CoreOS ecosystem

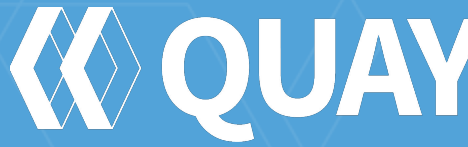
rkt, CoreOS, et al

Sergiusz Urbaniak

github.com/s-urbaniak - @_surbaniak

What is CoreOS?

The company behind ...



Major supporters, contributors of ...

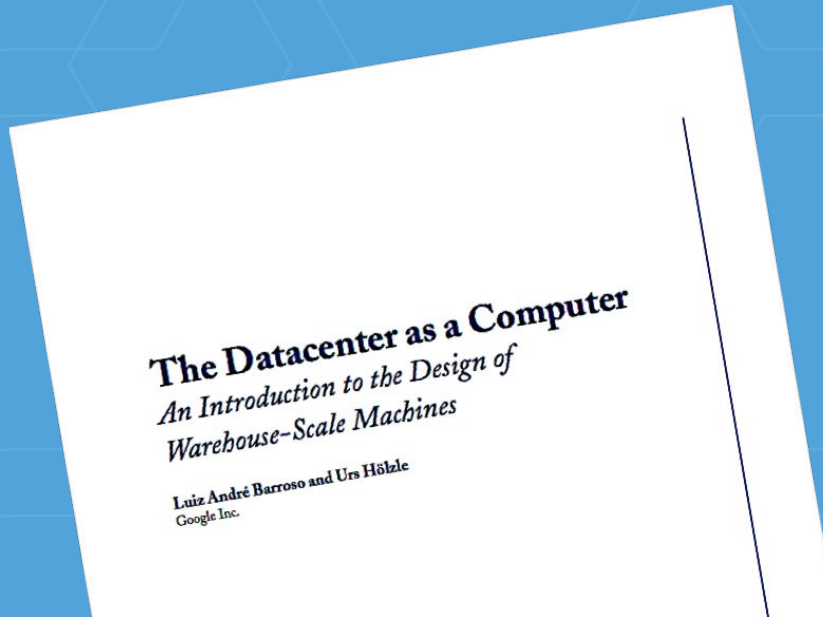
kubernetes

Prometheus

A brief standards history

- appc (December 2014)
  - container images, runtime environment, and pods
  - some adoption, but (intended to be) deprecated in favour of
- OCI (June 2015)
  - initially runtime only, now container images too
- CNCF (December 2015)
  - "harmonising cloud-native technologies"

Why build all this?


The Datacenter as a Computer
An Introduction to the Design of Warehouse-Scale Machines
Luiz André Barroso and Urs Hölzle
Google Inc.

you

you as a sw engineer

your

```
with Ada.Text_IO;

procedure Hello_World is
 use Ada.Text_IO;
begin
    Put_Line("Hello, world!");
end;
```

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
}
```

```
package main

import "fmt"

