rkt and Kubernetes

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

Jonathan Boulle
github.com/jonboulle
@baronboulle

CoreOS

Why rkt and Kubernetes?

~~Why rkt and Kubernetes?~~

Why container runtimes and orchestration?

CoreOS, Inc (2013 - today)

mission: *"Secure the Internet"*

Started at the OS level: **CoreOS Linux**
- modern, minimal operating system
- self-updating (read-only) image
- updates must be *automatic* and *seamless*

Automatic and *seamless*

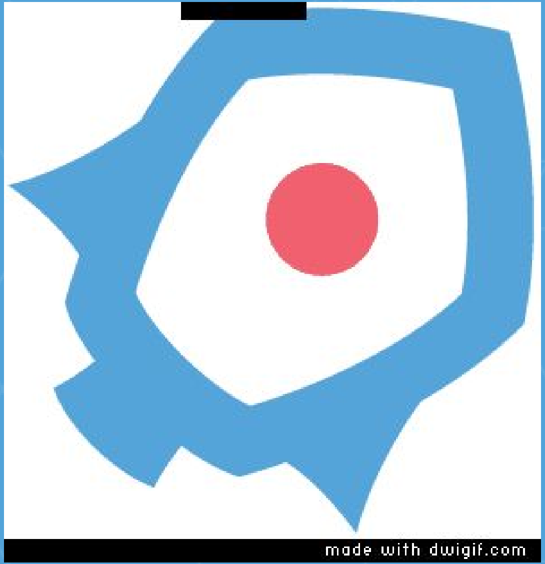
- If the OS is always updating, what about applications running on it?
- Classic use case for *containers* and *orchestration*
  - *containers* decouple the application and OS update lifecycles (update at different cadences)
  - *orchestration* decouples application and OS uptime (services can remain unaffected during OS downtime)

Why container runtimes and orchestration?

So we can provide *seamless updates* and push forward the security of application servers

Why rkt?

The simple answer:
Containers are cool



A long time ago in an ecosystem far, far away....

(2014, to be precise)

2014

- Large incumbent container tool (in CoreOS)
- *Common* practices, but few *best* practices
  - unsigned images (`curl | sudo sh -`)
  - `FROM ubuntu:14.04`
  - PID1 or not to PID1 (zombie reaping problem)
- New platforms emerging, difficult to integrate
  - `systemd` + `dockerd` = sad times had by all

Enter `rkt` (and appc)

- Create an alternative container runtime (competition drives innovation)
- Emphasise the importance of *security*
- Spur the conversation around *standards* in the application container ecosystem

rkt

a modern, secure container runtime
a simple, composable tool
an implementation of an open standard

appc

a standard application container

open specification

associated tooling

appc

github.com/appc/spec
github.com/appc/acbuild
github.com/appc/docker2aci
github.com/appc/cni
github.com/appc/...

appc

github.com/appc/spec ("appc spec")
github.com/appc/acbuild
github.com/appc/docker2aci
github.com/appc/cni (more on this later..)
github.com/appc/...

appc spec in a nutshell

- Image Format (ACI)
    - what does an application consist of?
- Image Discovery
    - how can an image be located?
- Pods
    - how can applications be grouped and run?
- Executor (runtime)
    - what does the execution environment look like?

appc spec in a nutshell

- Image Format (ACI)
  - what does an application consist of?
- Image Discovery
  - how can an image be located?
- **Pods**
  - **how can applications be grouped and run?**
- Executor (runtime)
  - what does the execution environment look like?

appc pods

- grouping of applications executing in a shared context (network, namespaces, volumes)

- shared fate

- the *only* execution primitive: single applications are modelled as singleton pods

appc pods ≈ Kubernetes pods

- grouping of applications executing in a shared context (network, namespaces, volumes)
- shared fate
- the *only* execution primitive: single applications are modelled as singleton pods

