Container, rkt, Kubernetes

顾宜凡 (Yifan Gu) Software Engineer @ CoreOS
github.com/yifan-gu

Overview

1. Containers, OCI, Appc
2. rkt
3. Kubernetes, and rkt + Kubernetes

Container Is HOT

Compare Search terms

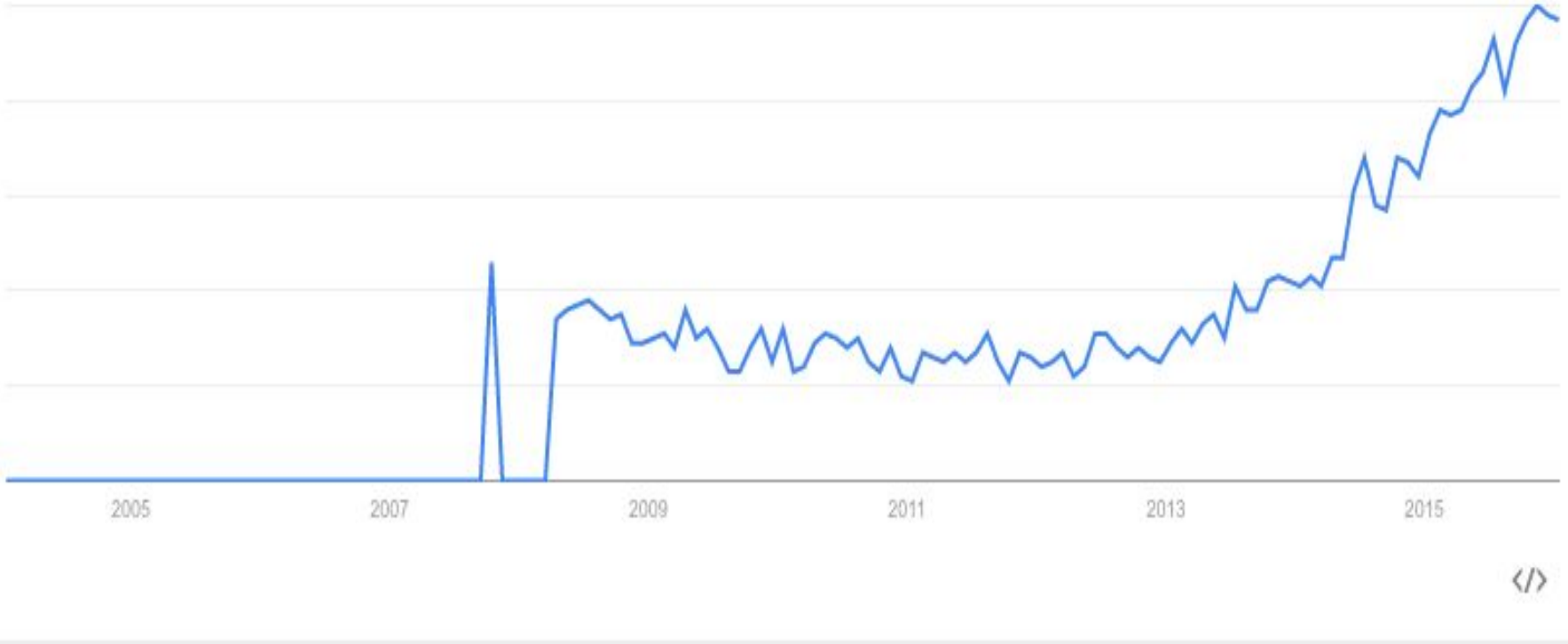
linux container
Search term

+ Add term

Interest over time

News headlines Forecast

Container is not a new technology, Why?



So, what is container?

Container = Docker ?

So, what is container?

Control group

- CPU
- Memory
- IO
- Devices
- ...
-

Namespaces

- Network
- IPC
- ProcessID
- ...

Linux

So, what is container?

Container = Docker ? No

So, what is container?

Container = Docker ? No
Container = cgroup + namespace ? No

So, what is container?

FreeBSD ~ 2000

FreeBSD®

solaris™

Solaris Zones ~ 2005

So, what is container?

Container = package + runtime !

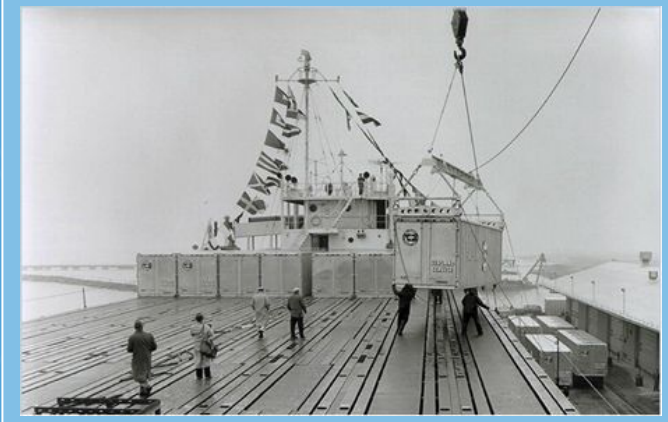
So, what is container?

Container = package + runtime !

