KubeCon | CloudNativeCon

Europe 2018

What’s Up With All The Container Runtimes

Ricardo Aravena

branch.io

@raravena80

Motivation

News from the source

Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Front page

Demystifying container runtimes

December 20, 2017
This article was contributed by Antoine Beaupré

KubeCon+CloudNativeCon NA

As we briefly mentioned in our overview article about KubeCon + CloudNativeCon, there are multiple container "runtimes", which are programs that can create and execute containers that are typically fetched from online images. That space is slowly reaching maturity both in terms of standards and implementation: Docker's containerd 1.0 was released during KubeCon, CRI-O 1.0 was released a few months ago, and rkt is also still in the game. With all of those runtimes, it may be a confusing time for those looking at deploying their own container-based system or Kubernetes cluster from scratch. This article will try to explain what container runtimes are, what they do, how they compare with each other, and how to choose the right one. It also provides a primer on container specifications and standards.

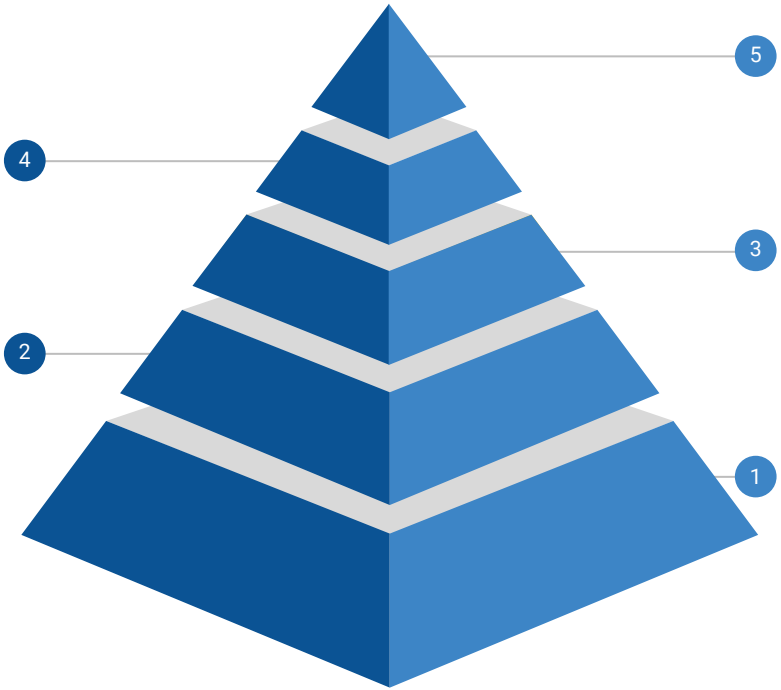
What is a container?

Before we go further in looking at specific runtimes, let's see what containers actually are. Here is basically what happens when a container is launched:

1. A container is created from a container image. Images are tarballs with a JSON configuration file attached. Images are often nested: for example this Libresonic image is built on top of a Tomcat image that depends (eventually) on a base Debian image. This allows for content deduplication because that Debian image (or any intermediate step) may be the basis for other containers. A container image is typically created with a command like `docker build`.
2. If necessary, the runtime downloads the image from somewhere, usually some "container registry" that exposes the metadata and the files for download over a simple HTTP-based protocol. It used to be only Docker Hub, but now everyone has their own registry: for example, Red Hat has one for its OpenShift project, Microsoft has one for Azure, and GitLab has one for its continuous integration platform. A registry is the server that `docker pull` or `push` talks with, for example.

https://lwn.net/Articles/741897/

Containers @ Branch



5 **DB & Big data apps**
Standalone

4 **Runtime?**
Docker/containerd/runC

3 **Mesos qa/dev/prod**
Data pipelines
Spark/Flink/HDFS/Druid

2 **K8s in qa/dev/prod**
API services - Lots of different languages

1 **Where?**
Everywhere

Where Do Runtimes Live?

HERE

Outline

History

- Containers

Early

- OpenVZ
- Mesos
- LXC
- Docker

K8s

- Docker/containerd
- Rkt
- CRIO
- Kata

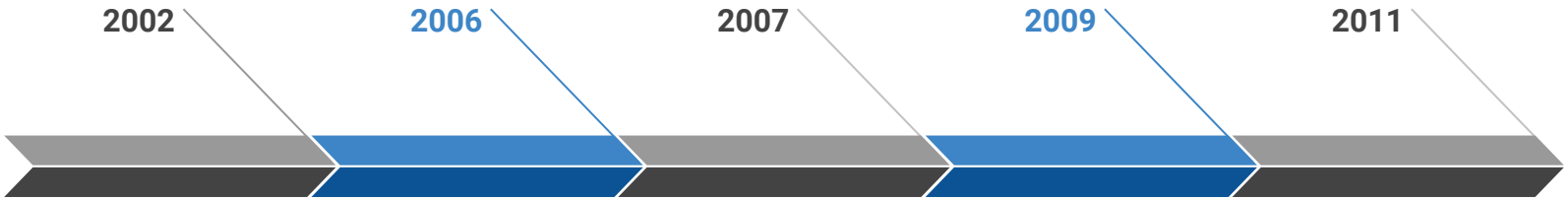
Which one?

- Performance
- Security
- K8s Integration

Future

- CRI
- OCI

History



2002 — Kernel Namespaces

2006 — OpenVZ

Containers with kernel level patch

2007 — cgroups

Kernel level support for control groups

2009 — Mesos

Started as a research paper, first use of cgroups

2011 — LXC

First standalone runtime

History

2013

Docker LXC

First version of Docker

2014

Docker Libcontainer

LXC didn't provide all the nitty gritty isolation.

2014

rkt

Alternative to Docker.

