rkt and Kubernetes

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

rkt + kubernetes

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?

- Update the OS without affecting application dependencies

kernel
systemd
rkt
ssh
docker

python
java
nginx
mysql
openssl

stro distro distro distro distro distro

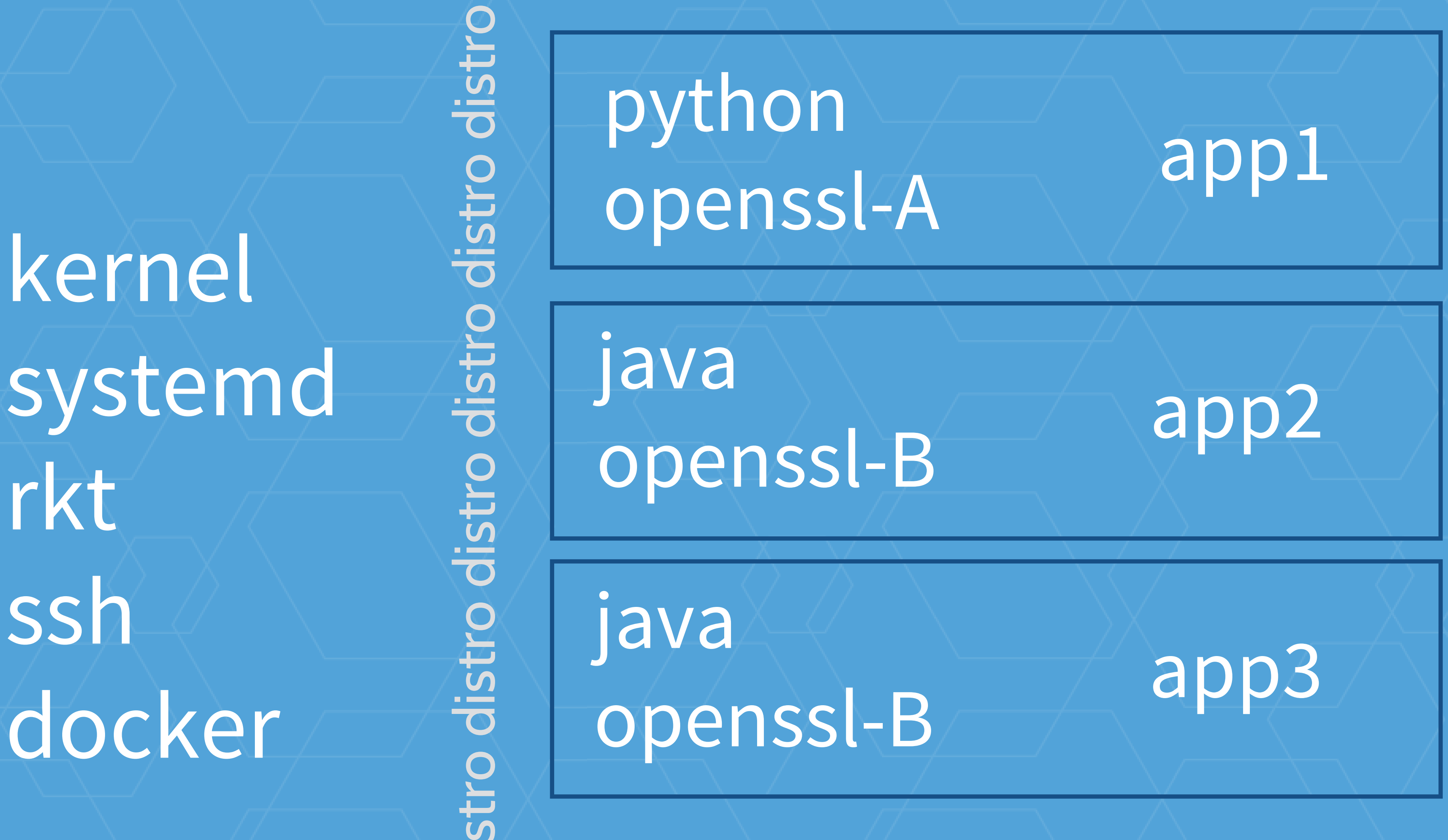
app

kernel
systemd
rkt
ssh
docker

stro distro distro distro distro distro

python
java
nginx
mysql
openssl

app



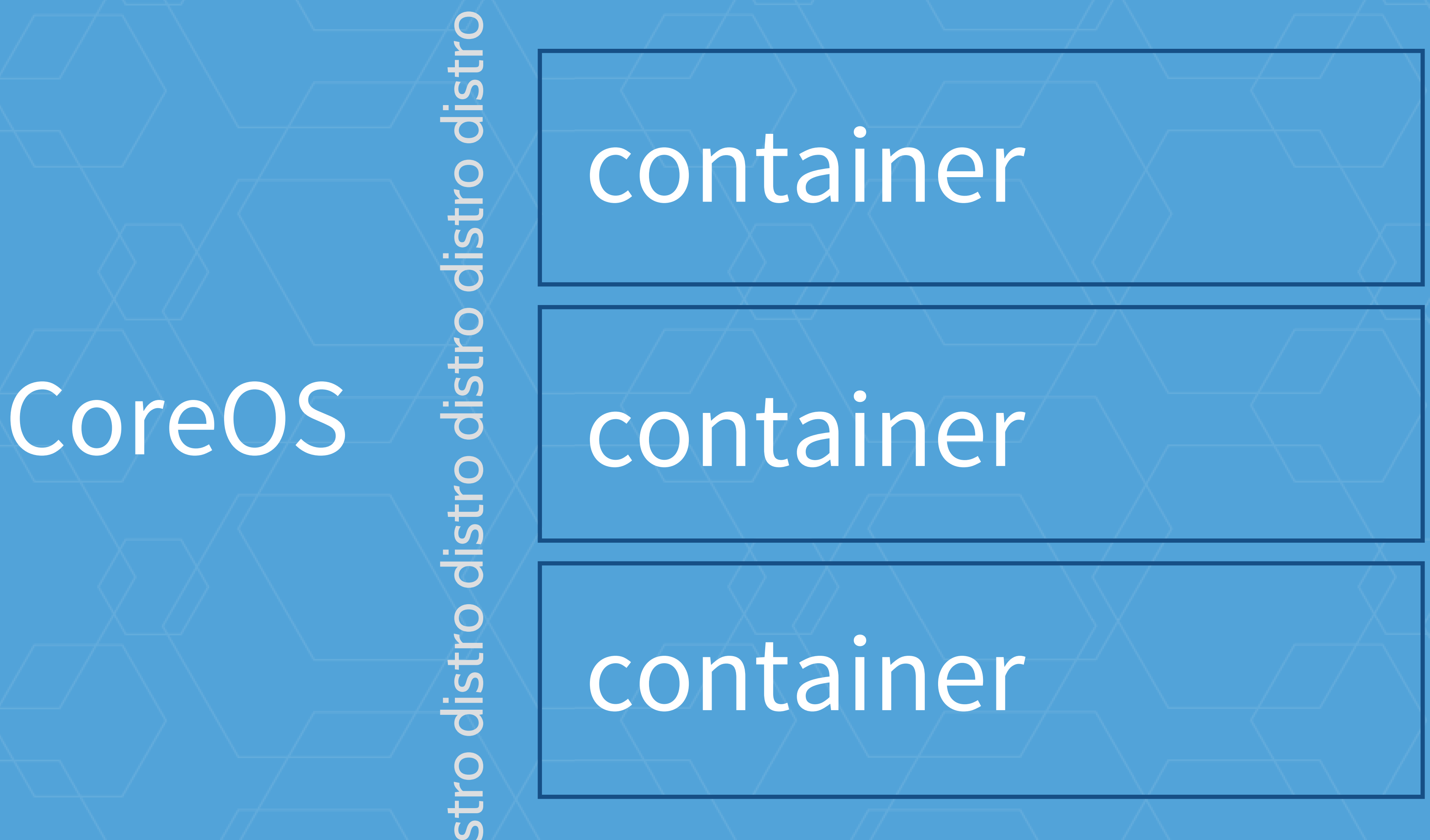
kernel
systemd
rkt
ssh
docker

stro distro distro distro distro

python
openssl-A
app1

java
openssl-B
app2

java
openssl-B
app3



CoreOS

stro distro distro distro distro distro

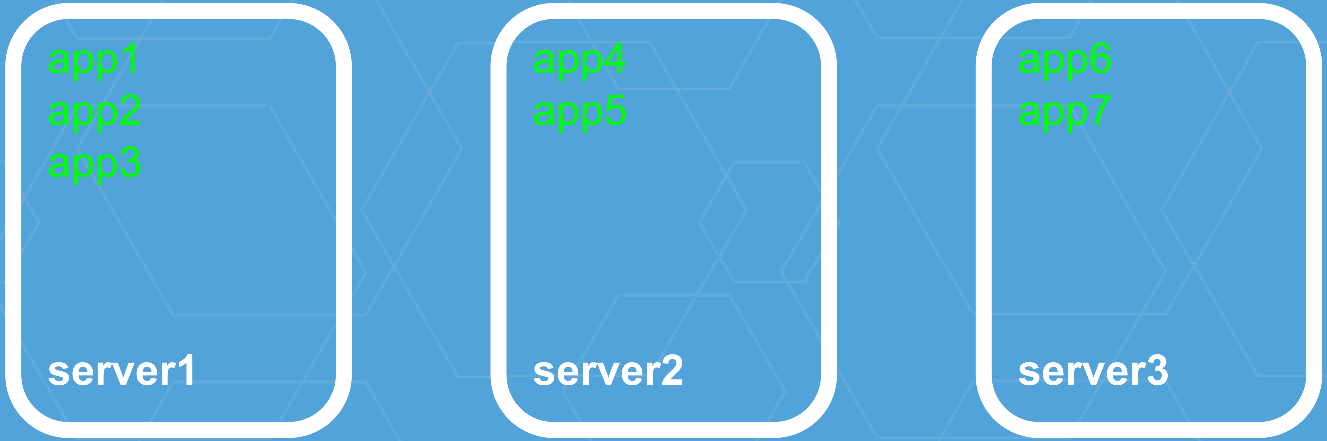
container

container

container

Why orchestration?