rkt

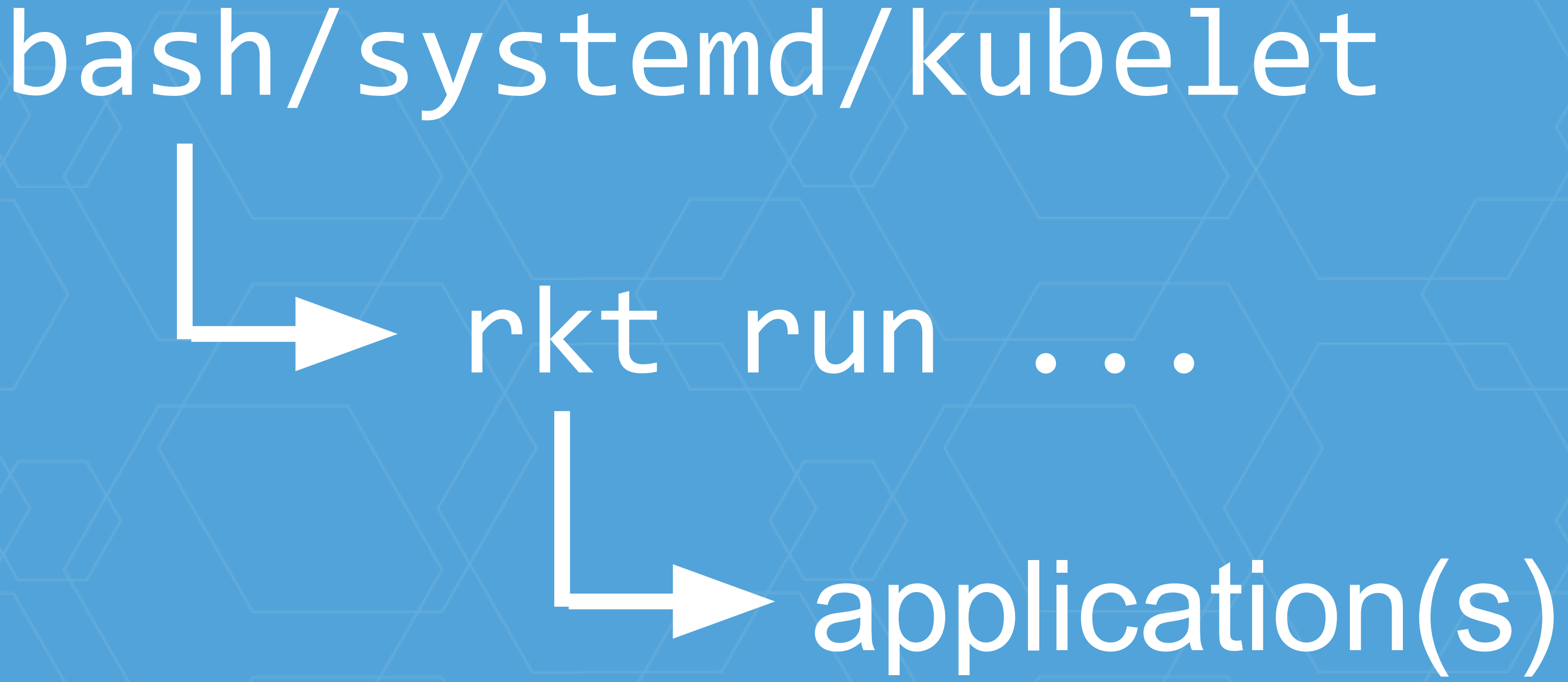
a modern, secure container runtime
a simple, composable tool (CLI)
an implementation of an open standard (appc)

rkt - simple CLI tool

no central daemon

no (mandatory) API

apps run directly under spawning process

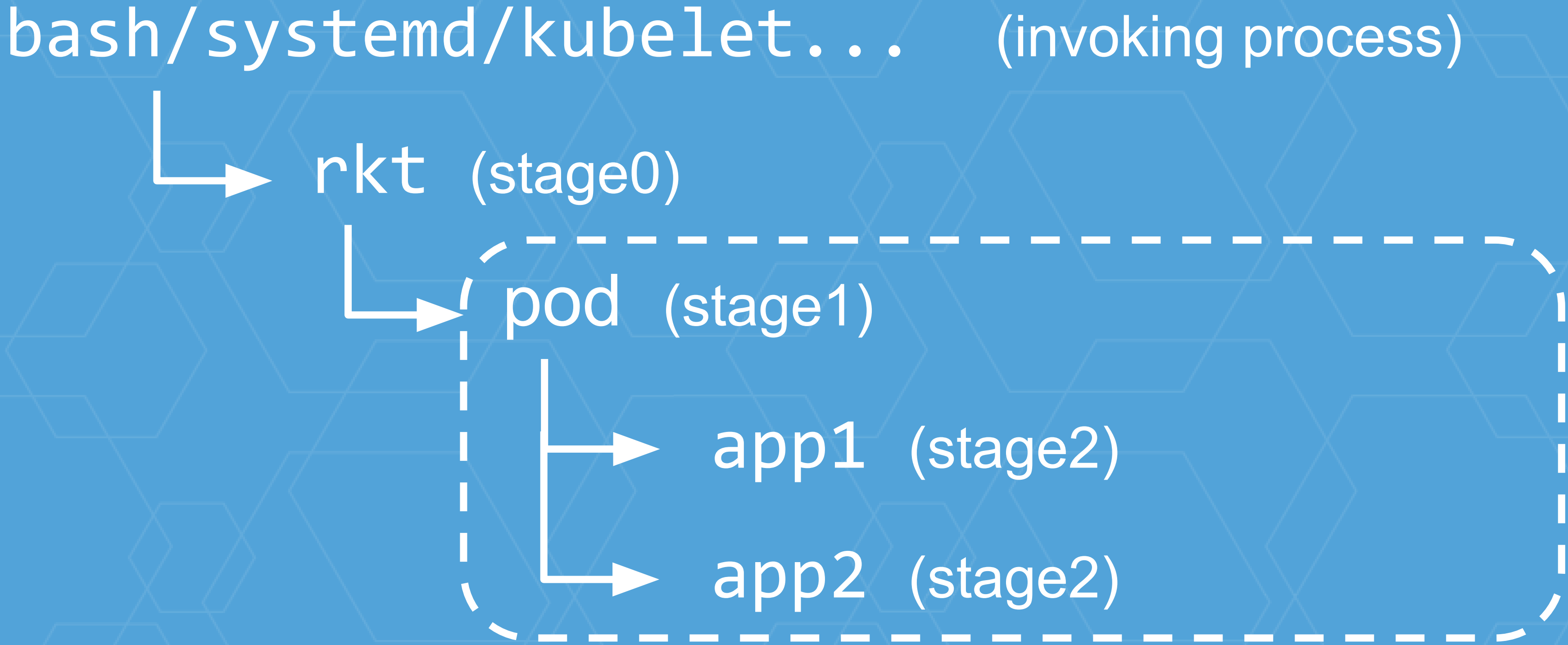

bash/systemd/kubelet
rkt run ...
application(s)

rkt internals

modular architecture

execution divided into *stages*

stage0 → stage1 → stage2


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

stage0 (rkt binary)

- primary interface to rkt
- discover, fetch, manage application images
- set up pod filesystems
- manage pod lifecycle
  - rkt run
  - rkt image list
  - rkt gc
  - ...

stage1 (swappable execution engines)

- default implementation
  - based on systemd-nspawn+systemd
  - Linux namespaces + cgroups for isolation
- kvm implementation
  - based on lkvm+systemd
  - hardware virtualisation for isolation
- others?

stage2 (inside the pod)

- actual app execution
- independent filesystems (`chroot`)
- shared namespaces, volumes, IPC, ...

