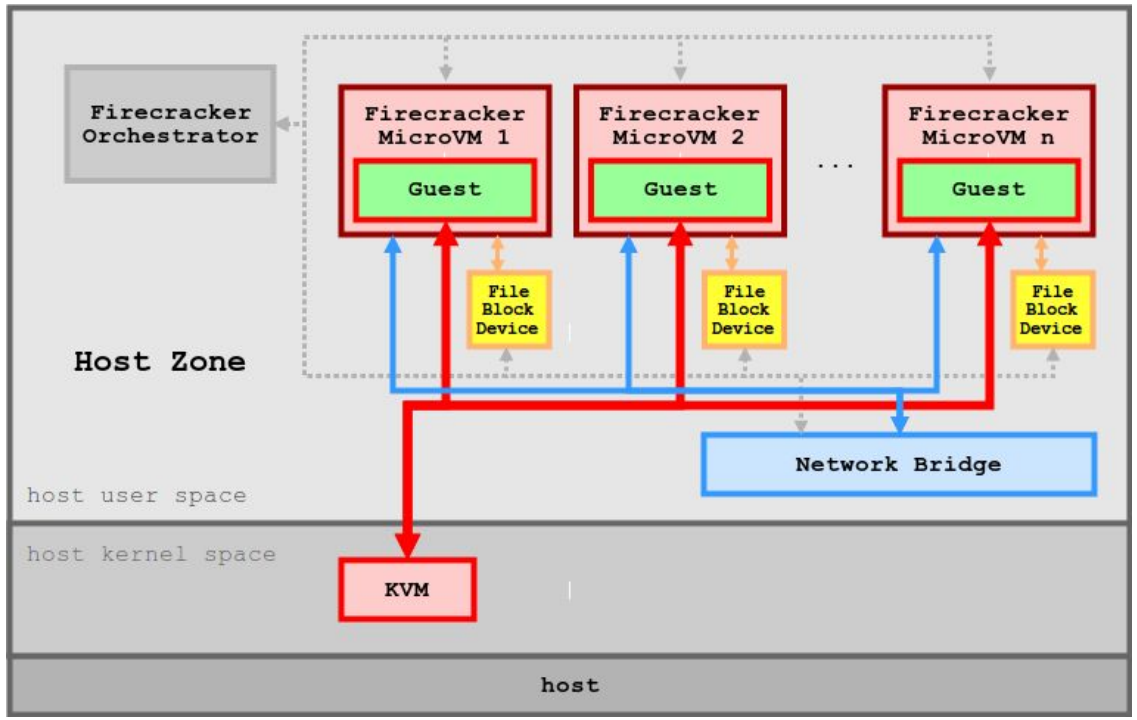
# AWS Firecracker

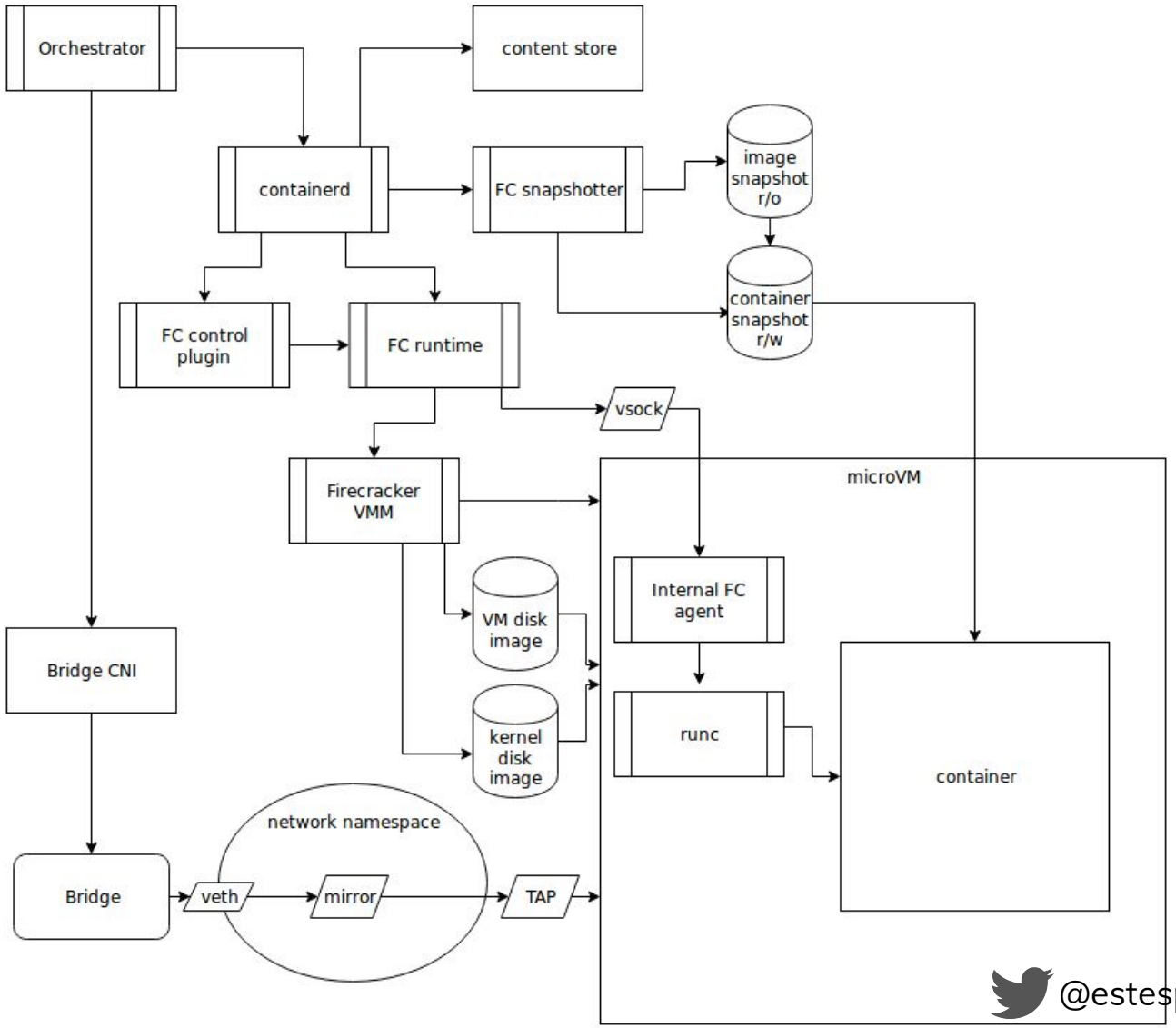


Firecracker

# AWS Firecracker: What






IBM Cloud

@estesp

# AWS Firecracker: What



- Lightweight virtualization via Rust-written VMM
  - originating from Google's crosvm project
  - narrow focus on the serverless runtime use case
  - open sourced by Amazon in November 2018
- Works standalone via API or via containerd
- cgroup + seccomp "jailer" to tighten down kernel access
- Integrated with containerd via shim and external snapshotter
  - Contributed devmapper snapshotter to containerd core
- Weaveworks has wrapped Firecracker in 2019 with a few interesting projects marrying VMs and containers: Ignite and Firekube; others investigating Firecracker for various use cases

# AWS Firecracker: Why



- Need a narrowly focused runtime without support (today) for general container use cases (volume mounts, general virtualization scenarios—e.g. full device emulation, etc.)
- Attracted to security promises of Rust-based VMM
- Use via other wrappers: Weaveworks Ignite/Firekube