func main() {
      fmt.Println("Hello, world!")
}
```

your

your

your

your

d474e8c57737625c

your

Signed By: Alice

d474e8c57737625c

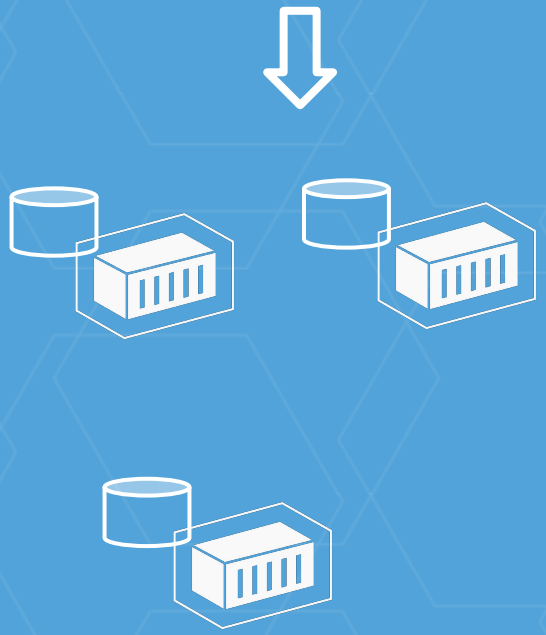
you as an ops engineer

your

com.example.webapp
x3

your

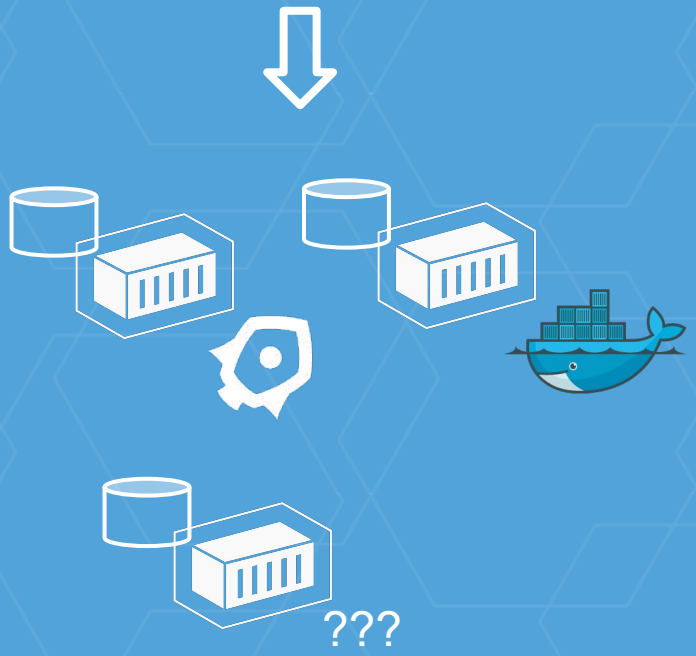
com.example.webapp
x3



your

your

com.example.webapp
x3



kernel
systemd
rkt
ssh
docker

python
java
nginx
mysql
openssl

stro distro distro distro distro distro

app

kernel
systemd
rkt
ssh
docker

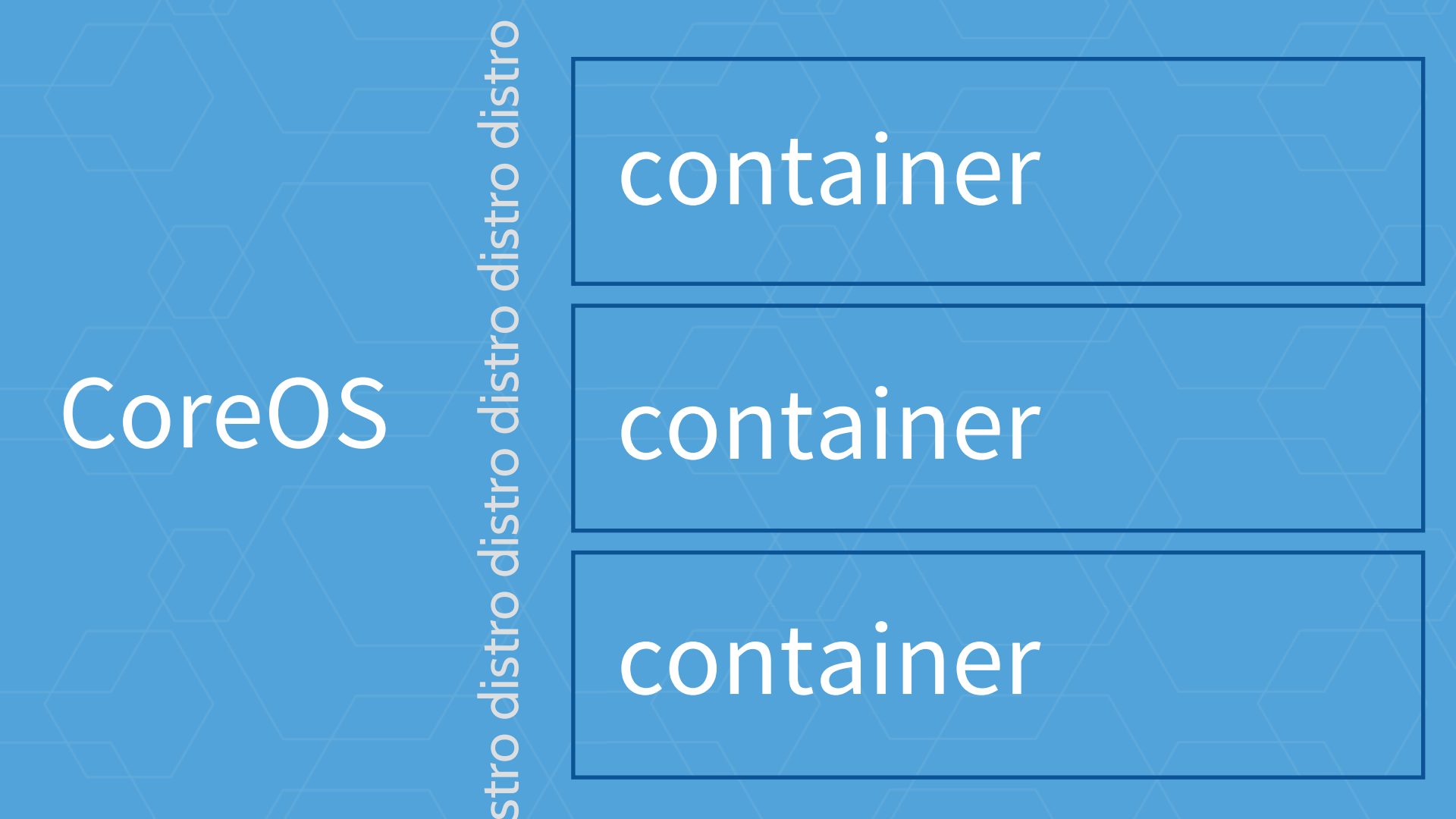
stro distro distro distro distro distro

python
java
nginx
mysql
openssl

app


kernel
systemd
rkt
ssh
docker
distro distro distro distro distro distro distro distro
python
openssl-A
app1
java
openssl-B
app2
java
openssl-B
app3


CoreOS
distro distro distro distro distro distro distro
python
openssl-A
app1
java
openssl-B
app2
java
openssl-B
app3


CoreOS
distro distro distro distro distro distro distro
container
container
container

What is rkt?

