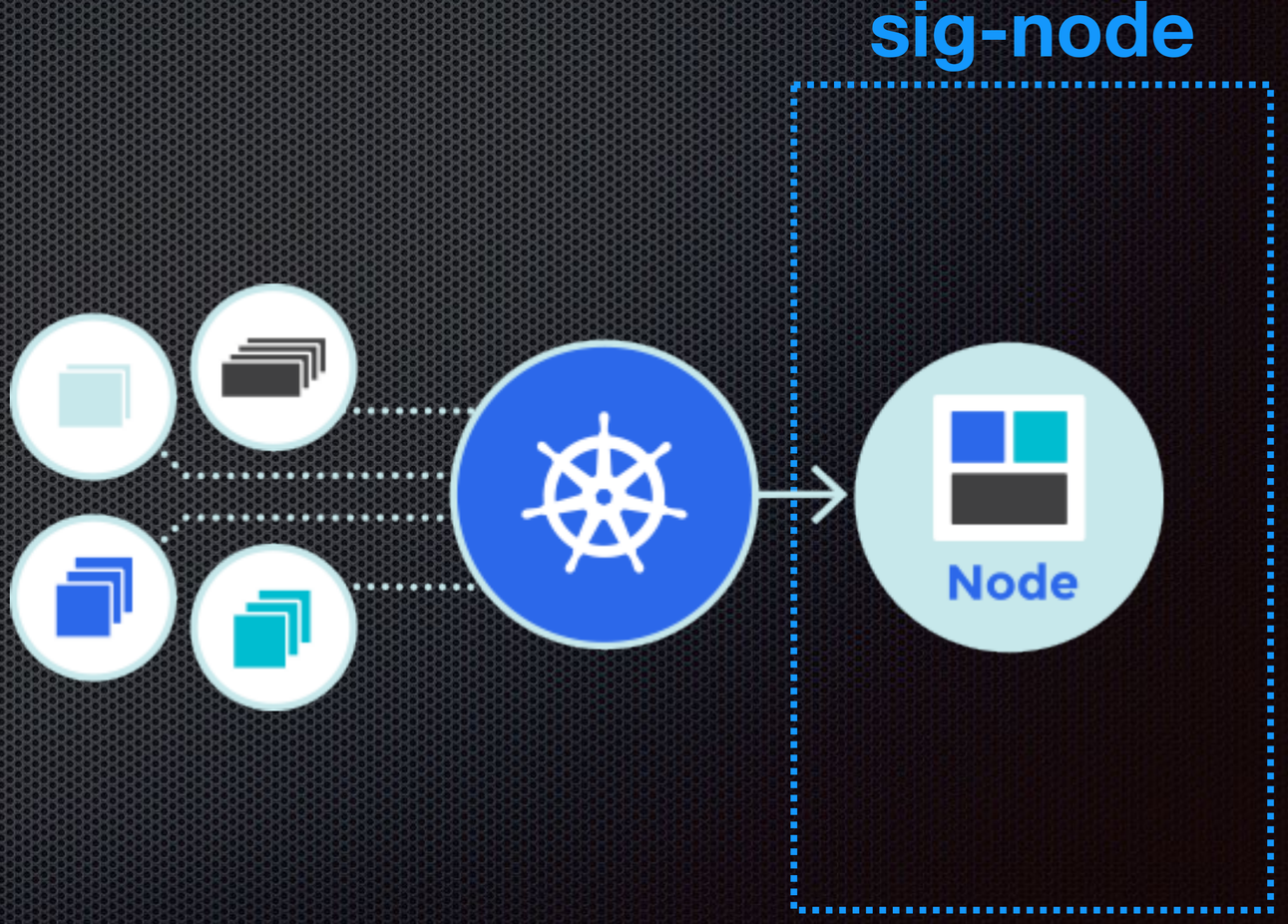
# Kubernetes “Node”

## Past, now, and the future

by Harry Zhang @resouer

# What is “Node”?

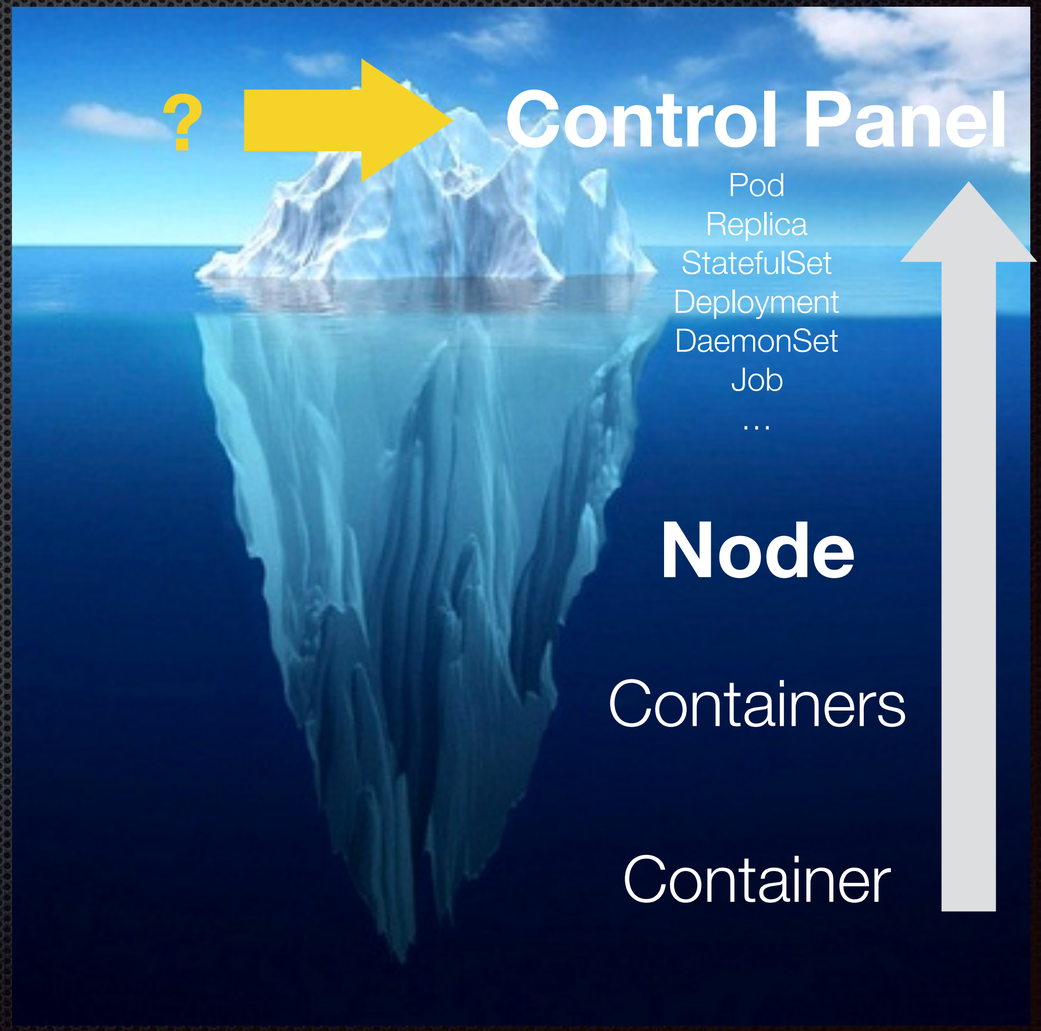
- Where the
  - **container** lives
  - Kubernetes cluster is **bootstrapped**
  - **container centric** features are implemented
  - **docker/rkt/runV/runC** is plugged in
  - **networking** is implemented
  - **volume** is enabled



# Why “Node” Matters?

- Kubernetes
  - a **bottom-up** design of container cloud
  - with special **bonus** from Google …

How Kubernetes is created?



