rkt and Kubernetes

What's new (and coming) with Container Runtimes and Kubernetes

(and a bit about the CNCF, too)

Jonathan Boulle

github.com/jonboulle - @baronboulle

Berlin site lead

CoreOS

A CLI for running app containers on Linux.

Focuses on:

- Security
- Composability
- Standards/Compatibility

rkt - a brief history

- **December 2014 - *v0.1.0***
  - Prototype
  - Drive conversation (security, standards) and competition (healthy OSS) in container ecosystem
- **February 2016 - *v1.0.0***
  - (already) used in production
  - API stability guarantees
- **June 2016 - *v1.8.0 (?)***
  - Packaged in Debian, Fedora, Arch, NixOS

A CLI for running app containers on Linux.

Focuses on:

- Security
- Composability
- Standards/Compatibility

A CLI for running app containers on Linux.

Focuses on:

- **Security**
- Composability
- Standards/Compatibility

How rkt does security

- **UX: "*secure-by-default*"**
  - Verify image signatures by default
  - Verify image integrity by default
- **Architecture: *Unix philosophy***
  - Well-defined operations
  - No central privileged long-running components
  - Separate privileges for different operations ("fetch" operations shouldn't need root)

How rkt does security

Classic and modern Linux technologies

- User namespaces
  - container euid != host euid
- SELinux contexts
  - isolate individual pods
- Support for VM containment
  - lightweight hypervisor (= hardware isolation)
- TPM measurements
  - Tamper-proof audit log of what's running

How rkt does security (soon)

- Finer-grained Linux capabilities
- Tighter seccomp defaults
- SELinux across more platforms
- cgroup2 + cgroup namespaces support
- Support for more hypervisors (QEMU)
- Unprivileged containers (run entirely as non-root)

A CLI for running app containers on Linux.

Focuses on:

- **Security**
- Composability
- Standards/Compatibility

A CLI for running app containers on Linux.

Focuses on:

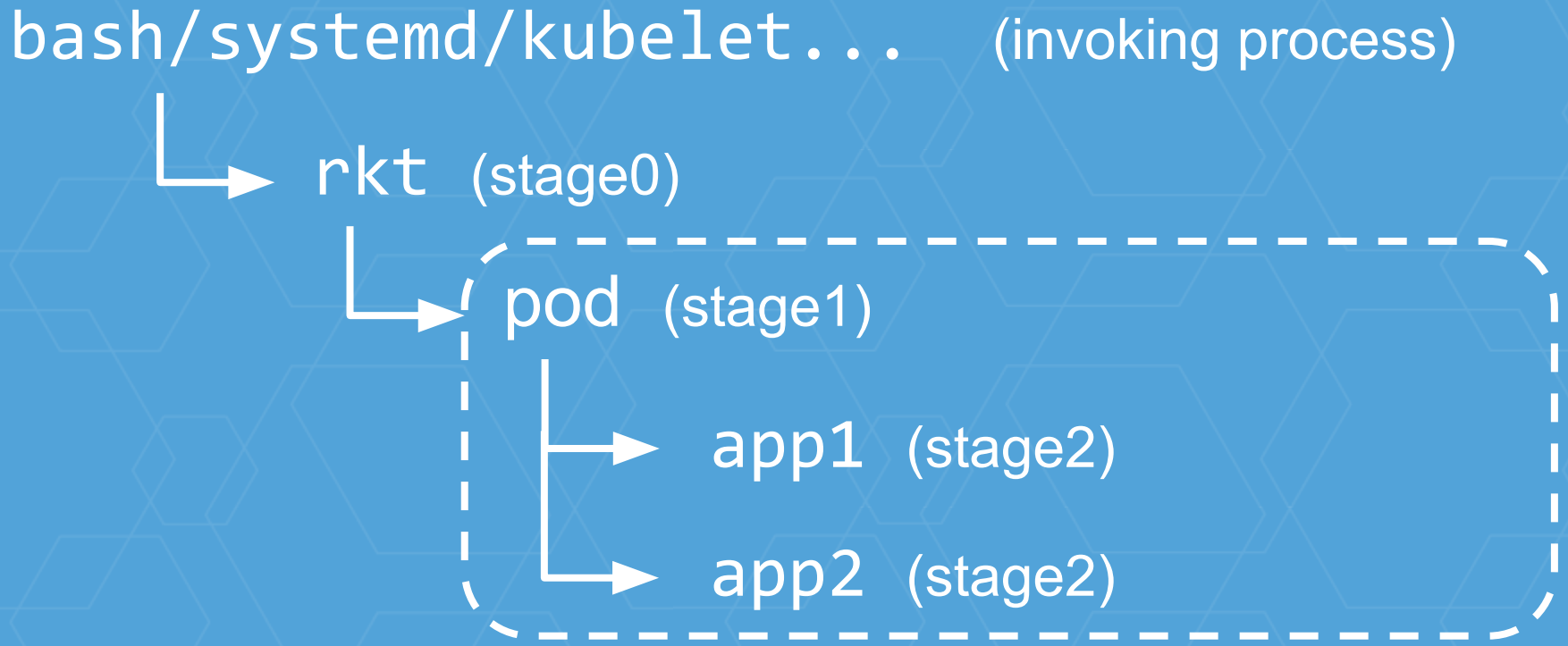
- Security
- **Composability**
- Standards/Compatibility

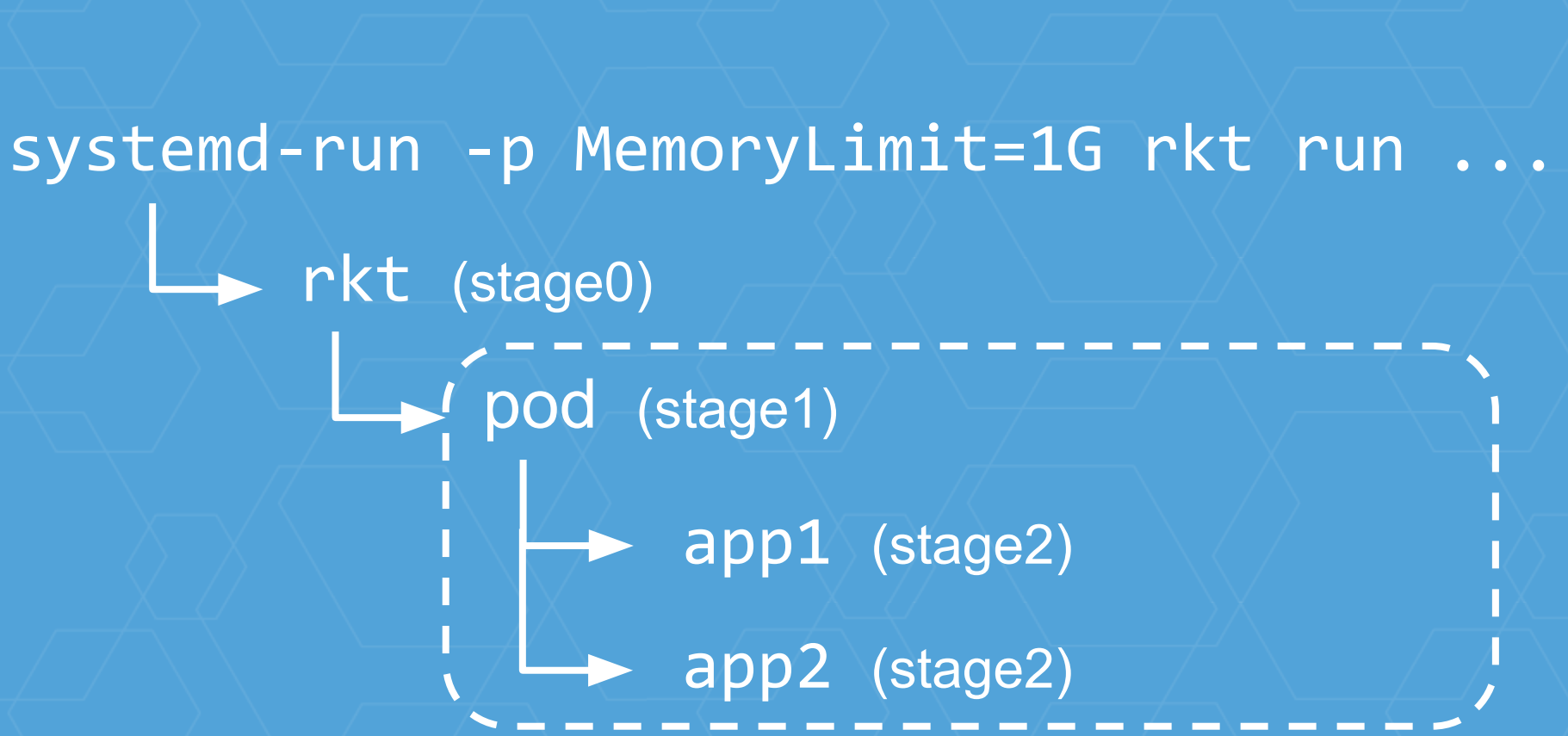
How rkt does composability

- **"*External*" composability**
  - Integrating well with init system(s) is a priority
  - Aims to work well with other projects, like cluster orchestration (more on this in a moment)
- **"*Internal*" composability**
  - Stage-based architecture
  - Swappable execution engines that actually runs the container

How rkt does composability

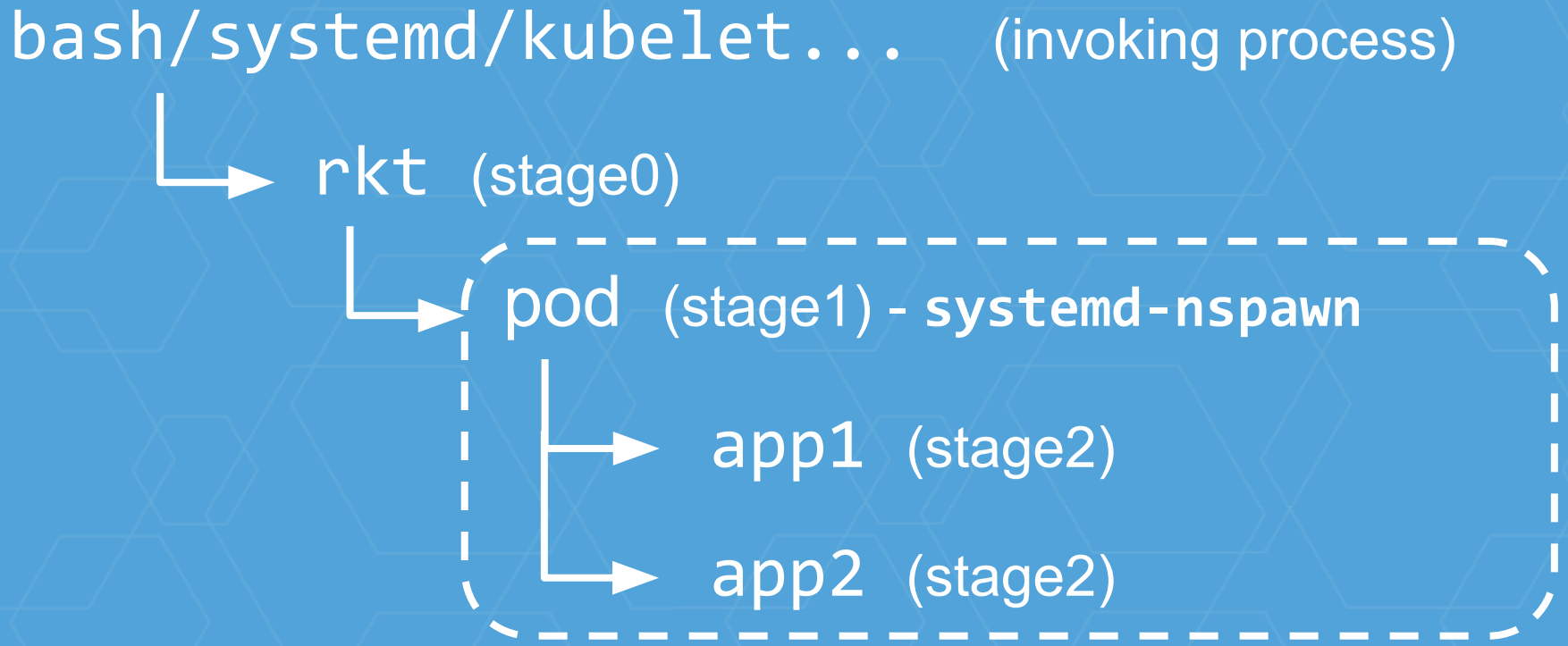
- **"*External*" composability**
  - Simple process model: rkt command *is* the container
  - Any context applied to rkt (cgroups, etc) applies transitively to the pod and the apps inside
  - Optional gRPC (HTTP2+Protobuf) API server to facilitate more efficient introspection

