K8S 技术社区 | IT大咖说 知识分享平台

KubeCon Europe 2017 Recap

王后明 2017-04-22

目录

Cloud Native Landscape

v0.9.4

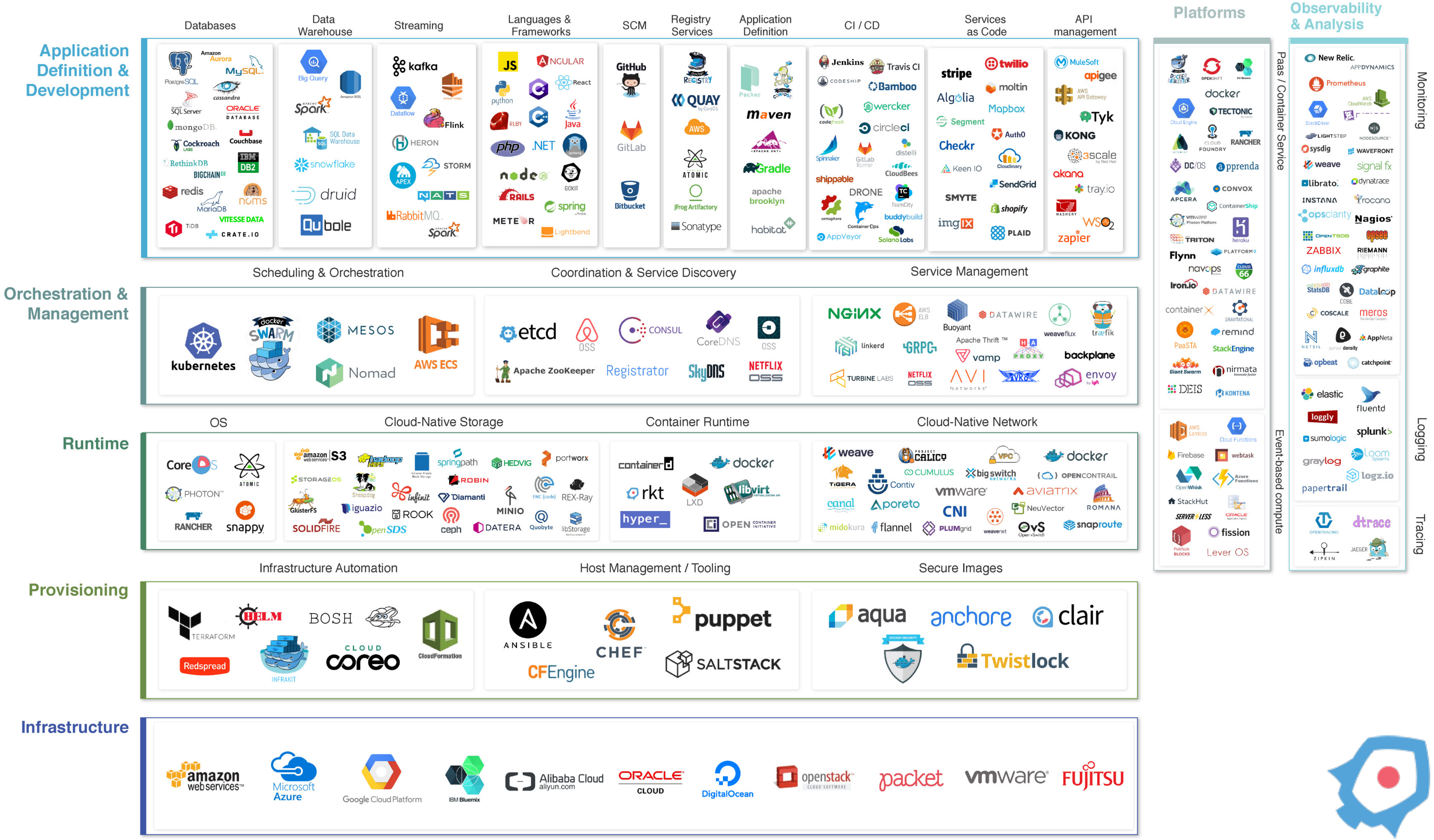
CLOUD NATIVE COMPUTING FOUNDATION

Redpoint