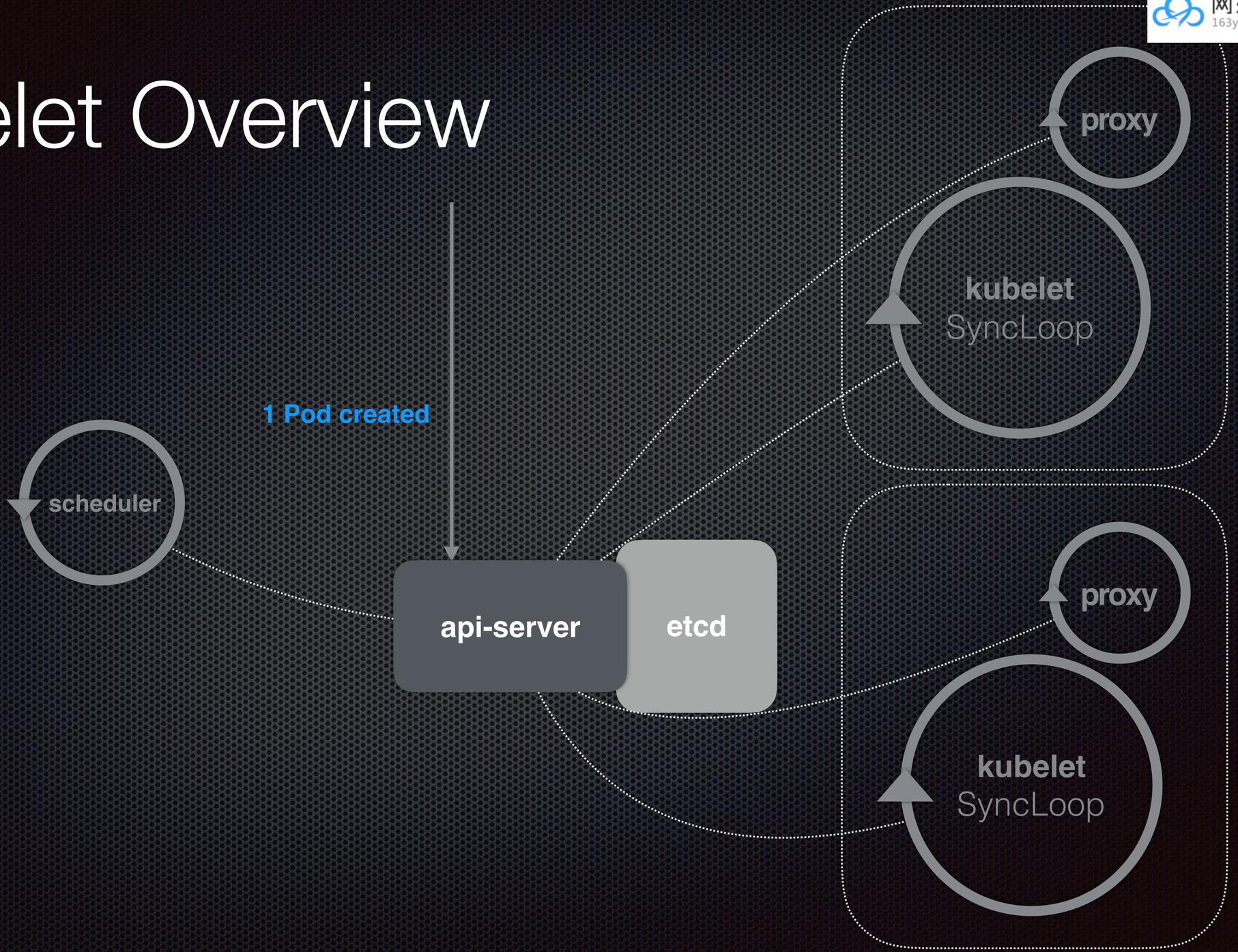
# Borg

- Borg = engineer oriented deployment scheduling & management system
- Google internal is massively using cgroups container, not **Container** :)
- **Kubernetes** = re-innovate **Borg** with **Container**