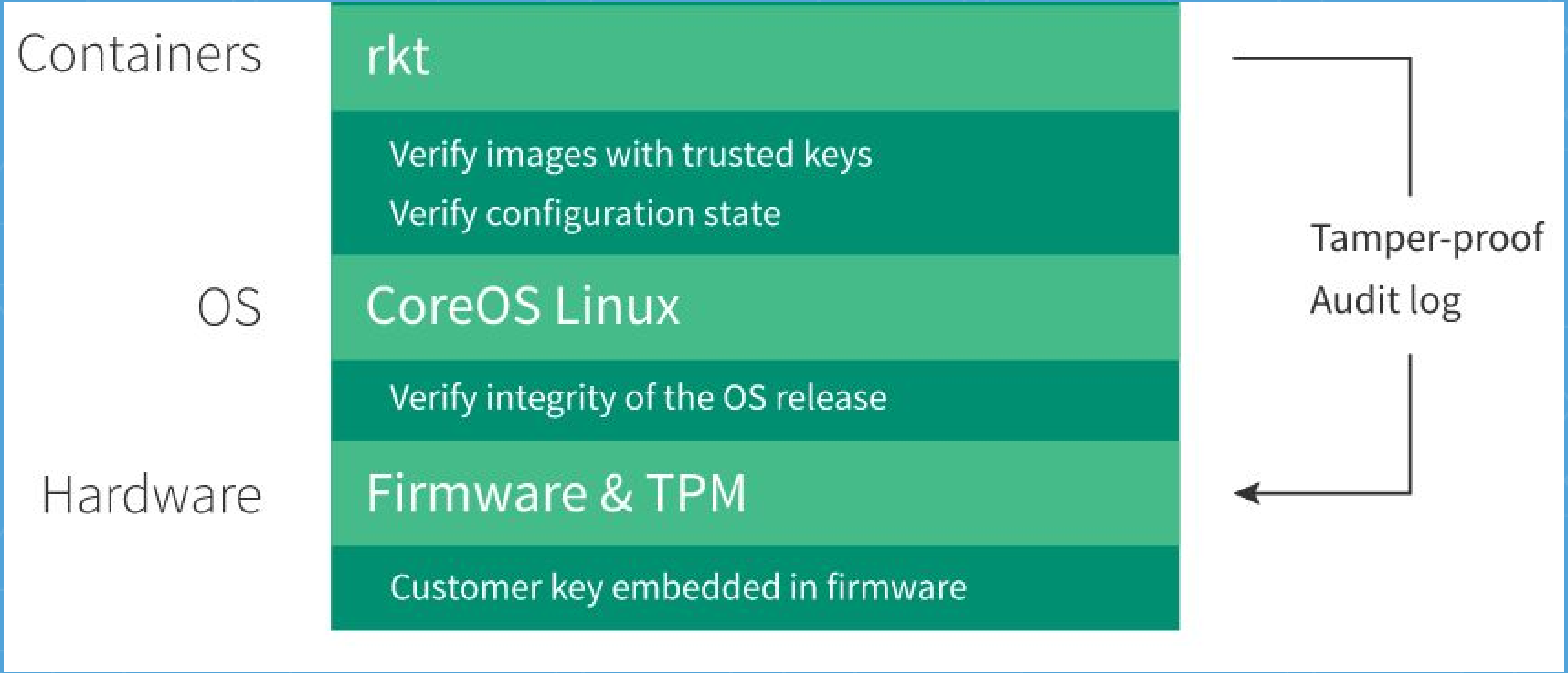
rkt security ("secure by default")

- image signature verification
- privilege separation
    - e.g. fetch images, expose API (new!) as non-root
- SELinux integration (although </3 overlayfs..)
- lkvm stage1 for true hardware isolation
- TPM attestation (new!)

rkt TPM measurement (new!)

- TPM, Trusted Platform Module: hardware module with cryptographic keys
- Used to "measure" system state
- Historically just use to verify bootloader/OS
- CoreOS added support to GNU Grub
- rkt can now record information about running pods in the TPM

rkt TPM measurement (new!)


Containers
rkt
Verify images with trusted keys
Verify configuration state
OS
CoreOS Linux
Verify integrity of the OS release
Hardware
Firmware & TPM
Customer key embedded in firmware
Tamper-proof
Audit log

rkt API service (new!)

- optional, gRPC-based API daemon
- exposes information on pods and images
- runs as unprivileged user
- read-only
- easier integration with other projects (ahem..)

Why rkt?

Secure

Standards

Composable

Why Kubernetes?

Why Kubernetes?