2015

OCI

runC started in 2014

2016

CRI

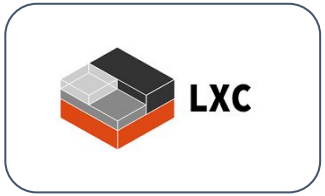
cri-o, rktlet, frakti, cri-containerd

Now?

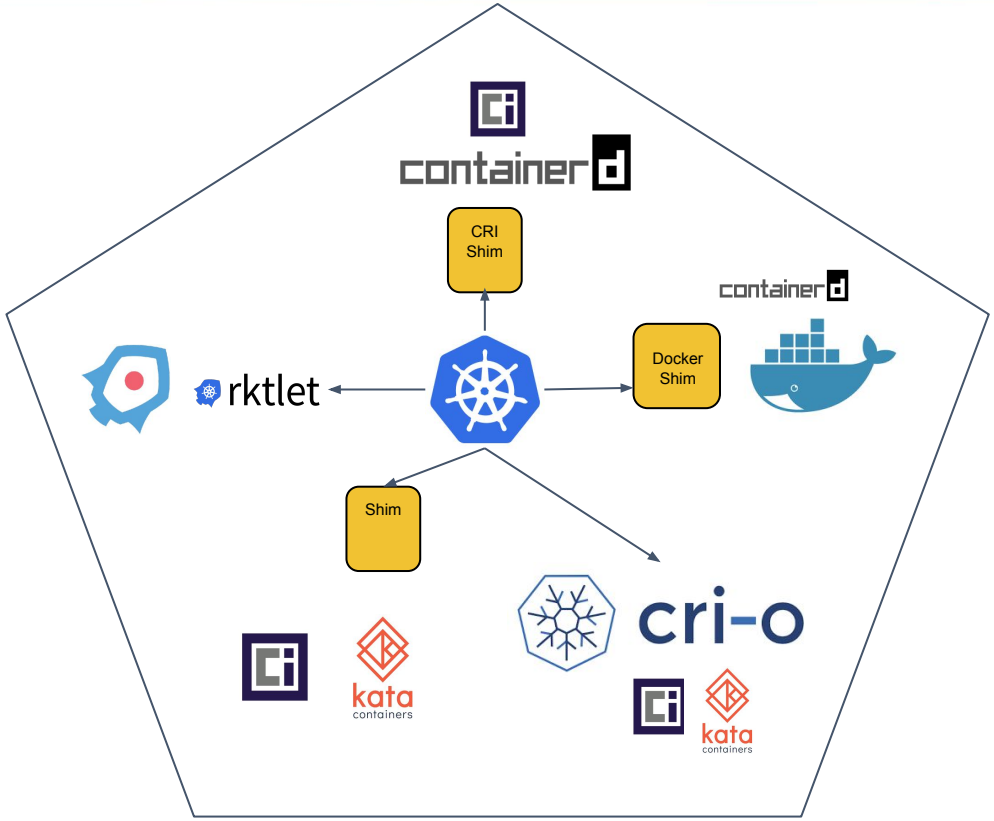
Seems like?

Landscape


OpenVZ
Linux Containers


LXC


appc
Containerizer
MESOS


containerd
CRI Shim
rktlet
Docker Shim
containerd
Shim
cri-o
kata containers
kata containers

OpenVZ Containers

OpenVZ
Linux Containers

First Release	• 2005
Implementation	• C • Kernel Level Patch
Notables	• Supports live migration
Cons	• Custom kernel • Runs on vzlinux based on RHEL 7 • OpenVZ 7 not as stable as OpenVZ 6 • No CRI or K8s yet

https://openvz.org/Main_Page

Mesos

Implementation	• C++ • cgroups
Image format	• Docker • Appc • OCI w/Docker
Best Use	• Spark • Flink • Stateful apps (Data)
Performance	• Very good
Cons	• No standalone mode • Needs Mesos to run

http://mesos.apache.org/

LXC

2012

lxc Linux Containers

container namespace cgroup virtualisation

PAGES
- Home
  - download
  - kernel namespaces
  - lxc development
  - man
- News

NEWS ARCHIVE
- April 2011
- March 2011
- June 2010
- January 2010
- November 2009
- March 2009
- December 2008
- August 2008

Home

LXC is the userspace control package for Linux Containers, a lightweight virtual system mechanism sometimes described as "chroot on steroids".

LXC builds up from chroot to implement complete virtual systems, adding resource management and isolation mechanisms to Linux's existing process management infrastructure.

Linux Containers (lxc) implement:

- Resource management via "process control groups" (implemented via the cgroup filesystem)
- Resource isolation via new flags to the clone(2) system call (capable of create several types of new namespaces for things like PIDs and network routing)
- Several additional isolation mechanisms (such as the "-o newinstance" flag to the devpts filesystem).

The LXC package combines these Linux kernel mechanisms to provide a userspace container object, a lightweight virtual system with full resource isolation and resource control for an application or a system.

Linux Containers take a completely different approach than system virtualization technologies such as KVM and Xen, which started by booting separate virtual systems on emulated hardware and then attempted to lower their overhead via paravirtualization and related mechanisms. Instead of retrofitting efficiency onto full isolation, LXC started out with an efficient mechanism (existing Linux process management) and added isolation, resulting in a system virtualization mechanism as scalable and portable as chroot, capable of simultaneously supporting thousands of emulated systems on a single server while also providing lightweight virtualization options to routers and smart phones.

The first objective of this project is to make the life easier for the kernel developers involved in the containers project and especially to continue working on the Checkpoint/Restart new features. The lxc is small enough to easily manage a container with simple command lines and complete enough to be used for other purposes.

SEARCH

Search

IN DISTRO
- ArchLinux
- Debian
- Exherbo
- Fedora
- Foresight
- Gentoo
- OpenSuse
- Ubuntu

LXC HOWTO
- Dwight Schauer GOOD !