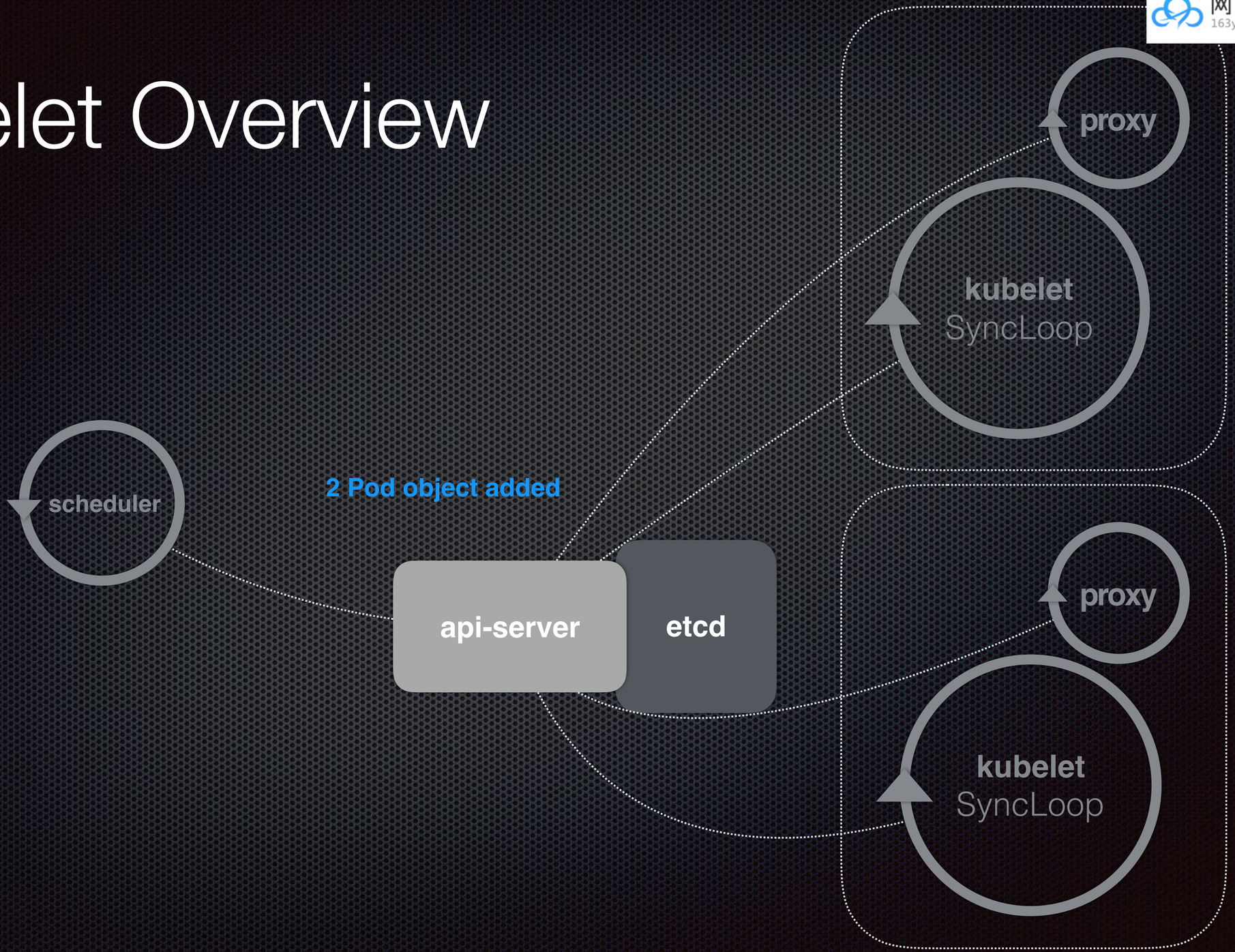
# Node

## Unsung hero to bridge Borg control panel with container

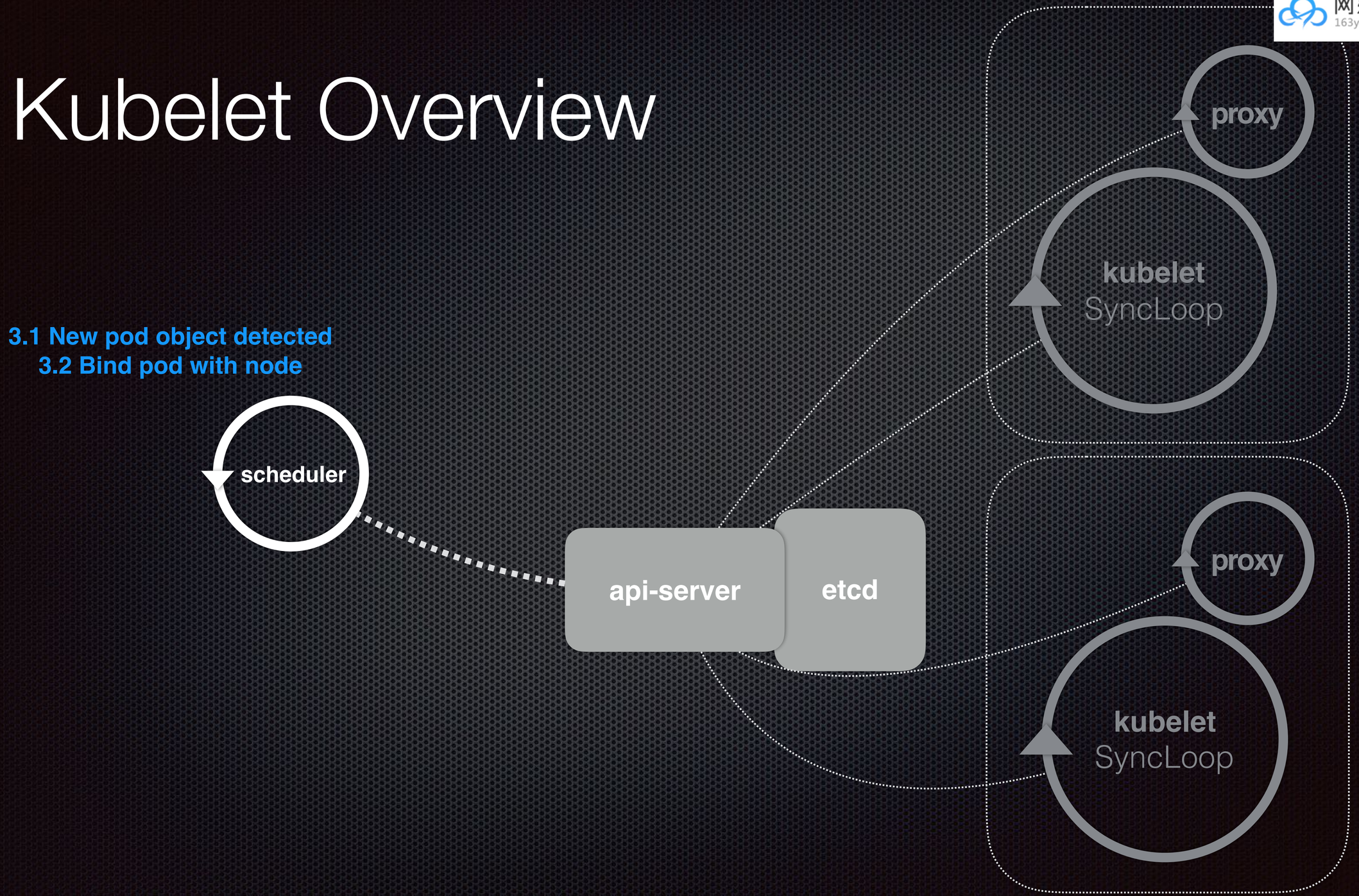
# Kubelet Overview



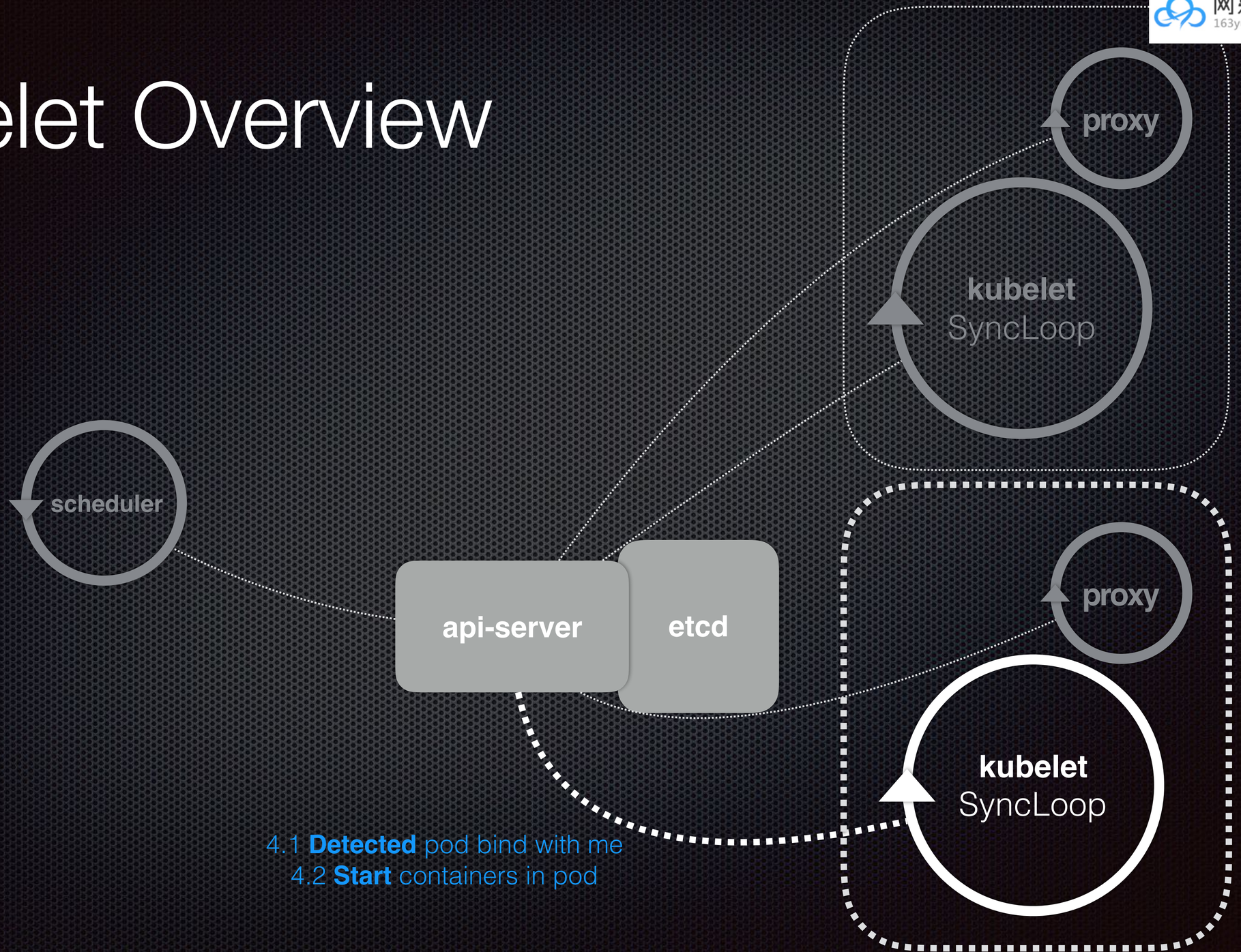
# Kubelet Overview



# Kubelet Overview



# Kubelet Overview



# Pod

- “Alloc” in Borg
- The atomic scheduling unit in Kubernetes
- Process group in container cloud
- Implemented in Node
- **But why?**

# Are You Using Container Like This?

1.use supervised/systemd to manage multi-apps in one container

2.ensure container order by tricky scripts

3.add health check for micro-service group

4.copy files from one container to another

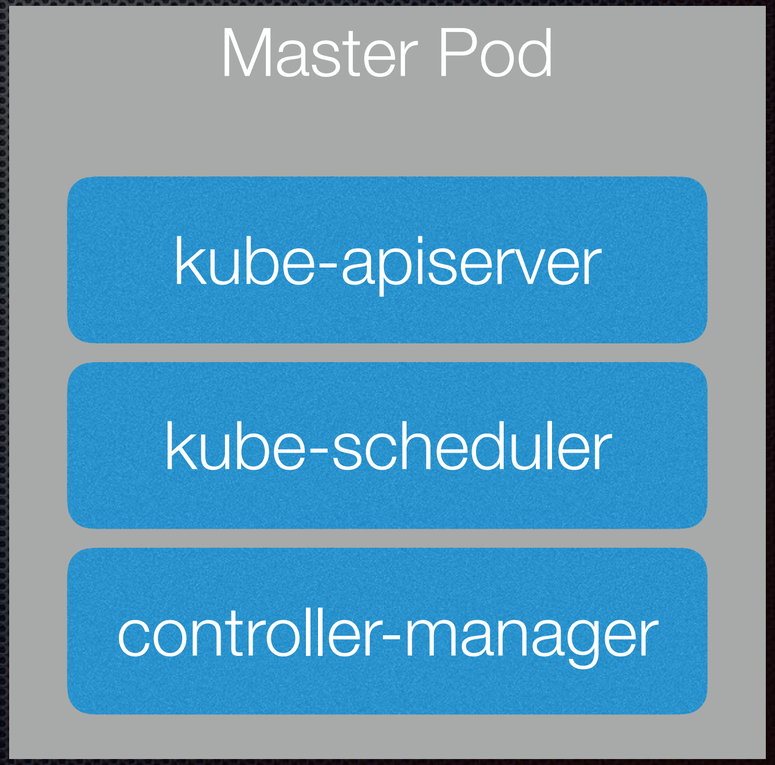
5.connect to peer container across whole network stack