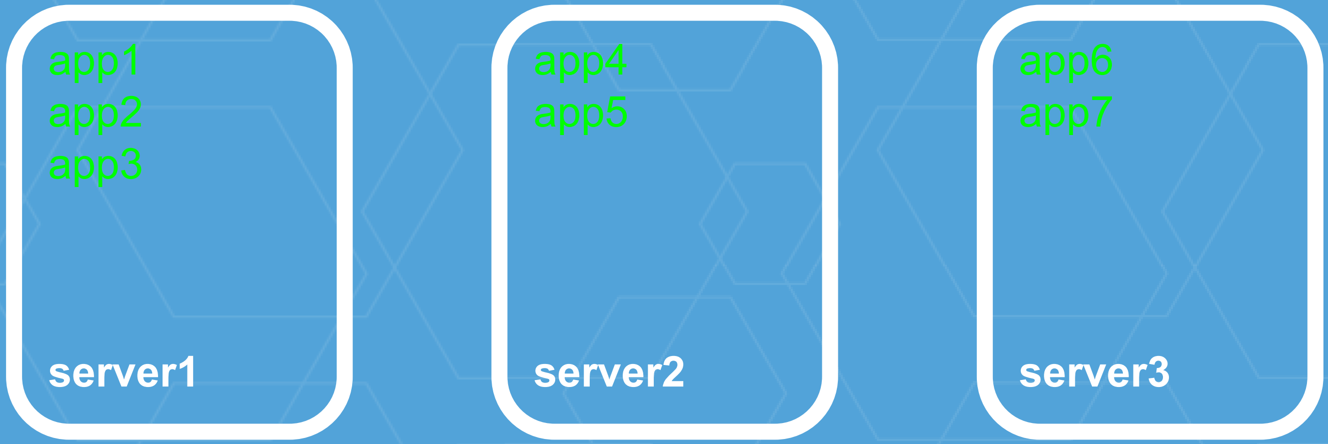
- Update the OS without affecting application uptime



app1
app2
app3
server1

app4
app5
server2

app6
app7
server3



app1
app2
app3

server1

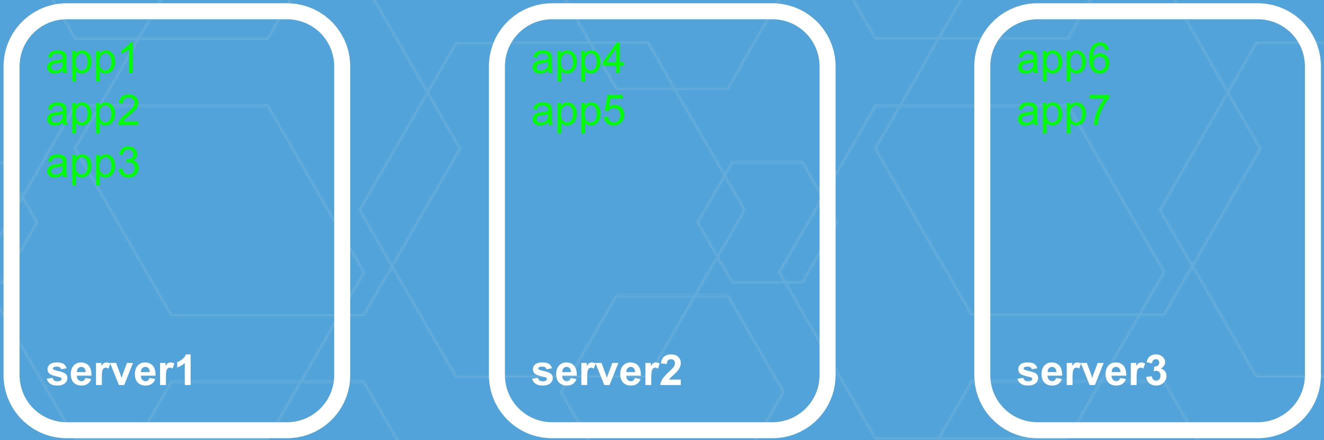
app4
app5

server2

app6
app7

server3

updating...



app1
app2
app3

server1

app4
app5

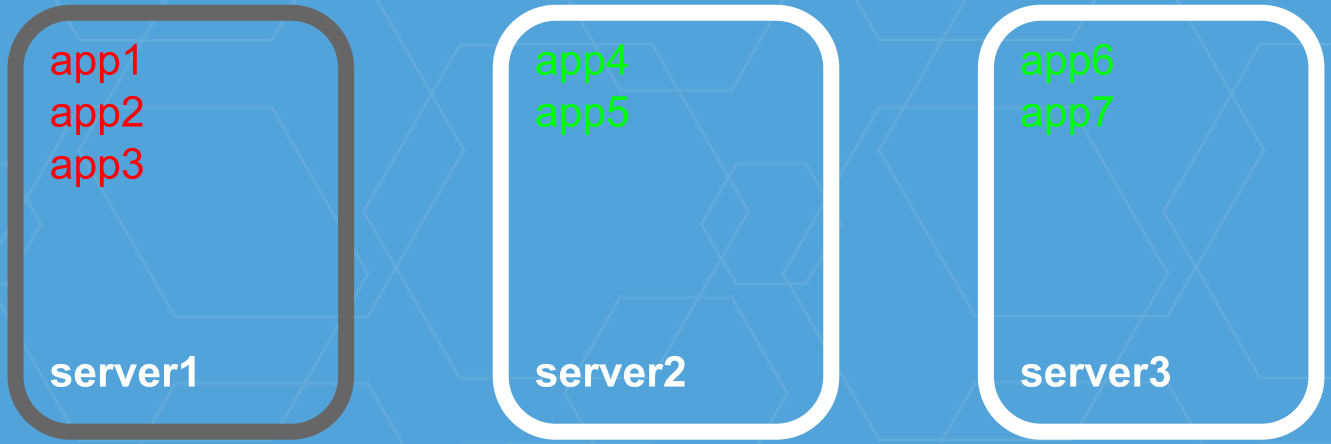
server2

app6
app7

server3

needs reboot

Without orchestration



app1
app2
app3

server1

app4
app5

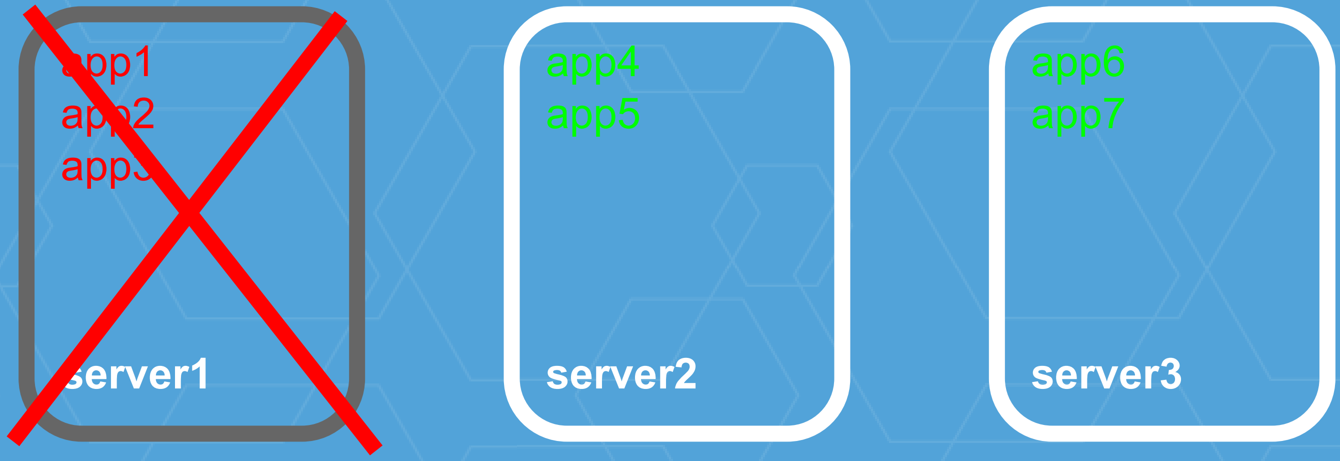
server2

app6
app7

server3

rebooting...