See the earlier talk...

rkt

+

kubernetes

rkt + Kubernetes

- rkt as container runtime (aka "*rktnetes*")
- rkt running Kubernetes ("*rkt fly*")
- rkt networking (CNI)

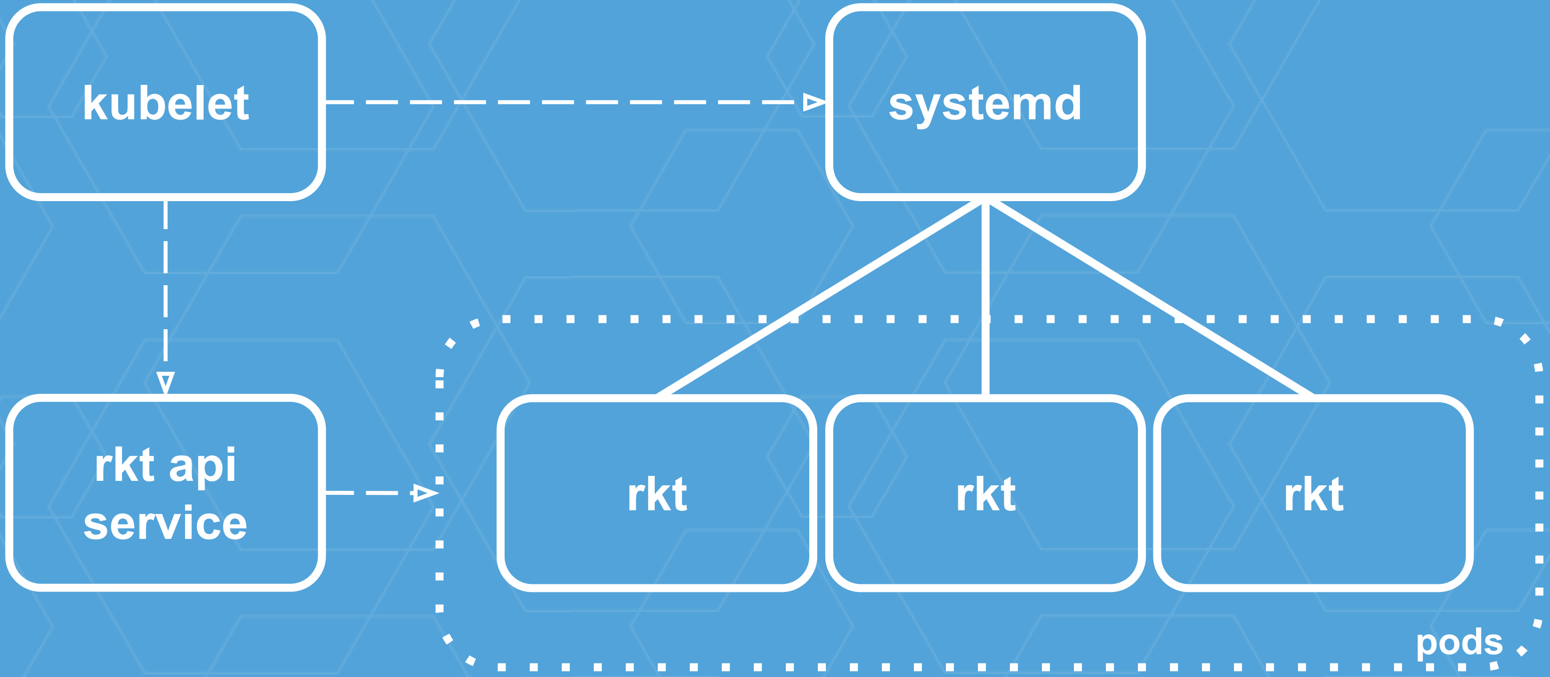
rkt + Kubernetes

- rkt as container runtime (aka "*rktnetes*")
- rkt running Kubernetes ("*rkt fly*")
- rkt networking (CNI)