Amplify PARTNERS

K8S 技术社区

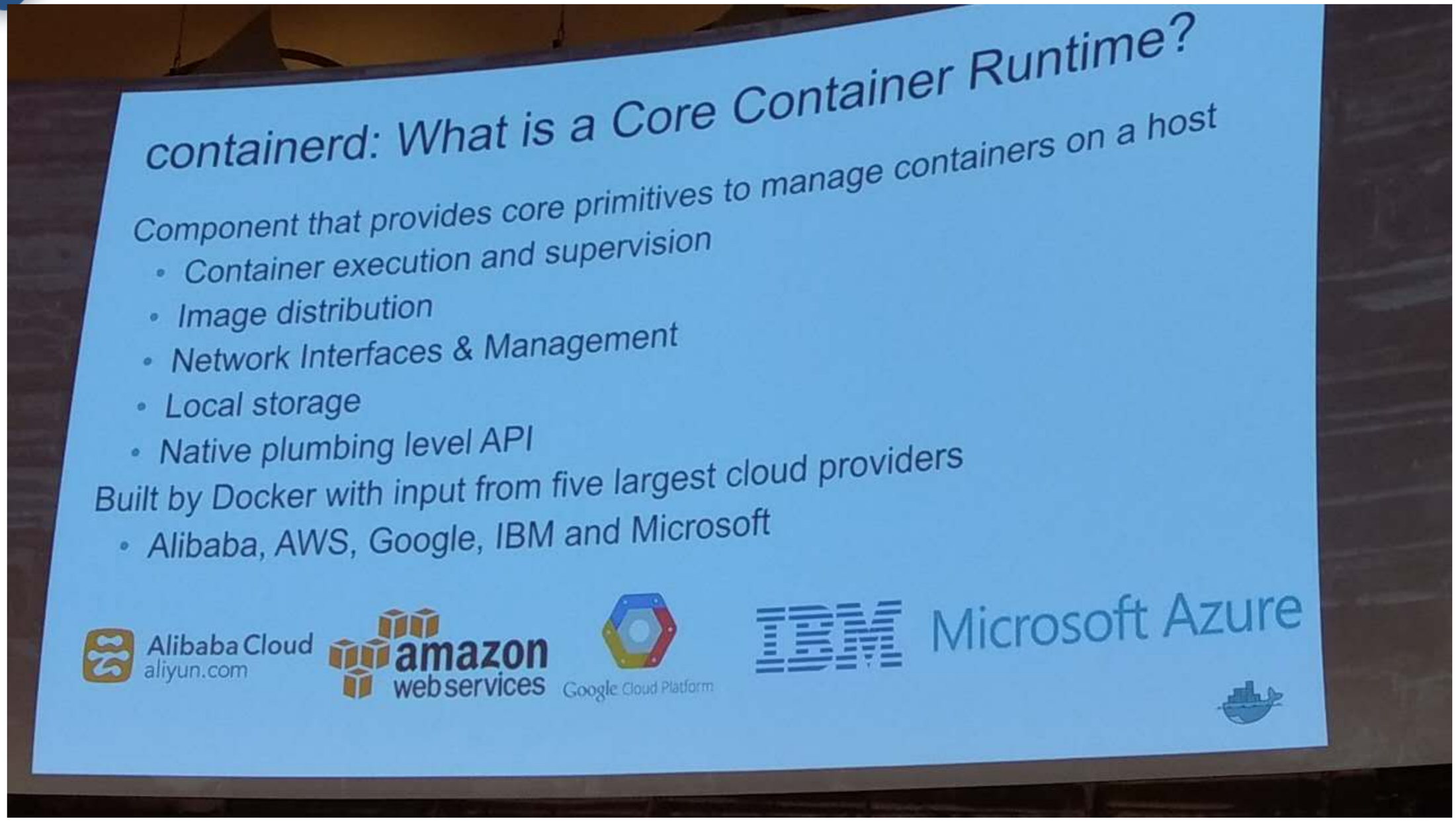
IT大咖说 知识分享平台



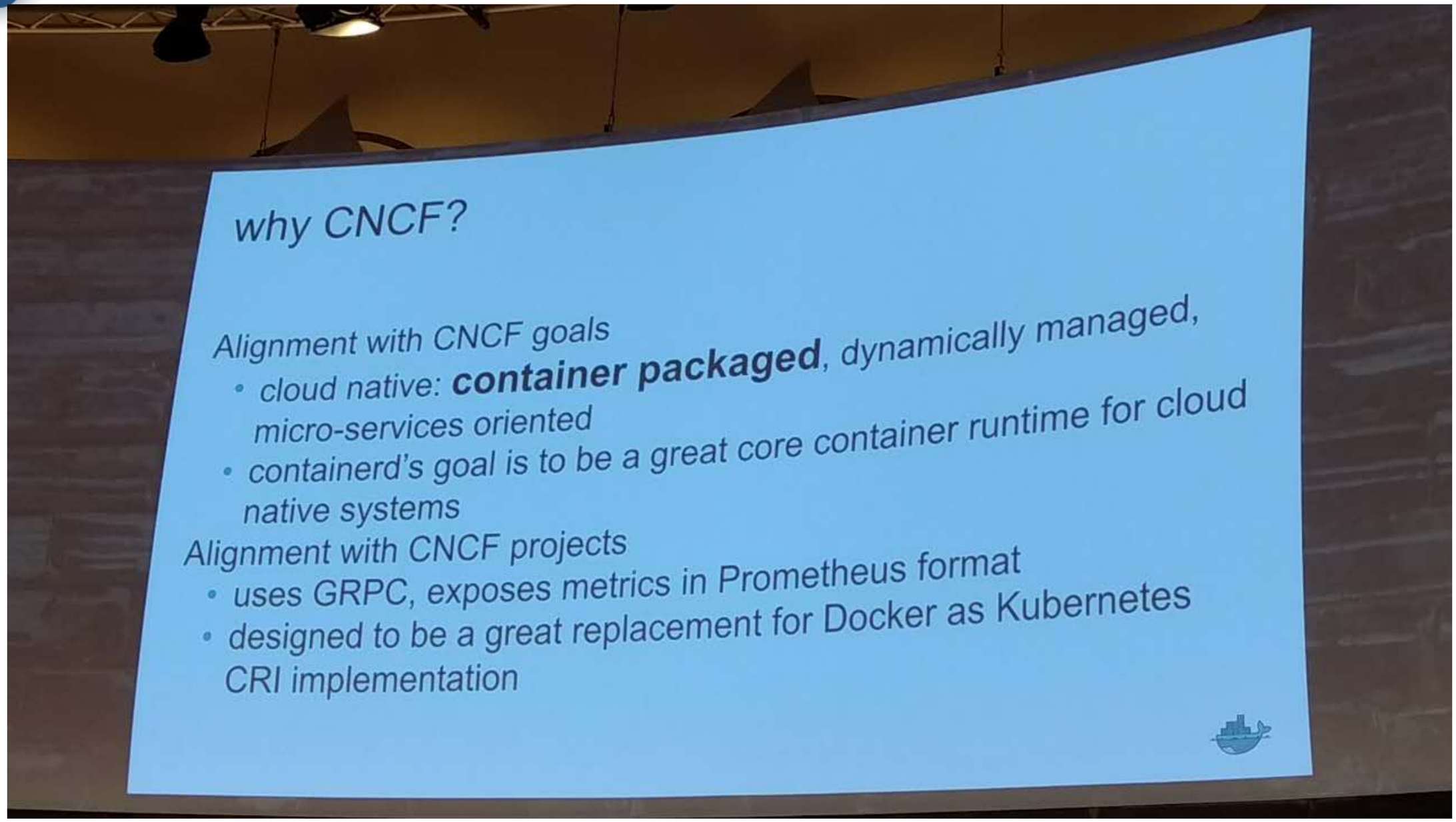
http://github.com/cncf/landscape

@dankohn1 @lennypruss @sraney