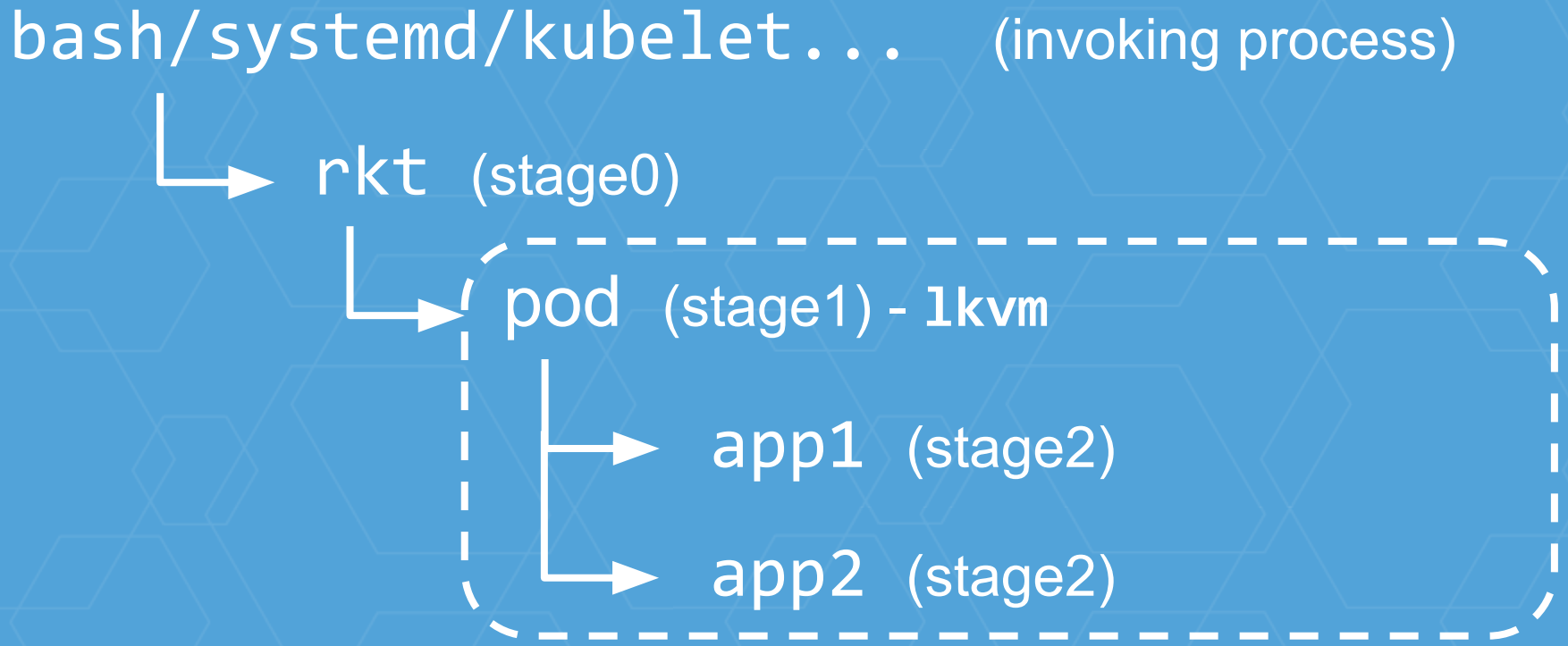

bash/systemd/kubelet...
(invoking process)
rkt (stage0)
pod (stage1)
app1 (stage2)
app2 (stage2)

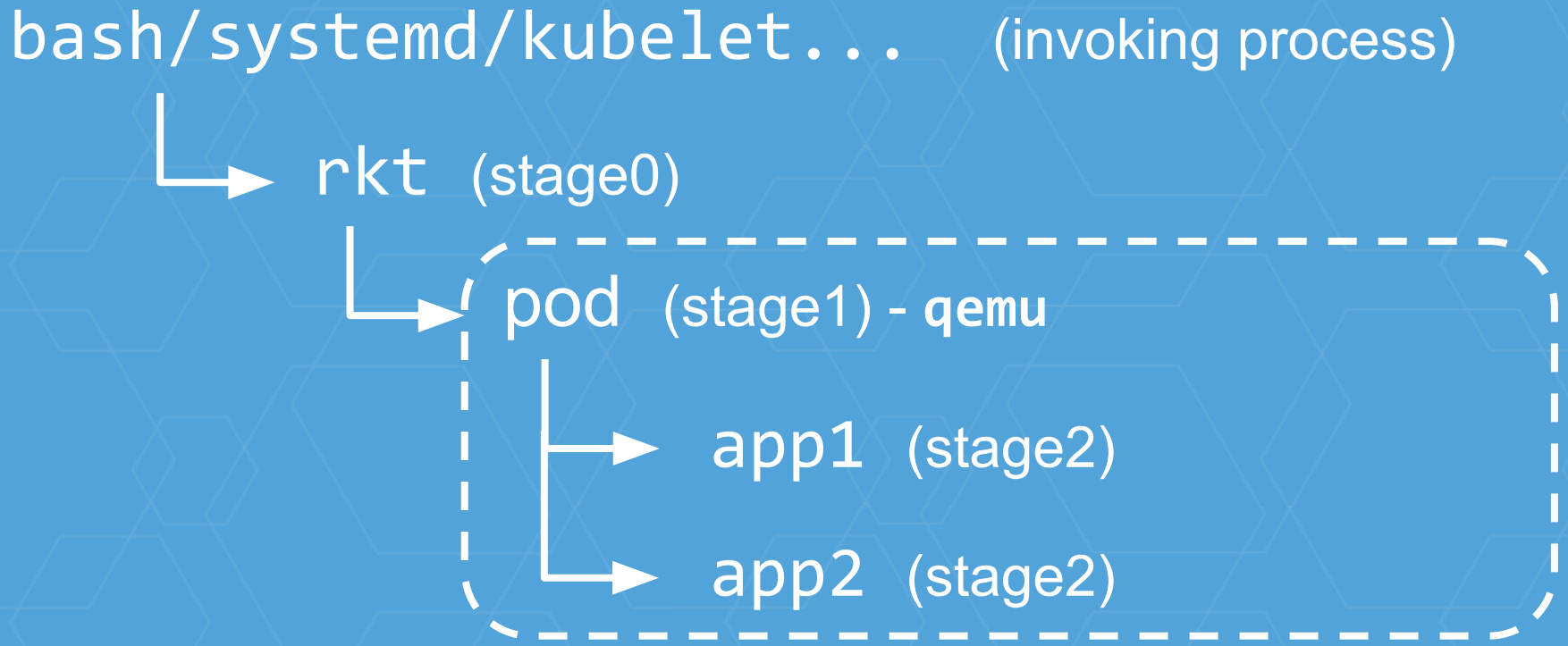

systemd-run -p MemoryLimit=1G rkt run ...
rkt (stage0)
pod (stage1)
app1 (stage2)
app2 (stage2)

How rkt does composability

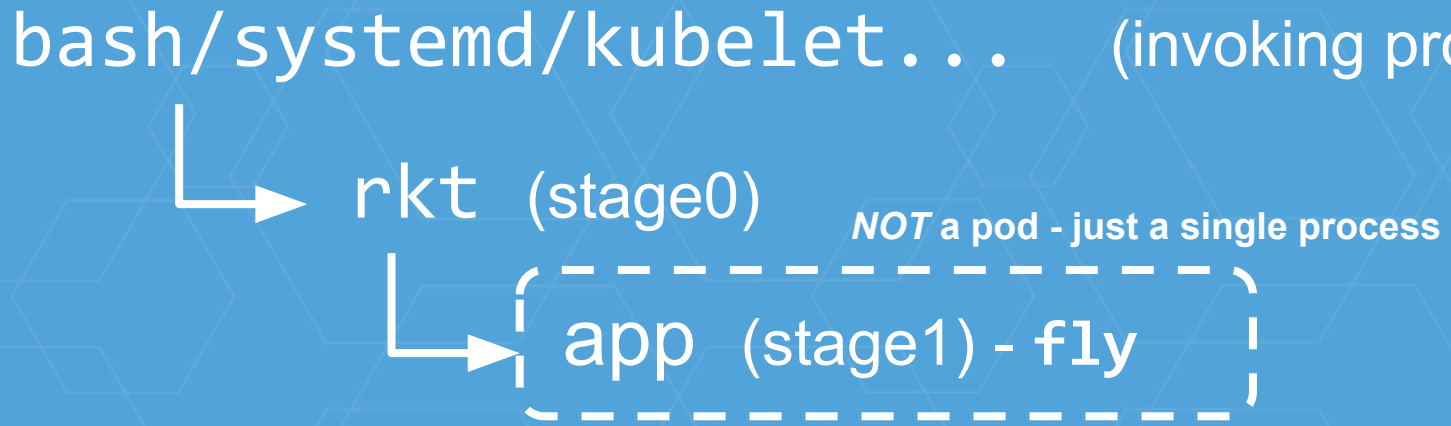
- **"Internal" composability**
  - Staged architecture
  - "rkt" is the UX/API, container technology is an implementation detail
- **Available stage1s**
  - cgroups + Linux namespaces (default)
  - LKVM
  - chroot ("fly")
  - QEMU (landing soon)


bash/systemd/kubelet...
(invoking process)
rkt (stage0)
pod (stage1) - systemd-nspawn
app1 (stage2)
app2 (stage2)


bash/systemd/kubelet...
(invoking process)
rkt (stage0)
pod (stage1) - lkvm
app1 (stage2)
app2 (stage2)


bash/systemd/kubelet...
(invoking process)
rkt (stage0)
pod (stage1) - qemu
app1 (stage2)
app2 (stage2)


bash/systemd/kubelet...
(invoking process)
rkt (stage0)
app (stage1) - fly


bash/systemd/kubelet...
(invoking process)
rkt (stage0)
NOT a pod - just a single process
app (stage1) - fly

rkt + `fly`

- Leverage the safety and security features of rkt (image retrieval, signature validation) without the containment
- Lightweight, isolated package manager
- Enables interesting use cases (more later)

A CLI for running app containers on Linux.

Focuses on:

- Security
- **Composability**
- Standards/Compatibility

A CLI for running app containers on Linux.

Focuses on:

- Security
- Composability
- **Standards/Compatibility**

How rkt does standards/compatibility

- (original) implementation of **appc**
  - first attempt at a well-defined container spec
- Uses **CNI** for networking
  - common plumbing used by many other projects (Kubernetes, Cloud Foundry, Project Calico, Weave, ...)
- Can run **Docker images** natively (V1, V2, ...)
- Developers participate actively in **standardisation efforts**
  - appc, OCI, CNCF
  - rkt will be fully OCI compliant

How do standards work?

- appc (December 2014)
  - some adoption, but (intended to be) deprecated in favour of
- OCI (June 2015)
  - container runtime and image format
- CNCF (December 2015)
  - "harmonising cloud-native technologies"

How do (image format) standards work?

	Docker v1	appc	Docker v2.2	OCI (in progress)
Introduced	2013	December 2014	April 2015	April 2016
Content-addressable	No	Yes	Yes	Yes
Signable	No	Yes, optional	Yes, optional	Yes, optional
Federated namespace	Yes	Yes	Yes	Yes
Delegatable DNS namespace	No	Yes	No	Yes

How do (networking) standards work?

CNI, the "Container Networking Interface"

Used by:

- rkt
- Kurma
- Kubernetes
- Cloud Foundry
- Weave
- Project Calico
- Contiv Networking

```
{
    "name": "mynet",
    "type": "bridge",
    "bridge": "cni0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "subnet": "10.22.0.0/16",
        "routes": [
            { "dst": "0.0.0.0/0" }
        ]
    }
}
```

How do (CNCF) standards work?

Or.. what even *is* the CNCF?

A brief digression...

CLOUD NATIVE
COMPUTING
FOUNDATION

CNCF:
a short overview

What is Cloud Native?

Software and patterns for...

Container based

Microservices oriented

Programmable infrastructure

CNCF - a commons under LF

What IETF is to Internet & email...

What Apache is to Big Data...

(Hadoop & ecosystem)

… CNCF is to Cloud Native

CNCF - what lives there?