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* (production)**
  - Runtime stability + interface guarantees
- < ... many more ... >
- **October 2016 - *v1.20.0***
  - Latest stable release

rkt vs. Docker

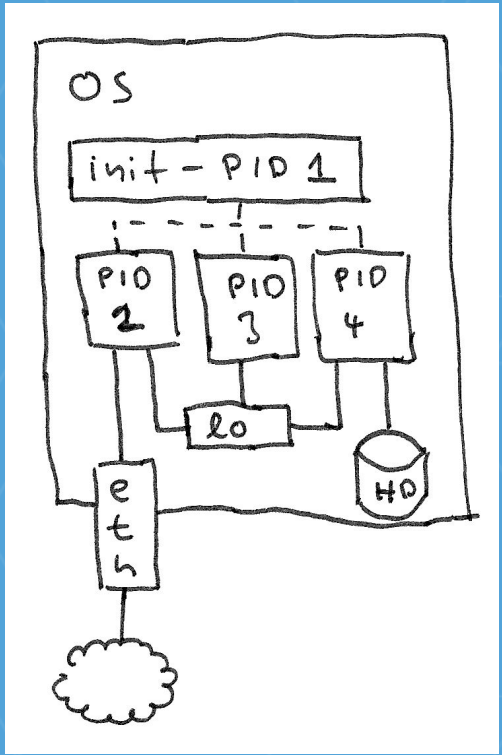
- **no** central daemon
  - each rkt app is a separate UNIX process
- 1st class **pod** support
  - run multiple containers in one isolated context
- Supports **multiple image formats**
  - Docker
  - appc / ACI
  - OCI

Pods - what's that again?

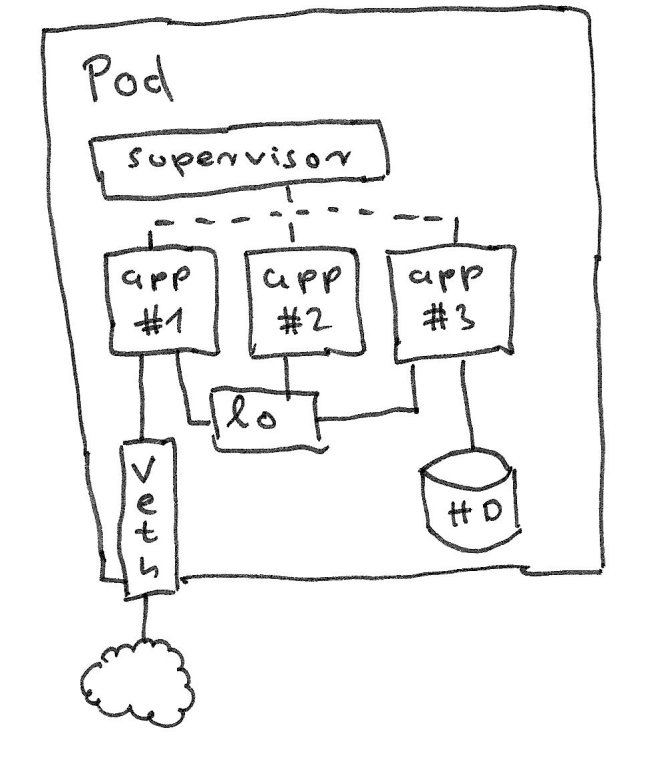
- A list of apps, where each app ...
  - references a container image
  - has a main executable

- Launched inside a shared execution context (namespace)
  - PID namespace
  - Network namespace
  - IPC namespace
  - UTS namespace

Before containers existed ...



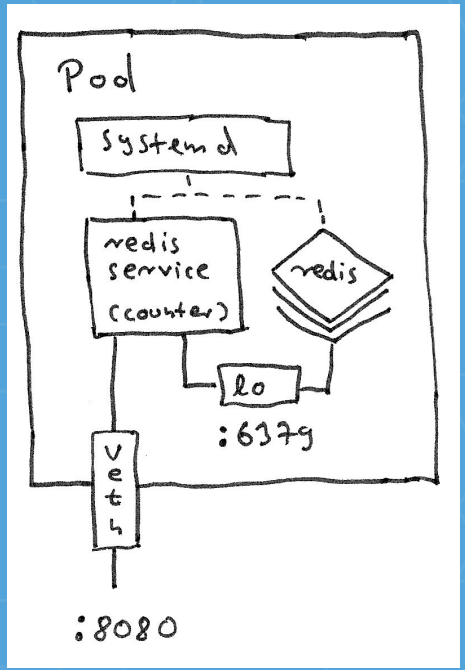
… now using rkt …



Pods - Example

- Redis Server
  - Docker image
  - Written in C

- Counter Microservice
  - appc image
  - Written in Go