6.schedule super affinity containers in cluster

# Multiple Apps in One Container

- Multiple containers

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
```

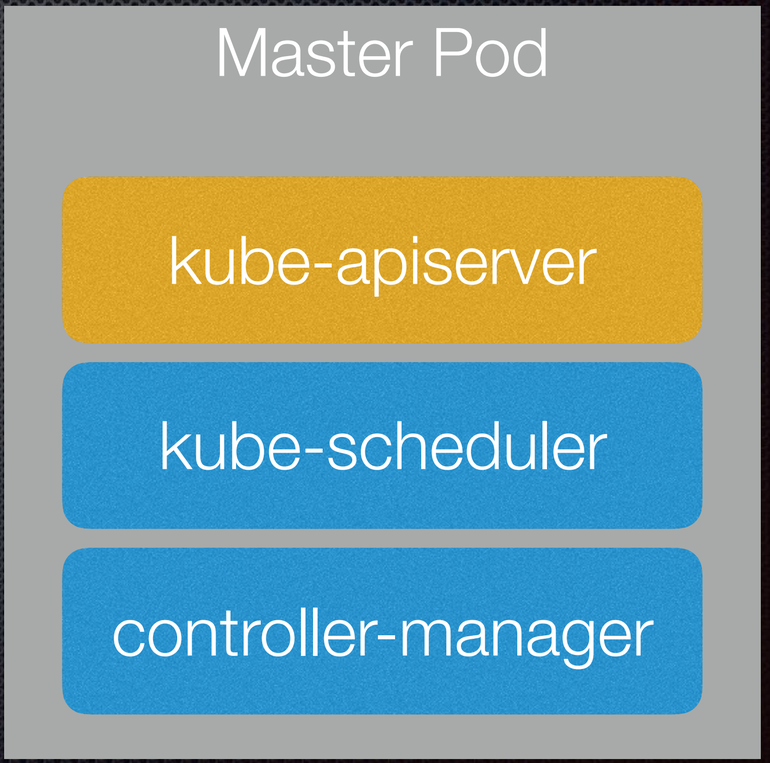



日志看不到
是否running没法判断
运维操作困难
出错定位麻烦，不知道是哪个挂了，频繁登陆容器

# Ensure Container Order

- InitContainer

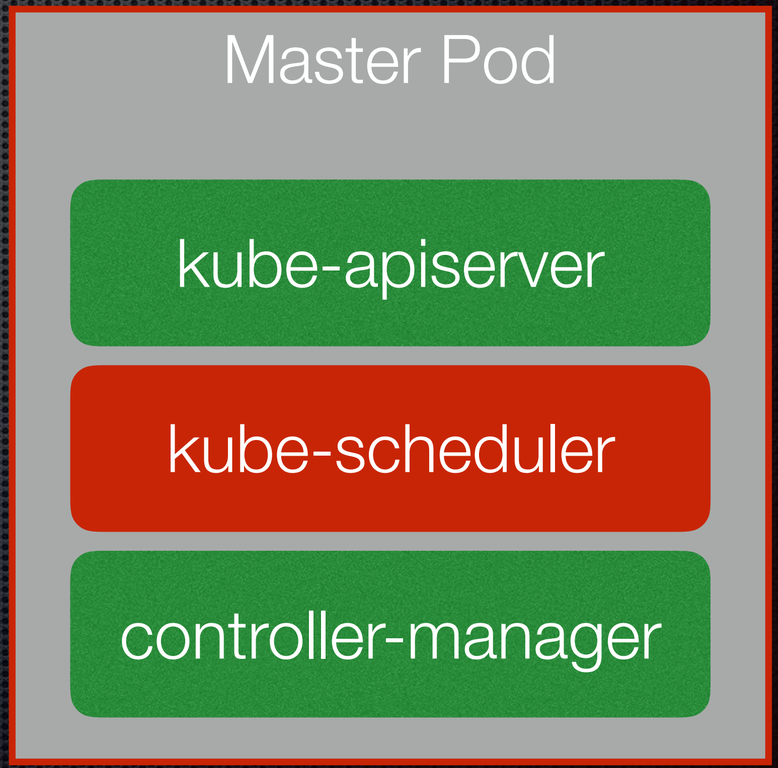
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
  annotations:
    pod.beta.kubernetes.io/init-containers: '[
        {
            "name": "kube-apiserver",
            "image": "kube-apiserver:v1.6.0",
        }
    ]'
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
```



# Health Check for Containers

- Liveness probe
  - Pod will be reported as Unhealthy

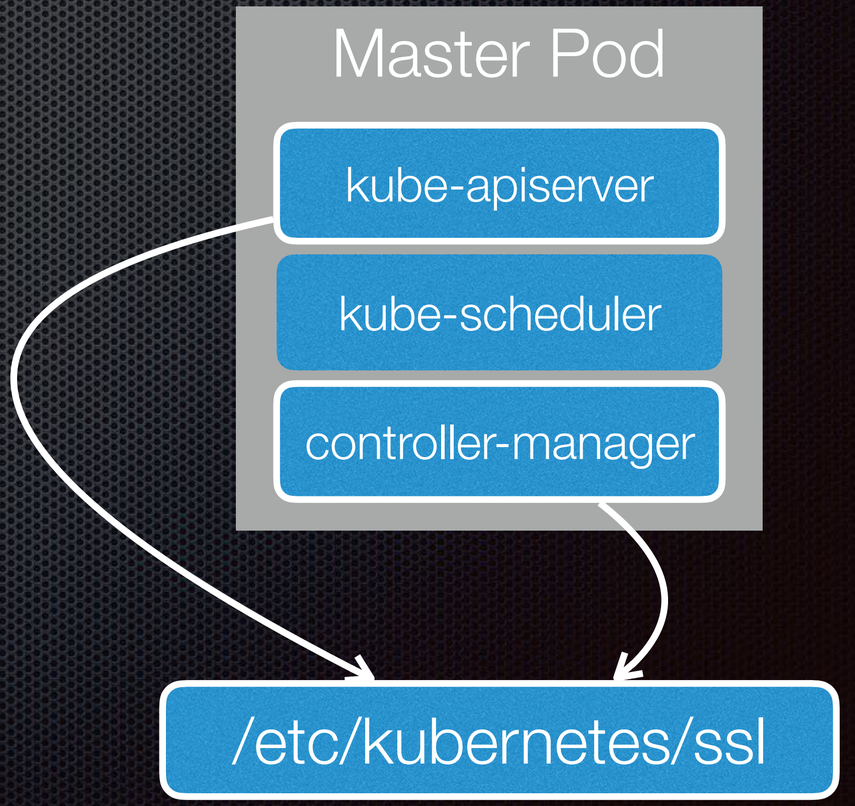
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10251
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        port: 10252
        path: /healthz
      initialDelaySeconds: 15
      timeoutSeconds: 15
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        port: 8080
        path: /healthz
      initialDelaySeconds: 15
      timeoutSeconds: 15
```



# Copy Files from One to Another

- Pod volumes is shared

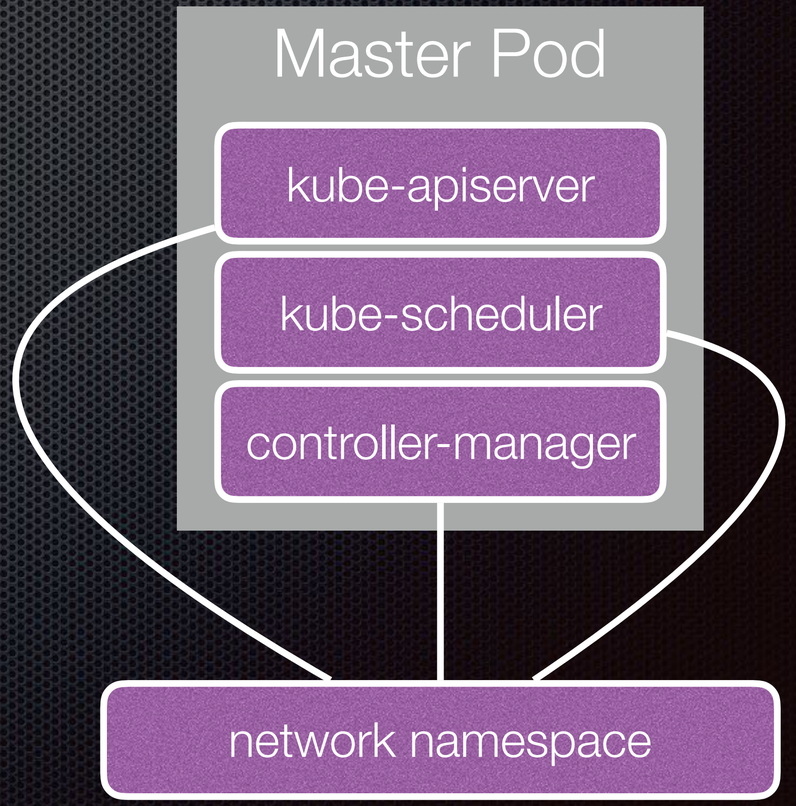
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
    volumeMounts:
    - mountPath: /etc/kubernetes/ssl
      name: ssl-certs-kubernetes
      readOnly: true
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
    volumeMounts:
      - mountPath: /etc/kubernetes/ssl
        name: ssl-certs-kubernetes
        readOnly: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/ssl
    name: ssl-certs-kubernetes
```