- Easy packaging (build/push/pull)
- Isolated, controlled (run/stop)

If it’s hot, then standardize it

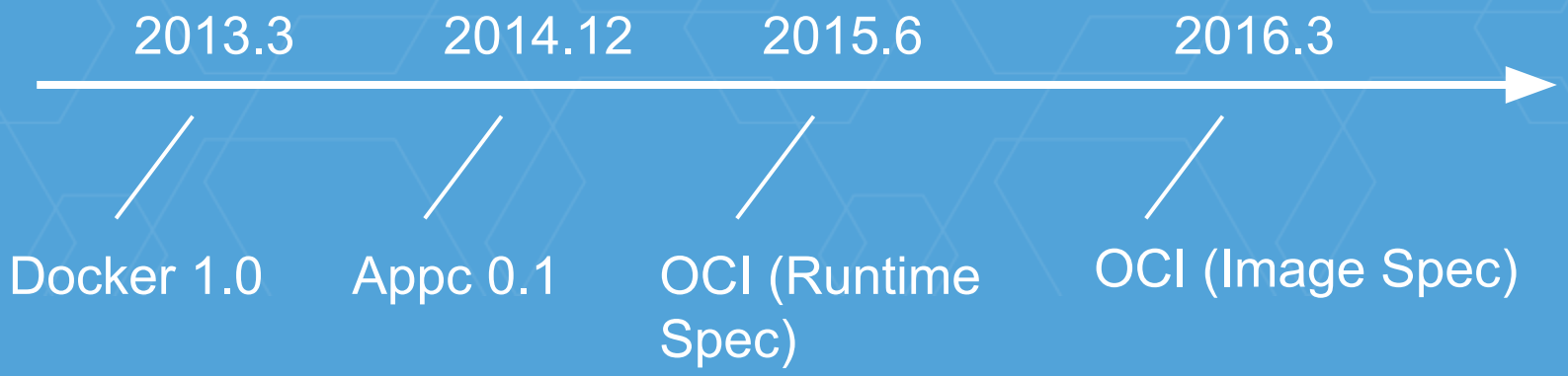
If it’s hot, then standardize it



Container ~ 1950

Container ~ 2010

Container Spec Timeline



Container Spec

Open Container Specifications

- **Runtime Spec**
  - config.json
  - runtime.json
  - rootfs
- **Image Spec**
  - started from Docker v2
  - absorb from Appc
    - discovery
    - signing
    - app configs

github.com/coreos/rkt

What is rkt ?

rkt is a CLI for running app containers on Linux. rkt is designed to be **secure**, **composable**, and **standards-based**.

rkt doesn’t require a long-running daemon and provides a powerful, pluggable, abstraction around isolation and runtime initialization.

How rkt does security

- GPG signatures to verify images
- SELinux contexts
- Can run containers in hypervisor
- Can do TPM measurements, provides a tamper-proof audit log

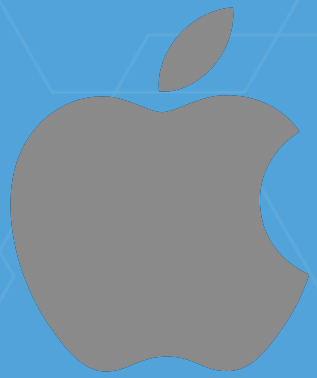
How rkt does composability

- Integrating well with init systems
- Aims to work well with other projects
- rkt has the concept of a “stage1”, which is a swappable component that actually runs the container
- Available stage1s
  - chroot
  - Linux namespaces (default)
  - LKVM

How rkt does standards/compatibility

- Implementation of AppC, a well defined spec
- Uses CNI for networking, common plumbing used by many other projects
- Can run docker images
- Will be fully OCI compliant