```
$ sudo rkt run \
  --insecure-options=image docker://redis \
  s-urbaniak.github.io/images/redisservice:0.0.2
```

A CLI for running app containers on Linux.

Focuses on:

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

How rkt does security

- **UX: "*secure-by-default*"**
  - Default image/integrity verification
  - Default capabilities restrictions

- **Architecture: *Unix philosophy***
  - Well-defined operational scope
  - Clean integration points as a classic Unix process
  - Separate privileges for different operations ("fetch" operations shouldn't need root)

How rkt does security

- User namespaces
- SELinux contexts
- Support for VM containment
- TPM measurements

How rkt does security (cont.)

- Fine-grained Linux capabilities(7)
- seccomp(2) enabled by default
- Mask sensitive `/proc` and `/sys` paths

Security will never be "complete"; always an iterative process, refining over time

A CLI for running app containers on Linux.

Focuses on:

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

How rkt does composability

- **"*External*" composability**
  - Unix architecture; integrating well with other tools (init systems, orchestration tools) is a priority

- **"*Internal*" composability**
  - Swappable execution engines (stage-based architecture) that actually runs the container

How rkt does composability

- **"*External*" composability**
  - Simple process model: a single rkt process *is* a pod
  - Any context applied to rkt (cgroups, etc) applies transitively to the pod and the apps inside
  - No mandatory daemon (but optional API server)


bash/systemd/kubelet...
(invoking process)
rkt run docker://nginx
fork()/exec()