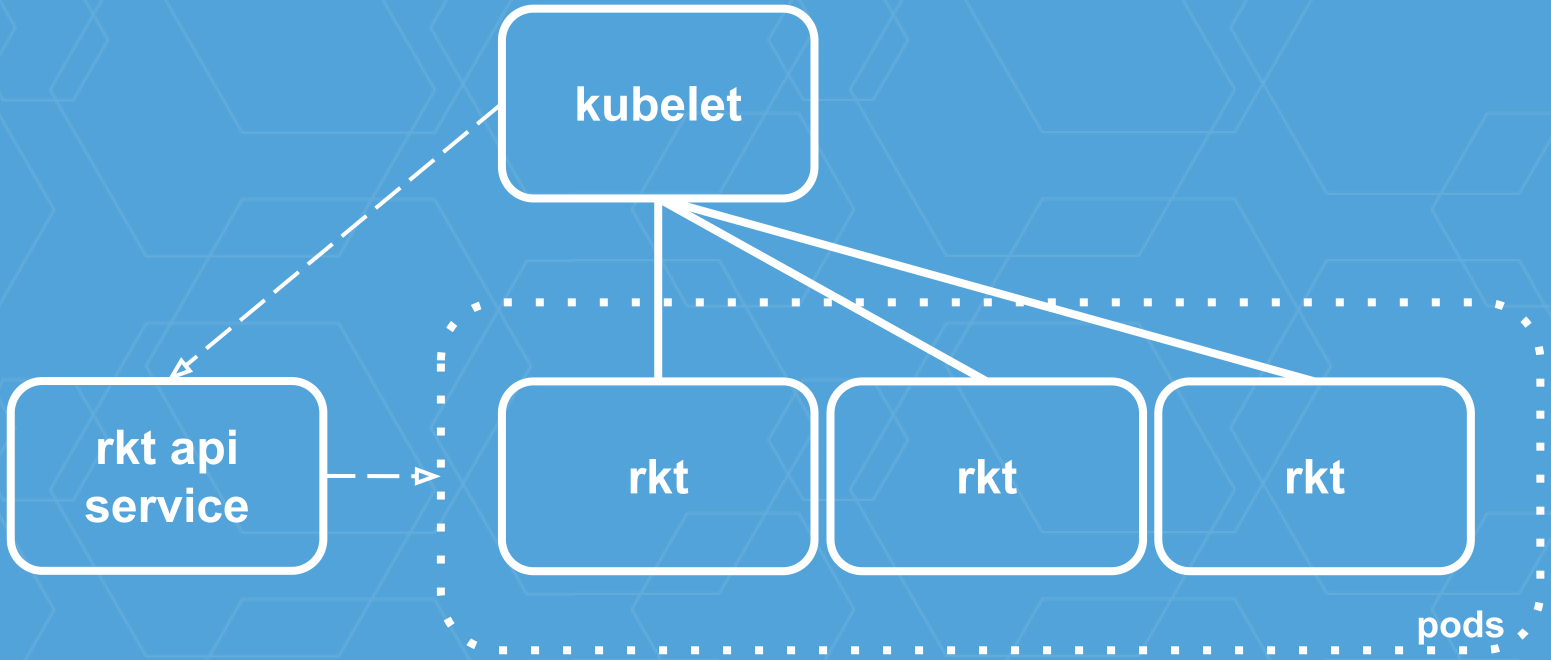
rktnetes

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
- Multiple-image compatibility (e.g. `docker2aci`)
- Transparently swappable container engines

Kubelet + rkt (rktnetes - with systemd)



Kubelet + rkt (rktnetes - without systemd)

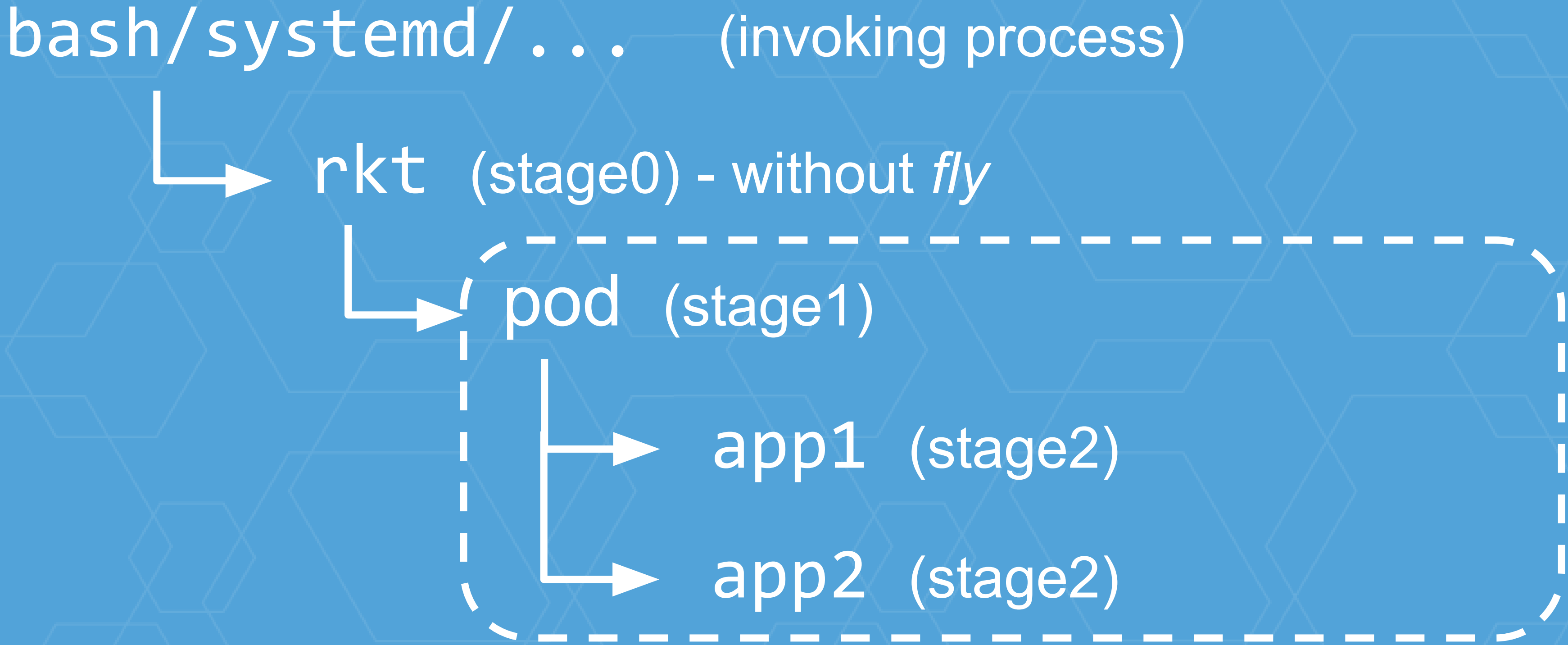


Using rkt to run Kubernetes

- Kubernetes components are largely self-hosting, but not entirely
- Need a way to bootstrap kubelet on the host
- On CoreOS, this means in a container..
- ... but kubelet has some unique requirements
  (like mounting volumes on the host)

Using rkt to run Kubernetes

- `rkt` "fly" feature (new in rkt 0.15.0)
- unlike `rkt run`, does *not* execute pods
- execute a single application in an unconstrained environment
- all the other advantages of rkt (image discovery, signing/verification, management)


bash/systemd/... (invoking process)
rkt (stage0) - without *fly*
pod (stage1)
app1 (stage2)
app2 (stage2)

`bash/systemd/...` (invoking process)