Distributed Trusted Computing



Distributed Trusted Computing Stack

Distributed Trusted Computing Stack

Cluster

Kubernetes

Only attested machines are allowed to join

Containers

rkt

Verify images with trusted keys

Verify configurat

OS

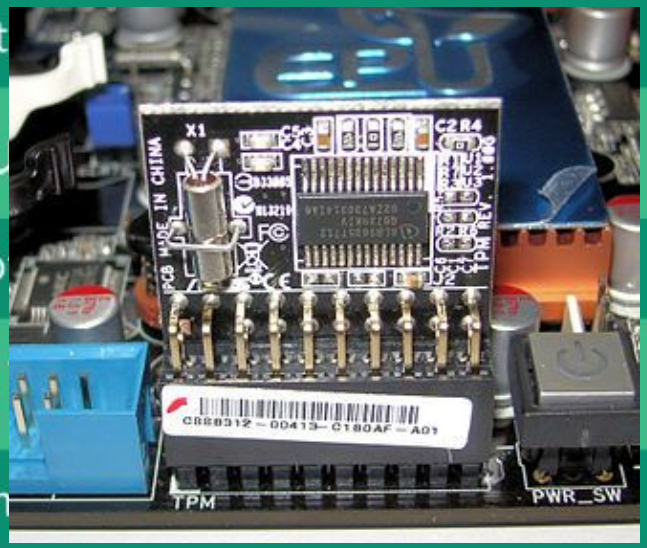
CoreOS Linu

Verify integrity o

Hardware

Firmware &

Customer key en



Tamper-proof
Audit log

rkt internals

modular architecture

execution divided into *stages*

stage0 → stage1 → stage2

rkt Stages

Stage 0

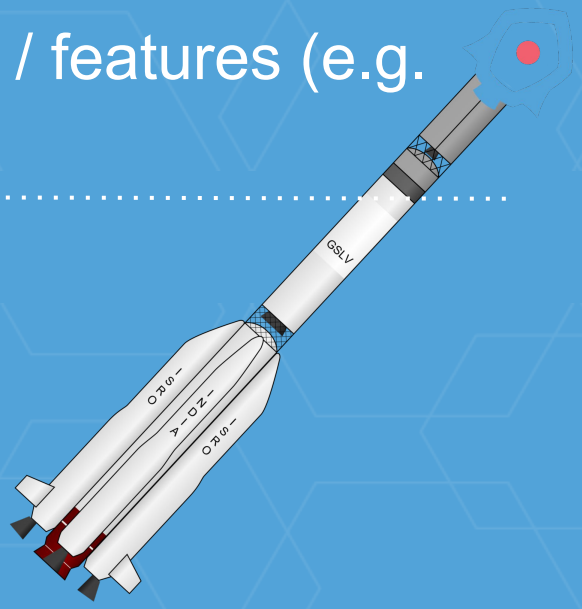
- Image discovery and fetching - Locate and download ACI and Docker images
- Unpacking/preparing the container manifest and filesystem

Stage 1

- Setting up network and namespace isolation
- Handle any needed runtime setup / features (e.g. systemd)

Stage 2

- Container entrypoint!



Stage 0 - Fetch

```
$ rkt fetch example.com/redis
$ rkt fetch docker://nginx
$ rkt fetch https://my_web_container.aci
$ rkt fetch ./my_container.aci
```

Stage 0 - Fetch

```
$ rkt fetch example.com/redis
# GET https://example.com/redis
#    <meta name="ac-discovery" content=”example.com/redis
https://example.com/redis.{ext}”>
# GET https://example.com/redis.aci.asc
# GET https://example.com/redis.aci
Downloading signature:
[=======================================] 287 B/287 B
Downloading ACI:
[=======================================] 10 MB/10 MB
image: signature verified:
  Example <example@example.com>
sha512-...
```

Stage 0 - Prepare

```
$ rkt prepare example.com/redis
uuid
$ tree /var/lib/rkt/pods/prepared/uuid
/var/lib/rkt/pods/prepared/uuid/
├── appsinfo
│   └── redis
│       ├── manifest
│       └── treeStoreID
├── overlay-prepared
├── pod
├── stage1
│   └── manifest
└── stage1TreeStoreID
```

Stage 1 run-prepared

```
$ rkt run-prepared uuid
```

Default: Systemd-nspawn

- Writes a unit file for each application based on its manifest
- Setup network namespaces (CNI)
- Handle mounts via systemd-nspawn (default)
- Hand off to systemd

Stage 1

```
$ rkt run-prepared uuid
```

Default: Systemd-nspawn

$ systemd-nspawn --boot --register=false --link-journal=try-guest --quiet --uuid=a1caebb1-948b-4486-8133-bb21133a7090 --machine=rkt-a1caebb1-948b-4486-8133-bb21133a7090 --directory=stage1/rootfs --capability=CAP_AUDIT_WRITE,CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_FSETID,CAP_FOWNER,CAP_KILL,CAP_MKNOD,CAP_NET_RAW,CAP_NET_BIND_SERVICE,CAP_SETUID,CAP_SETGID,CAP_SETPCAP,CAP_SETFCAP,CAP_SYS_CHROOT -- --default-standard-output=tty --log-target=null --show-status=0

Stage 1 (Continued)

- Distributed as a container image (rkt-fetchable, self-containerd)
- Run with no isolation on the host
- Support multiple stage1 for different purpose

Examples

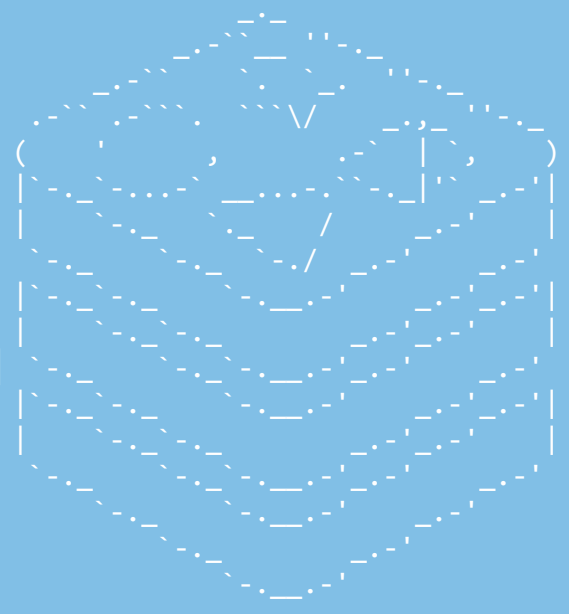
- stage1-coreos - Run all applications under systemd
- stage1-kvm - Run all applications under lkvm
- stage1-fly - Run an application under chroot “isolation”

Stage 2

- App + some nice features:

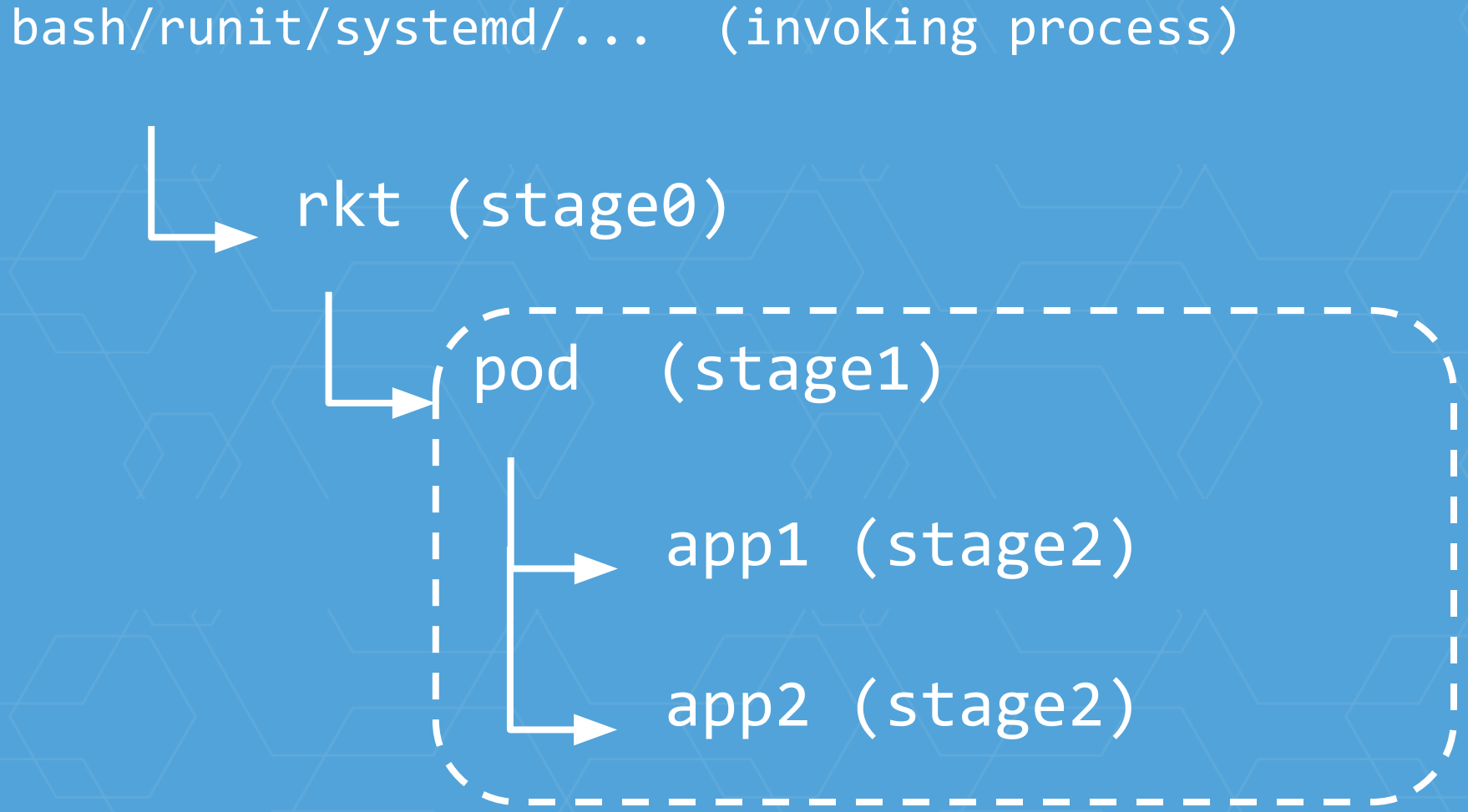
```
rkt———systemd-nspawn———systemd—┬—redis-server
                                └—systemd-journal
```

```
$ machinectl list
rkt-uuid
...
$ journalctl -M rkt-uuid
...
```



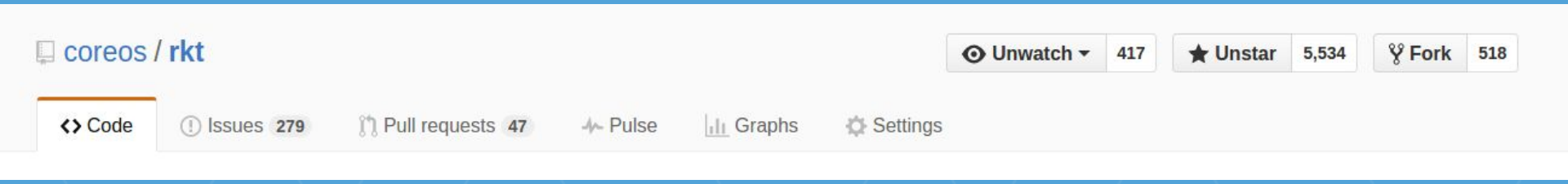
```
Redis 3.2.0 (00000000/0)

Running in standalone mode
Port: 6379
PID: 5


      http://redis.io
```