META
- Log in

https://web.archive.org/web/20121106023904/http://lxc.sourceforge.net/

LXC

Nickname	• Chroot in steroids
Community	• Active
Components	• lxc • lxd, lxfuse
OCI	• lxcrun OCI compliant runtime WIP
Downsides	• Image support and adoption • No k8s yet. WIP but not priority

rkt

1. **Images** — appc, docker
2. **Runtime ?** — Standalone runtime, uses its own command: rkt
3. **OCI ?** — Through runC (in dev)
4. **Security** — TPM, Isolated VMs, SElinux.

https://coreos.com/rkt/

rkt & K8s - Rktnetes


```
$ kubelet --container-runtime=rkt ...
```

rkt

Pros

- Works with K8s
- Pod based semantics
- Single process
- Runs Docker, appc, OCI (in dev) images
- KVM hypervisor for containers
- Process can be managed directly by systemd

Cons

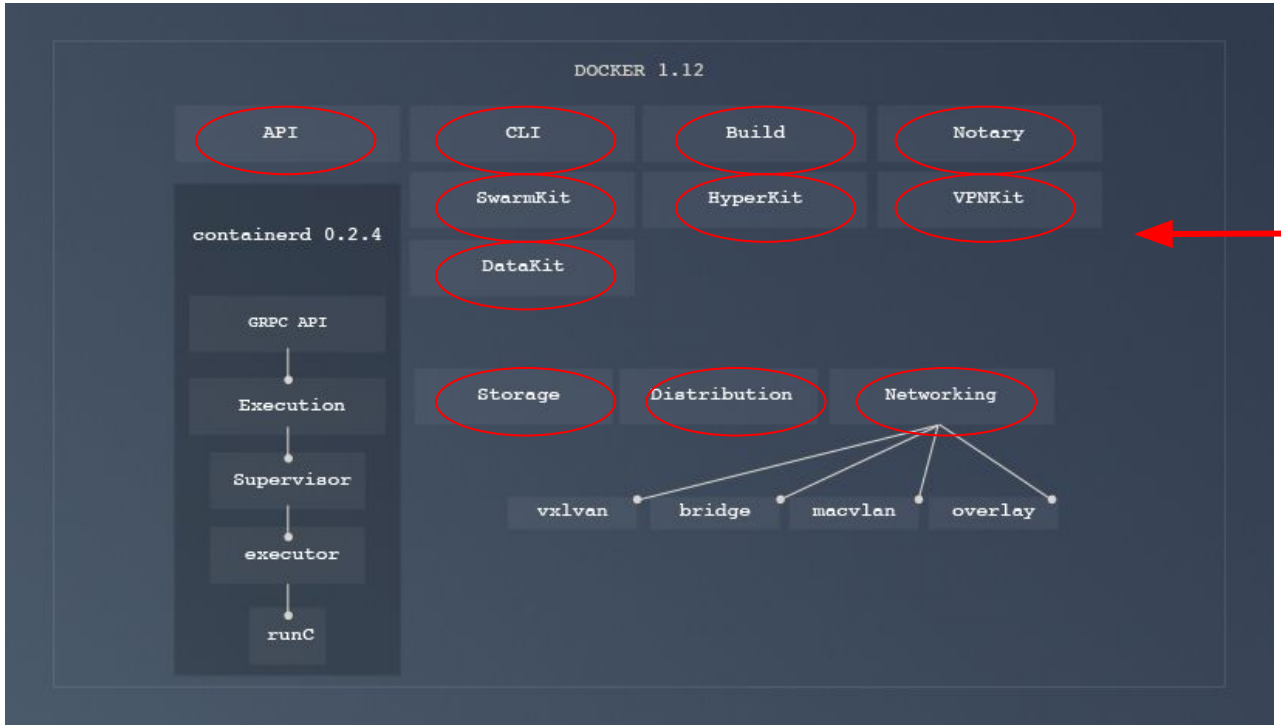
- CoreOS is less focused in runtimes
- Rklet (CRI) project is under development
- Minimal adoption in production
- No more dev into appc
  - OCI is the future

https://coreos.com/rkt/

Docker/Containerd

Initial nickname	• Docker
Now	• Docker daemon • containerd • runC • cri-containerd
containerd	• handles storage, image, networking, runC
cri-containerd now?	• Just “cri” plugin