# Connect to Peer Container

- Pod network is shared

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
    command:
    - apiserver
    - --bind-address=0.0.0.0
    - --insecure-port=8080
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
    command:
    - scheduler
    - --master=http://127.0.0.1:8080
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
    command:
    - controller-manager
    - --master=http://127.0.0.1:8080
```



# Schedule Super Affinity Containers

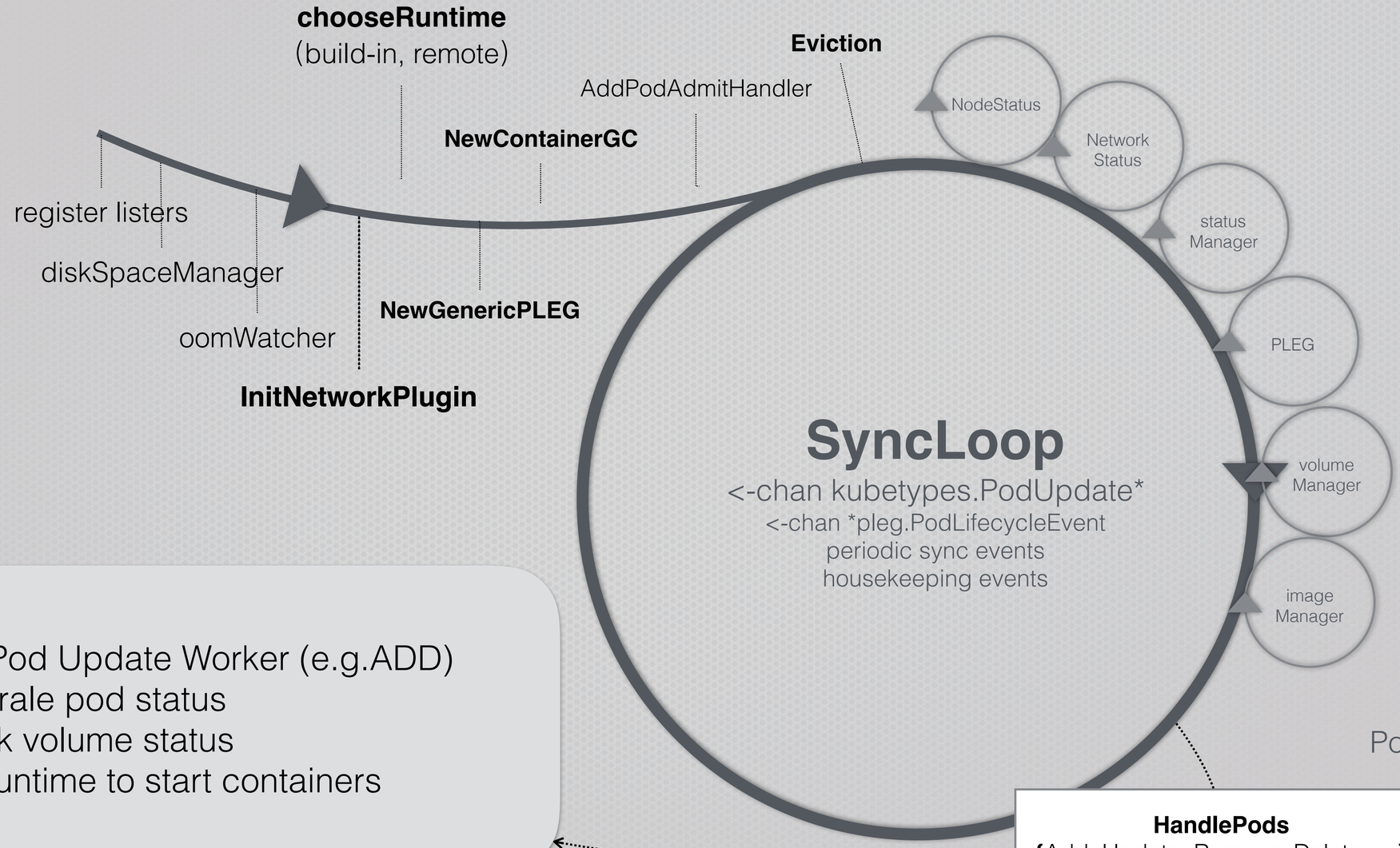
- Pod is atomic scheduling unit

- **controller** super affinity **apiserver**
- Request:
  - **controller**: 1G, **apiserver**: 0.5G
- Available:
  - **Node_A**: 1.25G, **Node_B**: 2G
- What happens if **controller** is scheduled to **Node_A** first?

# So, this is Pod

- **Design pattern** in container world
  - decoupling
  - reuse & refactoring
  - describe more complex workload by container
    - e.g. ML

# kubelet




***4-sources**
api-server (primary, watch)
http endpoint (pull)
http server (push)
file (pull)

Pod Update Worker (e.g.ADD)

- generale pod status
- check volume status
- call runtime to start containers