rkt Stages


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

rkt is in production

github.com/coreos/rkt



Only Container Runtime Is Not Enough

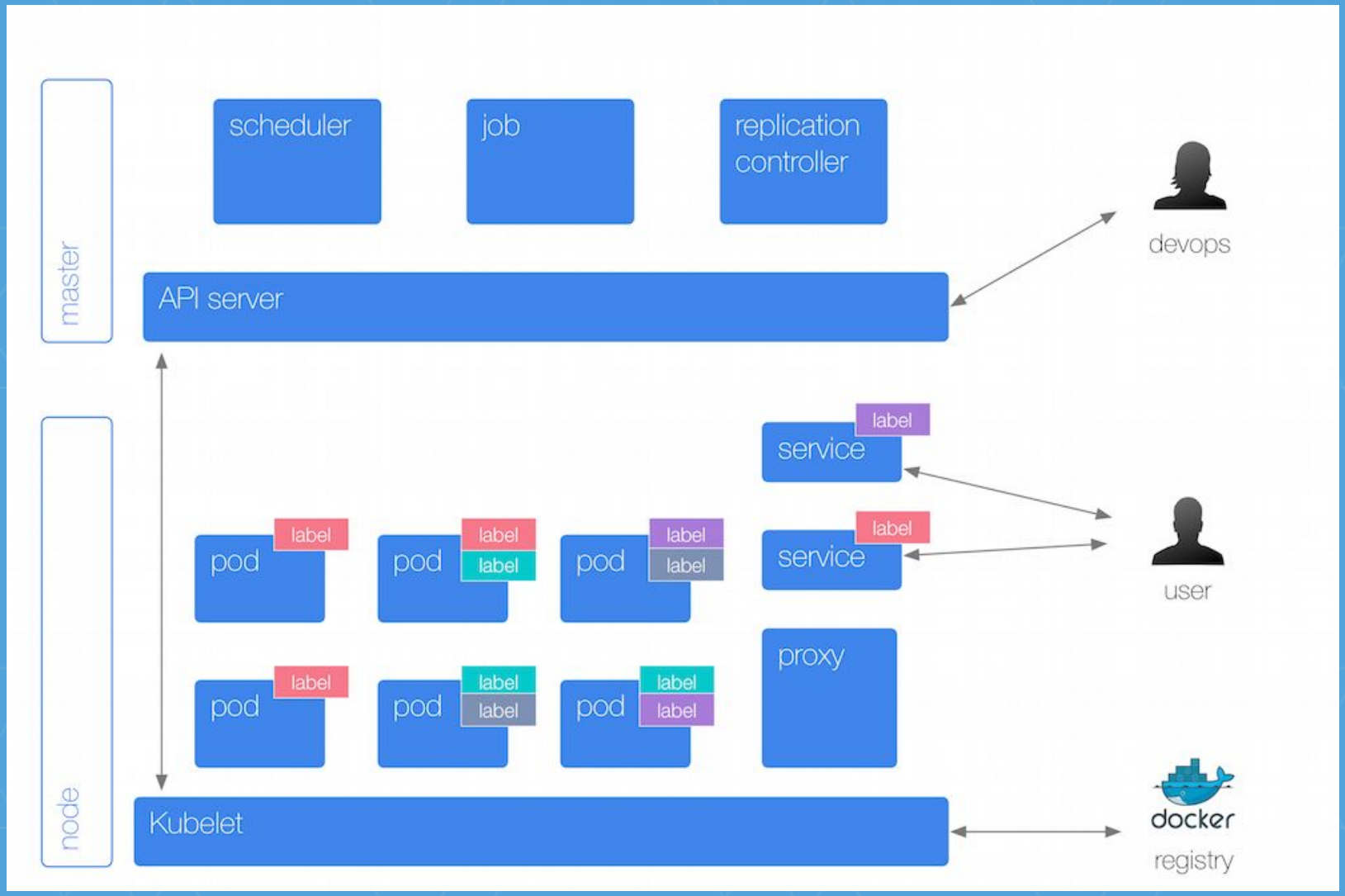
Only Container Runtime Is Not Enough

- Fleet,
- Mesos (marathon)
- Docker Swarm
- Kubernetes
- ...