IP:

- Open Source - running code
- Specifications for interoperability & patterns

Infra:

- eg. Intel cluster for large scale testing

CNCF - modern principles

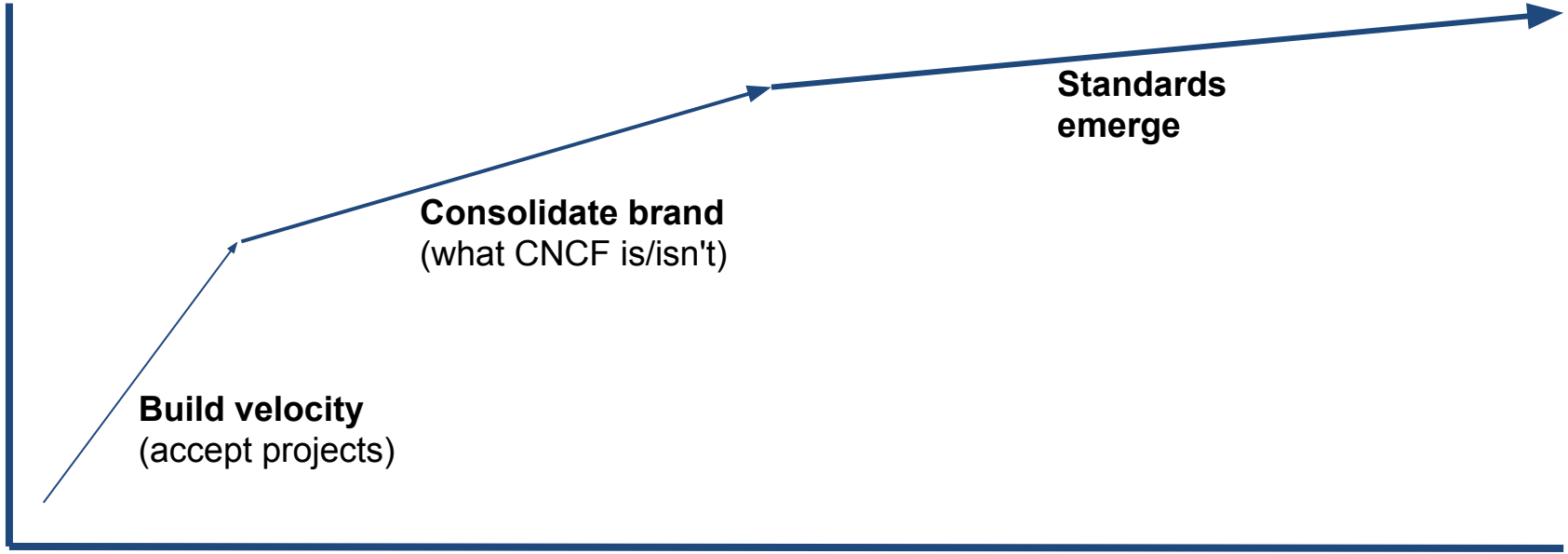
“No kingmakers”

- Competing approaches are OK
- Emergent standards based on real world USE

“GitHub way”

- Projects enjoy largely autonomous governance
- TOC as facilitator: helps, enables, encourages

CNCF - three phase plan



CNCF - Technical Oversight Committee

Company	Name	Email
Apache Software Foundation	Camille Fournier	skamille@gmail.com
Cisco	Ken Owens	kenowens@cisco.com
CoreOS	Jonathan Boulle	jonathan.boulle@coreos.com
Docker	Solomon Hykes	solomon.hykes@docker.com
GoDaddy	Elissa Murphy	elissam@godaddy.com
Google	Brian Grant	briangrant@google.com
Joyent	Bryan Cantrill	bryan@joyent.com
Mesosphere	Benjamin Hindman	benh@mesosphere.io
Weaveworks	Alexis Richardson	alexis@weave.works

CNCF - which projects?

High Quality & Velocity

Cloud Native

Foundation Affinity

Ok, no really, which projects?

Kubernetes (first project accepted)

... but this is NOT a "Kubernetes foundation"

We want lots more projects

Our doors are open

Ok, no really, which projects?

Prometheus (second project accepted)

Open source monitoring and metrics

Ok, no really, which projects?

Proposed:

- HyperContainer

Discussions:

- gRPC
- NATS
- CNI

CNCF - cluster

1000 node Intel cluster

For use by the CNCF community

Work on, test, and demo CNCF projects, at scale

Now live and accepting requests!

https://github.com/cncf/cluster

CNCF - cluster

1000 node Intel cluster

For use by the CNCF community

Work on, test, and demo CNCF projects, at scale

Now live and accepting requests!

https://github.com/cncf/cluster

CLOUD NATIVE
COMPUTING
FOUNDATION

CNCF tl;dr:
watch this space

A CLI for running app containers on Linux.

Focuses on:

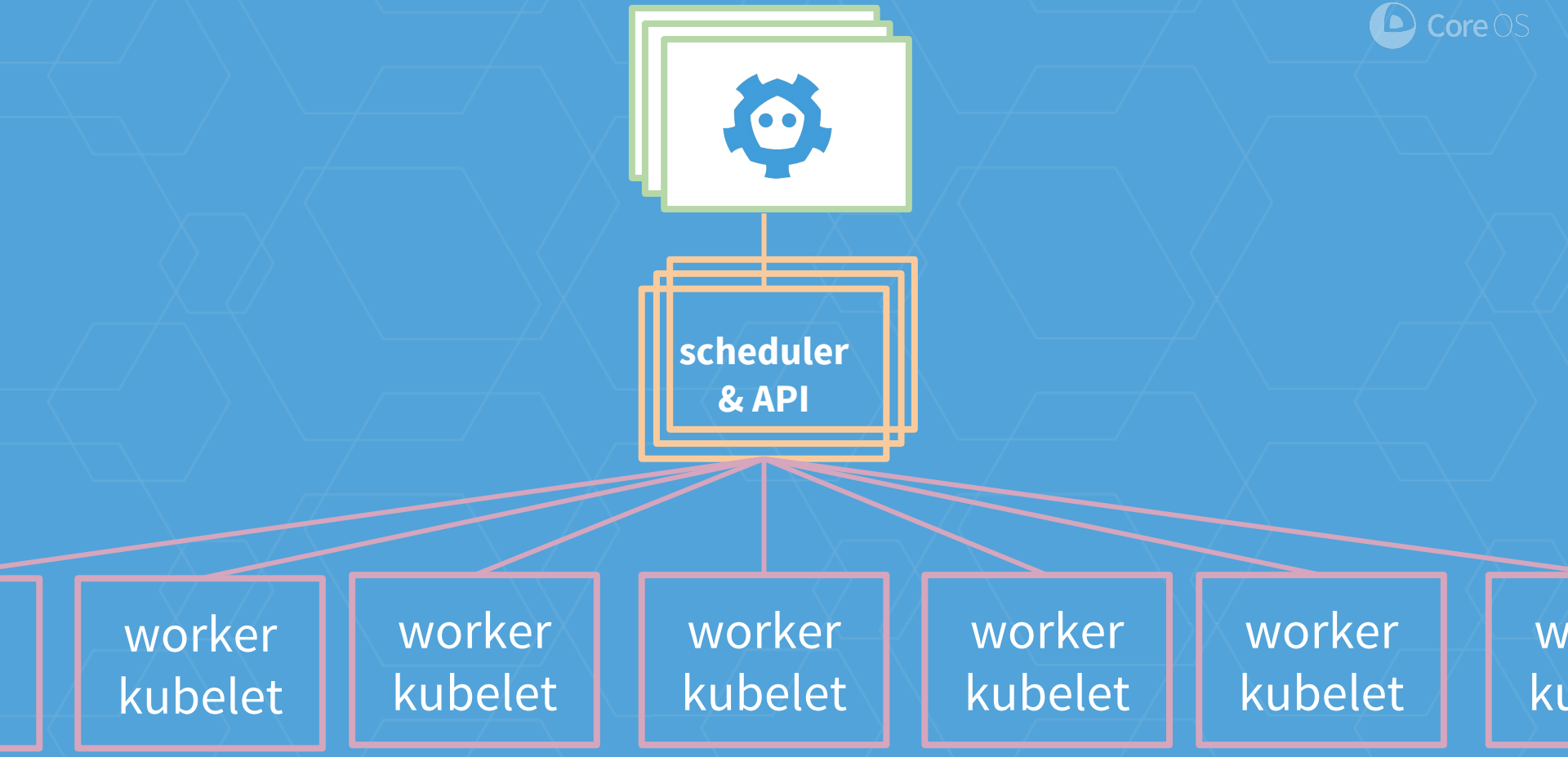
- Security
- Composability
- Standards/Compatibility

Kubernetes

Cluster-level container orchestration.

Handles:

- Scheduling/Upgrades
- Failure recovery
- Scaling


scheduler
& API
worker
kubelet
worker
kubelet
worker
kubelet
worker
kubelet
worker
kubelet

Kubernetes

Cluster-level **container** orchestration.

Handles:

- Scheduling/Upgrades
- Failure recovery
- Scaling

Kubernetes

Cluster-level **container** orchestration.

- Today , container = Docker container
- No reason for this strictly to be the case
- Kubernetes API (mostly) exposes only *pods*

Kubernetes

Cluster-level **pod** orchestration.

- Pod is a grouping of one or more applications sharing certain context (networking, volumes, ...)

How does it work?

- kubelet is a daemon that runs on every worker node in a Kubernetes cluster
- Kubernetes schedules *pods* for a kubelet to run
- kubelet delegates to container runtime to perform all container-related operations (start container, fetch images, ...)

How does it work?

- **Kubelet code provides a `Runtime` (golang) interface**
  - `SyncPod()`
  - `GetPod()`
  - `KillPod()`
  - `...`
- **in theory, anyone can implement this**
- **in practice, lots of Docker assumptions**

Kubernetes + rkt = rktnetes

Have Kubernetes use rkt as
the container runtime.

rkt handles:

- Image discovery
- Image fetching
- Pod execution

How does it work?

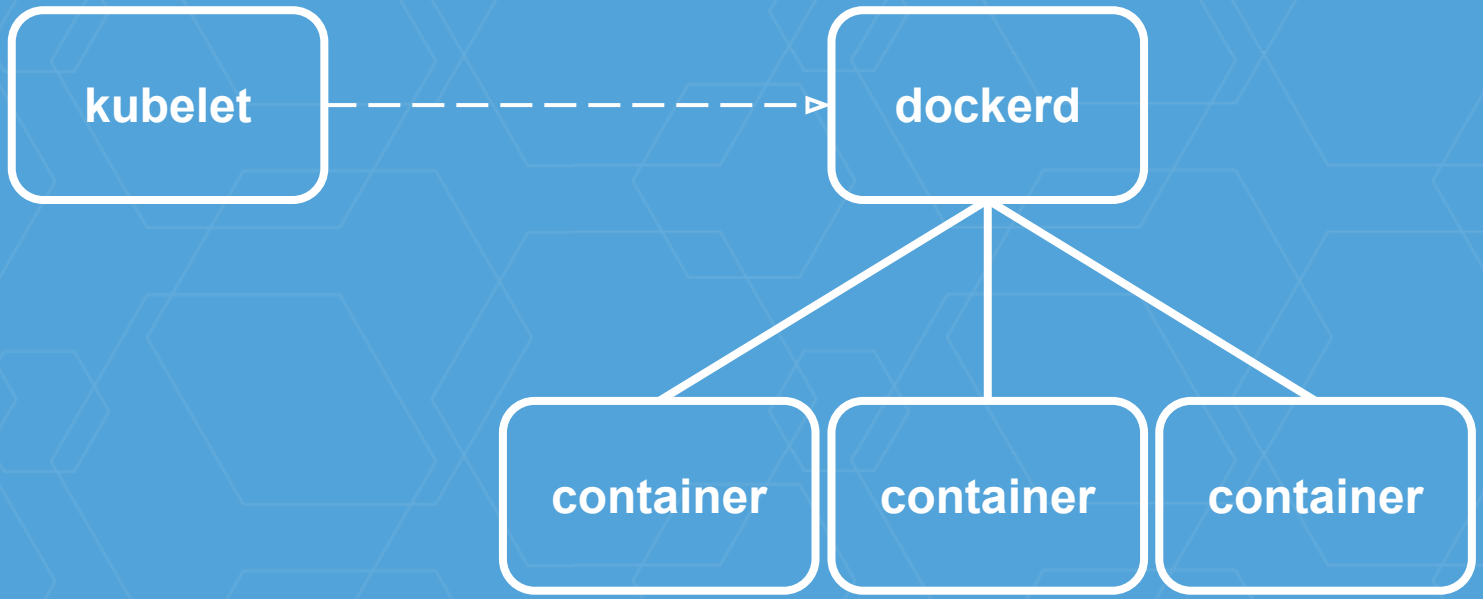
Pre-rktnetes (default):

- Kubelet talks to the Docker daemon for all tasks

With rktnetes changes:

- Kubelet talks to the rkt API daemon for read-only tasks
  - e.g. list pods, get logs
- Kubelet execs rkt directly for preparatory tasks
  - e.g. fetch images, create pod root filesystems
- Kubelet talks to systemd for running pods via rkt
  - e.g. launch containers

Kubelet + Docker (default)