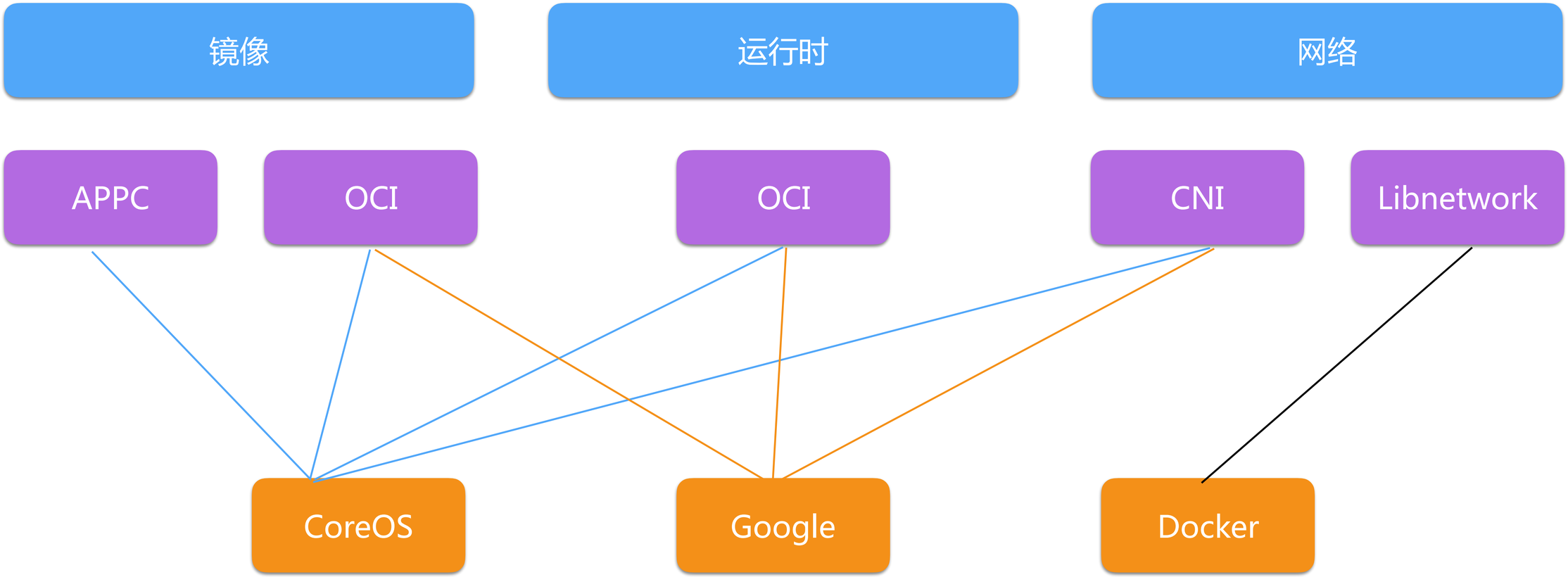
containerd: what is?



containerd: why CNCF ?

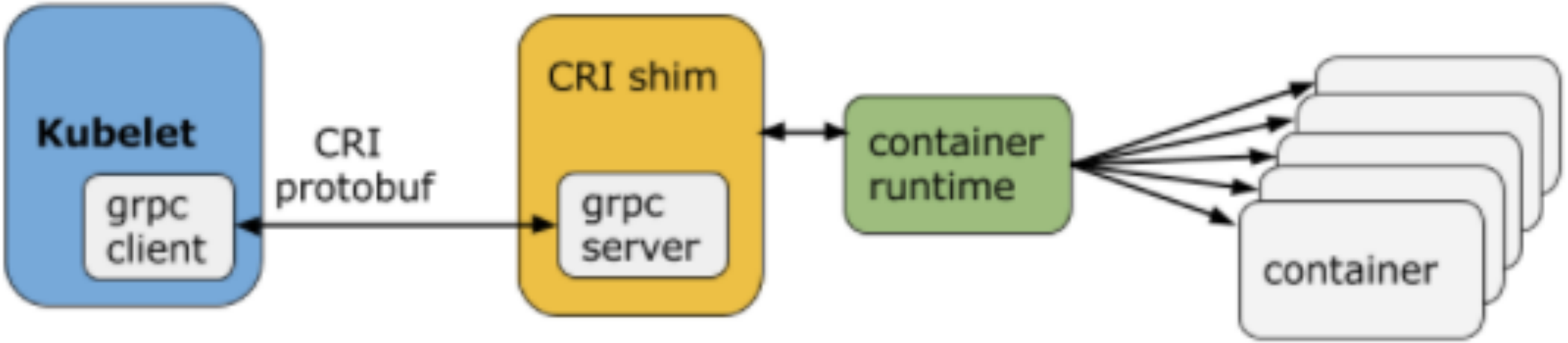


容器标准



CRI-O — OCI容器运行时实现、让K8S占据容器生态的中心地位

CRI（Container Runtime Interface，Kubernetes容器运行时接口，由kubelet调用，负责容器的生命周期管理）。
OCI（Open Container Initiative，推动容器镜像格式和运行时的标准化组织，由CoreOS、Google和Docker共同发起；目前拥有镜像规范、运行时规范，以及runc运行时软件）。
CRI-O当前为Kubernetes孵化项目，处于Alpha版本阶段。