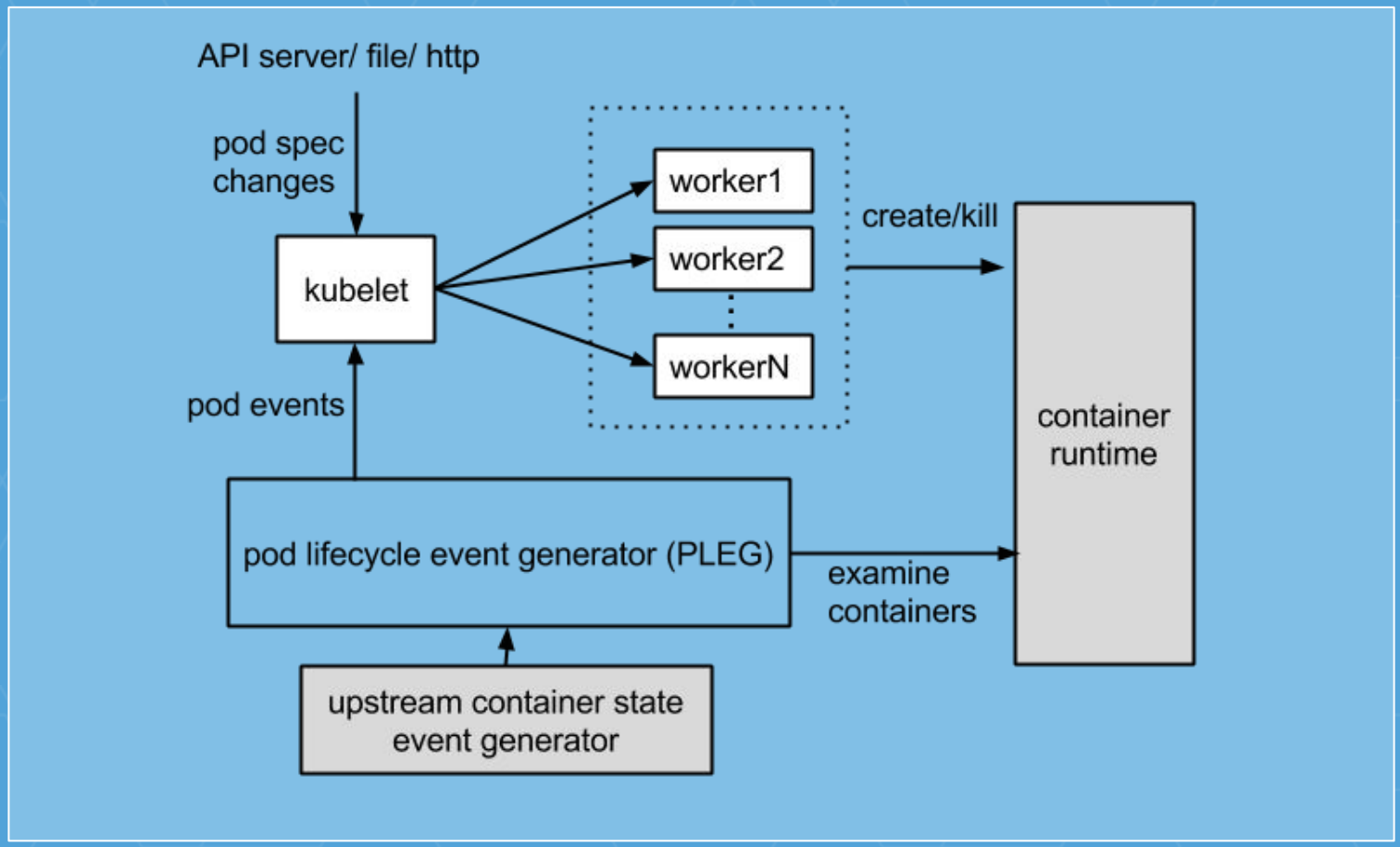
Kubernetes API

- Pod (Co-located containers)
- Replication Controller (HA & Resize)
- Service (Service discovery, Load balancer)

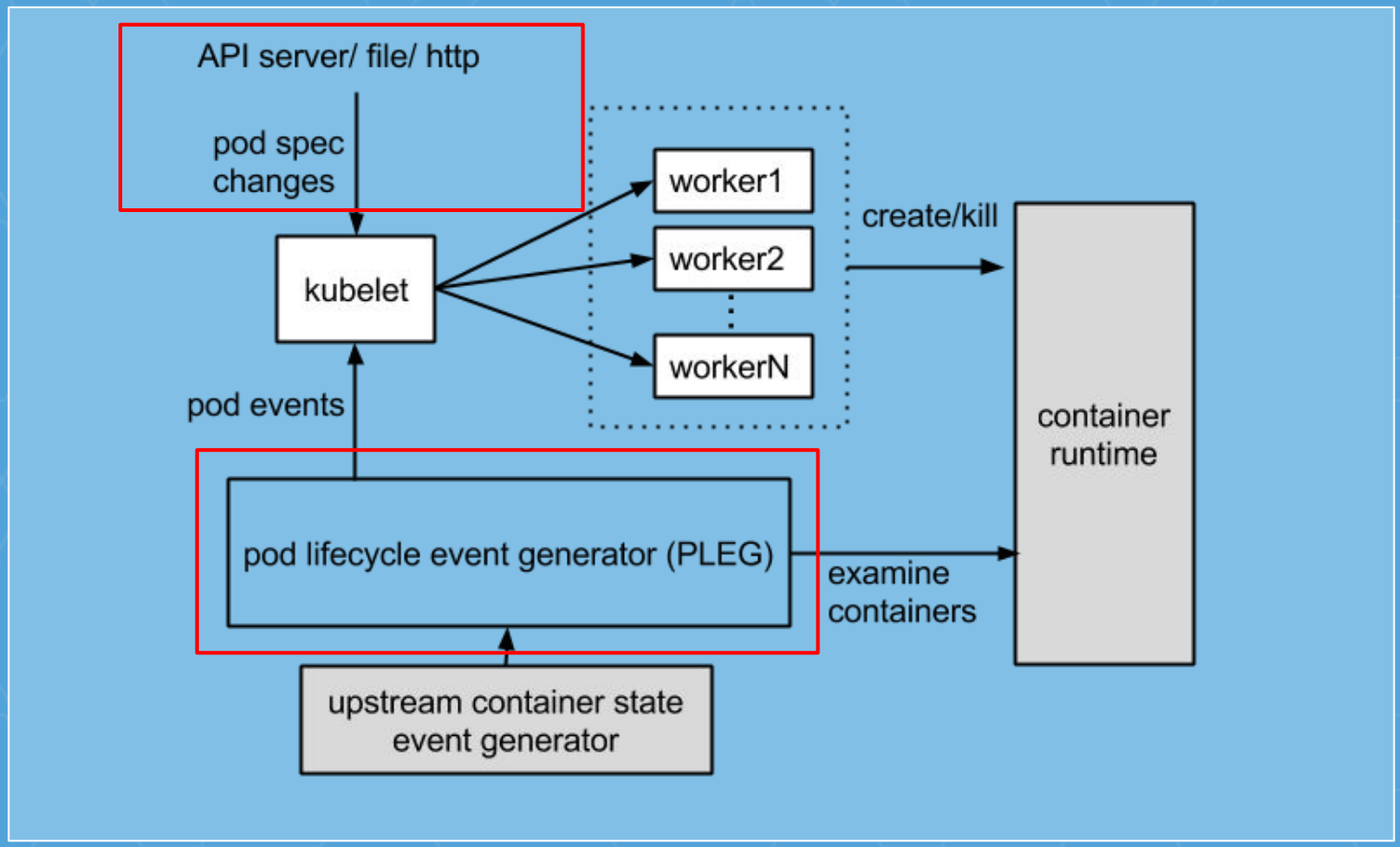
Kubelet


master
scheduler
job
replication controller
devops
API server
node
label
service
pod
label
pod
label
pod
label
service
user
proxy
pod
label
pod
label
pod
label
Kubelet
docker
registry