# Prepare Volume

- Get `mountedVolume` from `actualStateOfWorld`
- Unmount volumes in `mountedVolume` but not in `desiredStateOfWorld`
- `AttachVolume()` if vol in `desiredStateOfWorld` and not attached
- `MountVolume()` if vol in `desiredStateOfWorld` and not in `mountedVolume`
- Verify devices that should be detached/unmounted are detached/unmounted

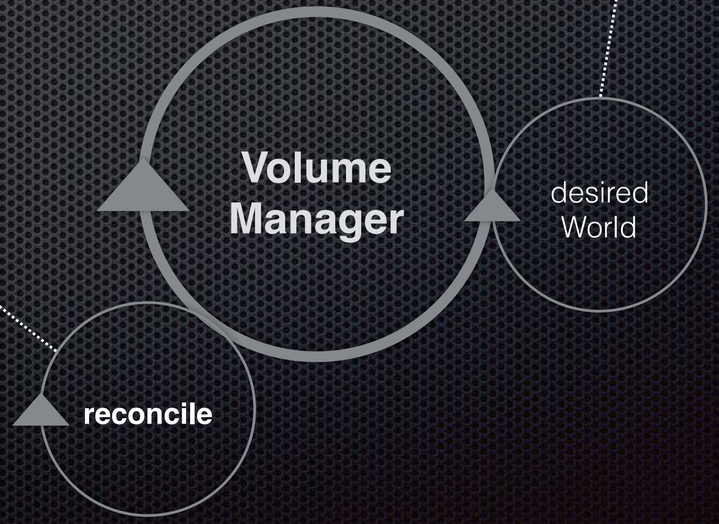
- Find new pods
- `createVolumeSpec(newPod)`
- Cache `volumes[volName].pods[podName] = pod`

- Tips:

**1. -v host:path**

2. attach VS mount

3. Totally **independent** from container management

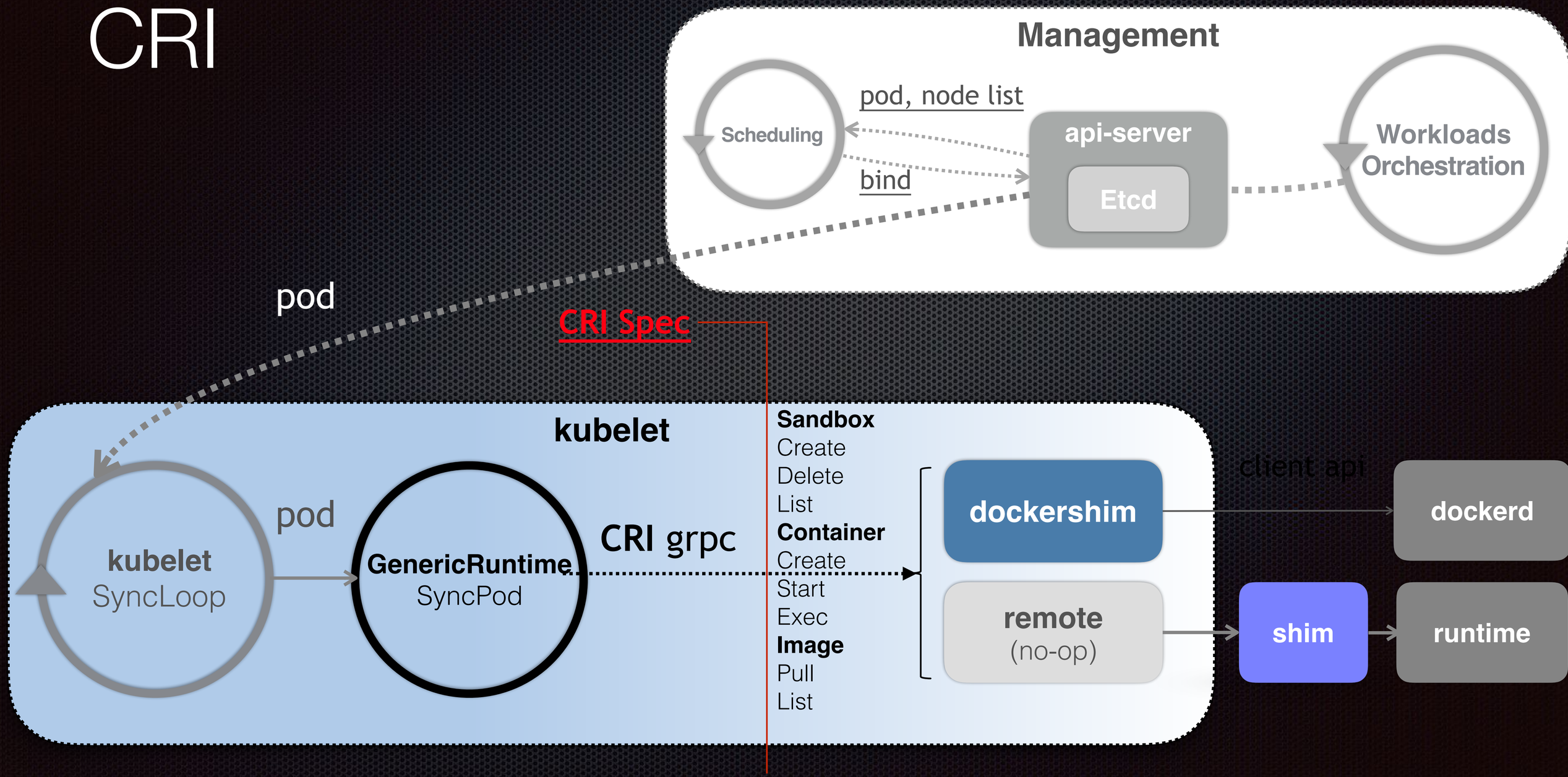


# Eviction

- **Guaranteed**
  - Be killed **until they exceed their limits**
    - or if the system is under memory pressure and there are no lower priority containers that can be killed.
- **Burstable**
  - killed once they **exceed** their requests and **no Best-Effort pods exist** when system under memory pressure
- **Best-Effort**
  - **First to get killed** if the system runs out of memory

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

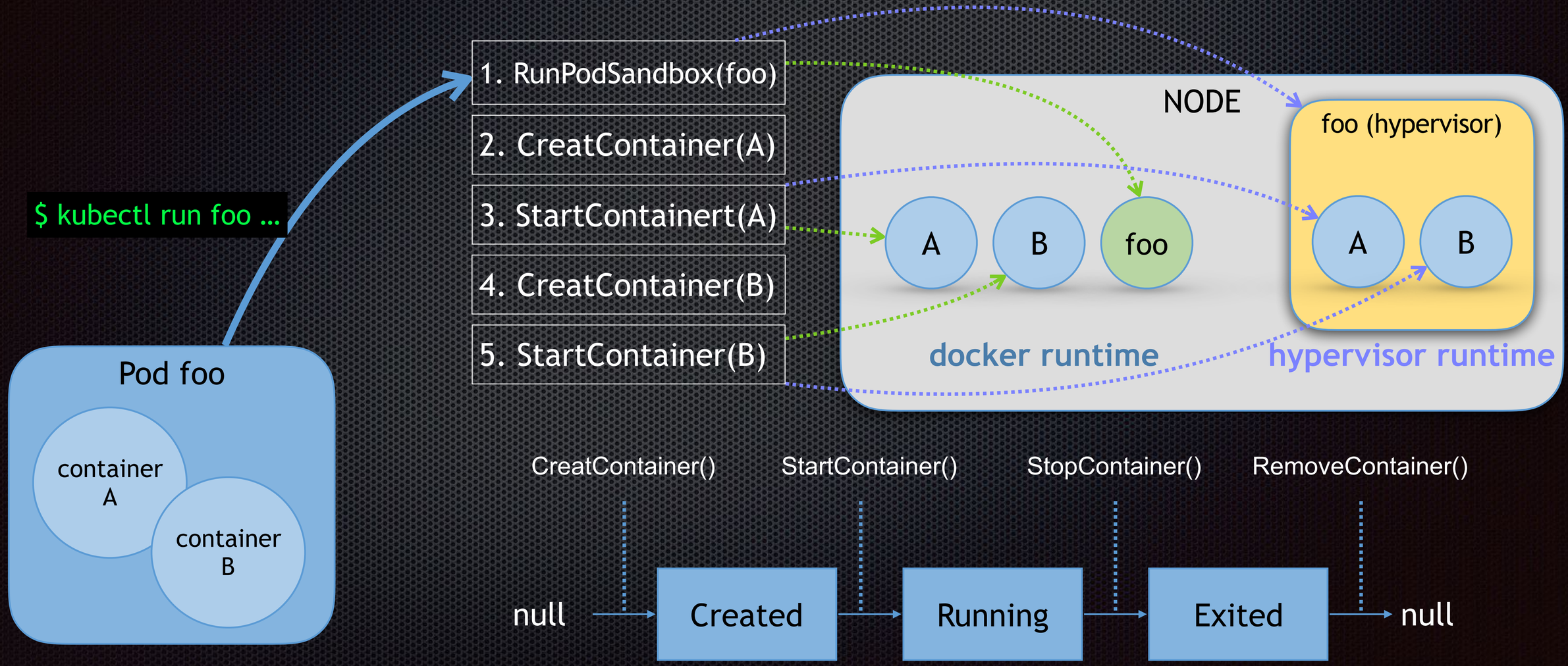
# CRI


Management
Scheduling
pod, node list
bind
api-server
Etcd
Workloads
Orchestration