```
systemd-run -p MemoryLimit=1G rkt run ...
```

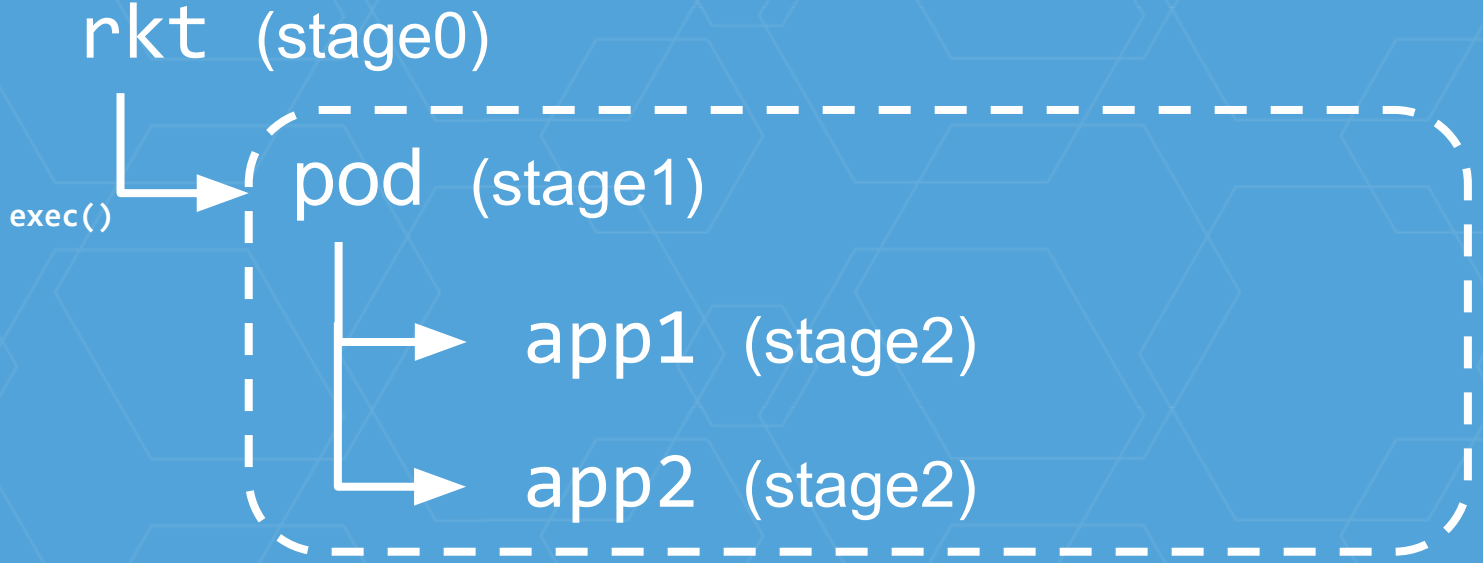