Without orchestration



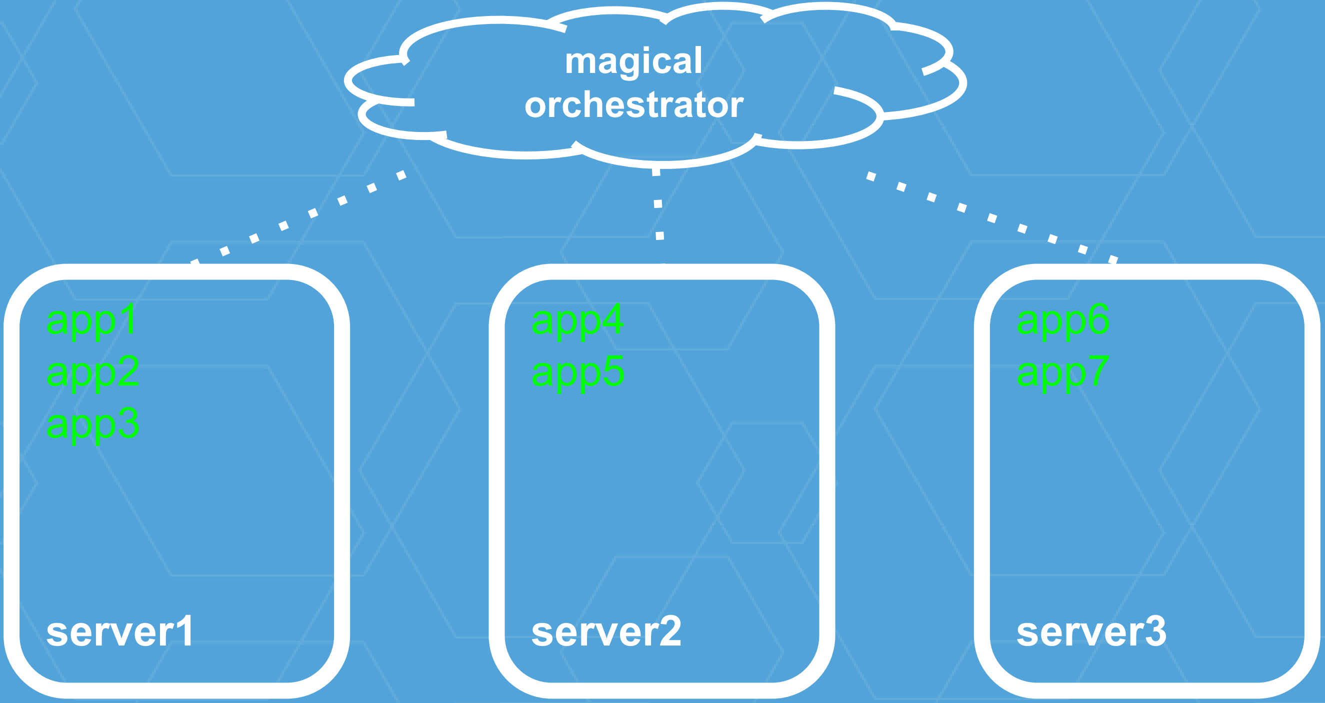
app1
app2
app3
server1

app4
app5
server2

app6
app7
server3

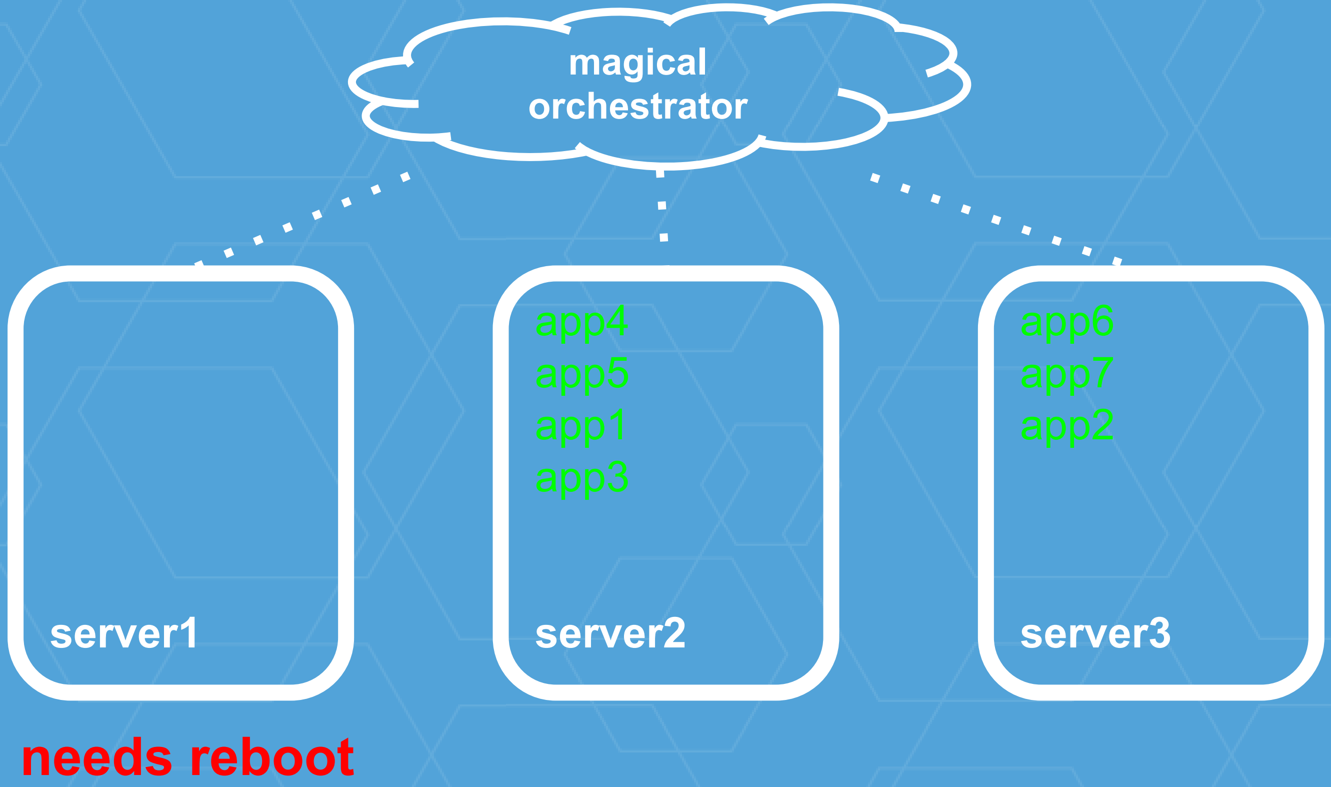
rebooting...

With orchestration



needs reboot

With orchestration



magical orchestrator

app4
app5
app1
app3

app6
app7
app2

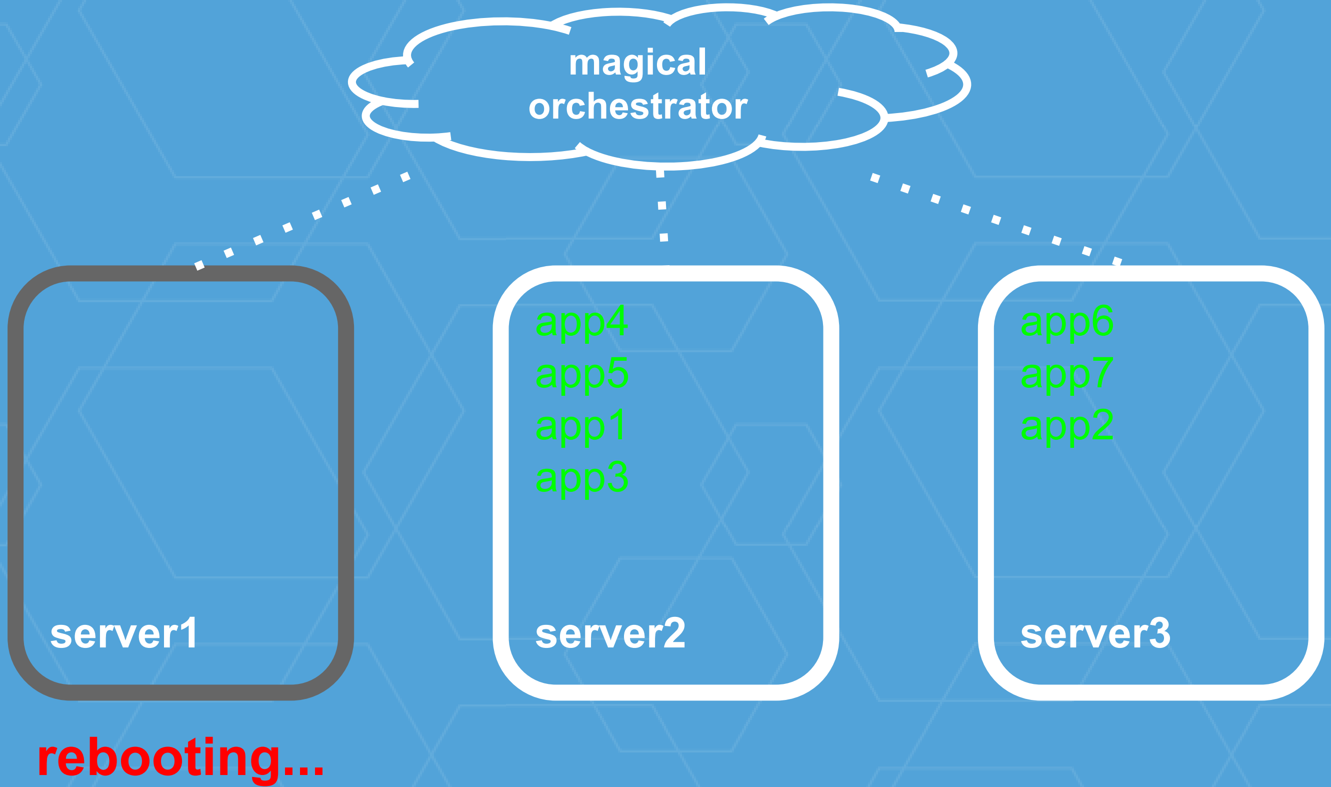
server1

server2

server3

needs reboot

With orchestration



magical orchestrator

app4
app5
app1
app3

app6
app7
app2

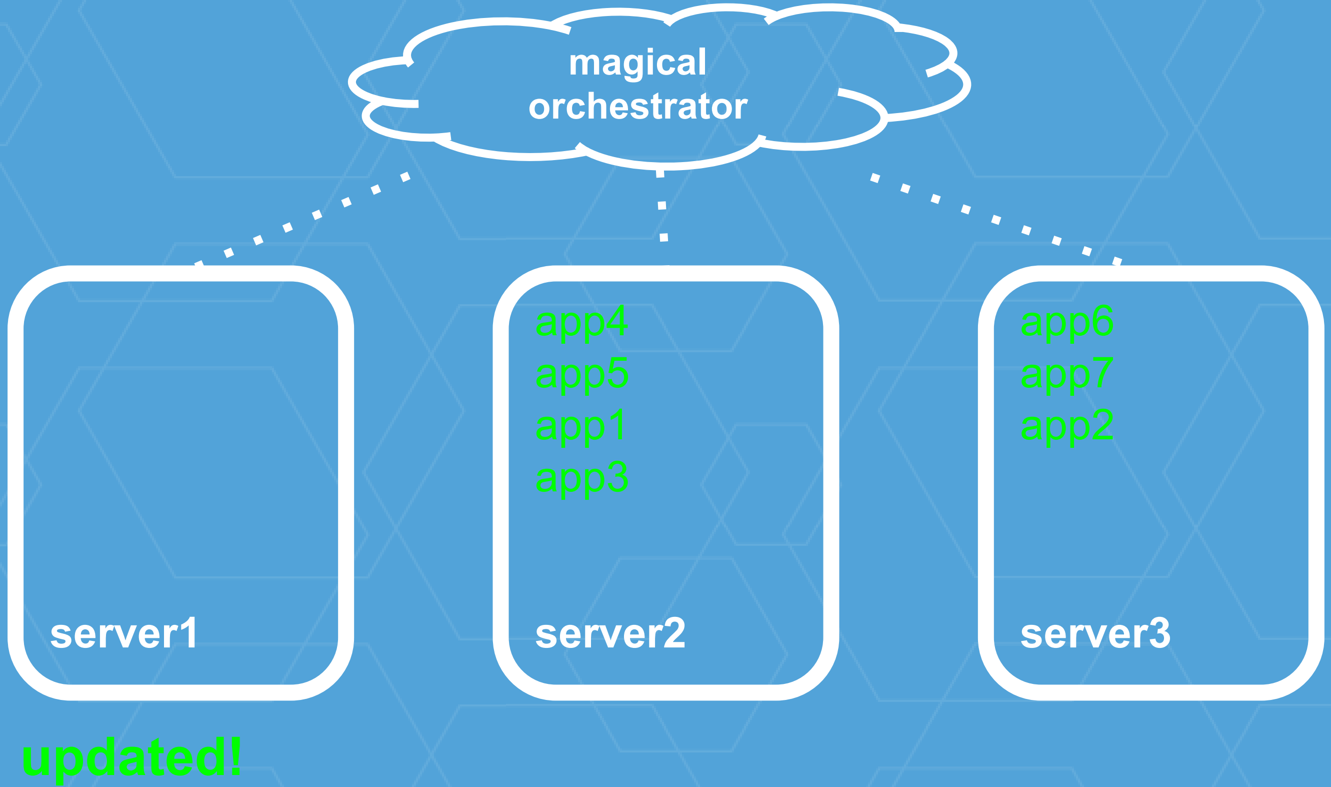
server1

server2

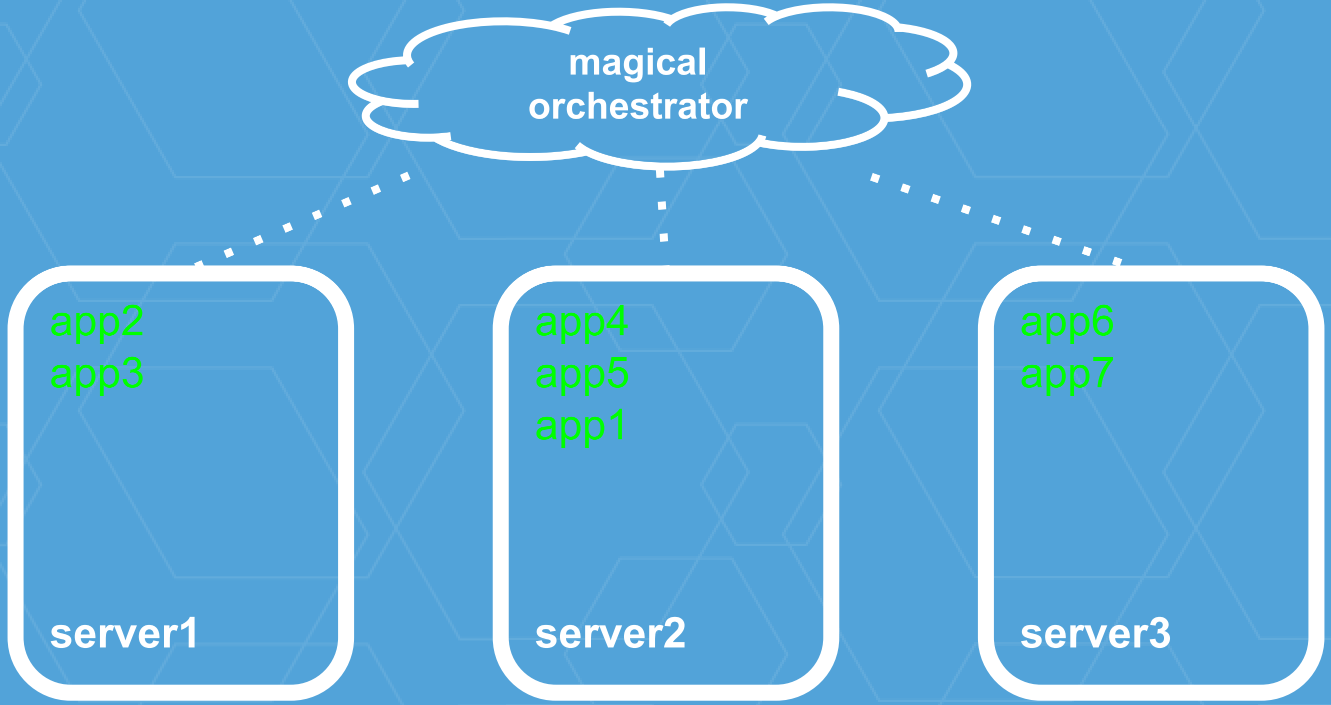
server3

rebooting...