pod
CRI Spec
kubelet
kubelet
SyncLoop
pod
GenericRuntime
SyncPod
CRI grpc
Sandbox
Create
Delete
List
Container
Create
Start
Exec
Image
Pull
List
dockershim
remote
(no-op)
client api
dockerd
shim
runtime

# CRI Runtime Shim

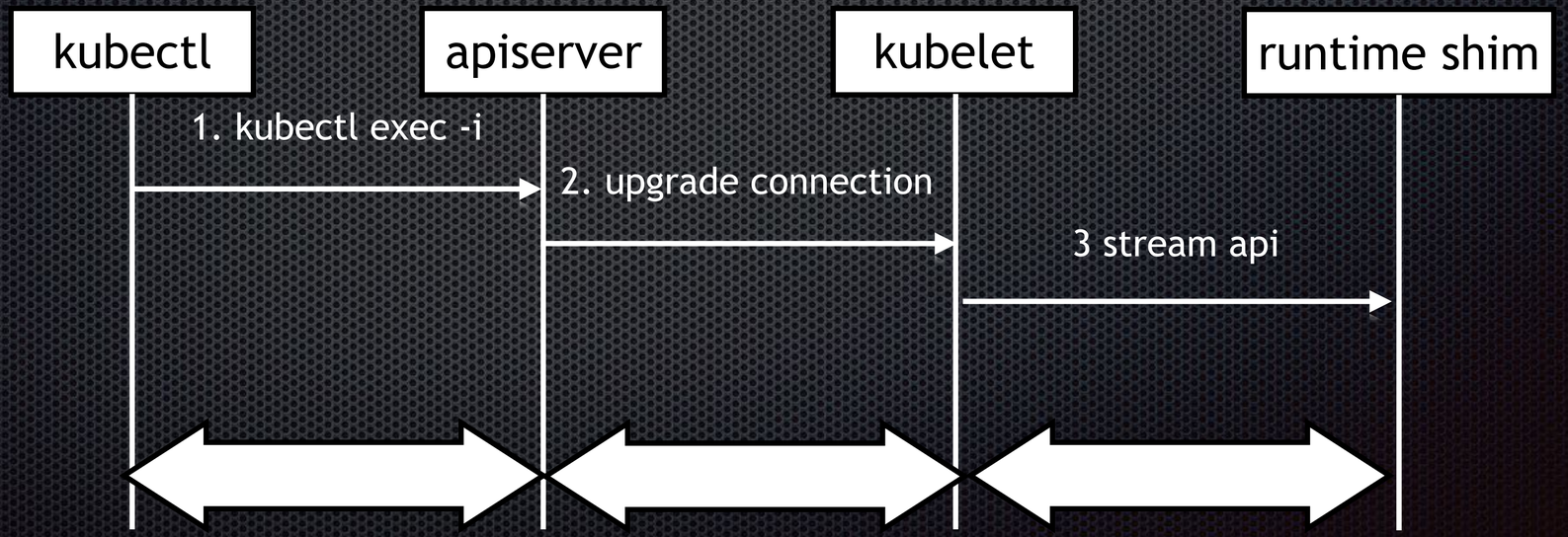
- **dockershim**: docker
- **frakti**: hypervisor container (runV)
- **cri-o**: runC
- **rktlet**: rkt
- **cri-containerd**: containerd

# Container Lifecycle



# Streaming (old version)

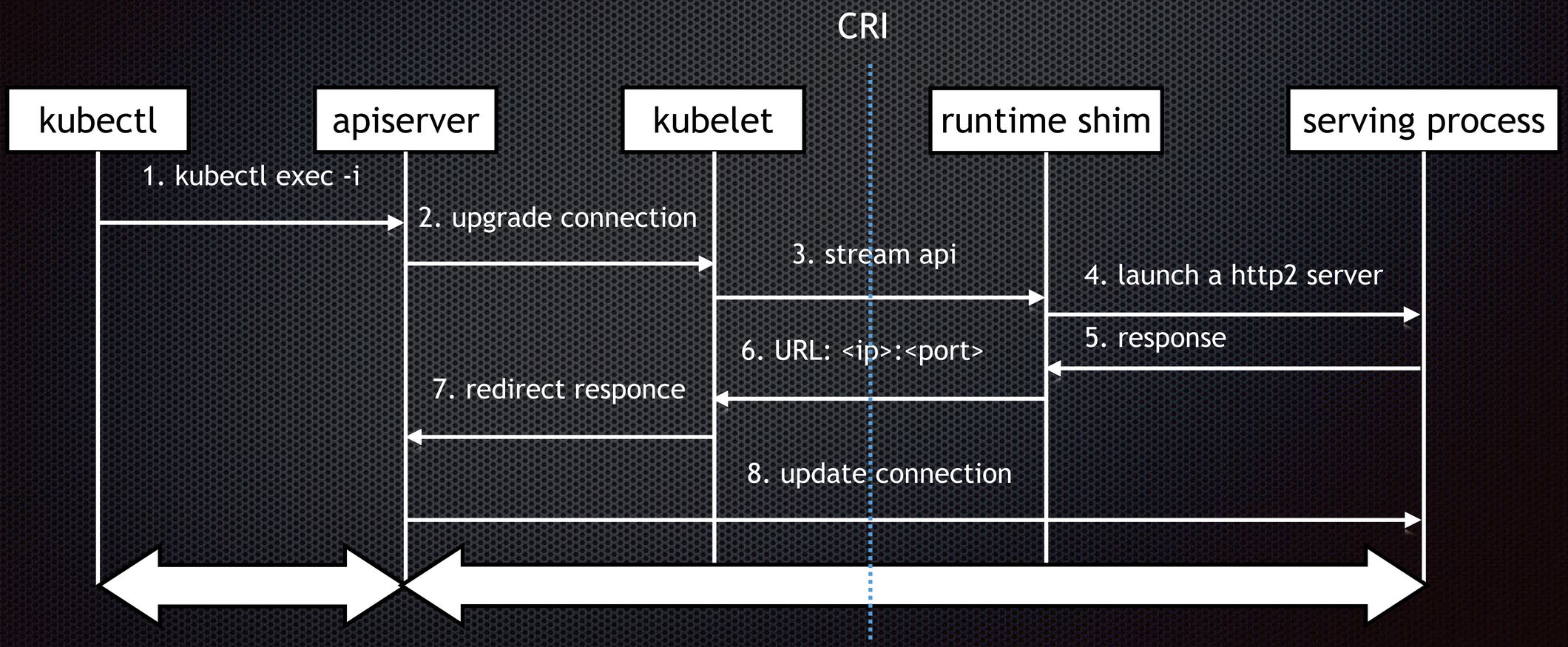
- kubelet becomes bottleneck
- runtime shim in critical path
- code duplication among runtimes/shims



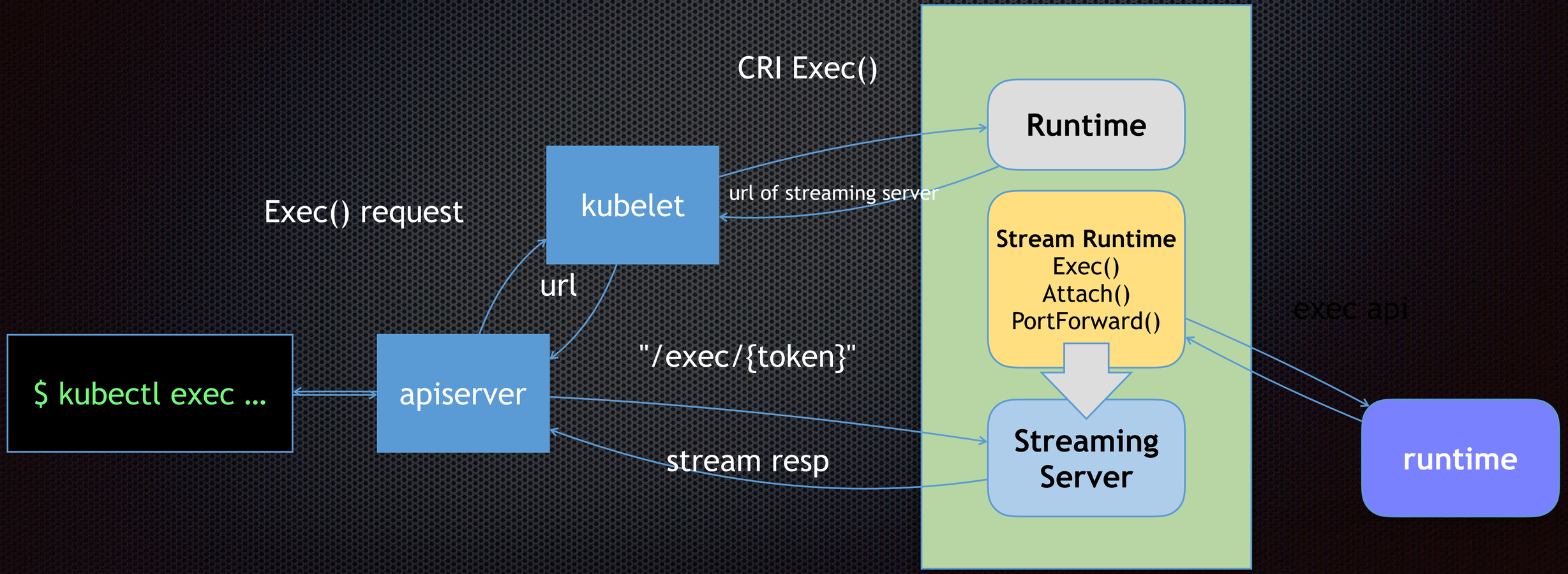

Design Doc

# Streaming (CRI version)

Design Doc

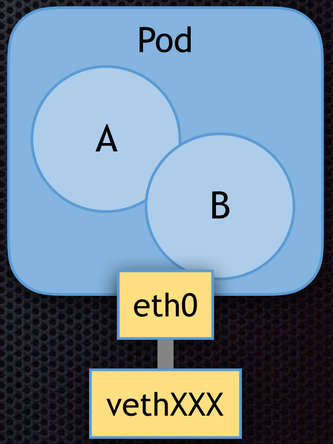


# Streaming in Runtime Shim



# CNI Network in Runtime Shim

- Workflow in runtime shim (may vary from different runtimes):
  1. Create a network NS for sandbox
  2. **plugin.SetUpPod(NS, podID)** to configure this NS
  3. Also checkpoint the NS path for future usage (TearDown)
  4. **Infra container join this network namespace**
     1. **or** scanning **/etc/cni/net.d/xxx.conf** to configure sandbox

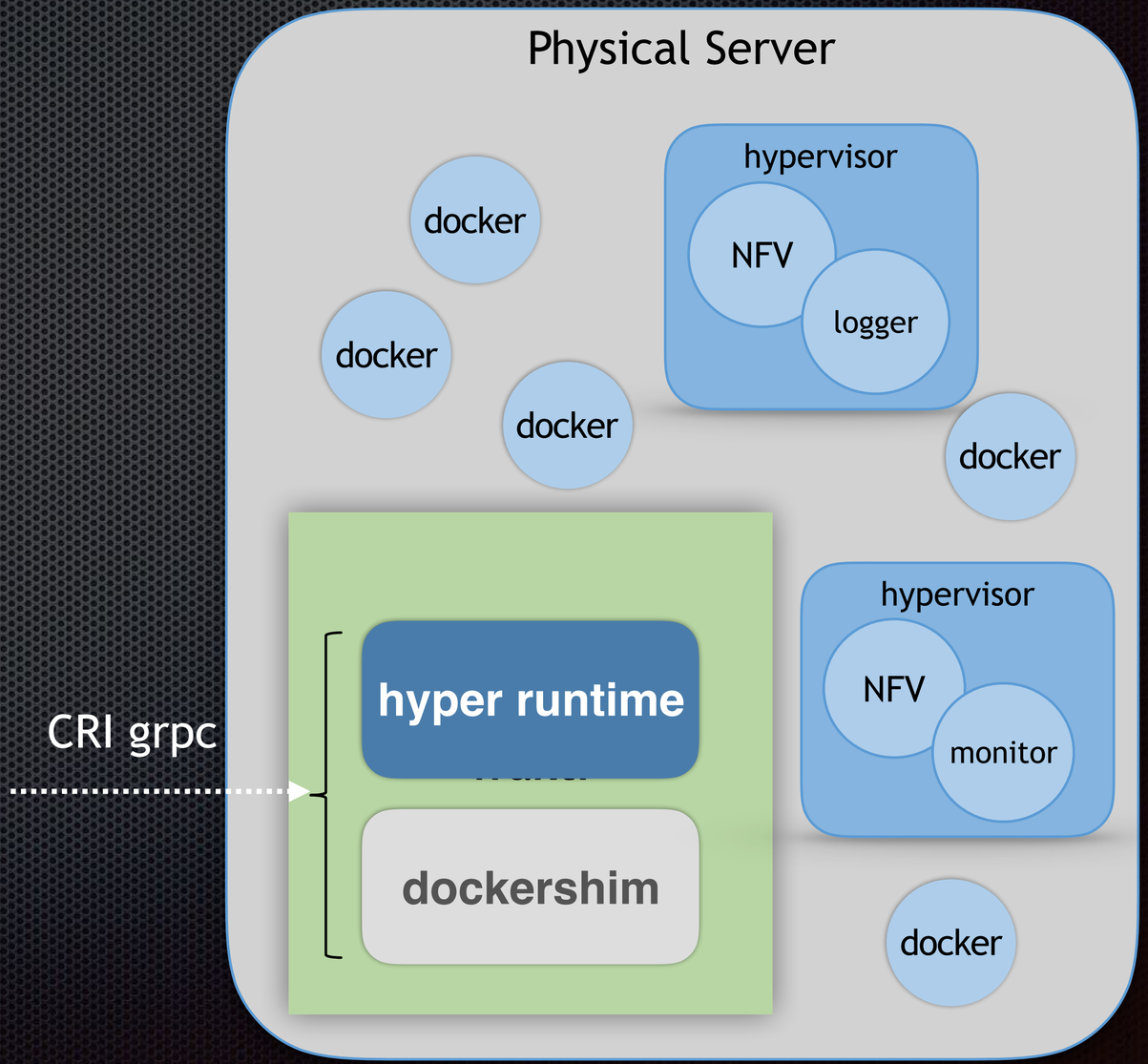