```
rkt run docker://nginx
```

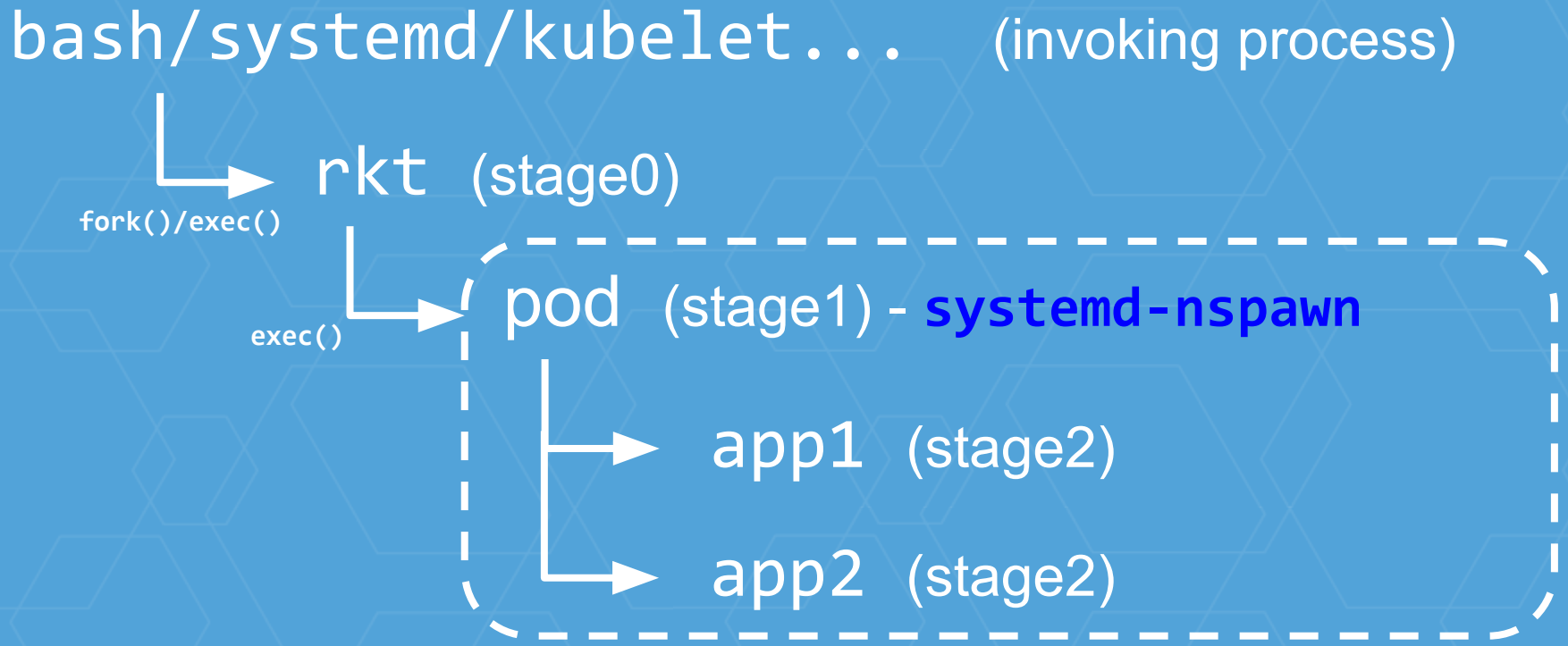
How rkt does composability

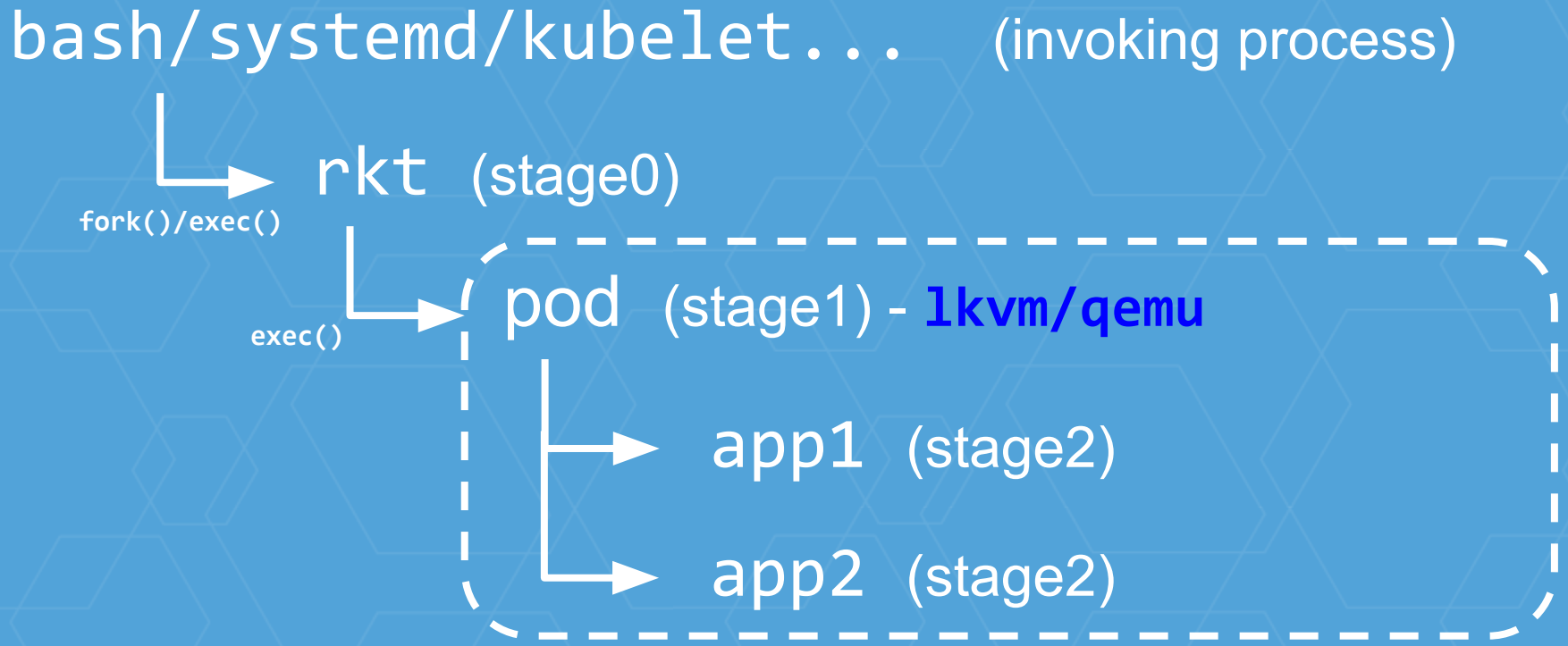
- **"Internal" composability**
  - Staged architecture
  - "rkt" is the UX/API, container technology is an implementation detail

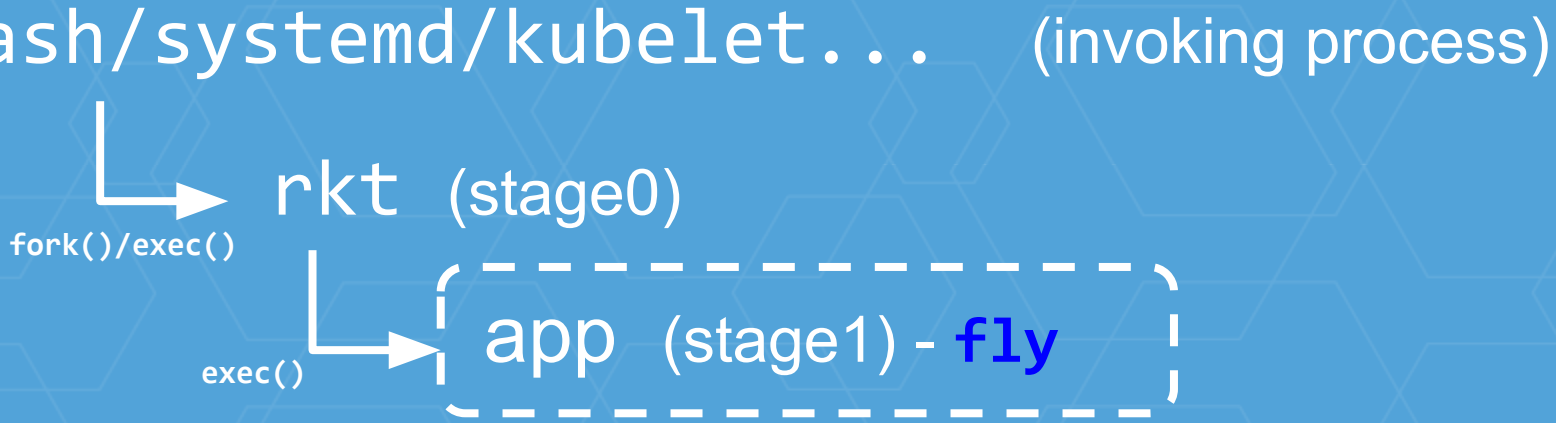
- **Available stage1s**
  - Linux namespaces+cgroup (default)
  - KVM (LKVM / QEMU-KVM)
  - chroot ("fly")


rkt
exec()
pod
app1
app2


rkt (stage0)
exec()
pod (stage1)
app1 (stage2)
app2 (stage2)


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


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


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

A CLI for running app containers on Linux.

Focuses on:

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

How rkt does standards/compatibility

- Started as an implementation of **appc**
- Networking plugin system became **CNI**
- Can run **Docker images** natively (V1, V2, ...)