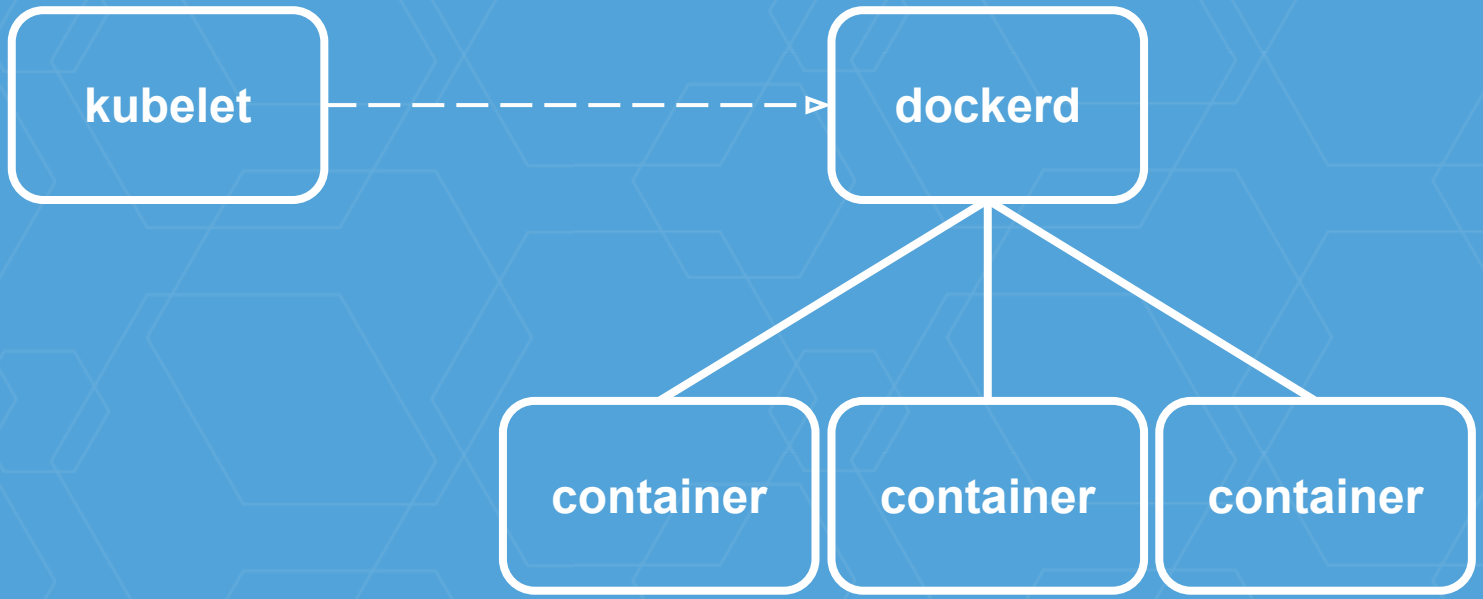


Kubelet + Docker (default)

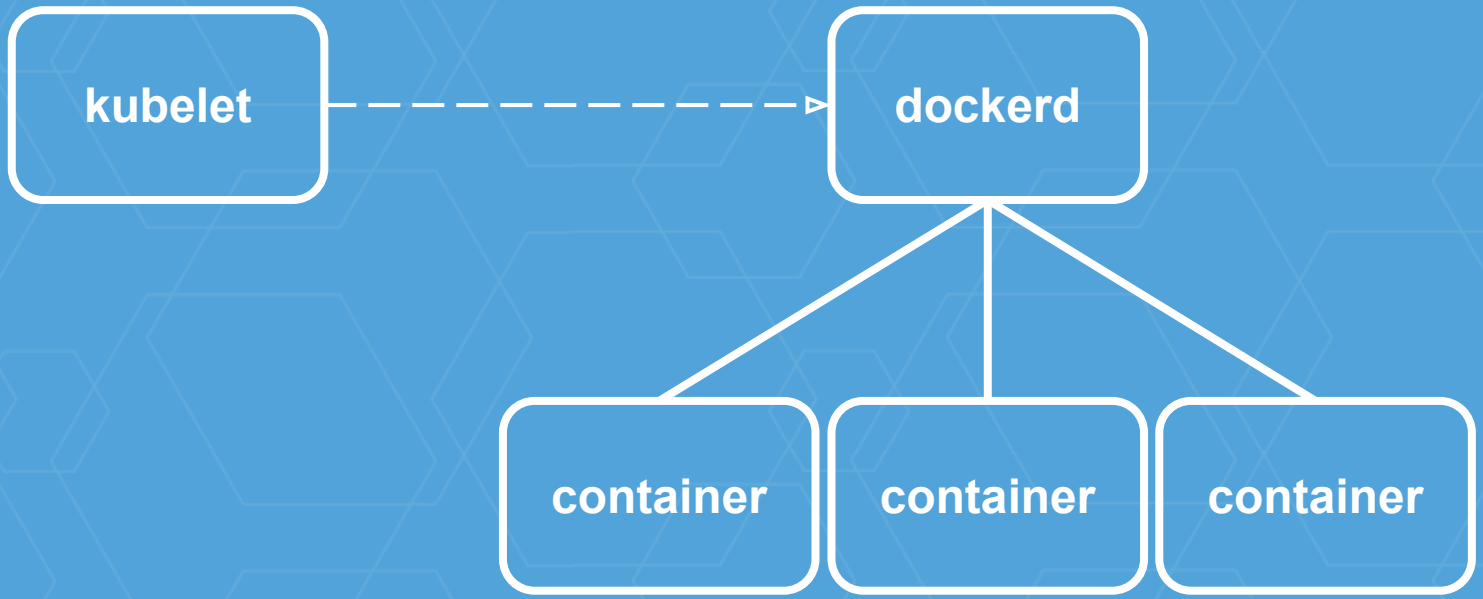
Problems

- ***Docker doesn't understand pods***
  - kubelet must maintain pod<->container mapping
  - "infra container" to hold namespaces for pod
- ***dockerd = SPOF for the node***
  - if Docker goes down, so do all containers
- ***Docker doesn't interact well with systemd***
  - who looks after cgroups?

Kubelet + Docker (default)



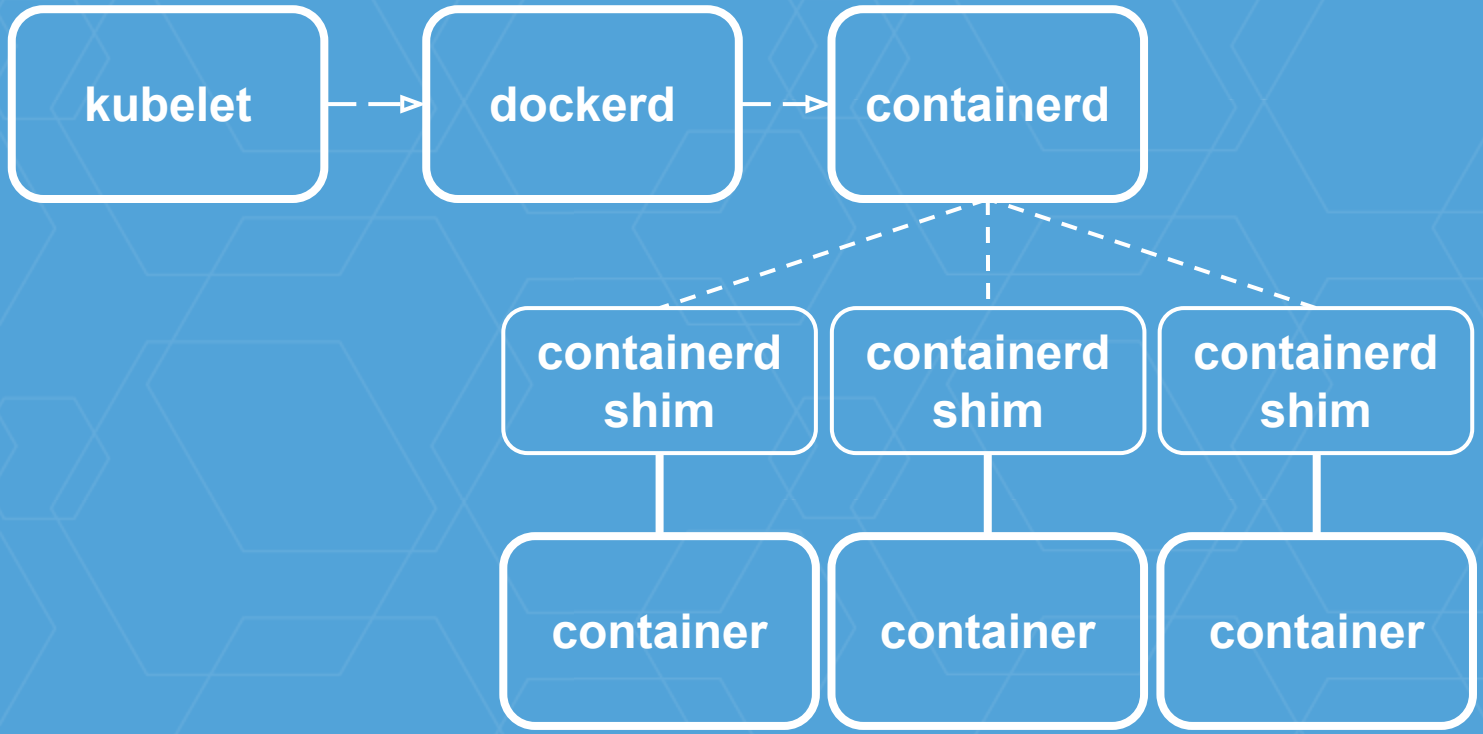
Kubelet + Docker (before Docker 1.11)



Kubelet + Docker (future)

- **Docker 1.11 introduced *containerd***
  - New daemon (outside of dockerd) to control container lifecycle
  - each Docker container is started and monitored by an individual "shim" process

Kubelet + Docker (1.11+ with containerd)



Kubelet + Docker (1.11+ with containerd)

- `dockerd` is no longer SPOF, but (for now) `containerd` is
- In future (1.12+), containerd will support persistence
  - "*Upgrade daemon without restarting containers*" **https://github.com/docker/docker/issues/2658**
- But...
  - per-container overhead, many moving parts
  - still have systemd integration issues

Kubelet + rkt (rktnetes)

- Using rkt as the kubelet's container runtime
- A *pod-native* runtime
- First-class integration with systemd hosts
- self-contained pods process model = no SPOF
- Multi-image compatibility (e.g. docker2aci)
- Transparently swappable - no user impact

How does it work?

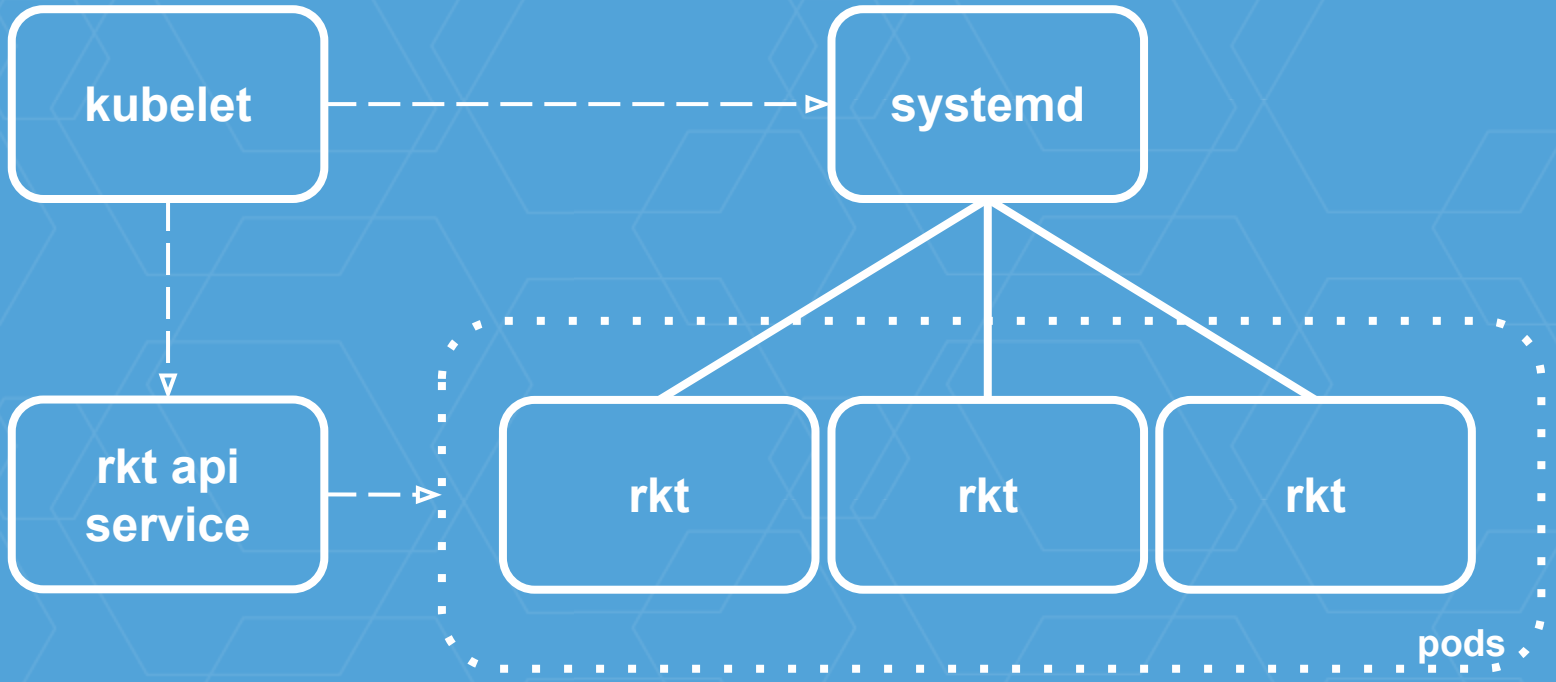
Pre-rktnetes:

- Kubelet talks to the Docker daemon for all tasks

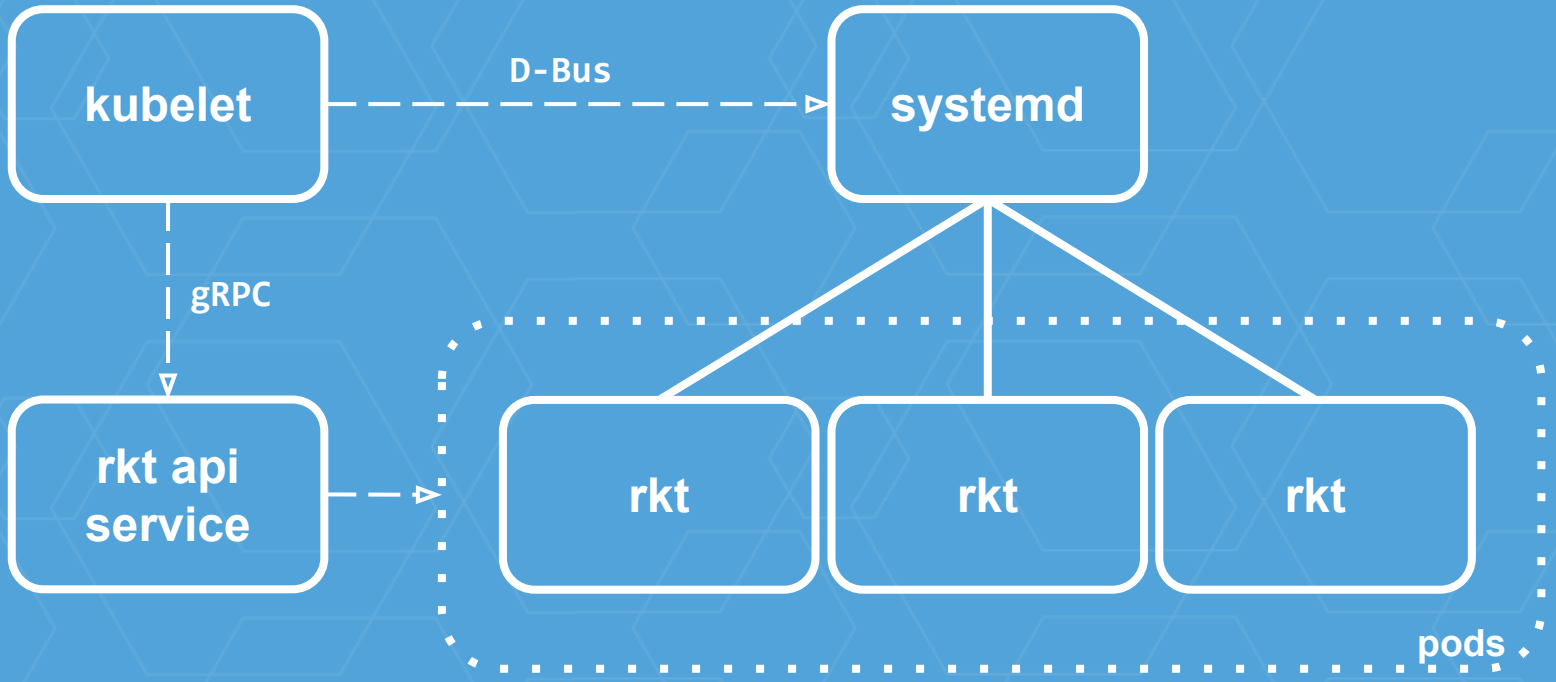
With rktnetes changes:

- Kubelet talks to the rkt API daemon for read-only tasks
- Kubelet execs rkt for preparatory tasks
- Kubelet talks to systemd for running pods via rkt

Kubelet + rkt (rktnetes - with systemd)



Kubelet + rkt (rktnetes - with systemd)

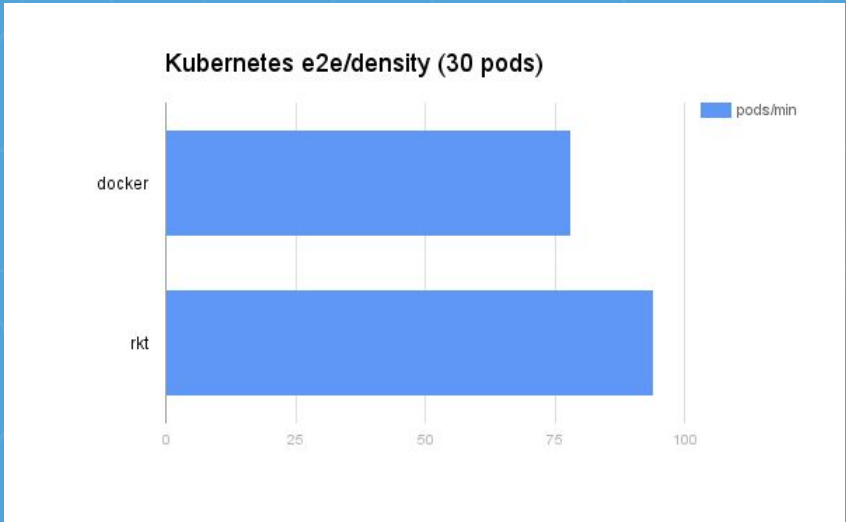


What's the benefit in this?

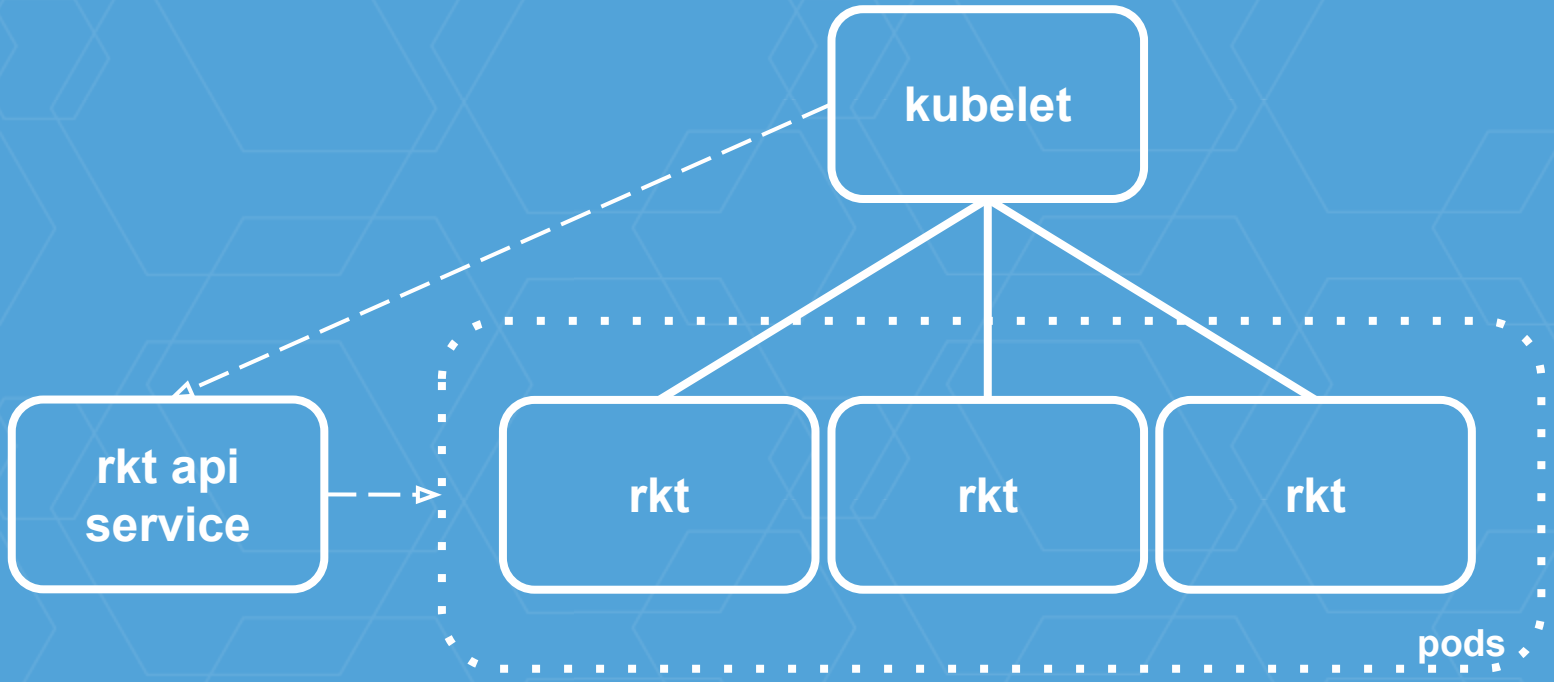
- **No daemon running the containers**
  - live upgrades of the container runtime without affecting existing pods
- **Multiple stage1s provides more flexibility**
  - Swap in more advanced isolation technologies without needing to modify Kubernetes
- **Seamless integration with systemd**
  - `machinectl`, `systemctl`, `journalctl` Just Work™
  - Increasingly important as systemd adoption grows

What's the benefit in this?

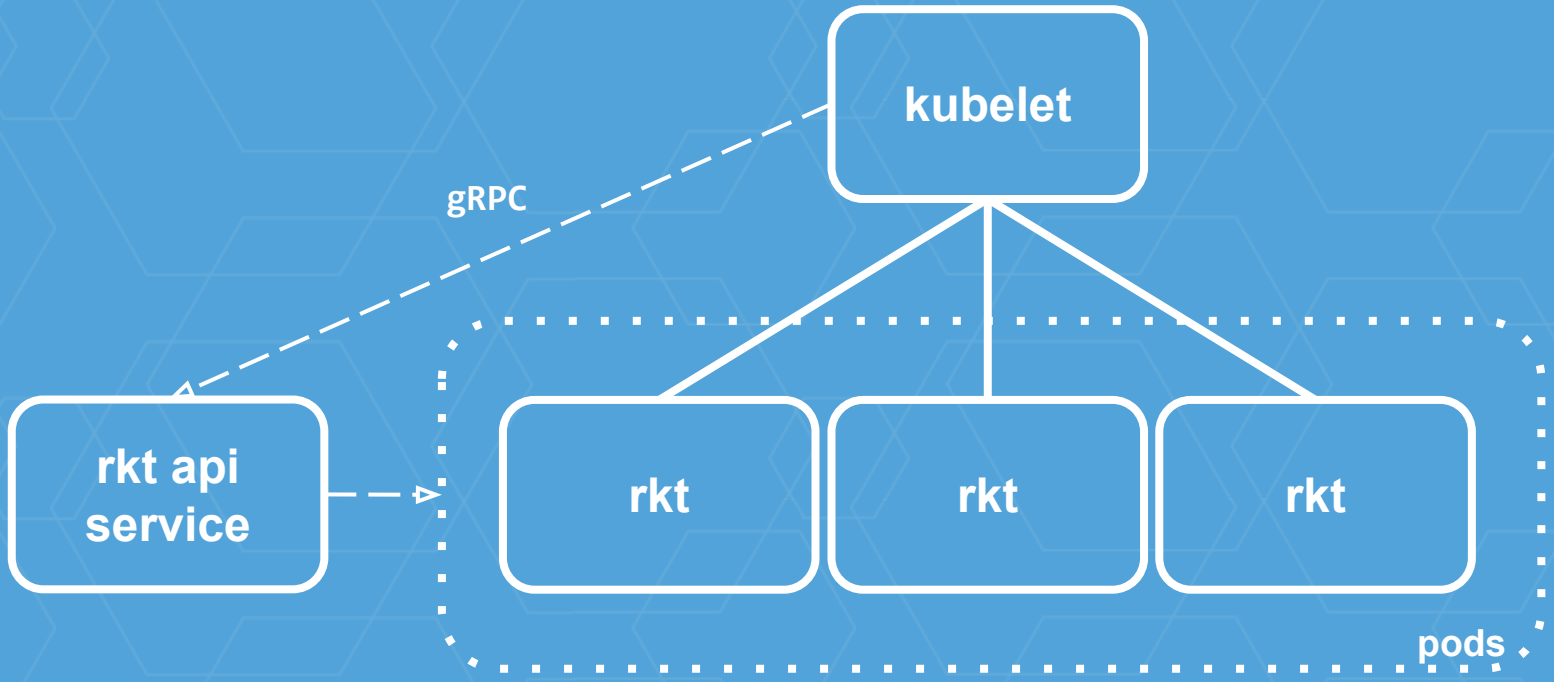
- **Paves the way for even more options**
  - runc
  - Hyper
- **Allows runtimes to compete on features/speed within Kubernetes**



Kubelet + rkt (rktnetes - without systemd)



Kubelet + rkt (rktnetes - without systemd)



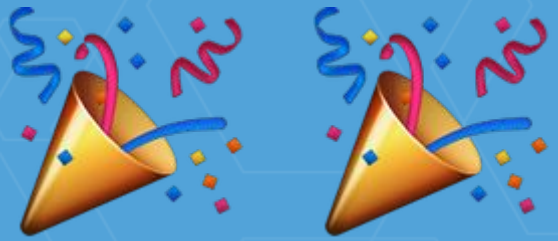
Kubelet + rkt (rktnetes - without systemd)

- **Benefits:**
  - Kubelet can retain complete/granular control over process lifecycle of container runtime
  - Remove one component from critical path (systemd)

- **Disadvantages:**
  - Kubelet needs to be well-behaved process manager (and still be compatible with systemd)
  - Kubelet is now SPOF for node

rktnetes: *does* it work?