With orchestration



With orchestration



magical orchestrator

app2
app3

server1

app4
app5
app1

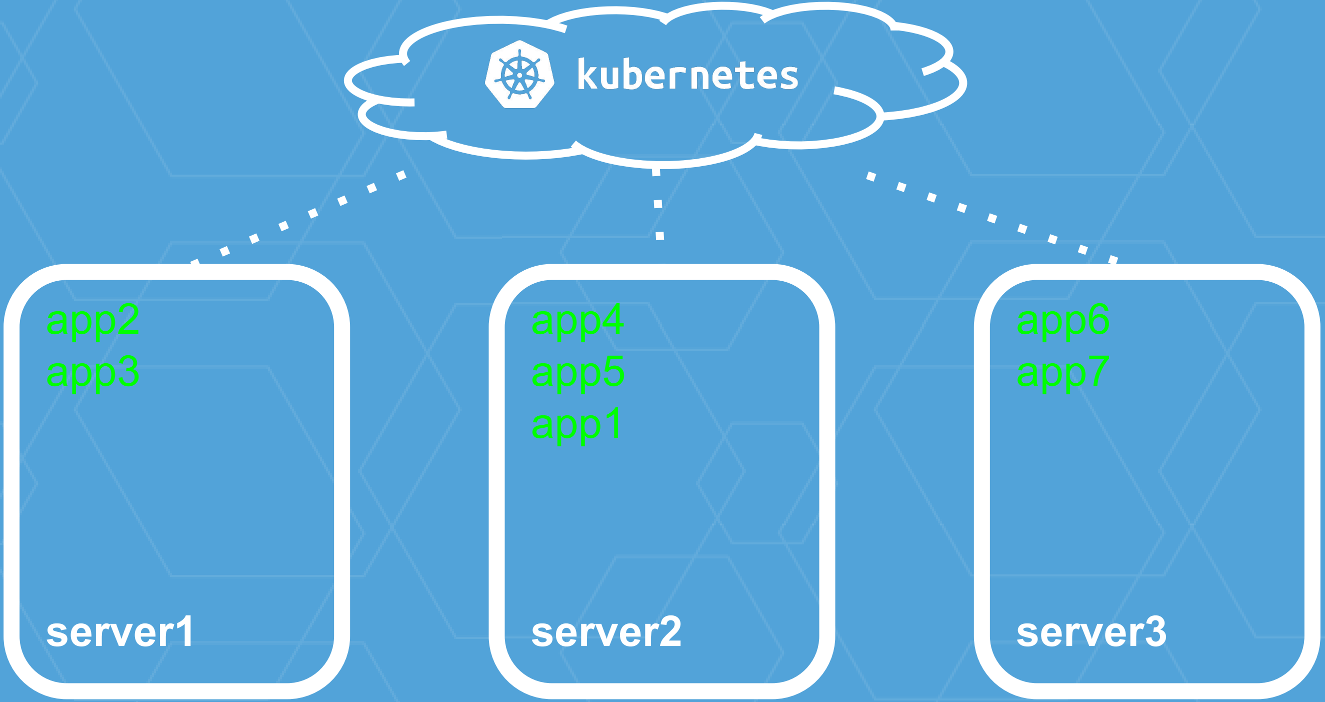
server2

app6
app7

server3

updated!

With orchestration



kubernetes

app2
app3

server1

app4
app5
app1

server2

app6
app7

server3

Why container runtimes and orchestration?

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

Why rkt?

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

(2014, to be precise)

Before the "Container Wars"

(please stop saying this)

2014

- Large incumbent container tool (in CoreOS)
- *Common* practices, but few *best* practices
    - unsigned images (`curl | sudo sh -`)
    - inefficient/insecure images (`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

2014 (December)

- Enter `rkt` (and appc)
  - Create an *alternative* container runtime (competition drives innovation)
  - Emphasise the importance of *security* and *composability*
  - Spur 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)

2016

- Docker and rkt both "production-ready"
- Kubernetes too!
- Standards?
    - appc (December 2014)
    - OCI (June 2015)
    - CNCF (December 2015)