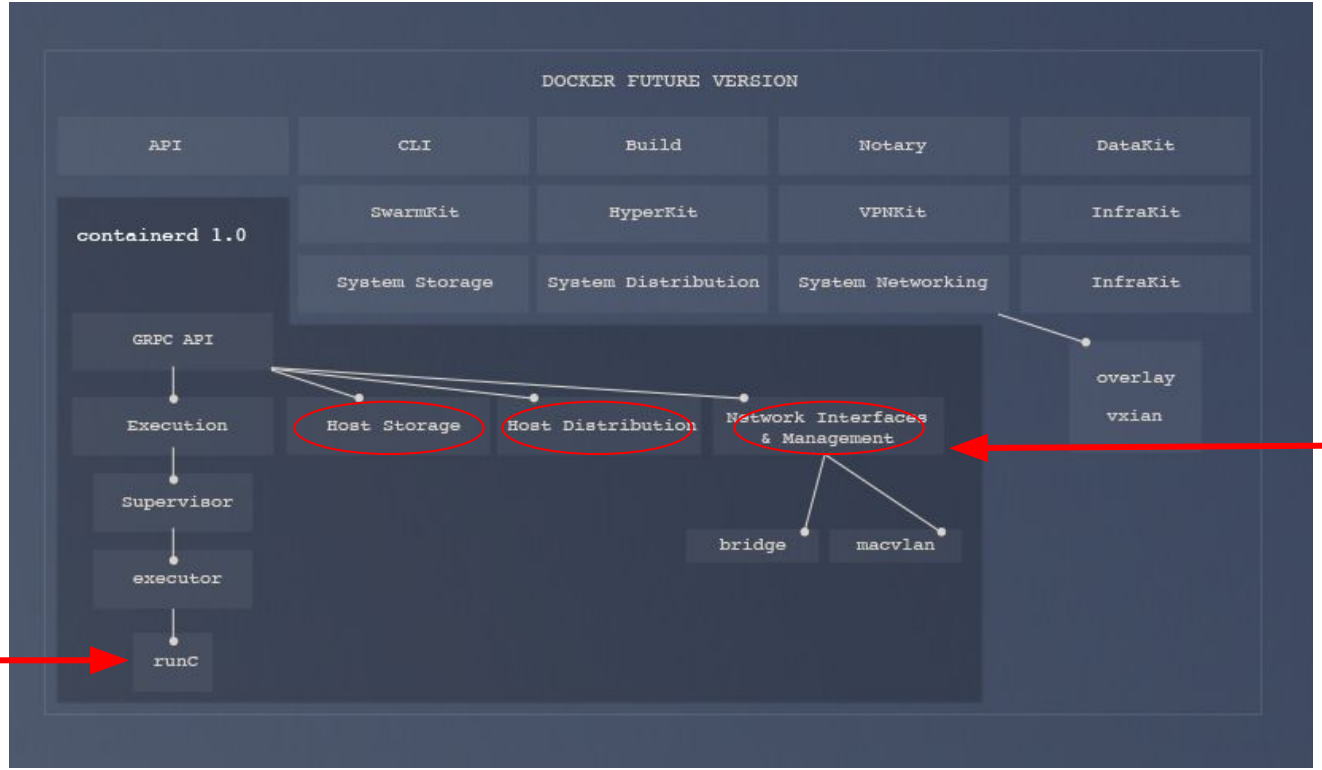
Containerd - Old



Docker

Credit: https://containerd.io/

Containerd



containerd

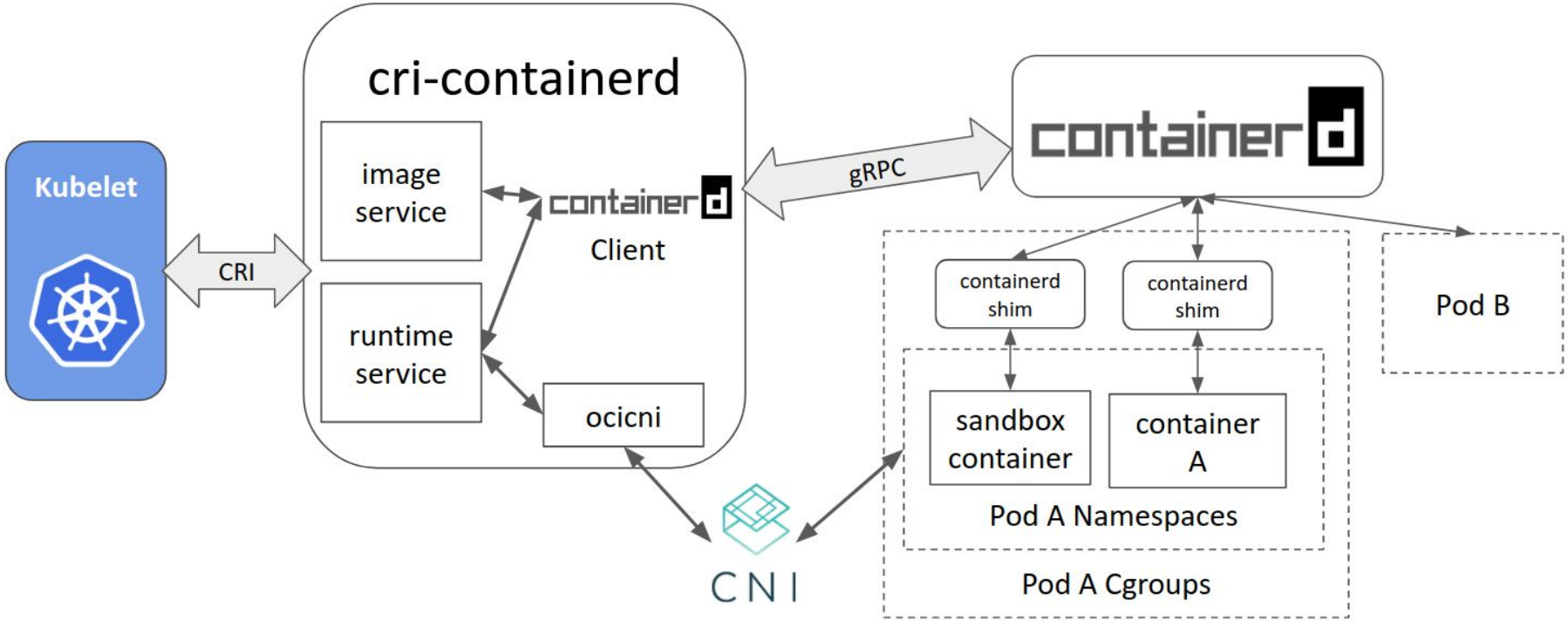
Runtime

Credit: https://containerd.io/

CRI - Container Runtime Interface

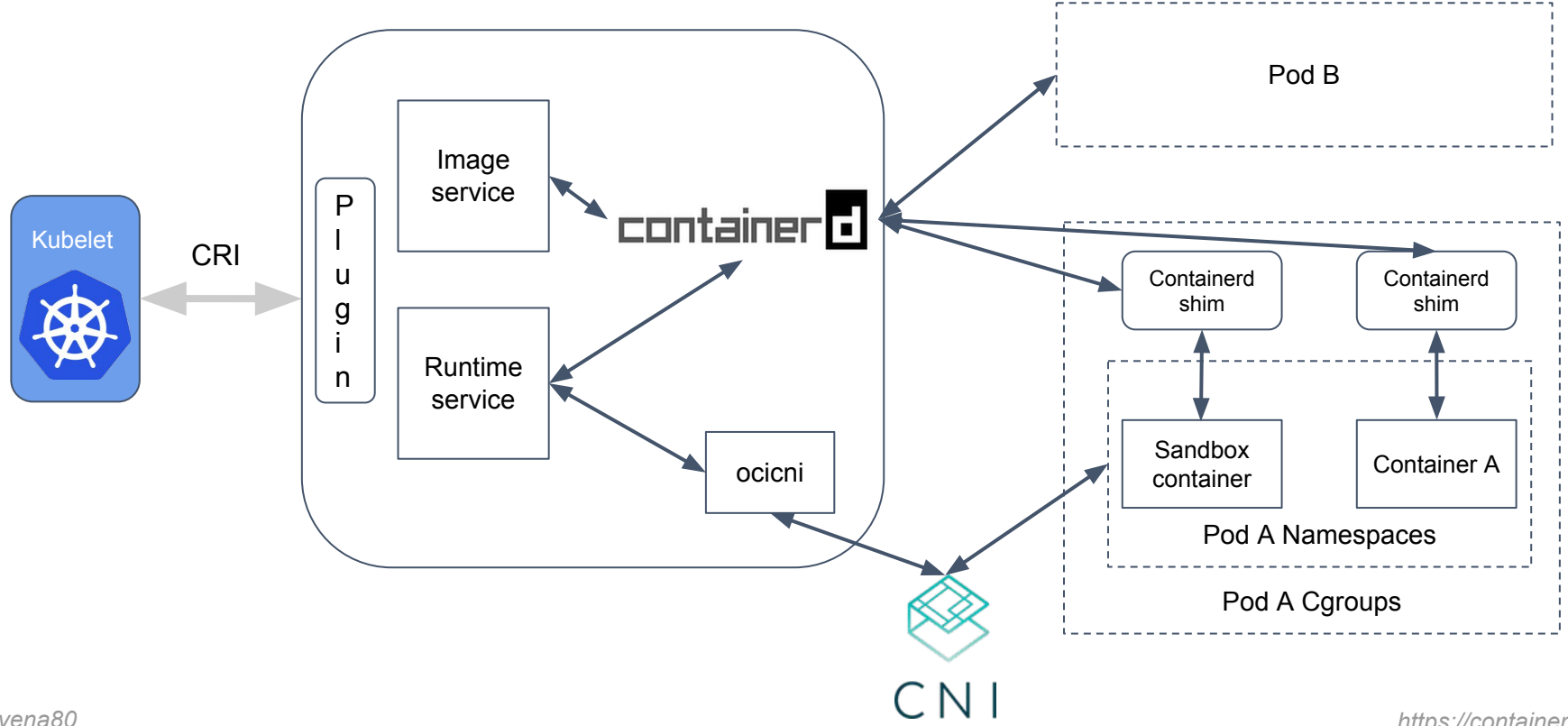
Introduced	• 2016
What?	• Only a GRPC interface • Different from **cri-containerd**
K8s	• Can talk to different runtimes through the same API
Current Plugins	• CRI-O • Rklet (rkt) • Frakti (vm) • CRI Plugin (formerly **cri-containerd**)

Containerd/CRI-containerd



Credit: http://blog.kubernetes.io/2017/11/containerd-container-runtime-options-kubernetes.html

Containerd/CRI Plugin - New



Containerd in K8s 1.10

CRI Plugin:

```
[plugins.cri]
...
   [plugins.cri.containerd]
   ...
     [plugins.cri.containerd.default_runtime]
     # default is runC
     [plugins.cri.containerd.untrusted_workload_runtime]
      # kata or cc
      runtime_engine = ""
```

Containerd in K8s 1.10

Startup:

```
$ systemctl start containerd
$ kubelet --container-runtime=remote --runtime-request-timeout=15m
--container-runtime-endpoint=unix:///run/containerd/containerd.sock
```

```
[Service]
Environment="KUBELET_EXTRA_ARGS=--container-runtime=remote --runtime-request-timeout=15m
--container-runtime-endpoint=unix:///run/containerd/containerd.sock"
...
```

Docker/Containerd

Pros

- Adoption
- Very stable
- Great performance
- Community and support
- Supports Windows - WIP
  - WCOW
  - LCOW