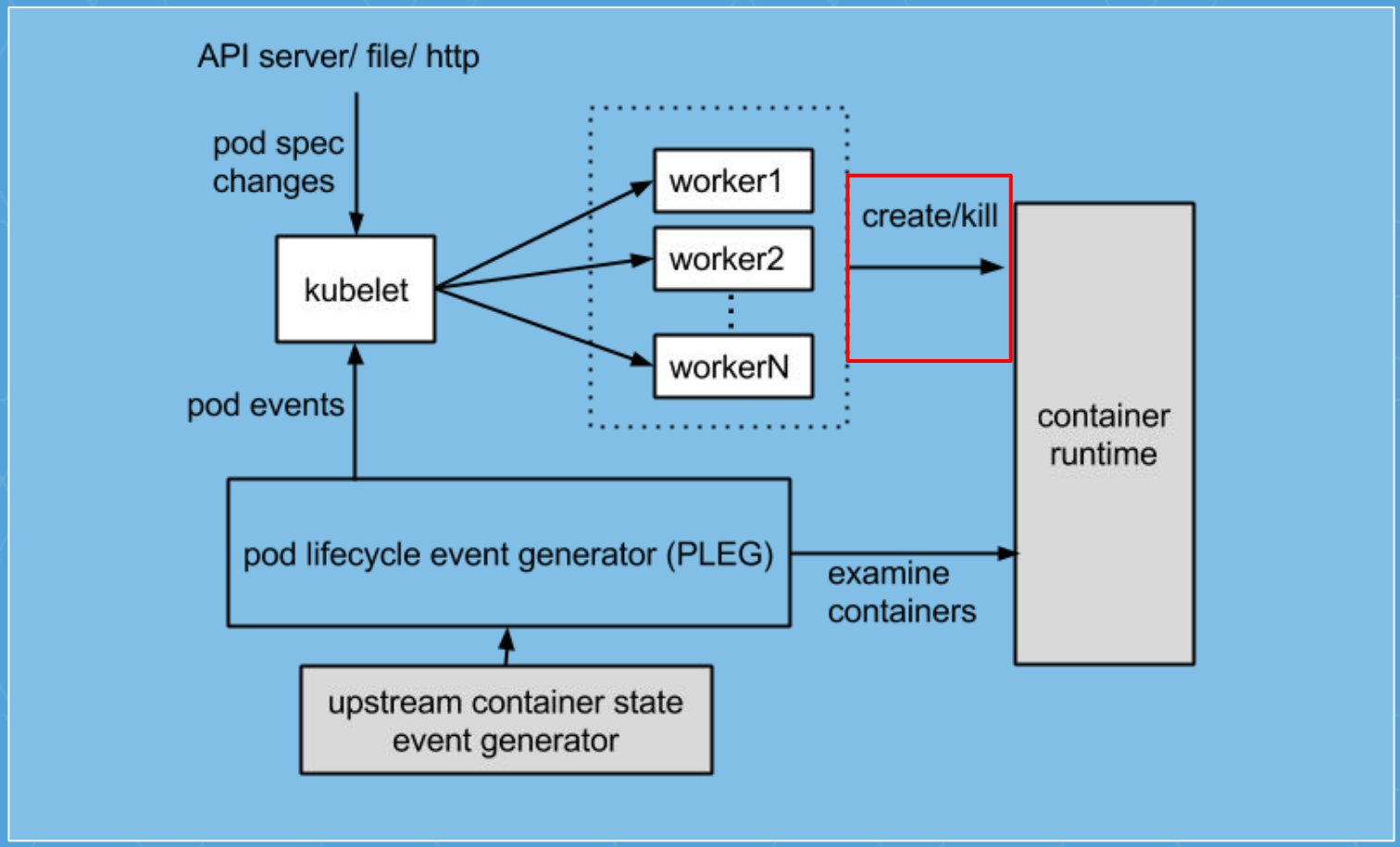
Kubelet Overview



Kubelet Overview



Kubelet Overview



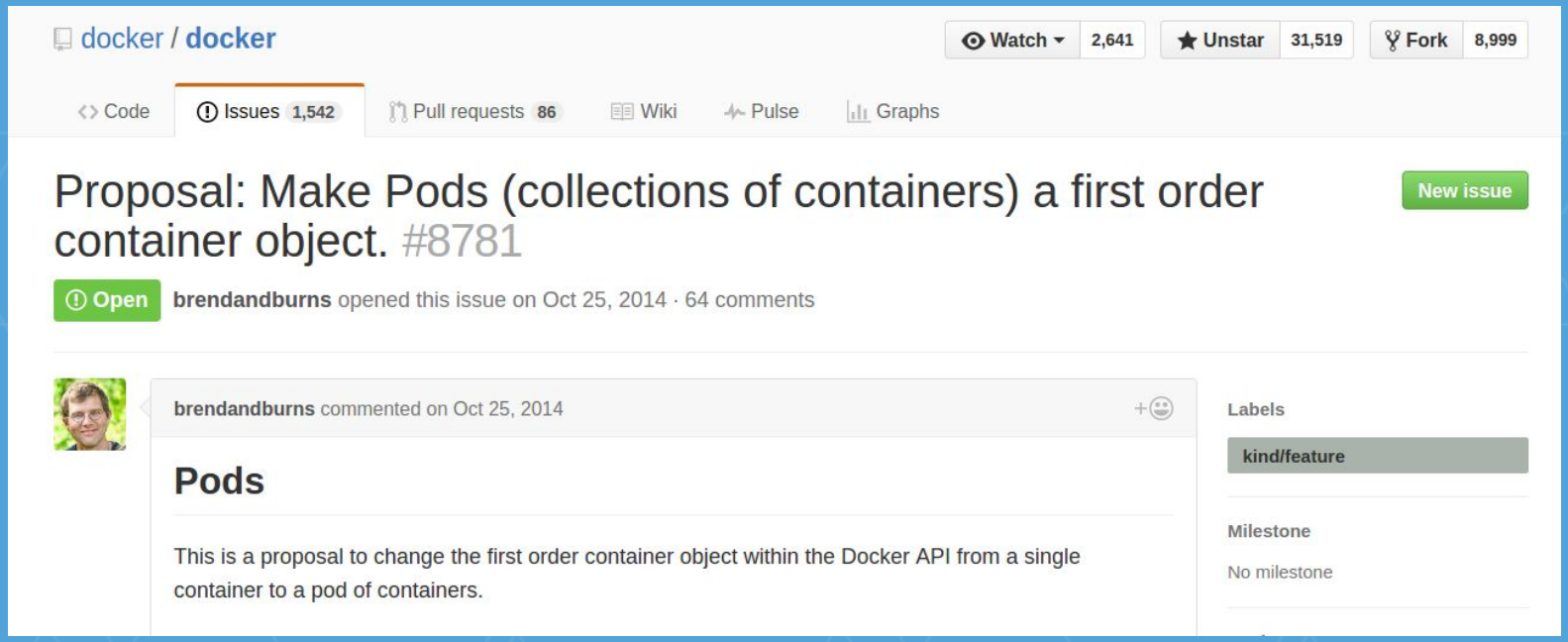
Kubelet -- Runtime Interface

Started from nothing but Docker
→ Deep-coupled with Docker

Kubelet -- Runtime Interface

Started from nothing but Docker
→ Deep-coupled with Docker
→ started rkt integration
→ Pod level runtime interface

Kubelet -- Runtime Interface



Kubelet -- Runtime Interface

- GetPods()
- SyncPod()
- KillPod()
- GetPodStatus()

- ListImages()
- PullImage()
- RemoveImage()
- ImageStats()

- GetContainerLogs()
- ExecInContainer()
- ...

Kubelet -- Runtime Interface

- GetPods()
- SyncPod(), declarative
- KillPod()
- GetPodStatus()

- ListImages()
- PullImage()
- RemoveImage()
- ImageStats()

- GetContainerLogs()
- ExecInContainer()
- ...

Kubelet -- Runtime Interface

```
func SyncPodIdeal (expectedPod, actualPod) {
    foreach container in actualPod {
        if container is not in expectedPod.Containers {
            KillContainer()
        }
    }
    foreach container in expectedPod {
        if container is not in actual.Containers {
            StartContainer()
        }
    }
}
```

Kubelet -- Runtime Interface

```
func SyncPodLessIdeal (expectedPod, actualPod) {
    foreach container in actualPod {
        Has the container spec changed?
        Is the container healthy?
        if container is not in expectedPod.Containers {
            KillContainer()
        }
    }
    foreach container in expectedPod {
        if container is not in actualPod.Containers {