└→ `rkt` (stage0) - with *fly*

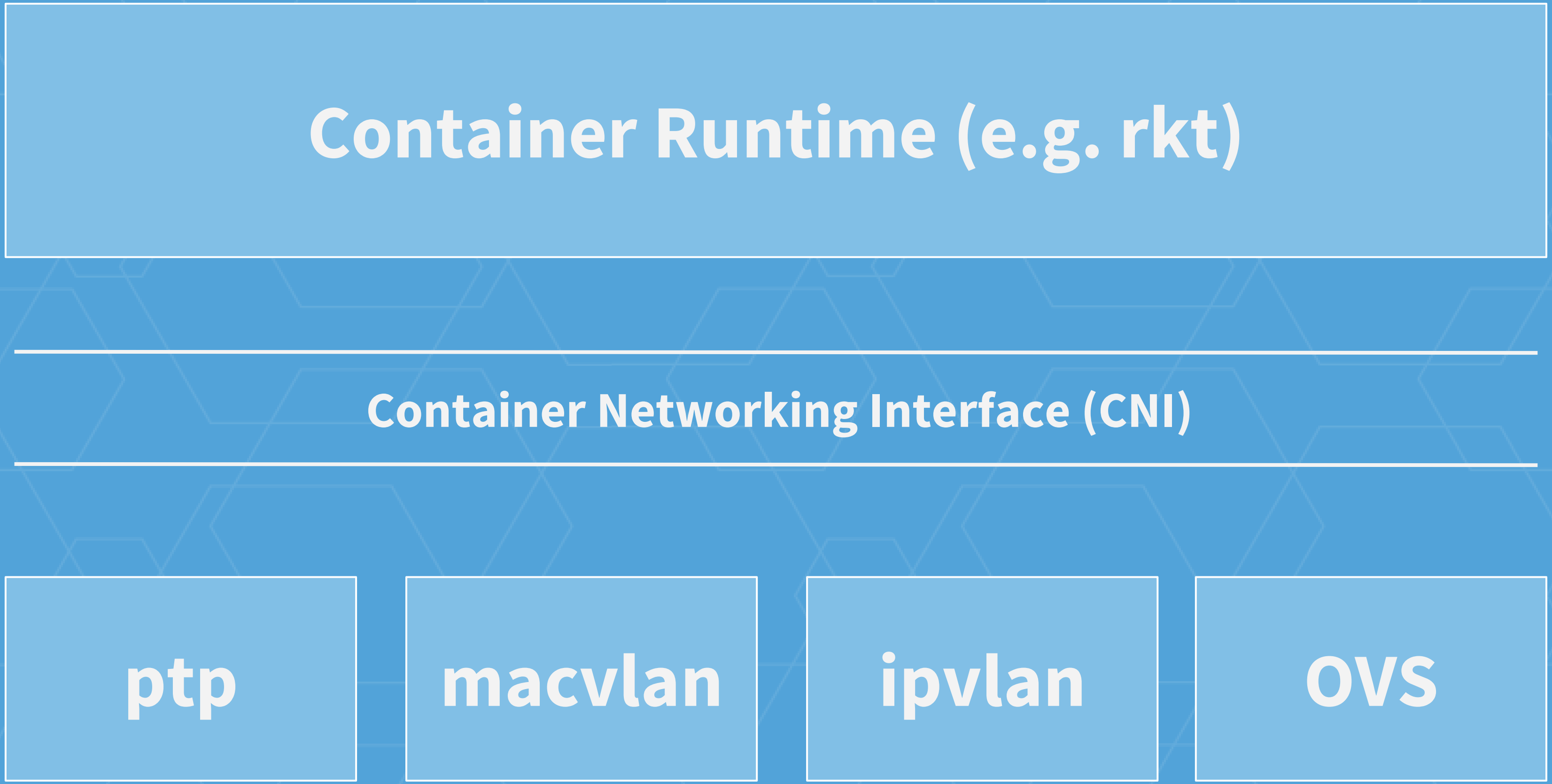
  └→ application (kubelet)

rkt networking

Plugin-based

IP(s)-per-pod

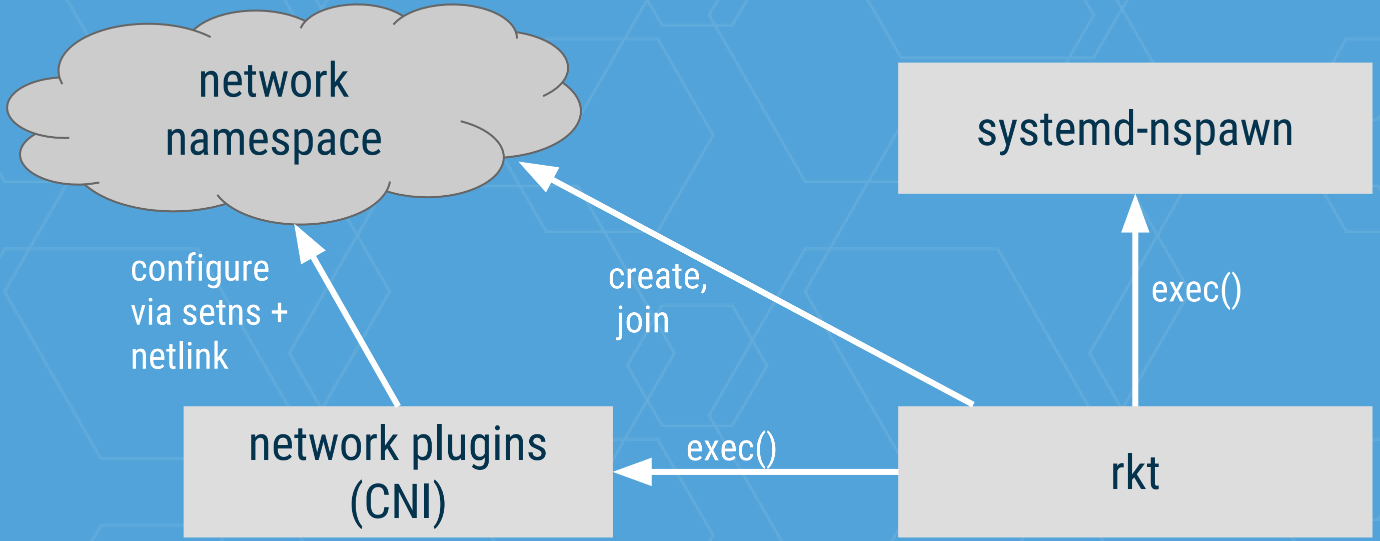
Container Networking Interface (CNI)