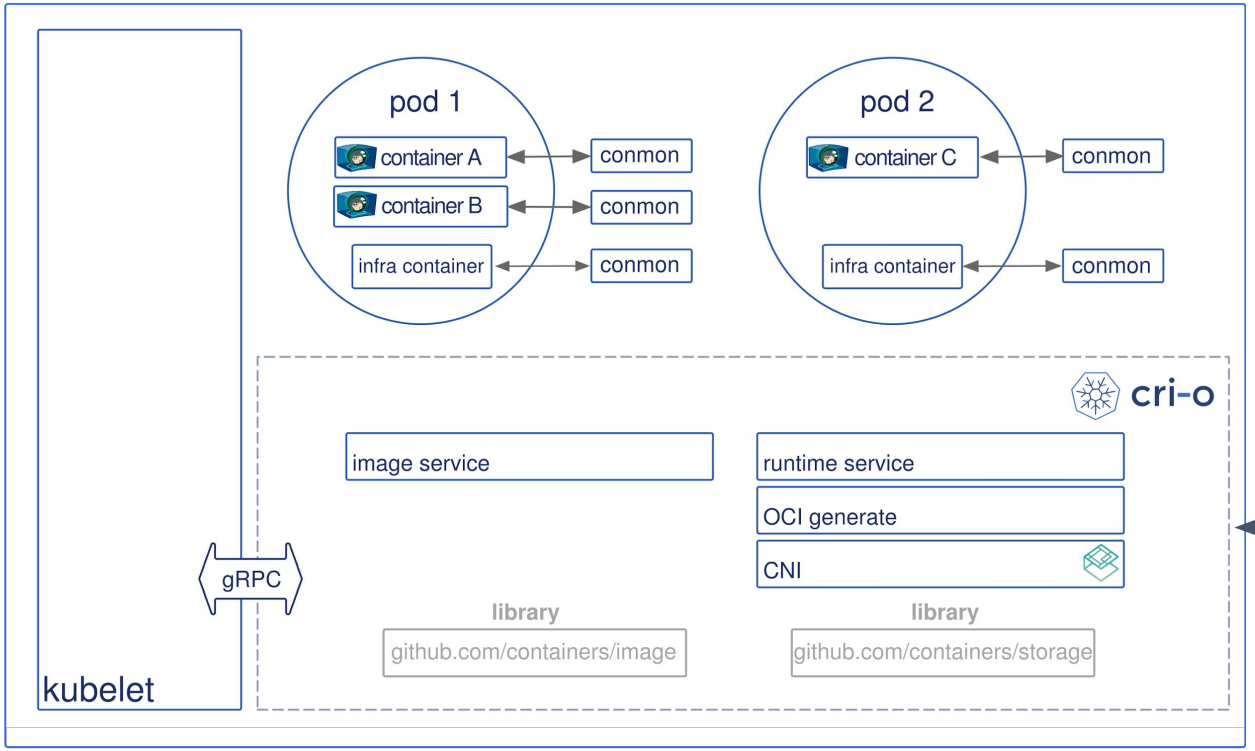
Cons

- Runtime coupled with daemon
- Too many daemons w/Docker (4)
- Daemons recently reduced to 3 in K8s
- Naming confusion

CRI-O

Nickname	• “cry-o”
crio daemon	• Handles storage, image, networking • Talks CRI
common daemon	• Monitors containers
OCI?	• Defaults to runC • Runs any OCI runtime

CRI-O



It's a Shim

Credit: http://cri-o.io/

CRI-O & K8s


```
$ systemctl start crio
$ kubelet --container-runtime=remote --runtime-request-timeout=15m
--container-runtime-endpoint /var/run/crio/crio.sock ...
```

CRI-O

Pros

- OCI & K8s ready
- Great performance
- 3 daemons by default
- Fast common systemctl container monitoring daemon
  - Restart safe
- Extendable
  - Kata

Cons

- GA recently
- Not battle tested in prod
- Cannot run containers outside k8s
  - Podman WIP
- No default for image management
  - buildah
  - Docker, skopeo or umoci

crun

When, Who	• Aug 2017, Red Hat
Implementation	• C
Performance	• Best
OCI?	• Yes
Downsides	• Little adoption • WIP

crun

/bin/true benchmark

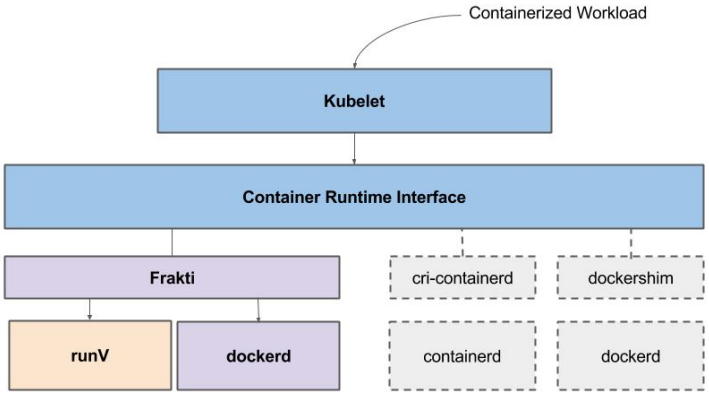
	crun	runC	%
100 /bin/true (no network namespace)	0m4.449s	0m7.514s	40.7%
100 /bin/true (new network namespace)	0m15.850s	0m18.986s	16.5%

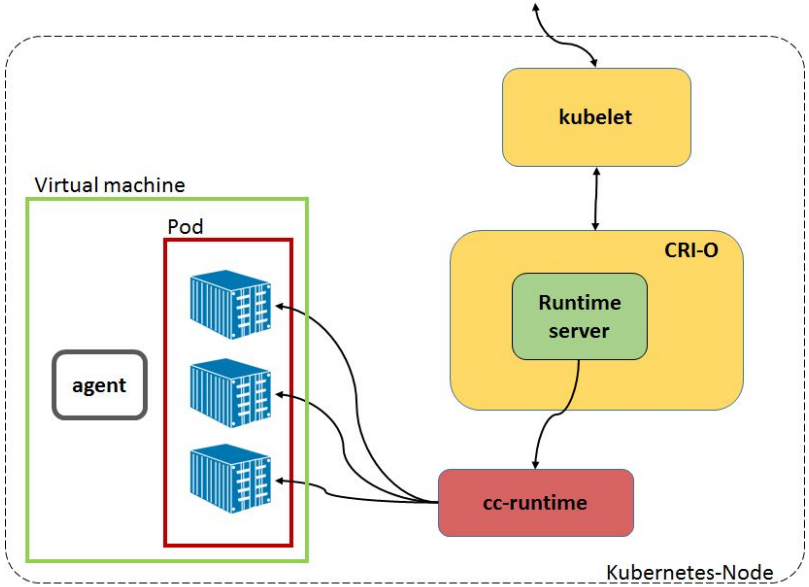
Kata

When, Who	• 2017, Openstack • First release June 2018
What	• runV & ClearContainers • Containers in VMs
Implementation	• Go
Security	• Very good
OCI?	• Yes