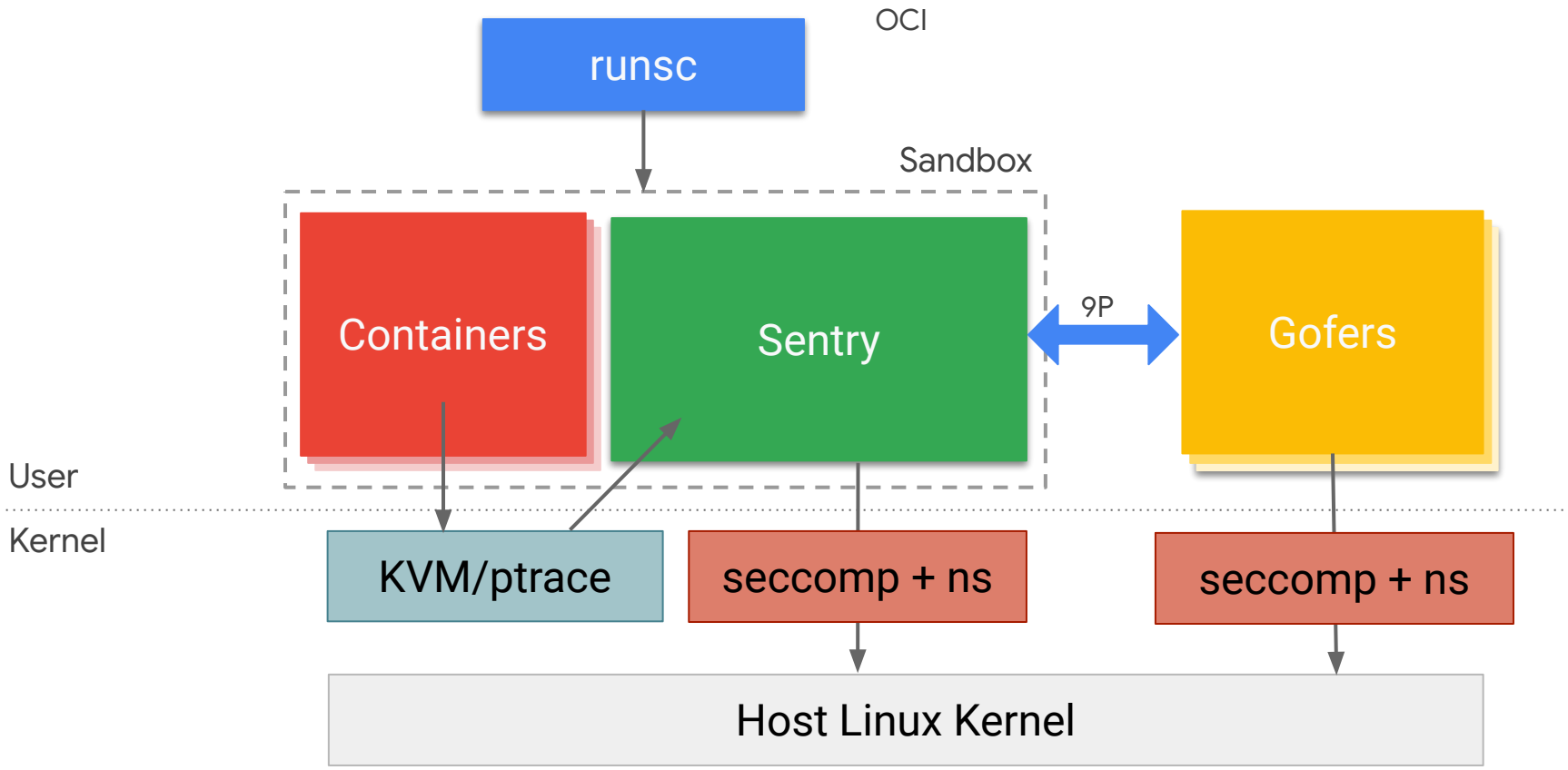
# gVisor



# gVisor: What



https://speakerdeck.com/ianlewis/the-enemy-within-running-untrusted-code-in-kubernetes



# gVisor: What



- Sentry: A kernel-in-userspace syscall implementation written by Google in Golang
- Gofer: filesystem access from the container
- Used in concert with GKE; for example with Google Cloud Run for increased isolation/security boundary
- Works standalone (OCI runc replacement) or via containerd shim implementation (Kubernetes integration)
- Not the entire syscall surface covered in Sentry implementation
- Intercepts syscalls via ptrace (some perf. impact); also experimental KVM-based method

# gVisor: Why



- Reducing syscalls used against “real kernel”; applications run against gVisor syscall implementations
- Limited functionality; some applications may not work if syscall not implemented or other incompleteness (/proc or /sys)
  - Incremental improvements always in development: https://opensource.googleblog.com/2019/05/gvisor-one-year-later.html
- Obvious alternative to hypervisor-based isolators; less management of a full guest (e.g. custom kernel, agents, etc.)

# Nabla



# Nabla: What and Why



- IBM Research created, open source, unikernel-based sandbox runtime
- Uses highly-restricted seccomp profile to reduce attack surface
- Similar to gVisor, but instead of user-mode kernel, uses unikernel+application packaged approach

---

- Currently requires building images against special set of unikernel-linked runtimes (Node, Python, Java, etc.)
- IBM Research pursuing ways to remove this limitation; until then doesn't allow generic use of any container image

# Singularity



Sylabs.io

# Singularity: What and Why



- An HPC/academic community focused container runtime
- Initially not implementing OCI, now has OCI compliant mode
- To meet HPC use model; not daemon-based, low privilege, user-oriented runtime (e.g. HPC end user workload scheduling)

---

- Sylabs, creator of Singularity have recently written a CRI implementation that drives Singularity runtime
- Uses OCI compliant mode; converts images to SIF, however
- Today is focused primarily on the academic/HPC use case

# Summary



- OCI specs (runtime, image, distribution) have enabled a common underpinning for **innovation** that maintains **interoperability**
- CRI has enabled a "**pluggable**" model for container runtimes underneath Kubernetes
- Options are growing; lots of innovation around sandboxes and K8s enablement via **RuntimeClass**
- **You** have to decide based on threat model/use cases


KubeCon
CloudNativeCon
North America 2019