- **Yes!**
- **It’s blasting off with a 1.0 release!***
  - With a small set of known issues
- **Official release in Kubernetes 1.3** (any day now)



How can I use it?

- A getting started guide is in the Kubernetes docs: http://kubernetes.io/docs/getting-started-guides/rkt/
- A simple, Vagrant-based rktnetes workshop: https://github.com/coreos/rkt8s-workshop
- Watch this space: http://rktnetes.io
- Fully supported, first-class backend in Kubernetes 1.3

What's coming with rkt and Kubernetes

Coming soon in rkt

- New architecture support (ARM64)
  - In upcoming 1.10 release
- New hypervisor support (QEMU)
  - https://github.com/coreos/rkt/pull/2684

Coming soon in rkt

- Unified cgroup hierarchy support
  - https://github.com/coreos/rkt/issues/1757
- Tighter privilege separation
  - Always drop euid when it's not needed
  - https://github.com/coreos/rkt/issues/2482

Coming soon in rkt (on CoreOS)

- Running Kubelet using rkt
  - `/usr/lib64/coreos/kubelet-wrapper`
  - `$KUBELET_VERSION` + `rkt run` (with fly)
- Running Docker using rkt
  - Custom Docker versions using `docker-in-rkt`
  - https://groups.google.com/d/msg/coreos-dev/icuel9OveRQ/0UIiE43yAwAJ
- Running everything using rkt!

Coming soon in rkt and Kubernetes

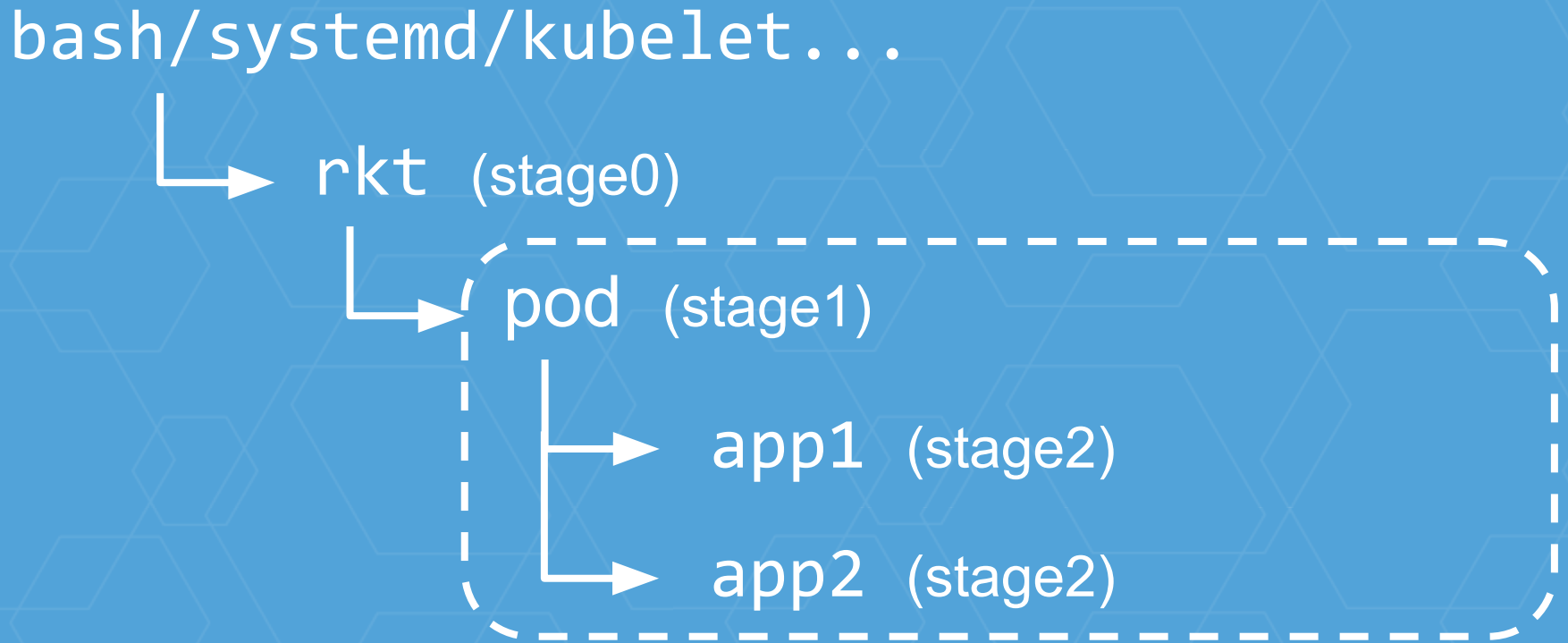
- seccomp isolation
  - https://github.com/kubernetes/kubernetes/pull/24602
  - https://github.com/coreos/rkt/pull/2753
- sysctl support
  - https://github.com/kubernetes/kubernetes/pull/26057
  - https://github.com/coreos/rkt/issues/2694

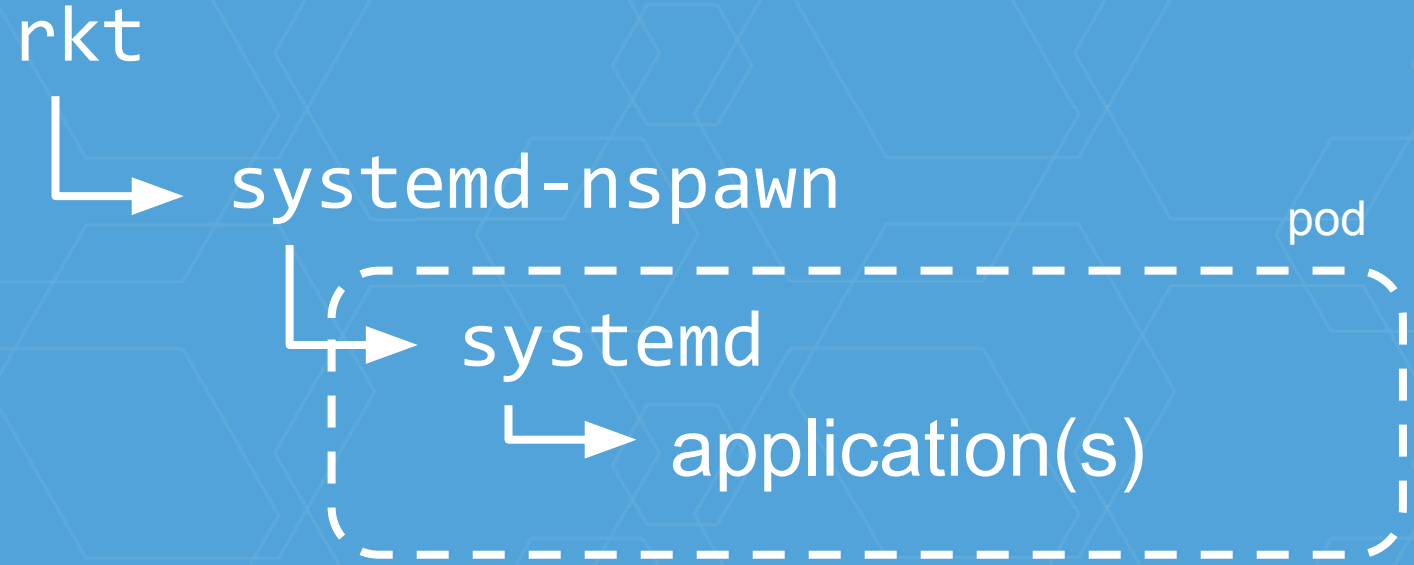
New Container Runtime Interface

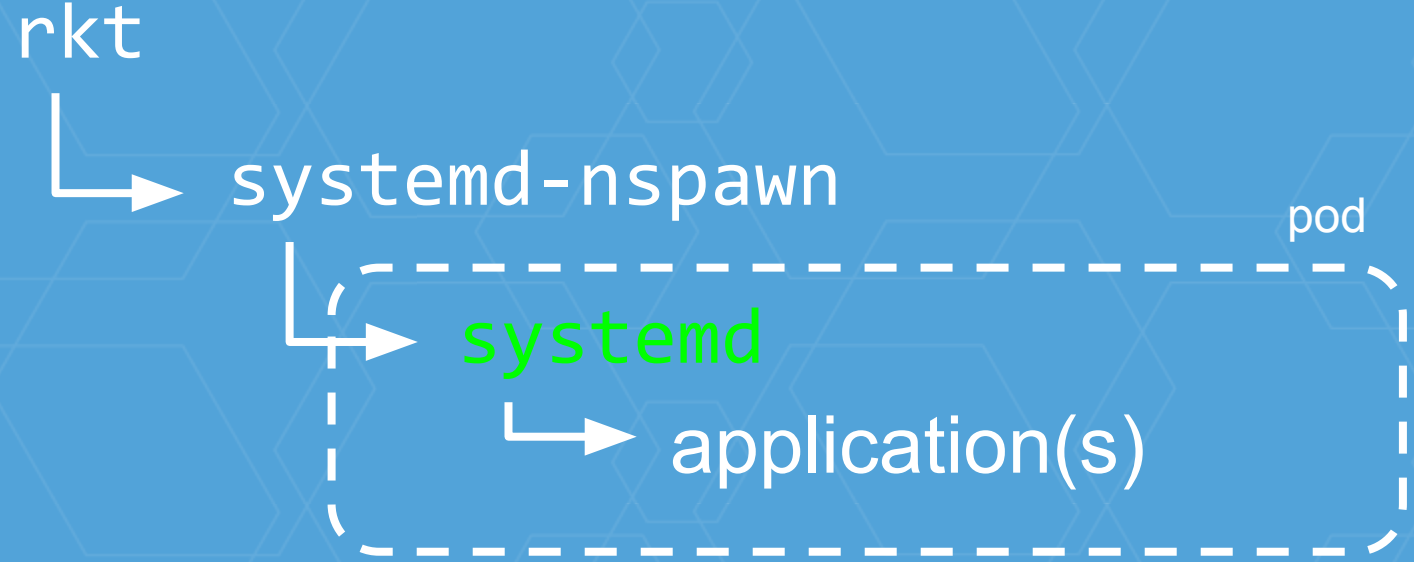
- k8s-sig-node wants to rework the interface between the kubelet and the container runtime
  - kubelet wants fine-grained control over containers
  - Move away from declarative monolithic function (`SyncPod`) to granular imperative operations (`CreatePod`, `CreateContainerInPod`, etc)
- Draft proposal up, targeted for Kubernetes 1.4+
  - https://github.com/kubernetes/kubernetes/pull/25899