Kata

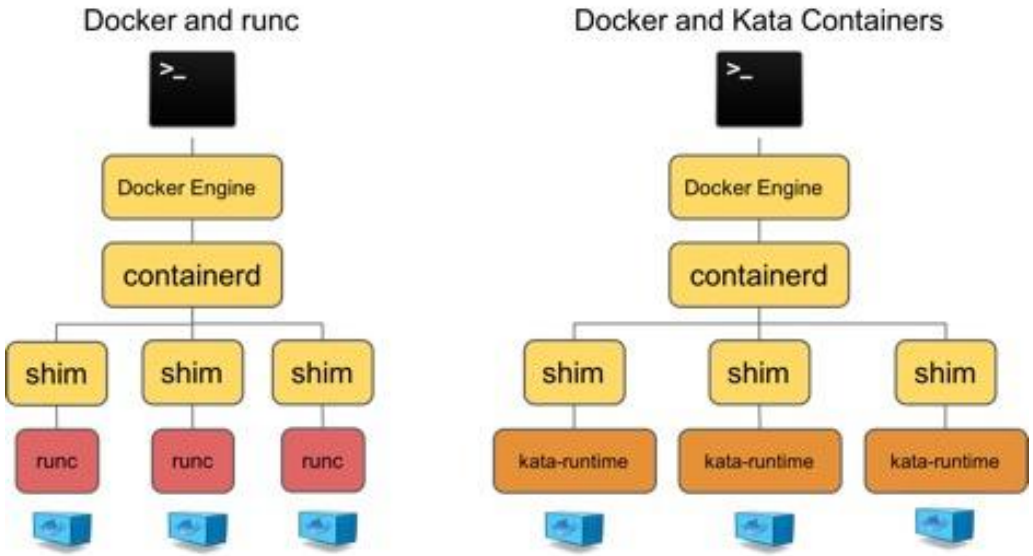
RunV and ClearContainers





Credits:
https://medium.com/cri-o/intel-clear-containers-and-cri-o-70824fb51811
https://github.com/kubernetes/frakti

Kata & Docker



Credit: https://katacontainers.io/posts/why-kata-containers-doesnt-replace-kubernetes/

Kata & Plain Docker

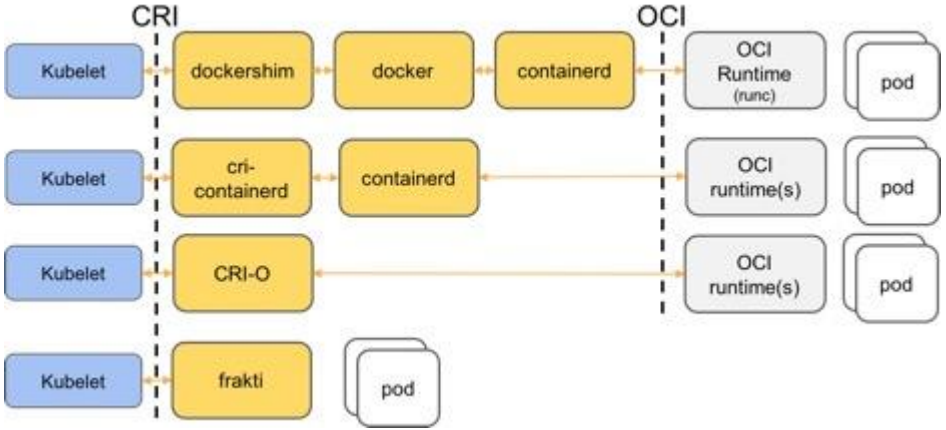

```
# /etc/containerd/config.toml

...
[plugins.linux]
...
          runtime = "kata"
```

Kata & Containerd CRI Plugin

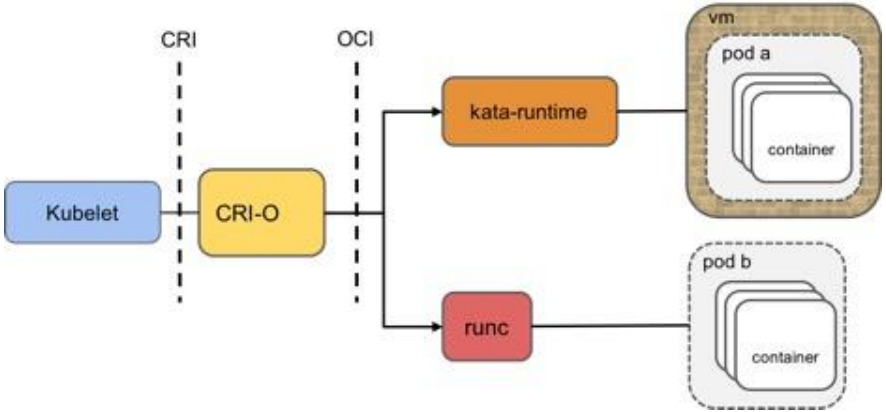

```
[plugins.cri]
...
   [plugins.cri.containerd]
   ...
     [plugins.cri.containerd.untrusted_workload_runtime]
      # kata or cc (?)
      runtime_engine = "kata"
```

Kata & K8s

K8s and any OCI



K8s (CRI) and Kata



Credit: https://katacontainers.io/posts/why-kata-containers-doesnt-replace-kubernetes/

Kata and CRI-O


```
# /etc/crio/crio.conf
...
runtime_untrusted_workload = "/usr/local/bin/kata-runtime"
default_workload_trust = "untrusted"
...
```