appc

OPEN CONTAINER INITIATIVE

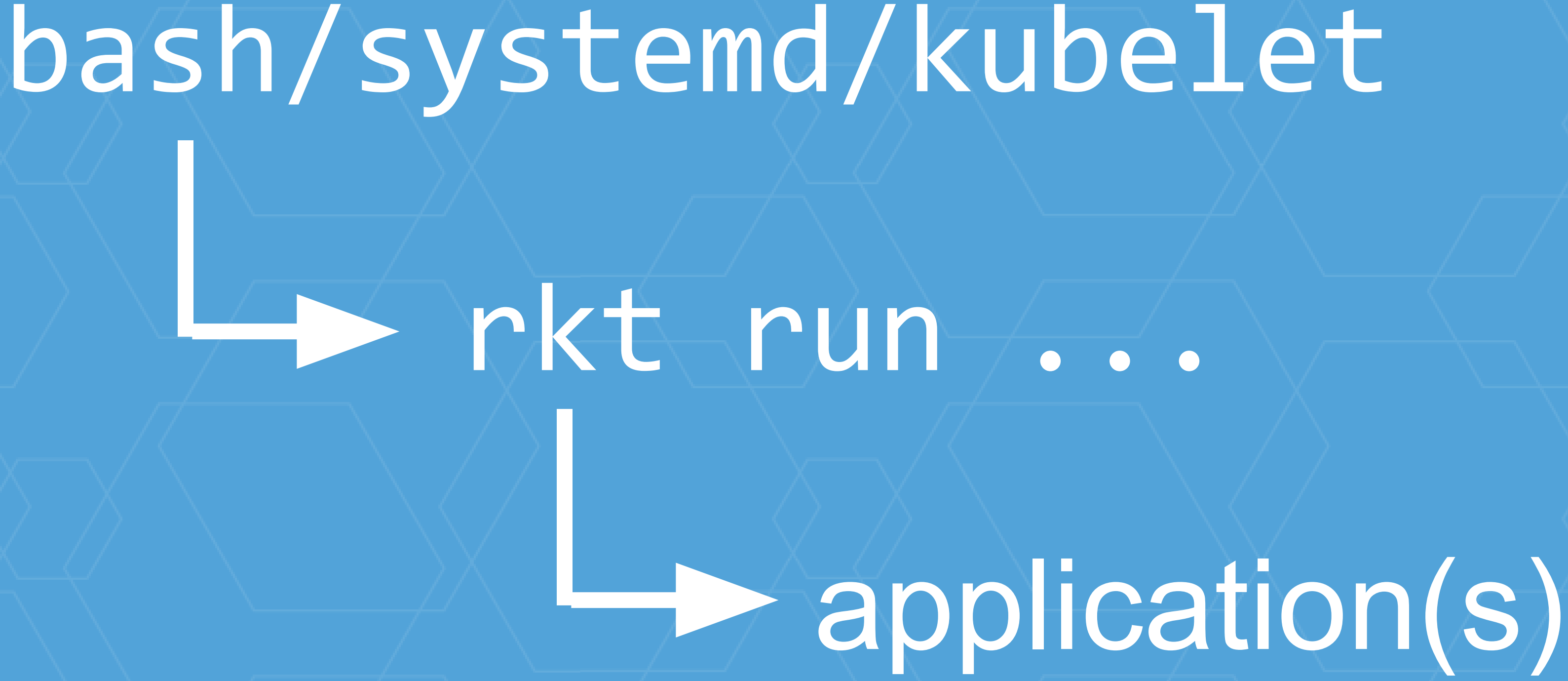
CLOUD NATIVE COMPUTING FOUNDATION

rkt architecture

A quick introduction

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

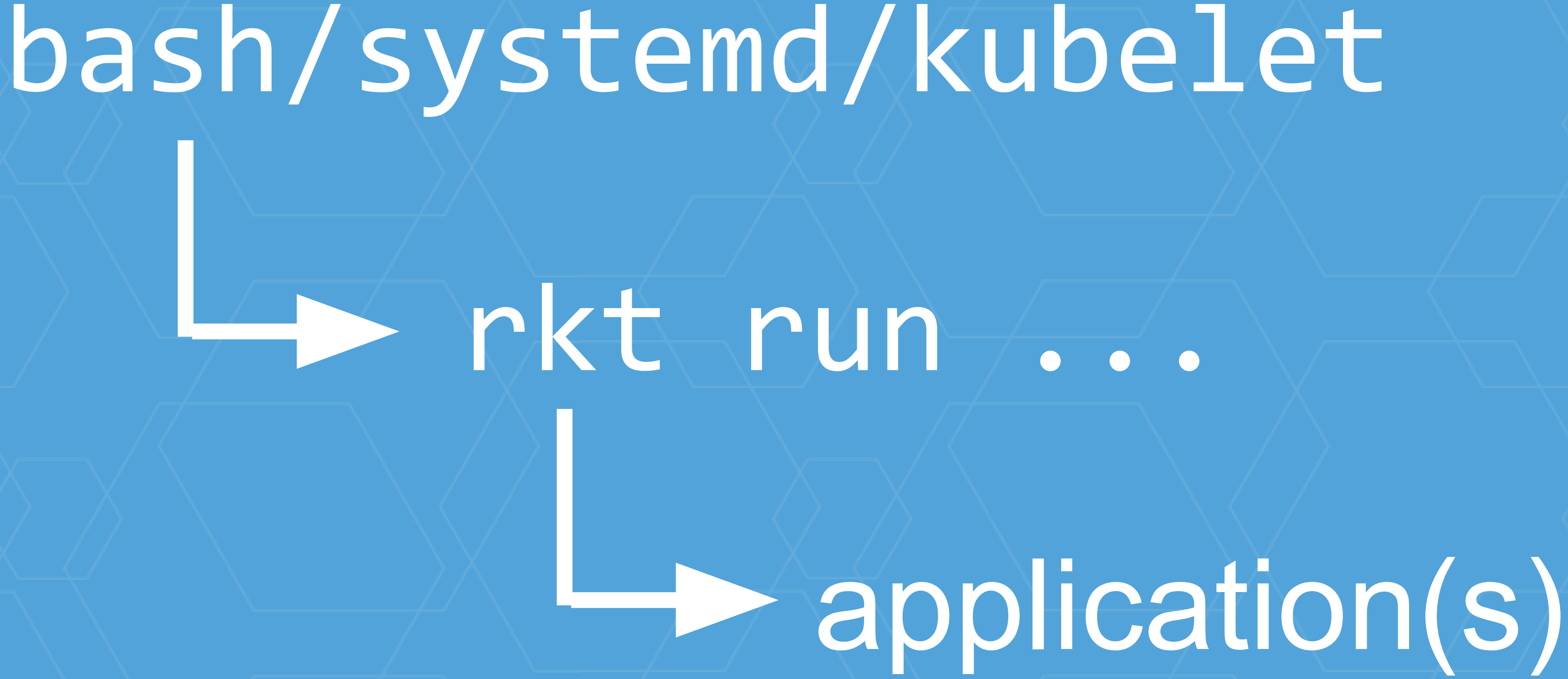
modular architecture

execution divided into *stages*

stage0 → stage1 → stage2

rkt internals

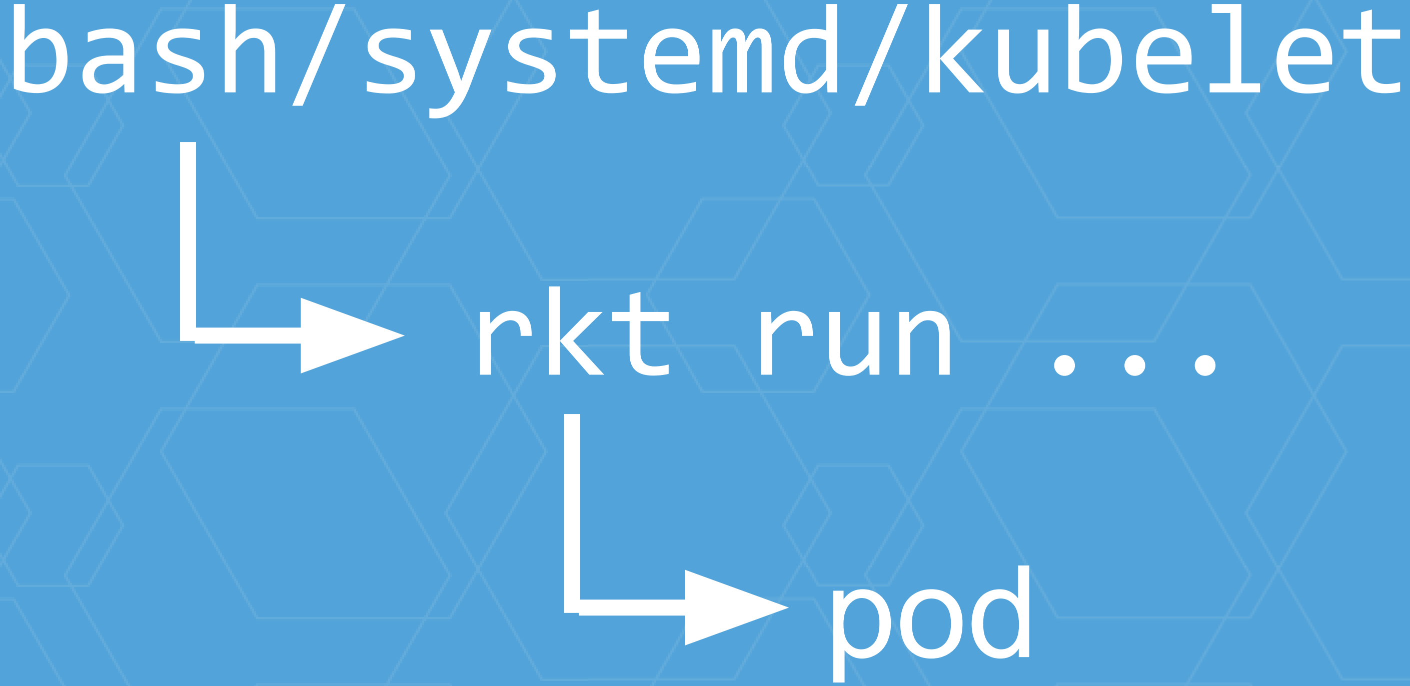
modular architecture
take advantage of different technologies
provide a consistent experience to users



bash/systemd/kubelet

rkt run ...

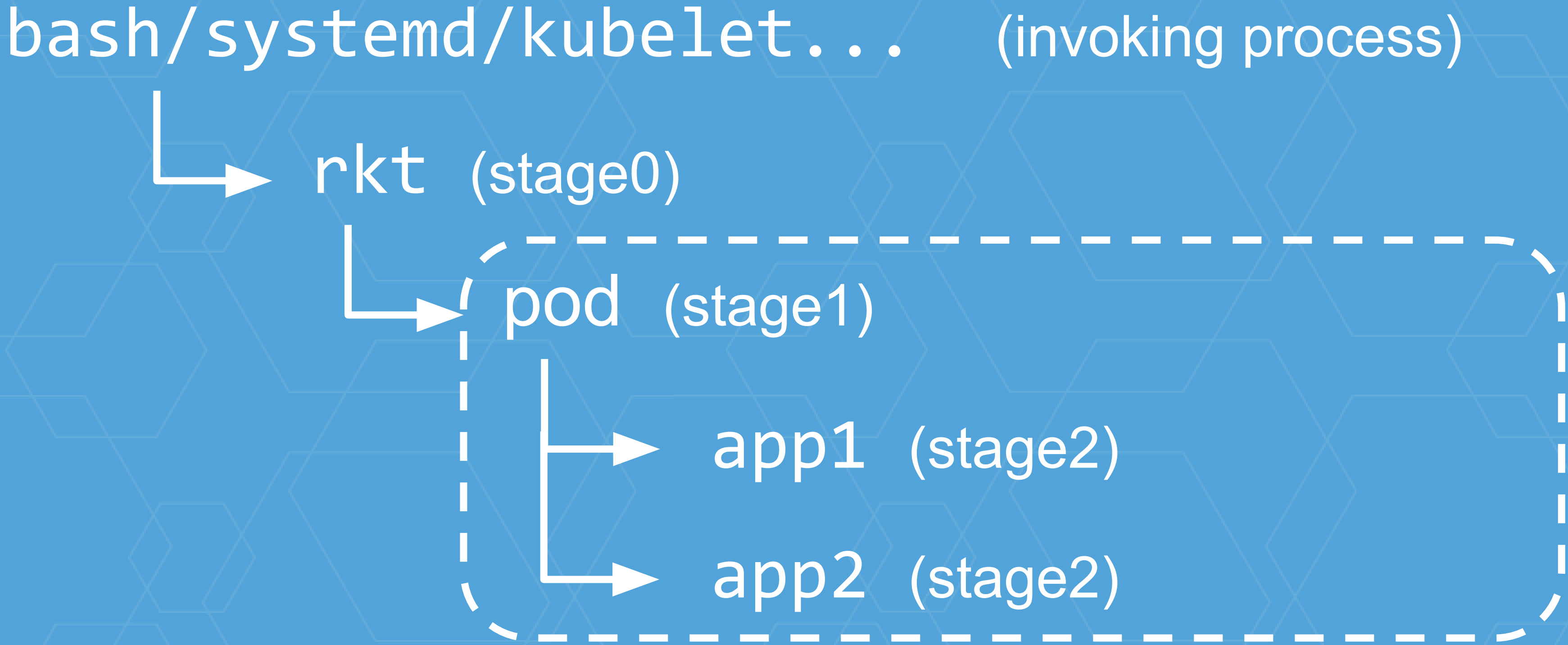
application(s)



bash/systemd/kubelet

rkt run ...

pod



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?
  - e.g. xhyve (OS X), unc (unprivileged containers)

stage2 (inside the pod)

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

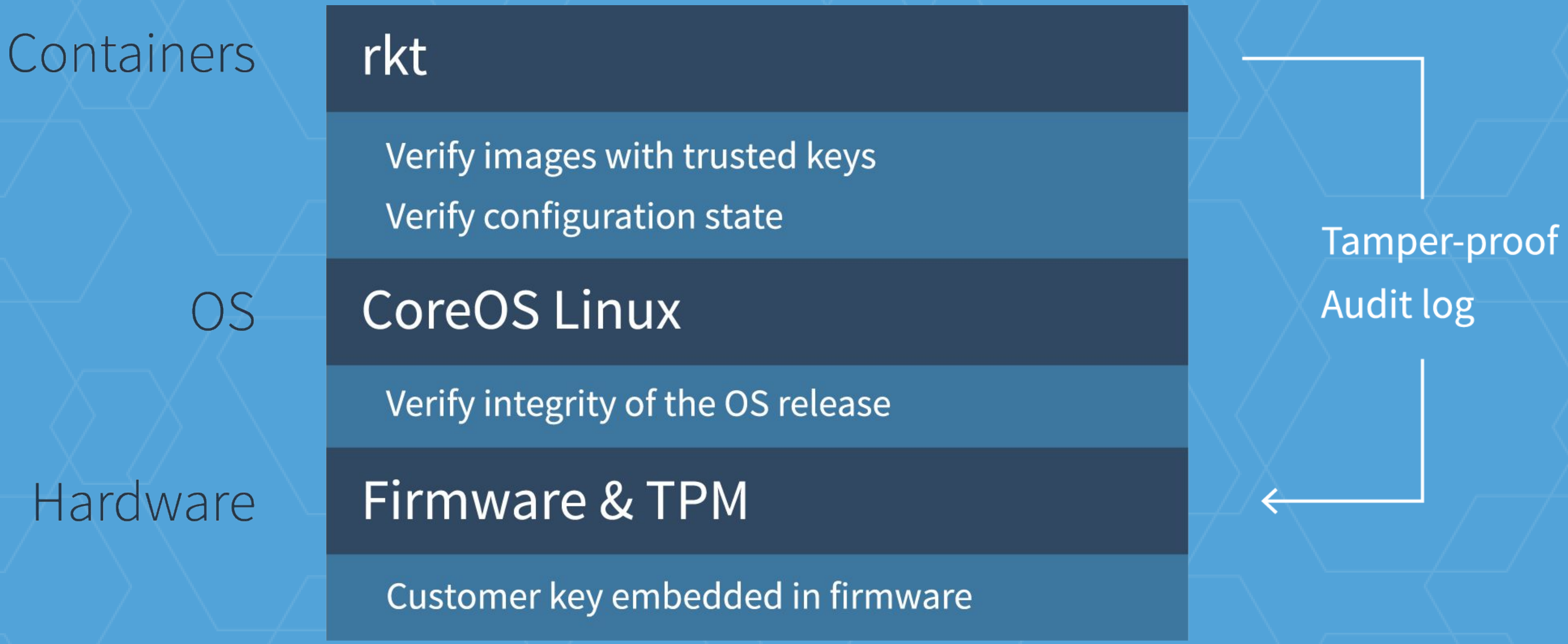
rkt TPM measurement (new!)

- TPM, Trusted Platform Module
  - physical chip on the motherboard
  - cryptographic keys + processor
- Used to "*measure*" system state
- Historically just use to verify bootloader/OS (on proprietary systems)

rkt TPM measurement (new!)

- CoreOS added support to GNU Grub
- rkt can now record information about running pods in the TPM
- attestable record of what images and pods are running on a system

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 TPM measurement (new!)

- For much, much more on TPM and rkt, see Matthew Garrett's talk:
  "Integrated trusted computing in Kubernetes"
- 11:30am tomorrow

rkt API service (new!)

- *optional*, gRPC-based API daemon
- exposes *read-only* information on pods/images
- runs as *unprivileged* user
- easier integration with other projects

Why rkt?