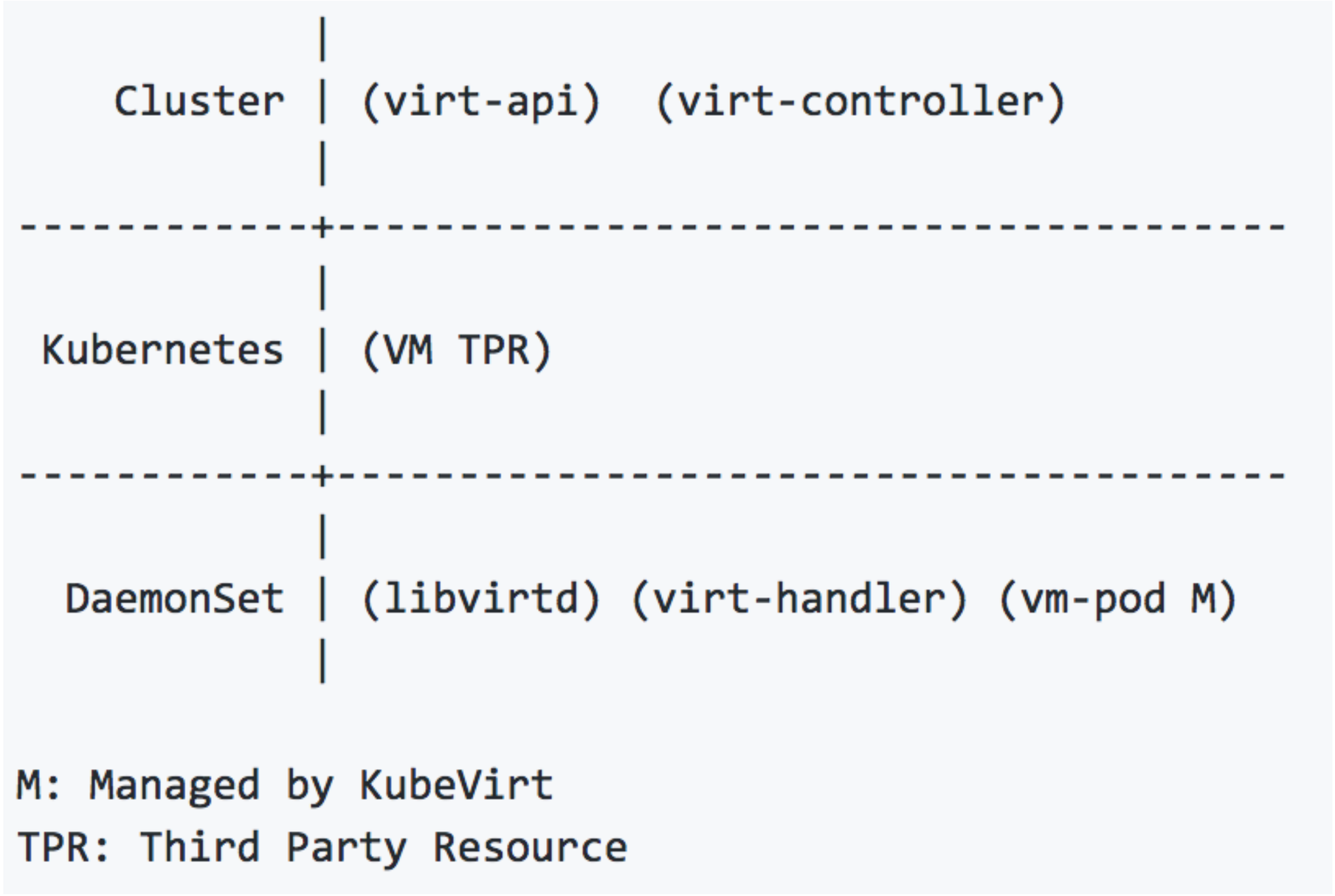


containerd: why CNCF ?