Container Runtime (e.g. rkt)
Container Networking Interface (CNI)
ptp
macvlan
ipvlan
OVS

CNI: example configuration

```
{
    "name": "mynet",
    "type": "ptp",
    "ipam": {
        "type": "host-local",
        "subnet": "10.1.1.0/24"
    }
}
$ rkt run --net=mynet coreos.com/etcd
```

How rkt uses CNI



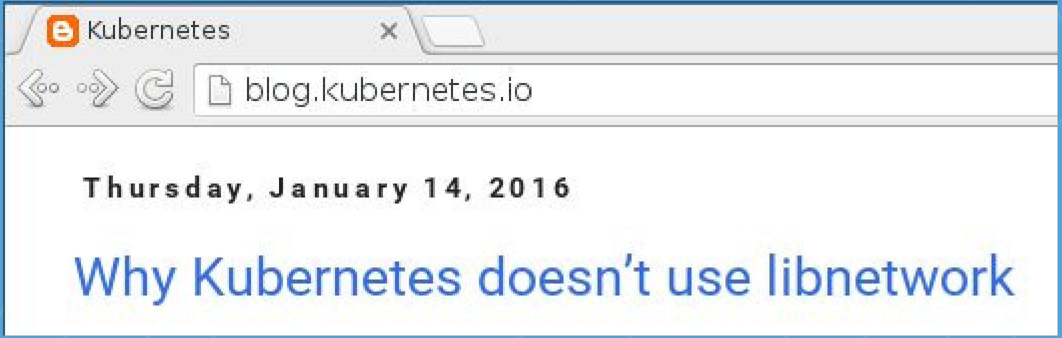
```
/var/lib/rkt/pods/run/$POD_UUID/netns
```

Kubernetes networking

Plugin-based (but never left alpha)
IP(s)-per-pod
(sound familiar?)

Kubernetes and CNI

Previously CNI was just another plugin type, but soon to be "*the* Kubernetes plugin model"

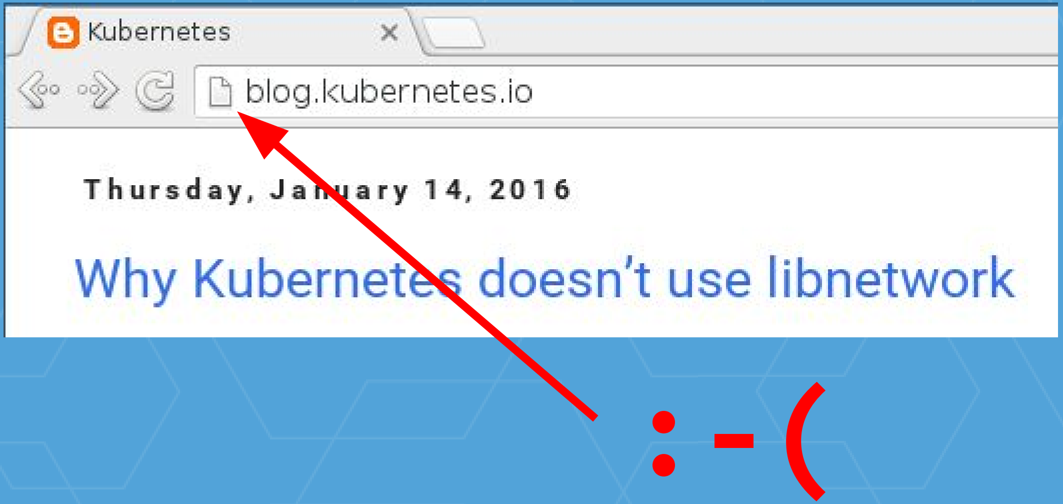


Thursday, January 14, 2016

Why Kubernetes doesn’t use libnetwork

Kubernetes and CNI

Previously CNI was just another plugin type, but soon to be "*the* Kubernetes plugin model"



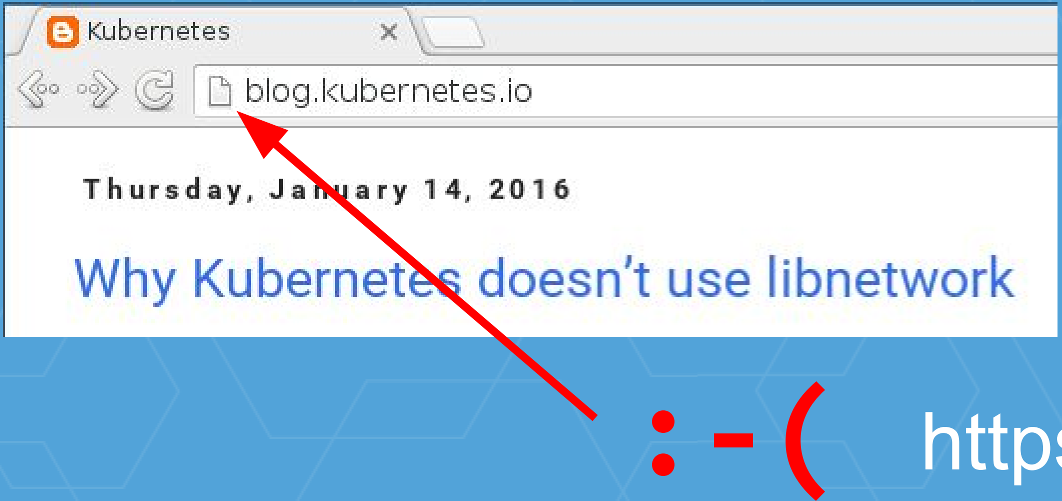
Thursday, January 14, 2016

Why Kubernetes doesn't use libnetwork

:-(

Kubernetes and CNI

Previously CNI was just another plugin type, but soon to be "*the* Kubernetes plugin model"