Secure

Standards

Composable

rkt

+

kubernetes

rkt + Kubernetes

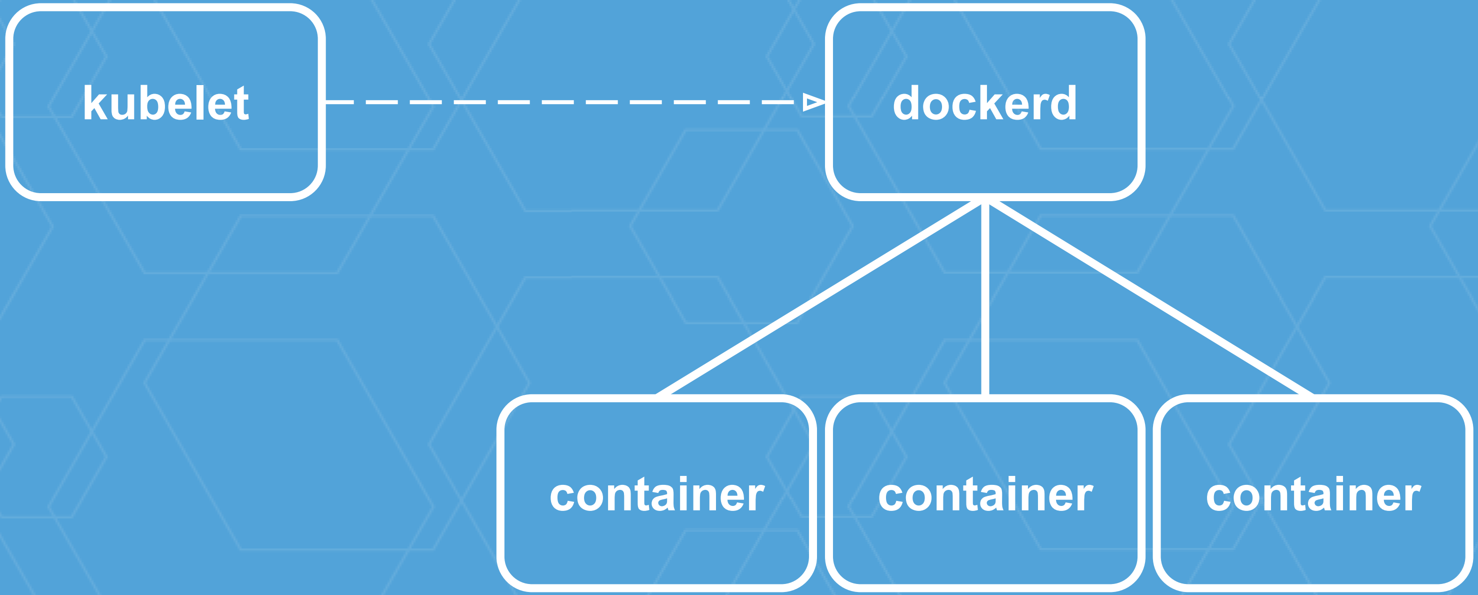
rkt ♥ k8s in a few ways:

- using rkt as container runtime (aka "*rktnetes*")
- using rkt to run Kubernetes ("*rkt fly*")
- integrating with rkt networking (CNI)

Kubelet + Container Runtimes

- Kubelet provides a `Runtime` interface
  - `SyncPod()`
  - `GetPod()`
  - `KillPod()`
  - …
- in theory, anyone can implement this
- in practise, lots of Docker assumptions

Kubelet + Docker (default)



Kubelet + Docker (default)

Problems:

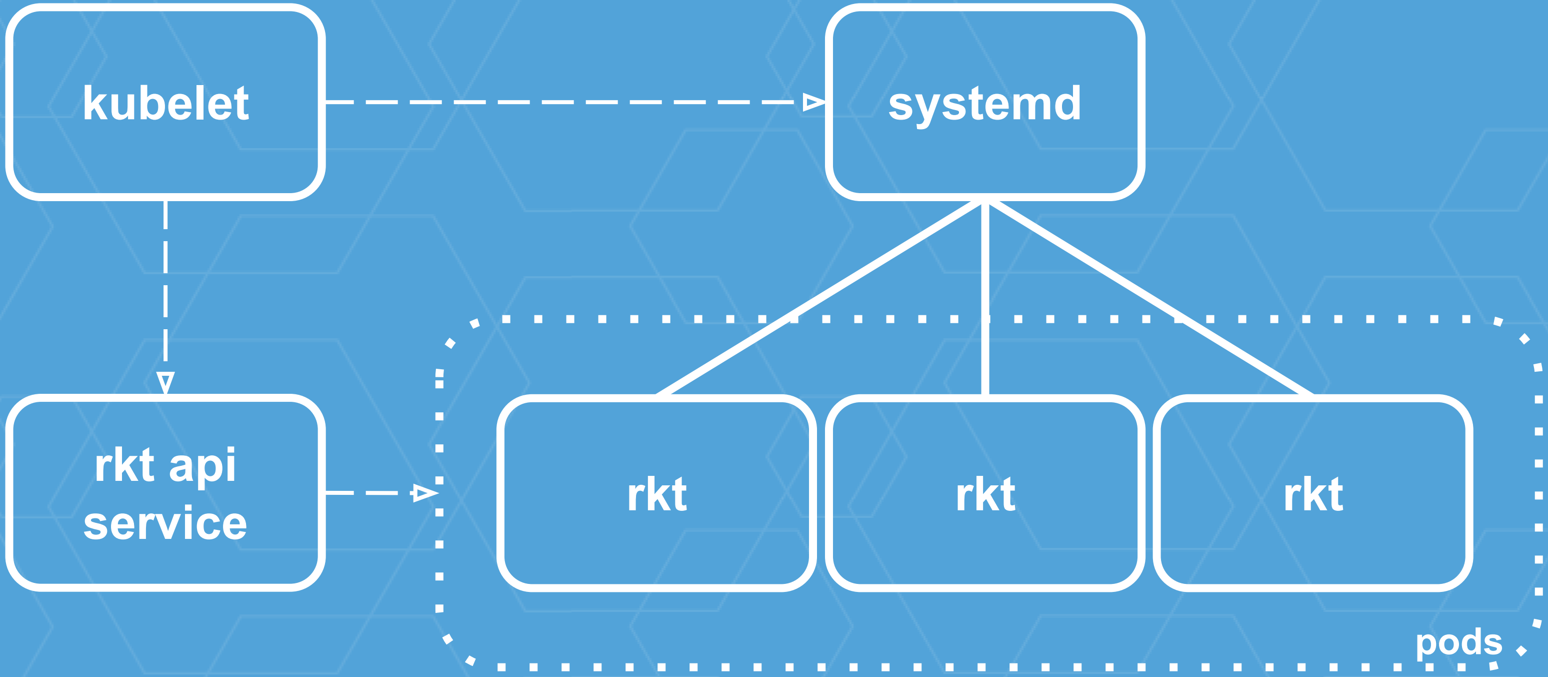
- Docker doesn't understand *pods*
  - kubelet must maintain pod<->container mapping
  - "infra container" to hold namespaces for pod
- `dockerd` = SPOF for node
  - if Docker goes down, so do all containers
- Docker doesn't interact well with systemd

Kubelet + rkt (rktnetes)

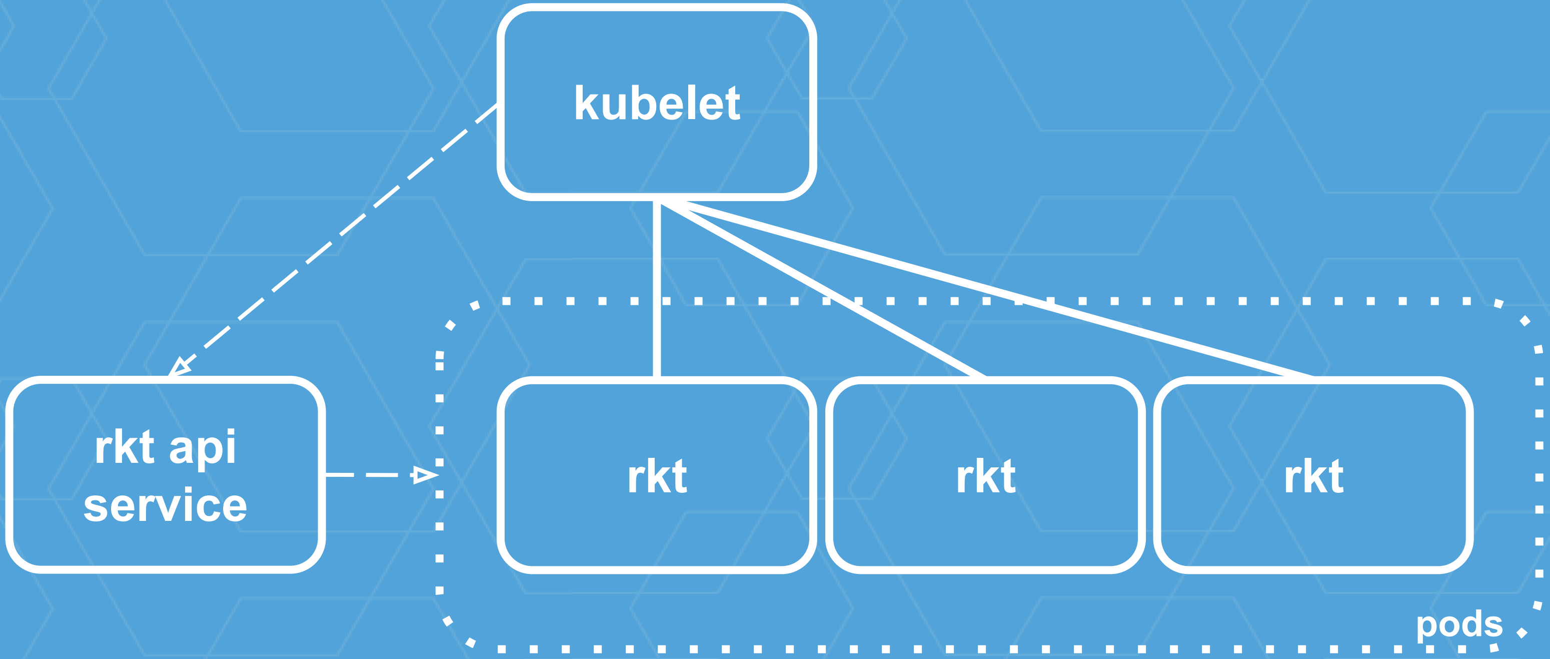
Using rkt as the kubelet's container runtime

- 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

Kubelet + rkt (rktnetes - with systemd)