Kata & Frakti

Startup:

```
$ frakti --v=3 --logtostderr --listen=/var/run/frakti.sock --hyper-endpoint=127.0.0.1:22318 &
$ kubelet --container-runtime=remote --runtime-request-timeout=15m
--container-runtime-endpoint=/var/run/frakti.sock
$ cat >/etc/hyper/config <<EOF
Kernel=/var/lib/hyper/kernel
Initrd=/var/lib/hyper/hyper-initrd.img
# Storage driver for hyperd, valid value includes devicemapper, overlay, and aufs
StorageDriver=overlay
# Hypervisor to run containers and pods, valid values are: libvirt, qemu, kvm, xen
Hypervisor=qemu
# The tcp endpoint of gRPC API
gRPCHost=127.0.0.1:22318
EOF
$ systemctl restart hyperd
```

Kata

Pros

- Isolation
- Security
- Good Performance
- Stability of VMs
- Stateful apps
  - Data Security
  - VM attaches to storage

Cons

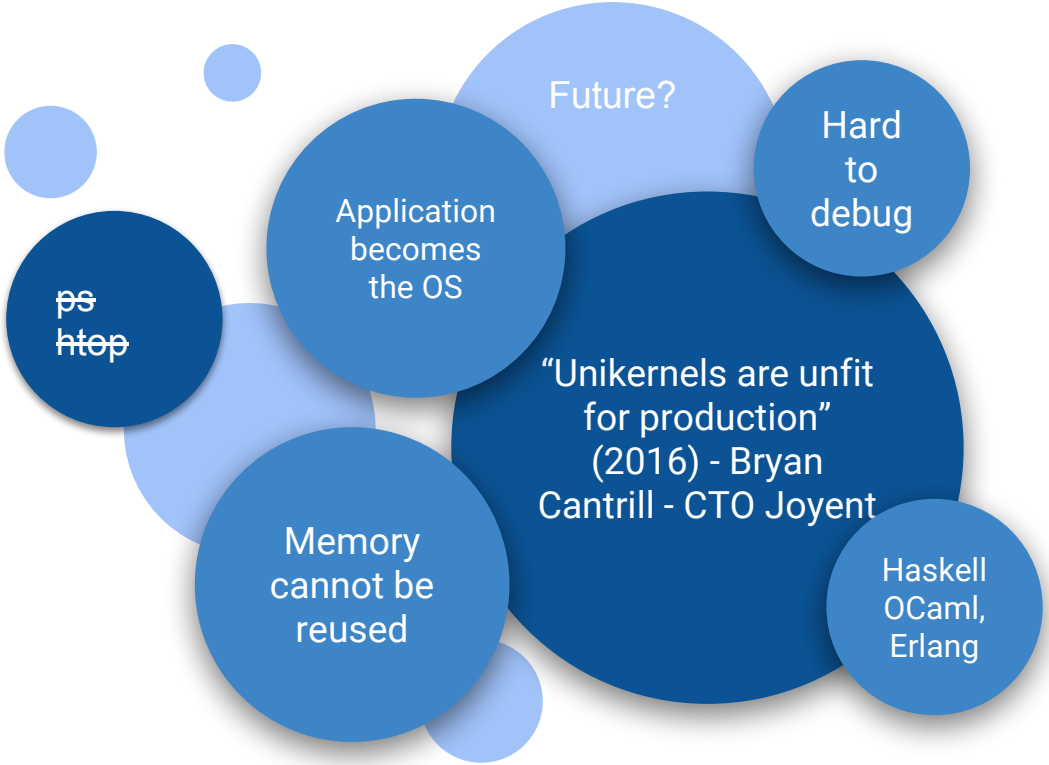
- Generally needs baremetal
- Startup time
  - VM template helps
  - No firmware helps
- AWS, GCP, Azure VMs cannot be used yet
  - C5s in AWS not ready
- For now slower than runC

Other Notables

Nvidia runtime	• Specific implementation for GPU • Modified runC with libnvidia-container
railcar	• OCI implementation in Rust
Pouch	• Alibaba's runtime • Shim that uses runC (OCI)
systemd-nspawn	• Based on systemd management • Some namespace isolation
lmctfy	• Google Internal Containers • FOSS - Development is stalled

https://en.wikipedia.org/wiki/Operating-system-level_virtualization

A Word On Unikernels



https://www.joyent.com/blog/unikernels-are-unfit-for-production

What Do I Use Now?

Stability	• Docker/containerd
Best K8s integration	• cri-containerd (Now CRI Plugin) • CRIO
Good Performance	• Kata
Better Performance	• Any with runC • rkt
Best Performance	• crun

What Do I Use Now?

K8s & Mesos Env	• Docker/containerd
Spark & Flink	• Mesos with Docker/containerd
Good Security & Iso	• Any with runC • rkt
Best Security & Iso	• Kata

Future

OCI — **CRI** — **Docker** — **CRIO** — **Kata** — **Cgroups V2**

- **OCI**: Convergence together with CNI, CSI and Distro spec
- **CRI**: K8s way of talking to all runtimes
- **Docker**: Adopt CRI Plugin for K8s
- **CRIO**: Continue as an alternative to containerd
- **Kata**: More security, isolation and performance
- **Cgroups V2**: Support in runtimes

Runtimes Running Towards OCI

Resources

Docker/containerd/cri	• https://github.com/containerd/cri • https://containerd.io/
Mesos	• http://mesos.apache.org/documentation/latest/containerizers/
CRI (Interface)	• http://blog.kubernetes.io/2016/12/container-runtime-interface-cri-in-kubernetes.html
CRI-O or CRIO	• http://cri-o.io/
Kata	• https://katacontainers.io/
Unikernels	• http://unikernel.org/

Thanks!