            Does the container needs to restart?
            Is the container a pod infra container?
            StartContainer()
        }
    }
}
```

Kubelet -- Runtime Interface

```
func SyncPodRkt (expectedPod, actualPod) {
    foreach container in actualPod {
        Has the container spec changed?
        Is the container healthy?
        if container is not in expectedPod.Containers {
            goto restart
        }
    }
    foreach container in expectedPod {
        if container is not in actualPod.Containers {
            Does the container needs to restart?
            goto restart
        }
    }
restart: RestartPod()
}
```

Observation

Pod level interface

- Simpler
- Coarse-grained
- Not every runtime implements “Pod”

Container level interface

- More complexity in kubelet
- Doesn’t make sense to VM based runtime
- Fine-grained
- Runtime implementation can be easy

Future of Runtime Interface

- Improve extensibility: Easier container runtime integration.
- Improve feature velocity
- Improve code maintainability

Container level, Pod level, or both?
It's under debate!
https://github.com/kubernetes/kubernetes/pull/25899

Summary

- Container is the future, standard is important
- rkt is a composable, secure container runtime
- Kubernetes, kubelet and container runtime interface

CoreOS is running the world's containers

We're hiring: careers@coreos.com

OPEN SOURCE

90+ Projects on GitHub, 1,000+ Contributors

coreos.com

ENTERPRISE

Support plans, training and more

sales@coreos.com