Kubelet + rkt (rktnetes - without systemd)



rktnetes today

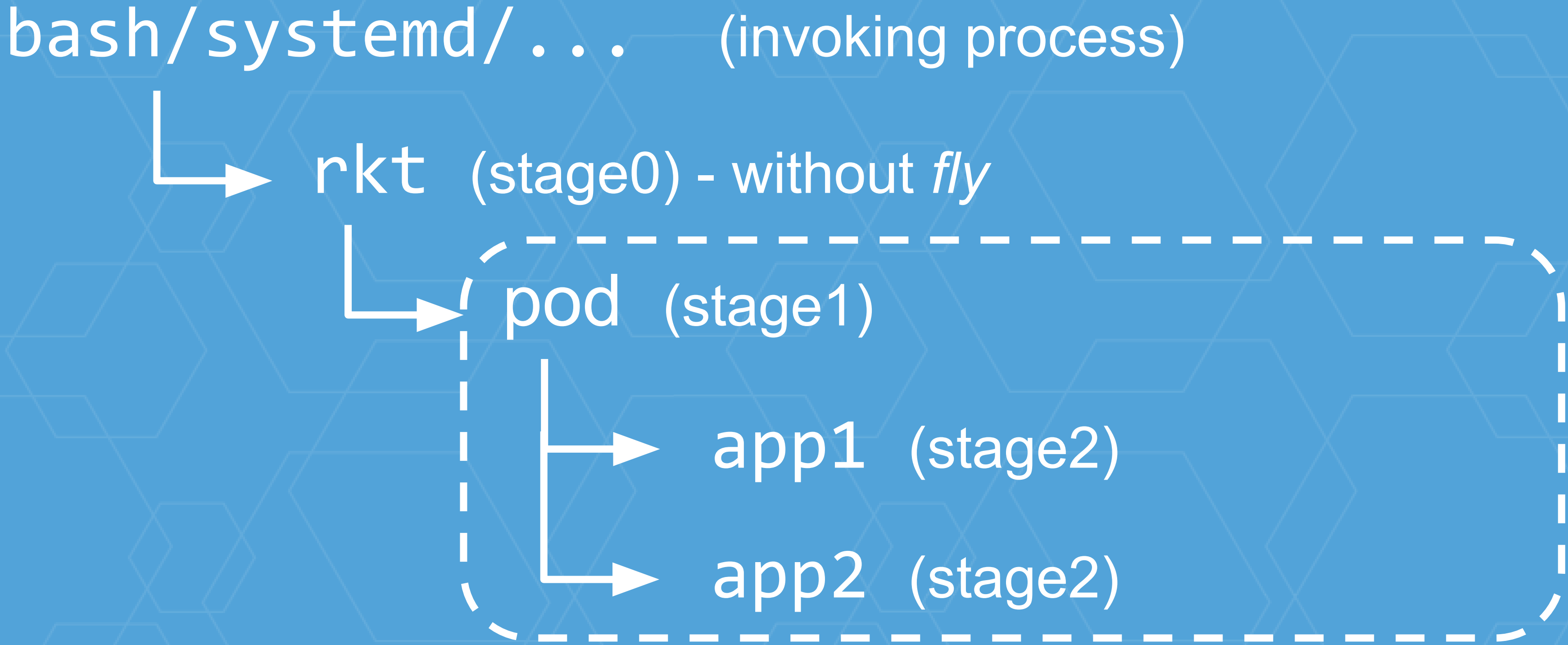
Nearly complete!
80% of end-to-end tests passing
cAdvisor integration in progress

Using rkt to run Kubernetes

- Kubernetes components are largely self-hosting, but not entirely
  - Need a way to bootstrap kubelet on the host
  - kubelets can then host control plane components
- 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 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)



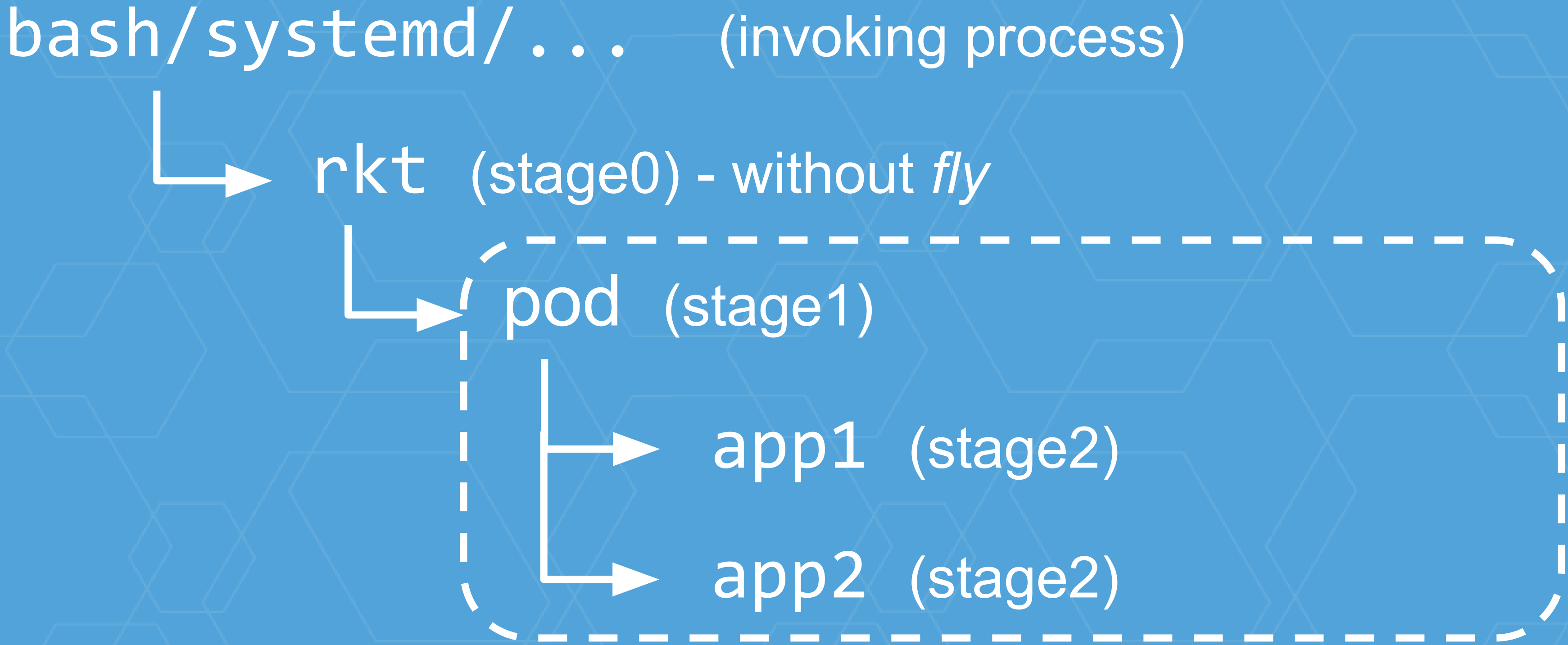
bash/systemd/... (invoking process)

rkt (stage0) - without *fly*

pod (stage1)

app1 (stage2)

app2 (stage2)



bash/systemd/... (invoking process)

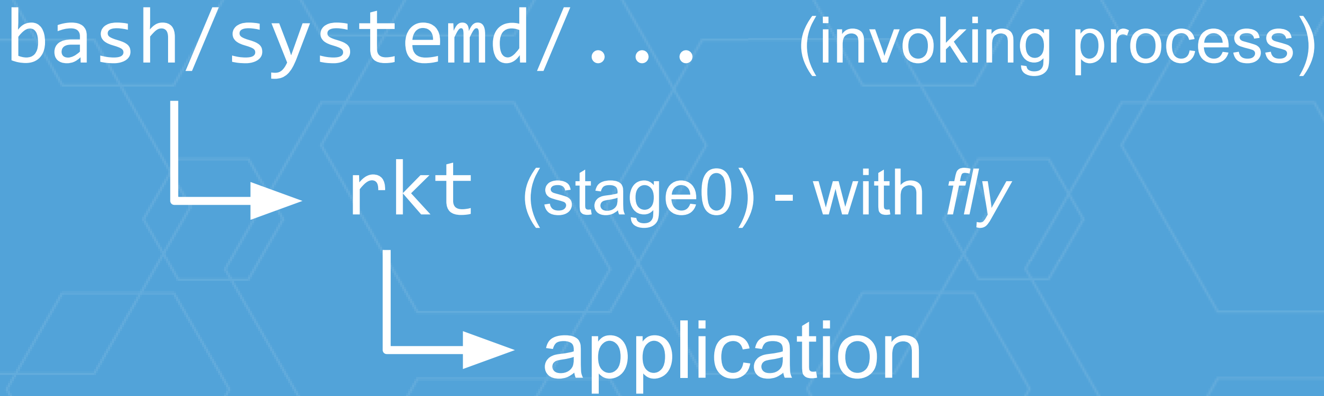
rkt (stage0) - without *fly*

pod (stage1)

app1 (stage2)

app2 (stage2)

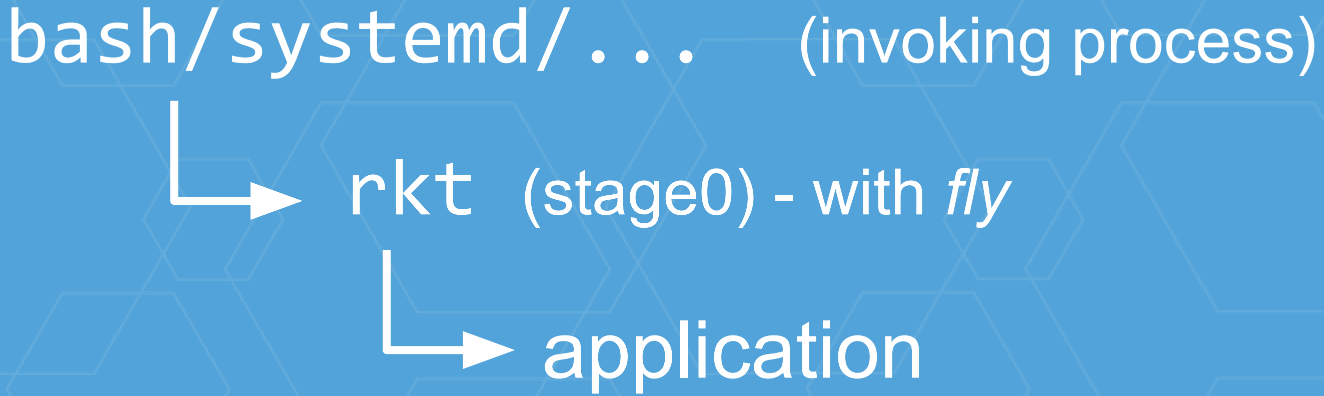
Isolated mount (and PID, ...) namespace



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

`rkt` (stage0) - with *fly*

application

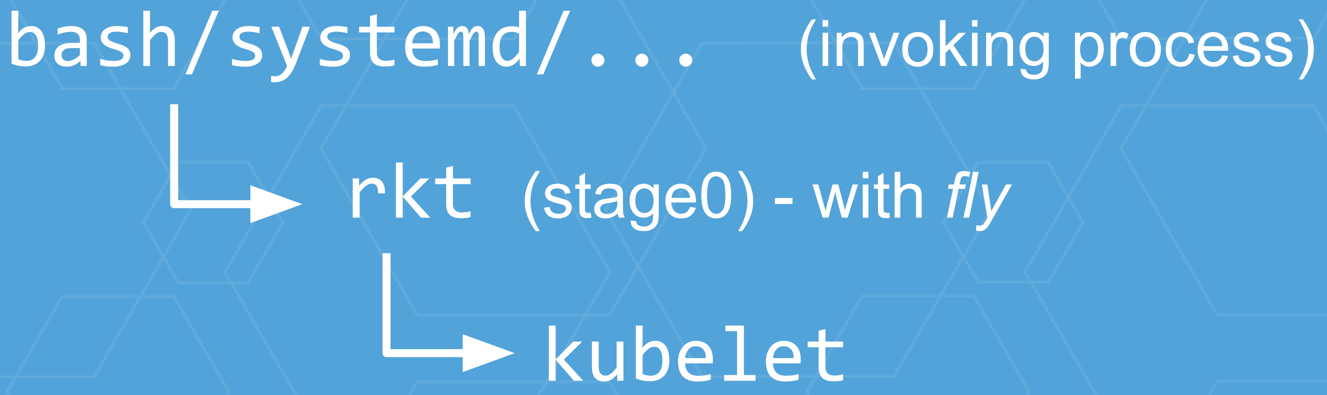


bash/systemd/... (invoking process)