Kubernetes

blog.kubernetes.io

Thursday, January 14, 2016

Why Kubernetes doesn't use libnetwork

:-(

https://letsencrypt.org/

Looking ahead

What's coming up for rkt and Kubernetes

First things first…

get things stable

rkt v1.0.0

targeting early February
stable API, CLI, on-disk format
ready to use in production!

rktnetes 1.0

2016Q1

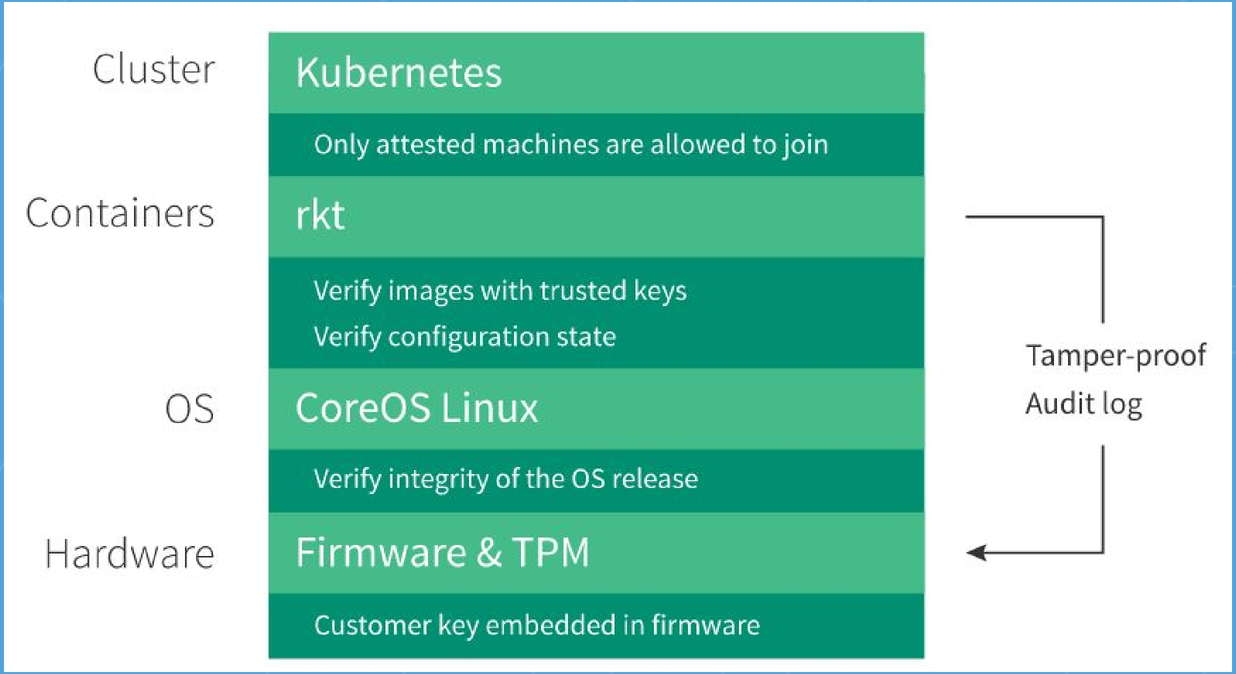
Fully supported, full feature parity

Automated testing on CoreOS

rktnetes 1.0+

LKVM backend by default
Native support for ACIs in Kubernetes
TPM at the Kubernetes level

Tectonic Trusted Computing



Cluster
Kubernetes
Only attested machines are allowed to join
Containers
rkt
Verify images with trusted keys
Verify configuration state
OS
CoreOS Linux
Verify integrity of the OS release
Hardware
Firmware & TPM
Customer key embedded in firmware
Tamper-proof
Audit log

https://coreos.com/blog/coreos-trusted-computing.html

Kubelet upgrades

- Remember from CoreOS mission:
  "updates must be *automatic* and *seamless*"
- If kubelet is in OS, must be upgraded in lock-step
- But mixed-version clusters don't always work

(e.g. upgrading from 1.07 - 1.1.1: https://github.com/kubernetes/kubernetes/issues/16961)

Kubelet upgrades

- Solution: API driven upgrades
- Small agent living on host, invoking kubelet (using rkt fly)
- Reading annotations from the kubelet API server
- Follow along: https://github.com/coreos/bugs/issues/1051

tl;dr:

- Use `rkt` (it's secure, cool, (soon to be) stable)
- Use Kubernetes (for all those earlier reasons)

- Get involved and help define the future of application containers

FEST

CoreOS

May 2016 in Berlin

https://coreos.com/fest (updated soon!)

- Earlybird tickets
- Sponsorships
- Talk submissions

Questions?

Join us!

rkt github.com/coreos/rkt

CoreOS coreos.com/careers *(now in Berlin!)*