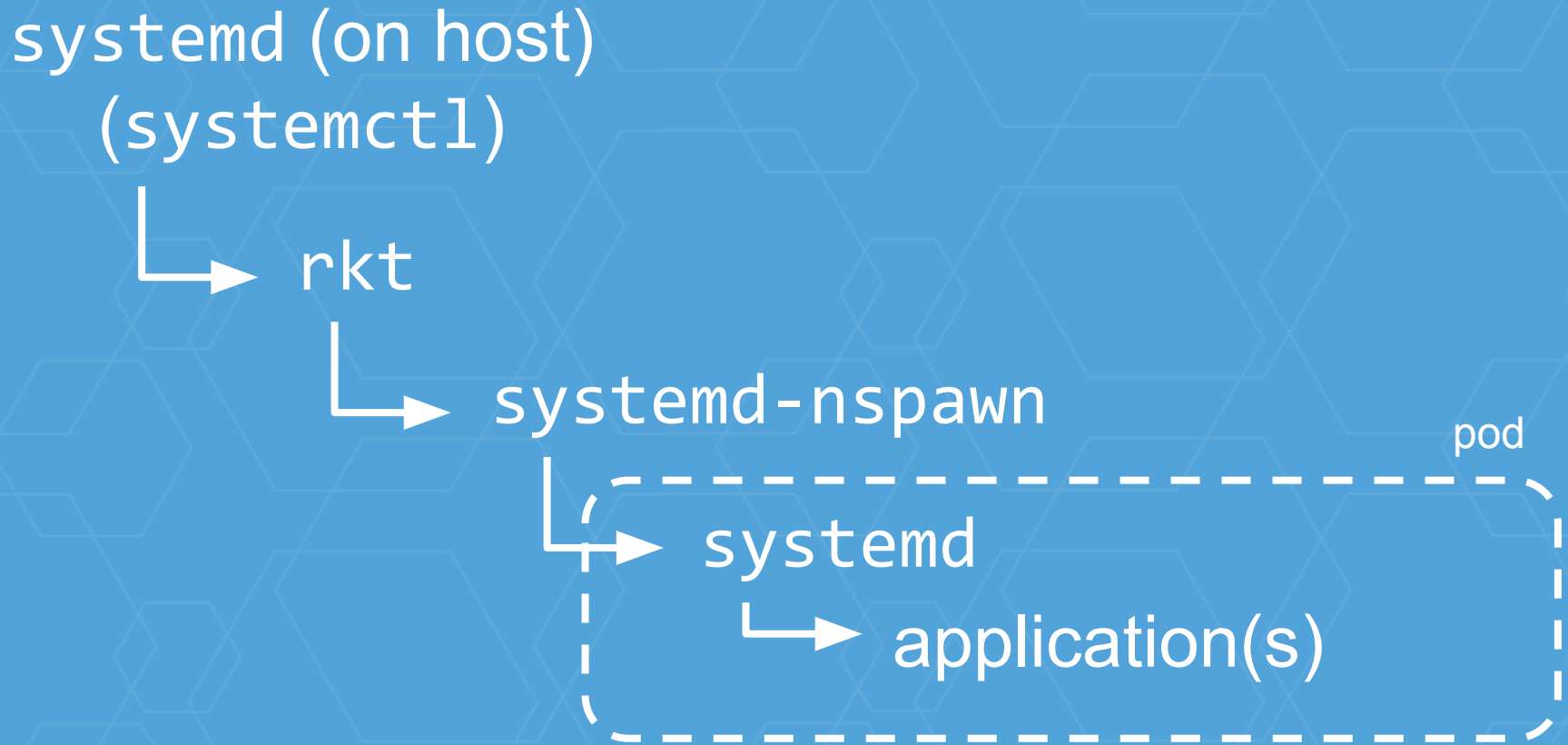
New Container Runtime Interface

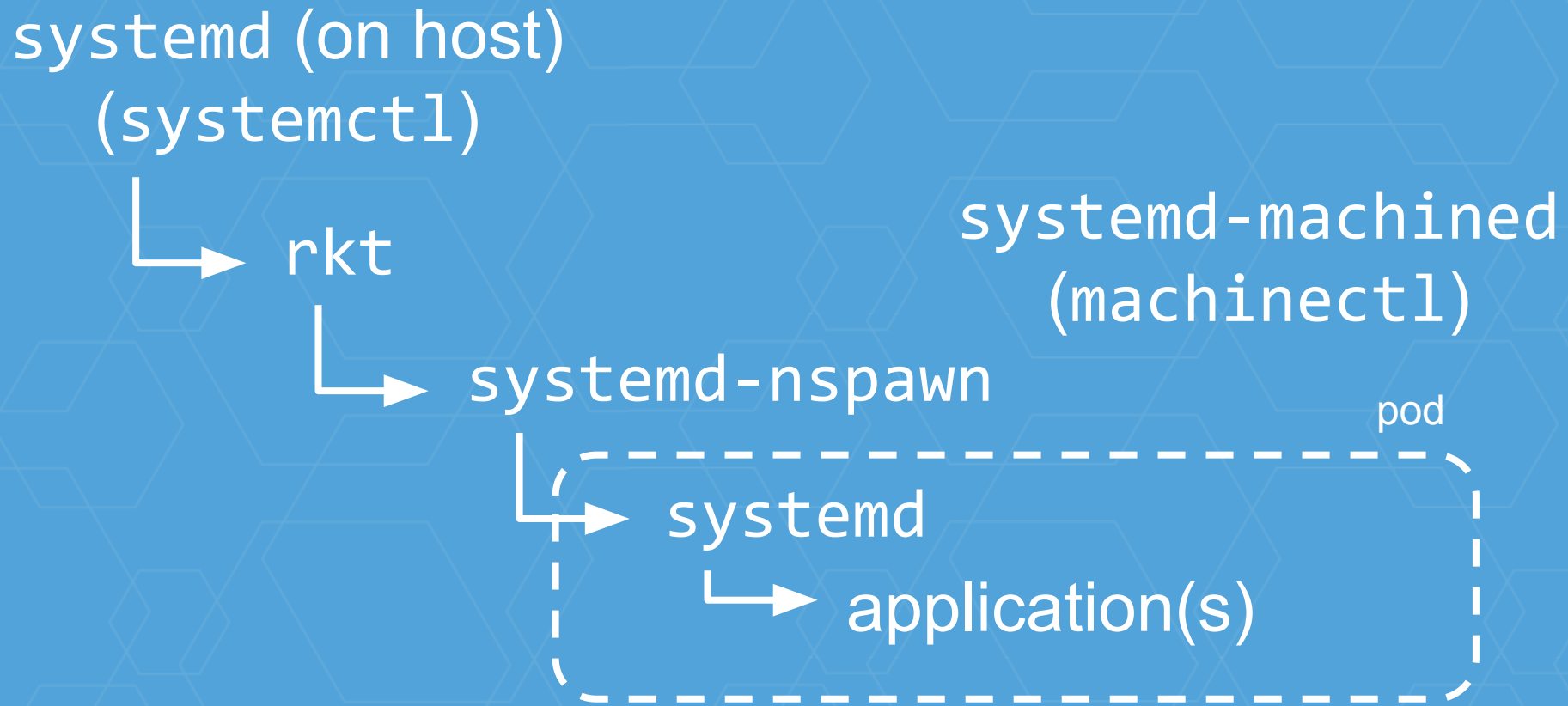
- Container-level operations: but what about pods?!
  - rkt gets container level operations "for free" by leveraging systemd:
    - `journalctl -M <podname> -u <app>`
    - `systemctl -M <podname> start <app>`
  - Still retain benefits of first-class pods
- https://github.com/coreos/rkt/issues/2375
- https://github.com/coreos/rkt/issues/1463

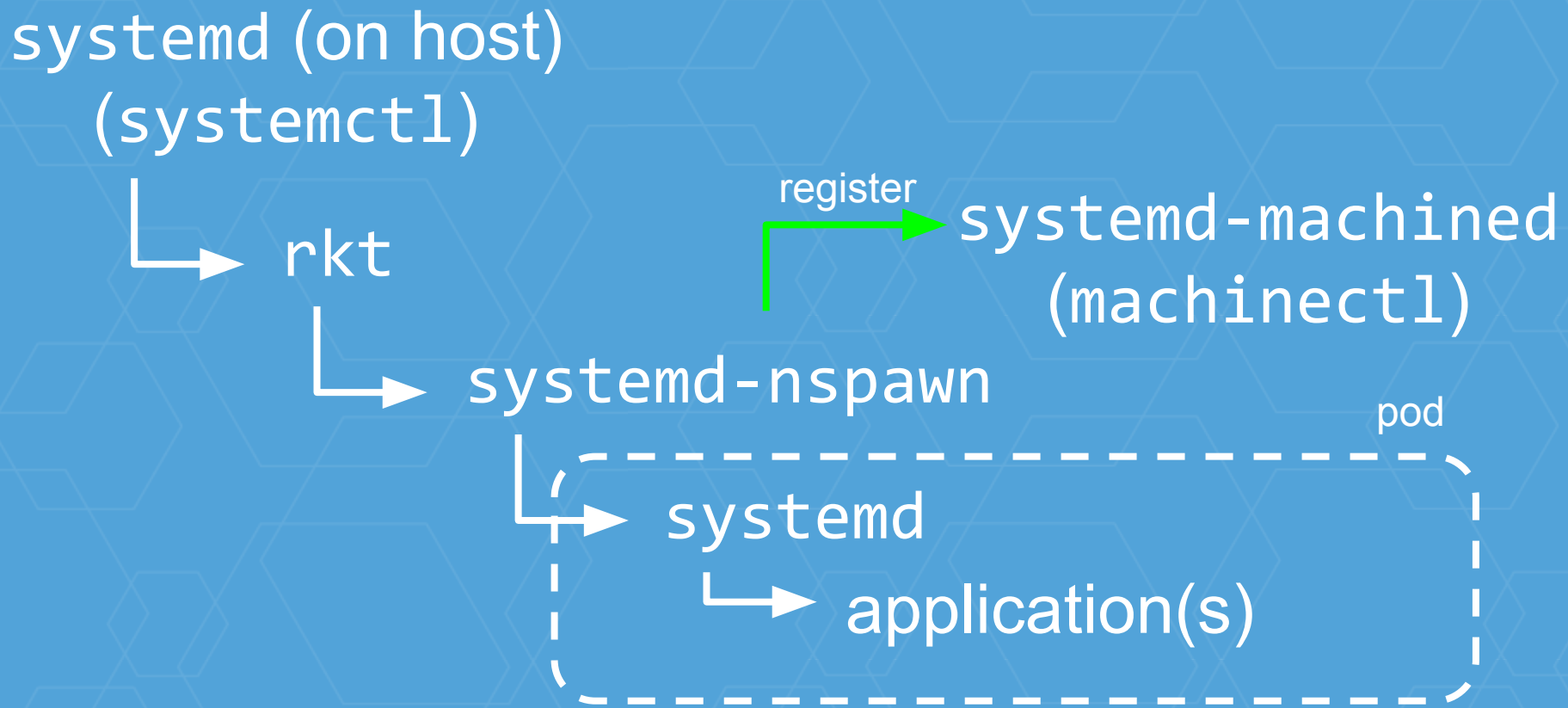

bash/systemd/kubelet...
rkt (stage0)
pod (stage1)
app1 (stage2)
app2 (stage2)

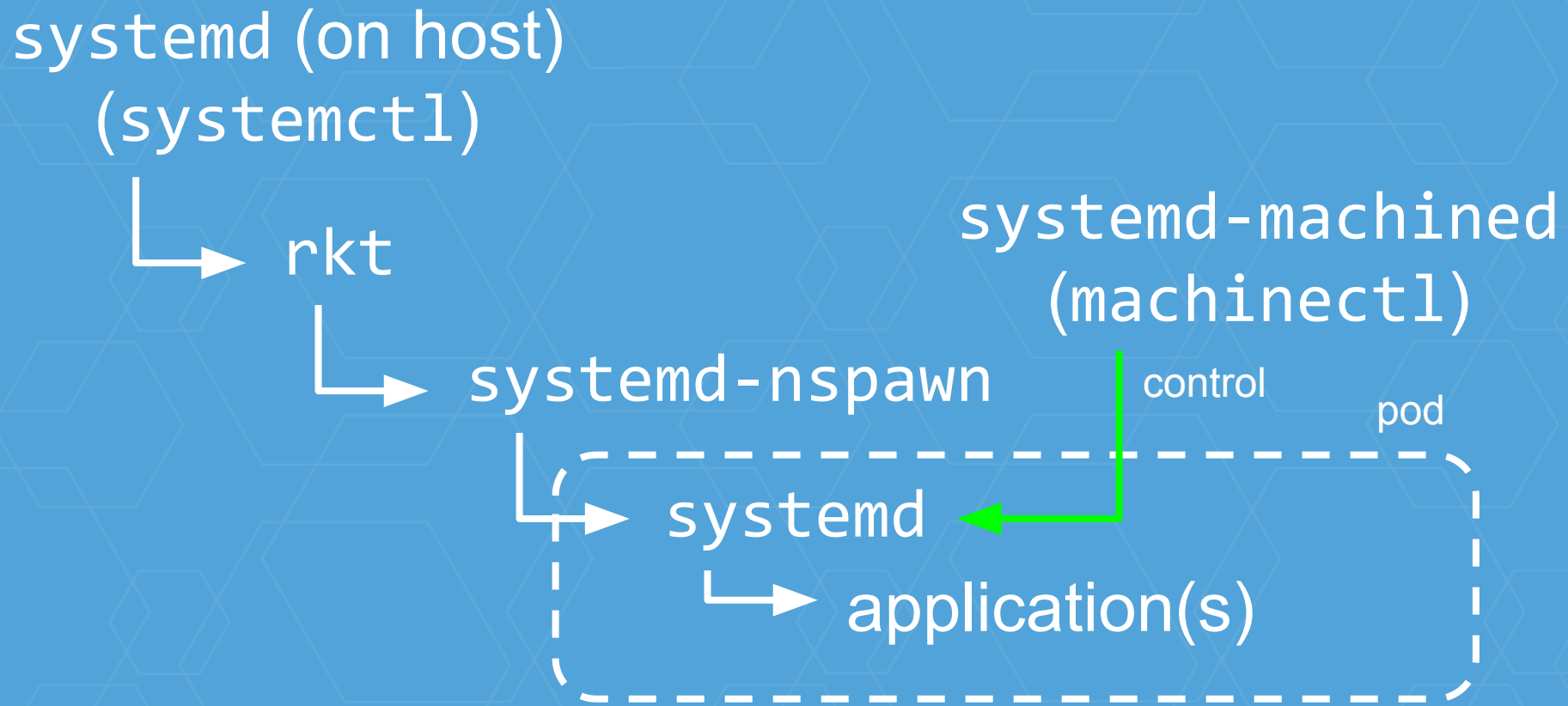

rkt
systemd-nspawn
pod
systemd
application(s)