- Developers participate actively in **standardisation efforts**
  - appc, CNI, OCI, CNCF
  - rkt will be fully OCI compliant

Kubernetes

Cluster-level container orchestration.

Handles:

- Scheduling/Upgrades
- Failure recovery
- Scaling

How does Kubernetes contain?

- **kubelet** is the daemon that runs on every worker node in a Kubernetes cluster
- kubelet runs the *pods* scheduled to it by Kubernetes
- kubelet delegates to container runtime to perform all container-related operations

Kubernetes + rkt = rktnetes

Have Kubernetes use rkt as
the container runtime.

rkt handles:

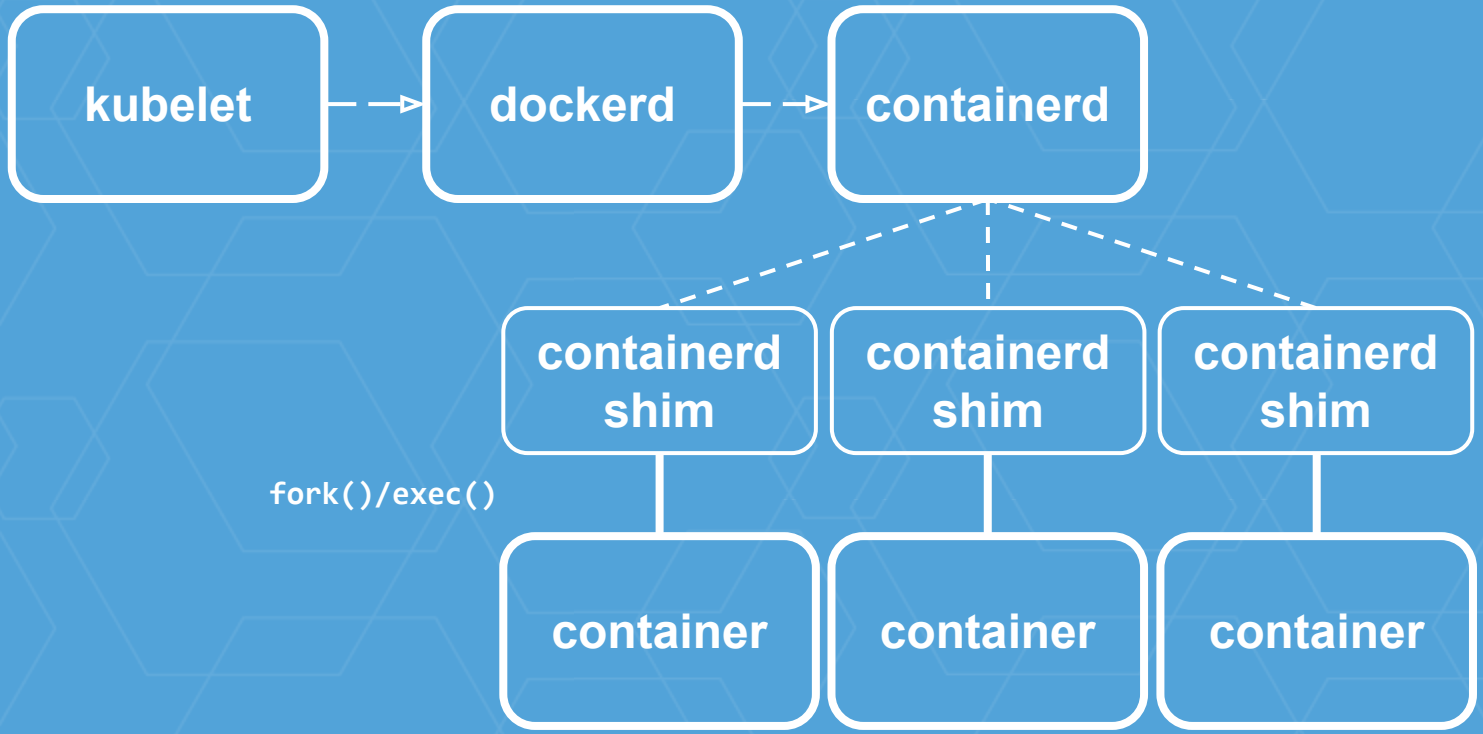
- Image discovery
- Image fetching
- Pod execution

How does it work?

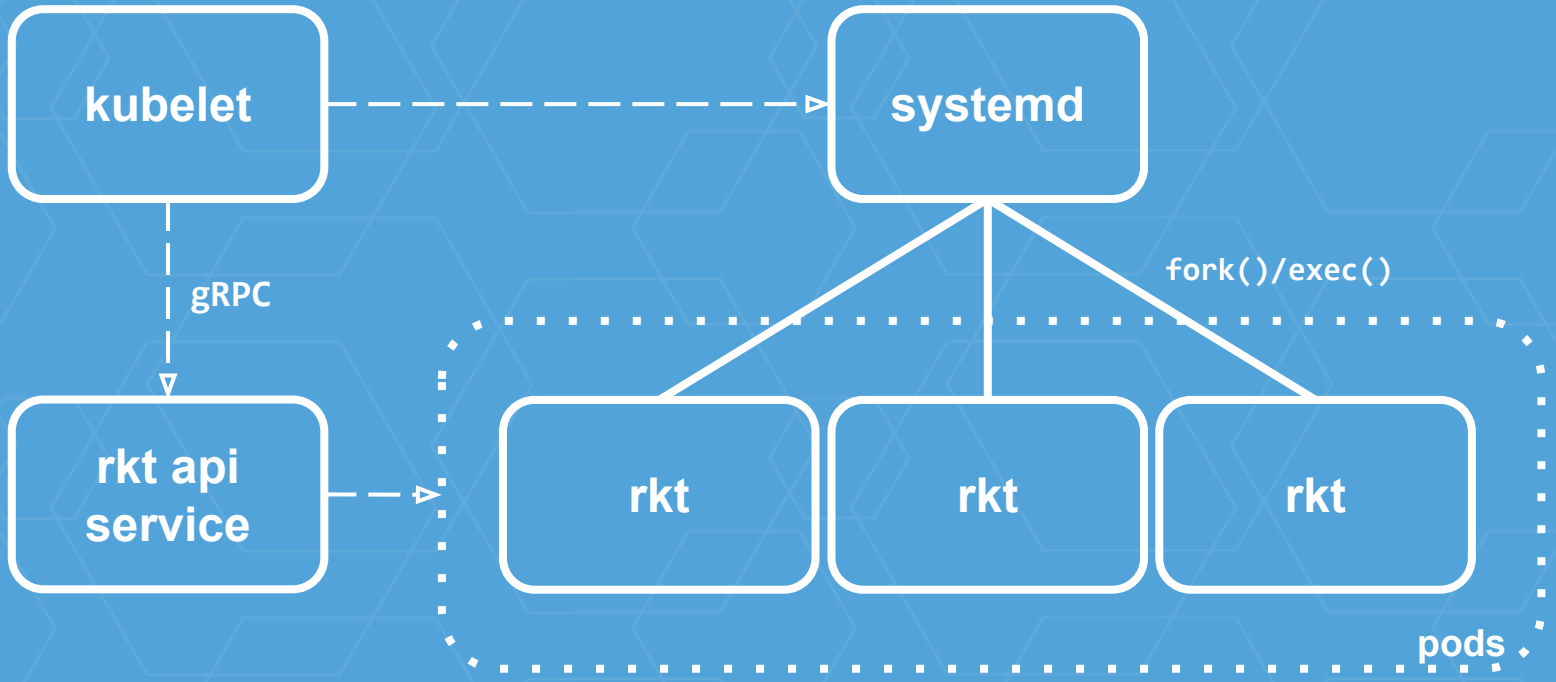
Pre-rktnetes (current default):

- Kubelet talks to the ***Docker daemon*** for all tasks

Kubelet + Docker (>= 1.11)



Kubelet + rkt (rktnetes)



What's the benefit in this?

- **no single daemon** running the containers
- pod native environment
- multiple stage1s provides more flexibility
- seamless integration with systemd
  - `machinectl`, `systemctl`, `journalctl` Just Work™

rktnetes: *does* it work?

- **Yes!**

- Official release in Kubernetes 1.3

http://blog.kubernetes.io/2016/07/rktnetes-brings-rkt-container-engine-to-Kubernetes.html

- Minikube Support

https://tectonic.com/blog/minikube-and-rkt.html

How do I find out more?

- **Reach out on GitHub or IRC**
  - github.com/coreos/rkt, #rkt-dev / #rkt on Freenode

- **Join a Kubernetes Special Interest Group (SIG)**
  - https://groups.google.com/forum/#!forum/kubernetes-sig-node
  - https://groups.google.com/forum/#!forum/kubernetes-sig-rktnetes
  - `#sig-node` / `#sig-rktnetes` on Kubernetes Slack

Some interesting papers

- Paper: Large-scale cluster management at Google with Borg
- The Datacenter as a Computer
- Paper: Design patterns for container-based distributed systems
- CAP basics