rkt (stage0) - with *fly*

application

Host mount (and PID, ...) namespace



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

`rkt` (stage0) - with *fly*

`kubelet`

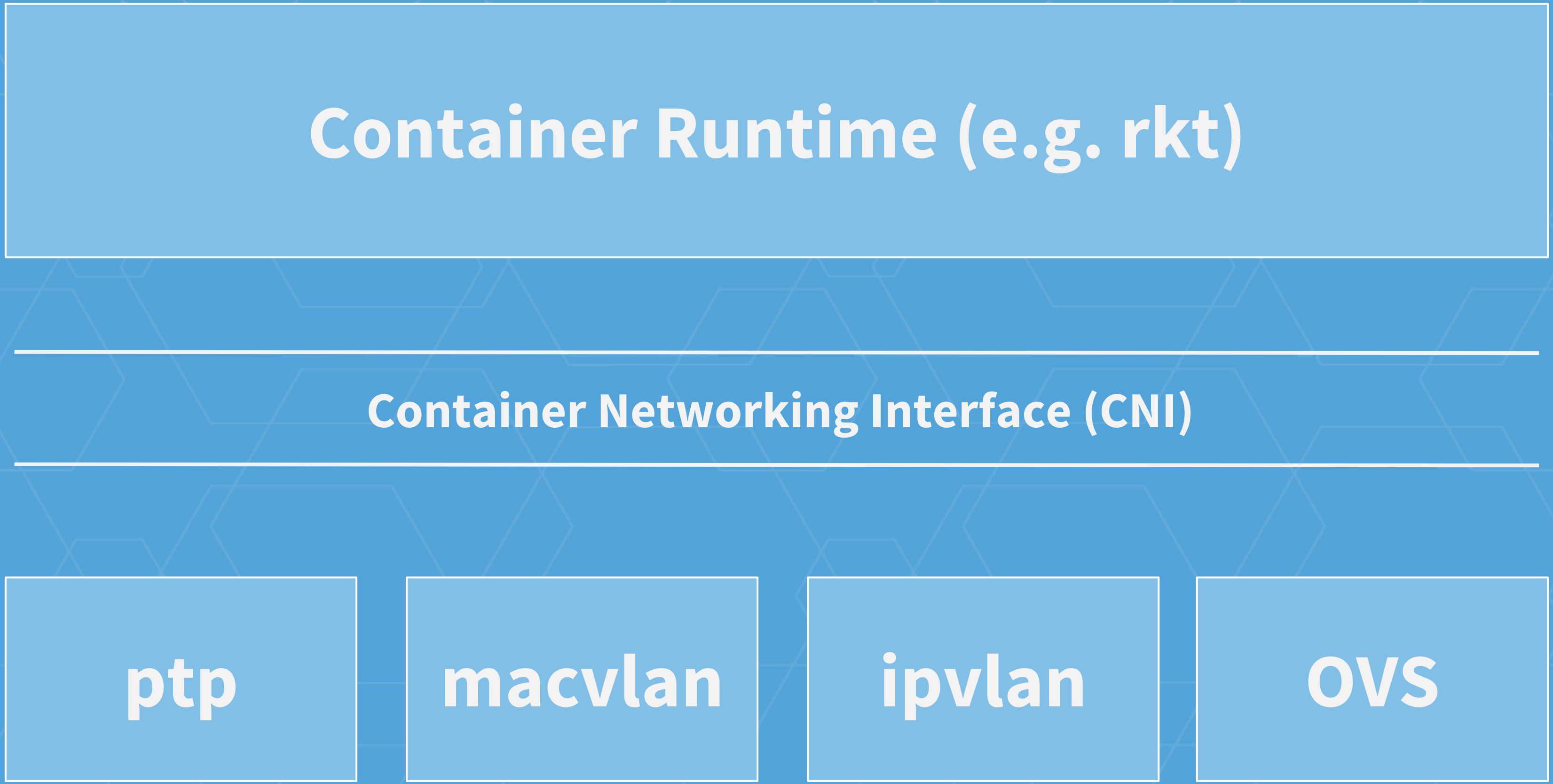
Host mount (and PID, ...) namespace

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 in a nutshell

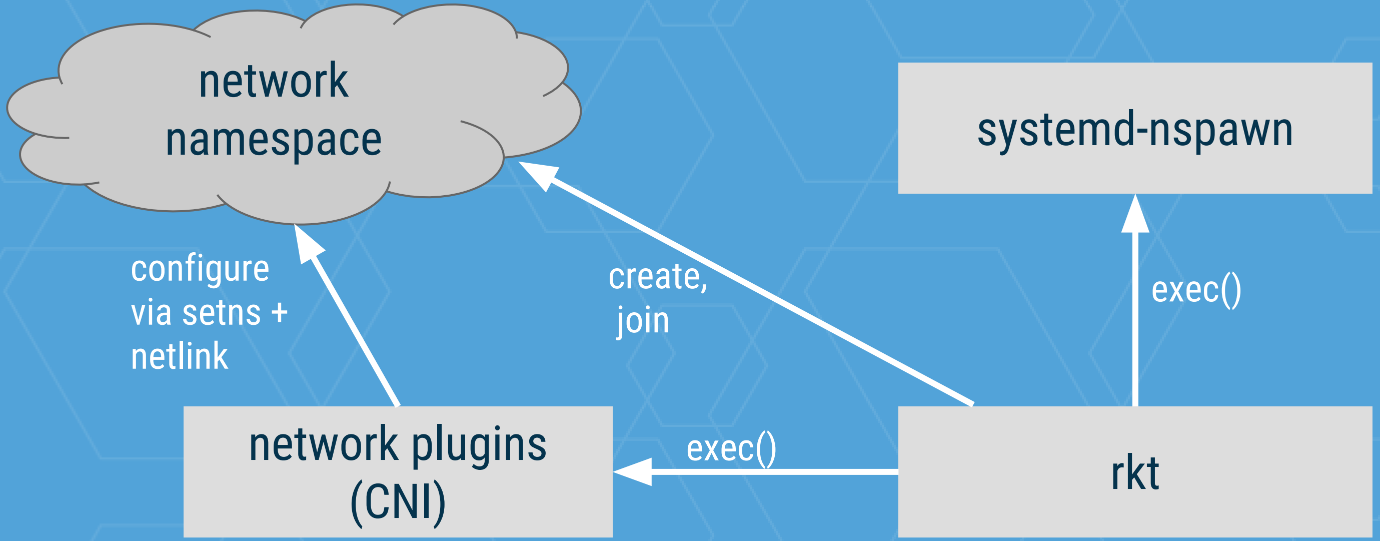
- Container can join multiple networks
- Network described by JSON config
- Plugin supports two commands
    - ADD container to the network
    - REMOVE container from the network
- Plugins are responsible for all logic
    - allocating IPs, talking to backend components, ...

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

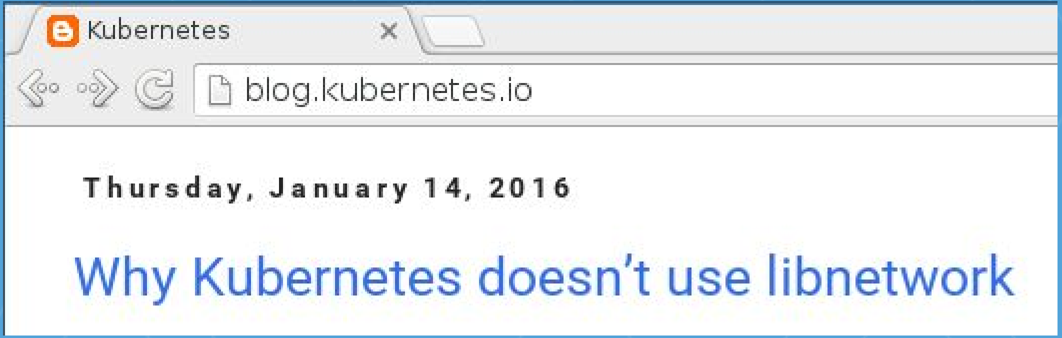
Plugin-based (but never left alpha)

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

CNI today

v0.x

Handles all networking in rkt

Integrations with Project Calico, Weaveworks

Looking ahead

What's coming up for rkt and Kubernetes

rktnetes 1.0

end of 2016Q1

Fully supported, full feature parity

Automated end-to-end testing on CoreOS

rktnetes 1.0+

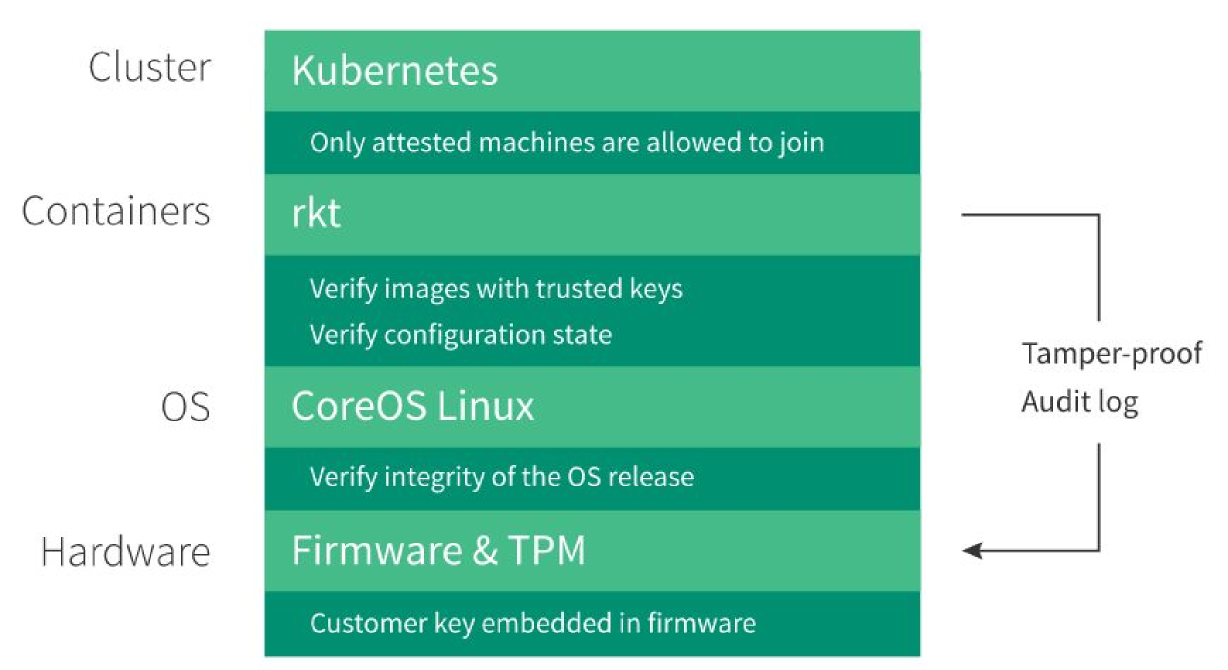
LKVM backend by default

Native support for ACI* in Kubernetes API

TPM up to the Kubernetes level

(*and/or OCI, ...)

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

CNI 1.0

Stable configuration

Stateless plugins (runtime responsibility)

IPv6

<your suggestions here>

Kubernetes 1.3

Move all networking code into CNI plugins

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

Graceful kubelet shutdown

- When an update is ready, locksmith signals kubelet to gracefully shut down
- Kubernetes can then gracefully migrate apps before shutdown
- https://github.com/coreos/bugs/issues/1112
- https://github.com/kubernetes/kubernetes/issues/7351

tl;dr:

- Use rkt
- Use Kubernetes
- Use rkt + Kubernetes (rktnetes)

- Get involved and help define the future of application containers and Kubernetes

FEST

CoreOS

May 9 & 10, 2016 - Berlin, Germany

coreos.com/fest - @coreosfest

Questions?

rkt + kubernetes + CoreOS

Join us!

contribute: github.com/coreos/rkt

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

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!)