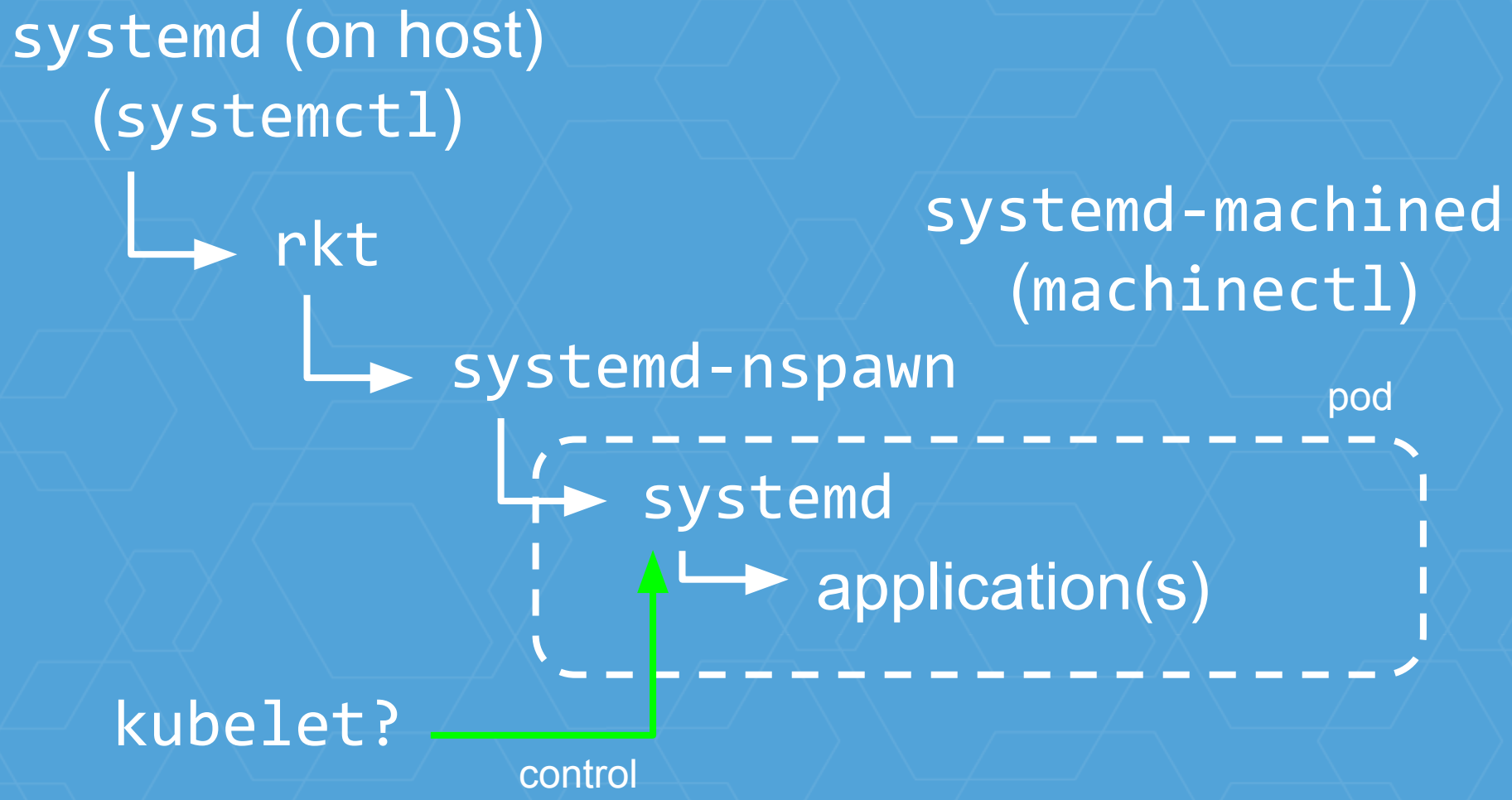

rkt
systemd-nspawn
pod
systemd
application(s)

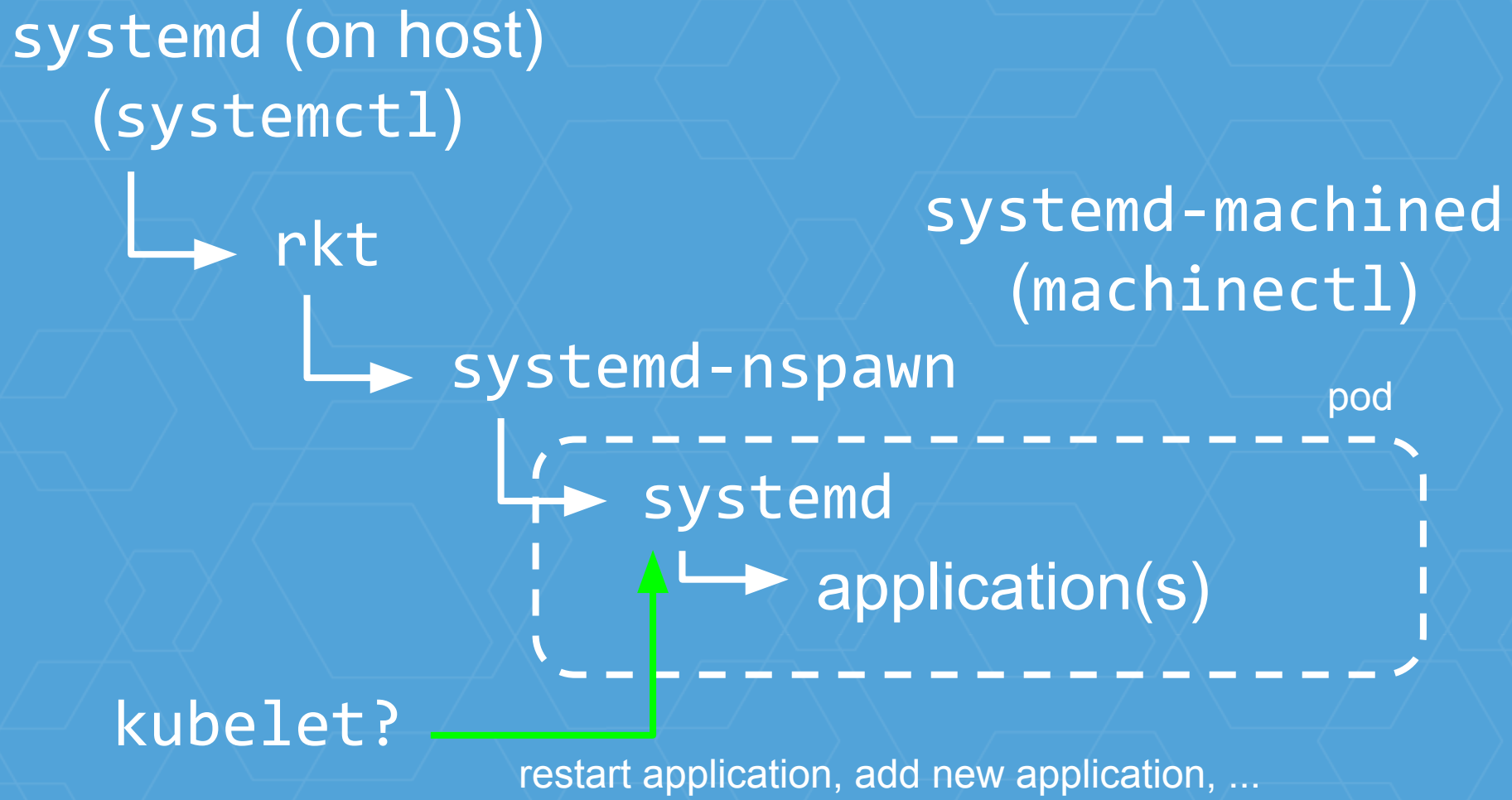

systemd (on host)
(systemctl)
rkt
systemd-nspawn
pod
systemd
application(s)


systemd (on host)
(systemctl)
rkt
systemd-machined
(machinectl)
systemd-nspawn
pod
systemd
application(s)


systemd (on host)
(systemctl)
rkt
register
systemd-machined
(machinectl)
systemd-nspawn
pod
systemd
application(s)

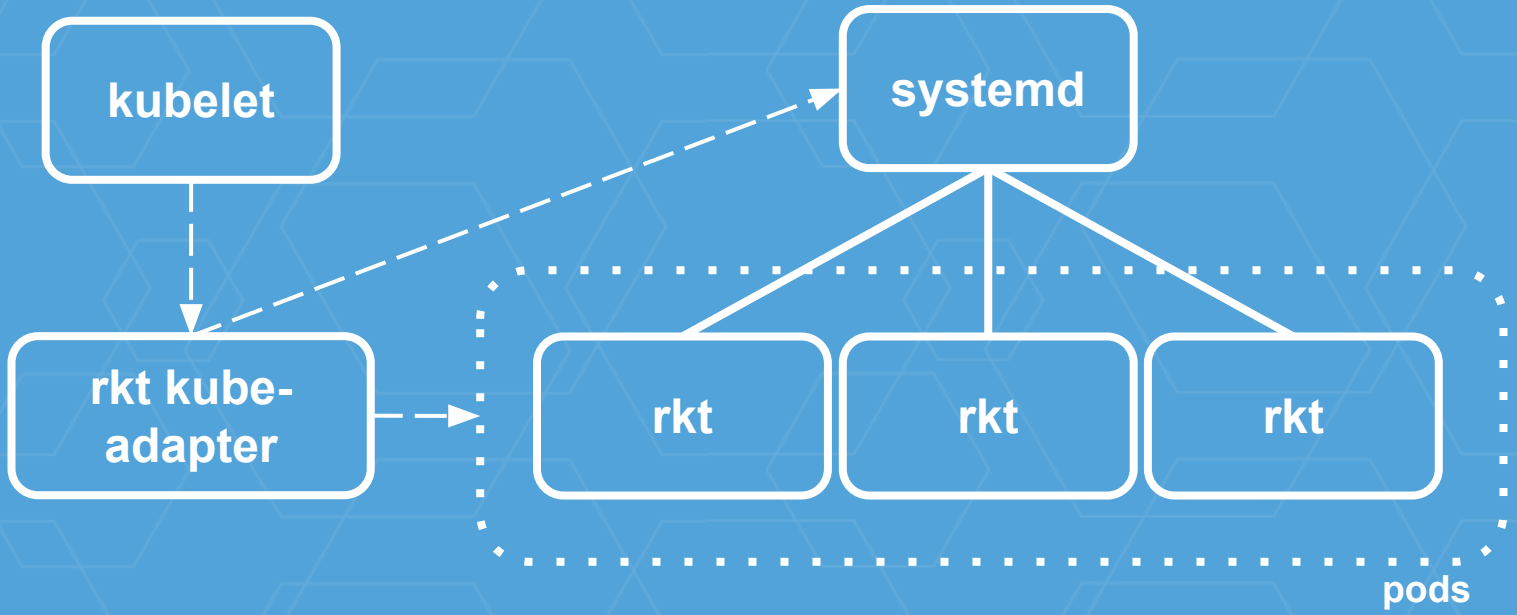

systemd (on host)
(systemctl)
rkt
systemd-machined
(machinectl)
systemd-nspawn
control
pod
systemd
application(s)


systemd (on host)
(systemctl)
rkt
systemd-machined
(machinectl)
systemd-nspawn
pod
systemd
application(s)
kubelet?
control


systemd (on host)
(systemctl)
rkt
systemd-machined
(machinectl)
systemd-nspawn
pod
systemd
application(s)
kubelet?
restart application, add new application, ...

Kubelet: Client-Server Runtimes

Change in architecture to decouple Container Runtimes from kubelet binary



New* container image format

- **OCI: a container image format we can *all* agree on**
  - Based on Docker v2.2 image format (*not really "new")
  - Adding optional components like signing and naming
  - Active work on tooling (octool)
  - First, reach a 1.0: then, push this image format into the Kubernetes API

How do I find out more?

- **Reach out on GitHub or IRC**
  - github.com/coreos/rkt
  - #rkt-dev / #rkt on Freenode
- **Join the Kubernetes "Node" special interest group**
  - https://groups.google.com/forum/#!forum/kubernetes-sig-node
- **Join us!**
  - Hiring rkt developers in Berlin

Get Involved!

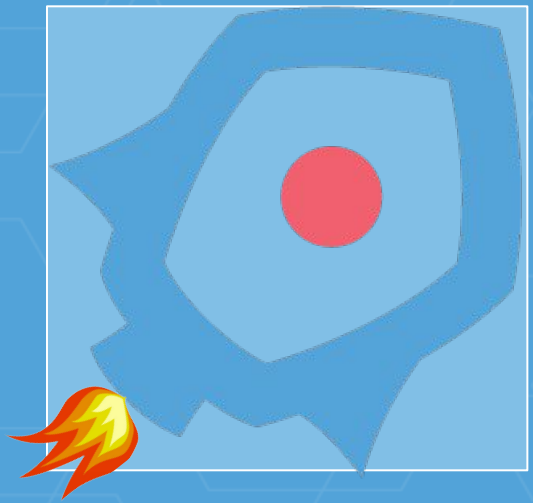
Use it

http://rktnetes.io

Talk about it

#sig-rktnetes (in k8s slack),
kubernetes-sig-rktnetes@googlegroups.com

Blast off with it!



CoreOS

CoreOS is Running the World's Containers

OPEN SOURCE

90+ Projects on GitHub, 1,000+ Contributors

CoreOS.com - @coreoslinux - github/coreos

CoreOS rkt etcd

ENTERPRISE

Secure solutions, support plans, training + more

sales@coreos.com - tectonic.com - quay.io

TECTONIC QUAY

We're hiring in all departments! Email: careers@coreos.com Positions: coreos.com/ careers