```
type NetworkInfo struct {
    IfName      string
    Mac         net.HardwareAddr
    Ip          *net.IPNet
    Gateway     string
    BridgeName string
}
```

# Mixed Runtimes: IaaS-less Kubernetes

- Hypervisor container + Docker
- Handled by:
  - https://github.com/kubernetes/frakti/

1. Share the same CNI network
2. Fast **responsiveness** (no VM host needed)
3. High resource **efficiency** (k8s QoS classes)
4. Mixed run micro-services & **legacy application**
   1. independent kernel + hardware virtualization
   2. high I/O performance + host namespace



# Node & Kubernetes is Moving Fast

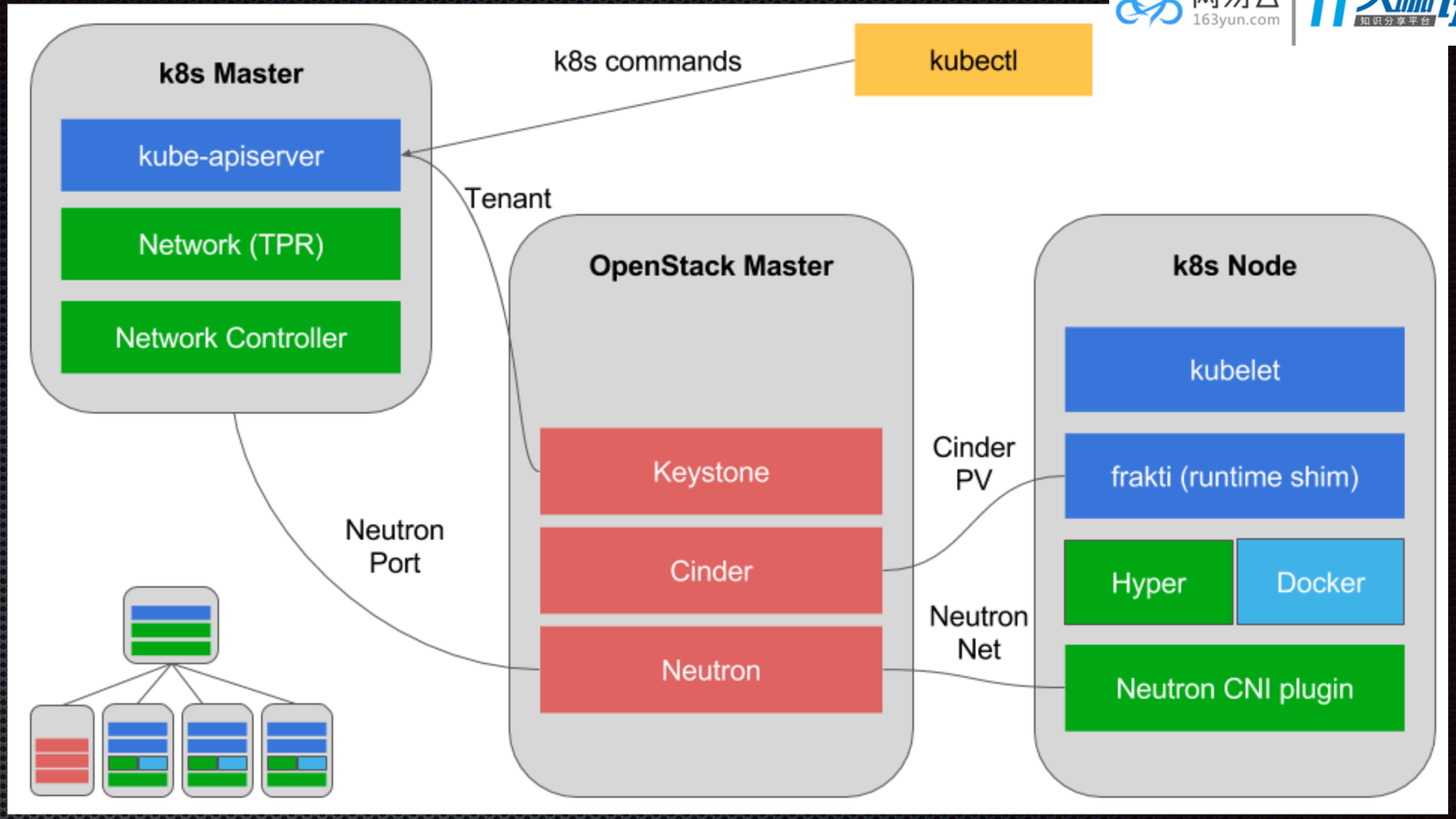
- GPU isolation
  - libnvidia-container is proposed
- CRI enhancement
  - cri-containerd (promising default), cri-tools, hypervisor based secure container
- CPU pin (and update) and NUMA affinity (CPU sensitive workloads)
- HugePages support for large memory workloads
- Local storage management (disk, blkio, quota)
- **“G on G”**: run Google internal workloads on Google Kubernetes

# Recently

- **Kubernetes**
  - CRI enhancement, equivalence class scheduler (Borg), NodeController, StatefulSet
  - **https://github.com/kubernetes/frakti** (Secure container runtime in k8s)
- **Mentoring**
  - cri-containerd, cri-tools
  - Unikernels & LinuxKit + k8s (Google Summer of Code 2017)
  - ovn-kubernetes (coming soon)
- Newly started: **Stackube**

# Stackube

Milestone: 2017.9




- **https://github.com/openstack/stackube** (Hypernetes v2)
  - 100% upstream Kubernetes + OpenStack plugins + Mixed Runtime
  - A IaaS-less, multi-tenant, secure and production ready Kubernetes distro