选择把containerd捐赠给 CNCF 基金会，主要原因在于两者的目标一致，并且 containerd 的技术实现也跟 CNCF 其它项目保持一致，能很快融入当前生态。Kubernetes 从1.5版本对Docker版本的支持由1.10.3升级到了1.12.3，这意味着 containerd 已经成为 Kubernetes 的关键部分。技术实现方面，containerd通过本地 UNIX Socket 暴露 gRPC API给上层系统调用，并且以 Prometheus 的数据格式对外提供监控数据。同时，containerd 也完全兼容 OCI 标准以及 runC实现并将通过 OCI 认证。

从1.6版本开始，Kubernetes 已经默认启用了 CRI 机制，并且默认使用 Docker-CRI 实现避免了 kubelet 直接调用 Docker API ，减轻了 Kubernetes 对 Docker 的直接依赖，Kubernetes CRI 对 containerd 的支持也已经有 POC 代码（https://github.com/kubernetes/kubernetes/pull/43655），可以看出，Docker 引擎将会逐渐淡出 Kubernetes 的视野，containerd 将会通过 CRI 接口成为 Kubernetes的默认容器引擎和生命周期管理工具。

KubeVirt - Kubernetes 作为容器和虚拟机统一控制平面



容器和虚拟机统一控制平面的其它方案

除了 KubeVirt 之外，当前也有另外几种虚拟机和容器统一调度的实现方案：

1. virtlet

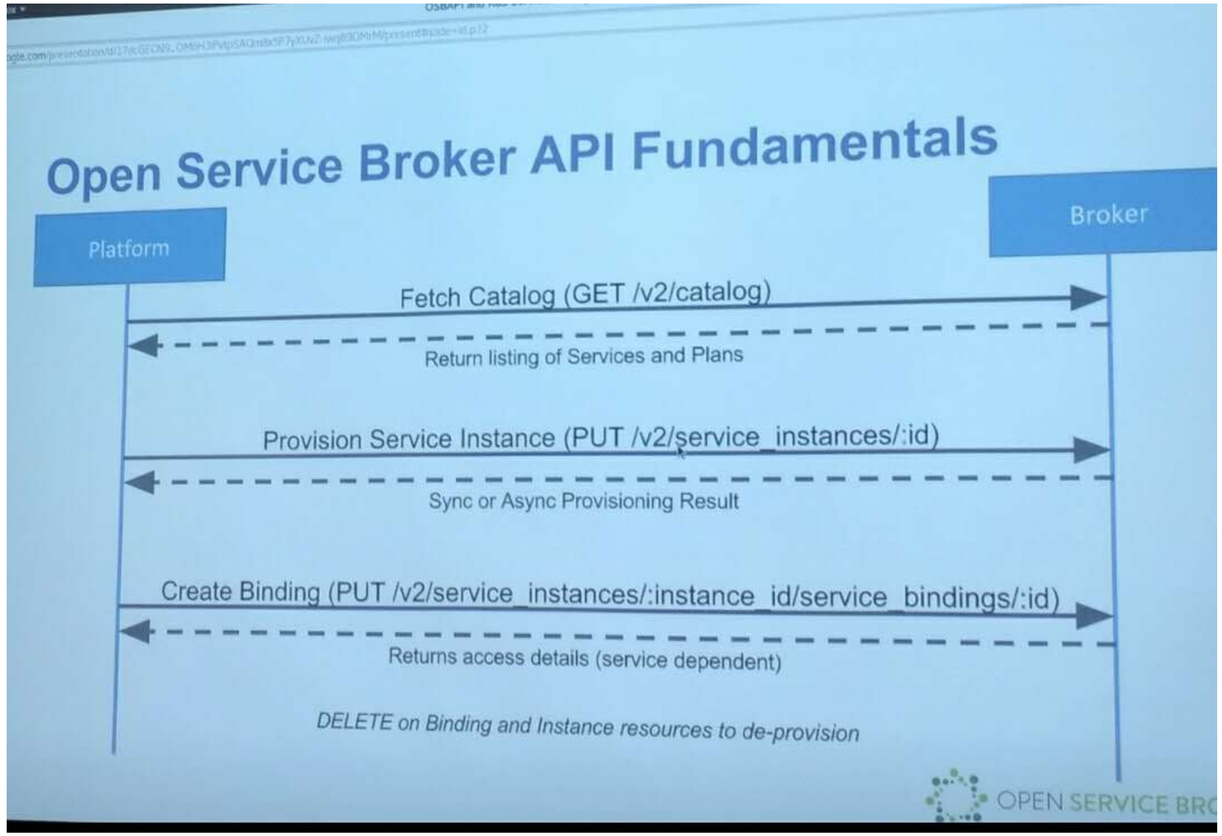
同样是为了让 Kubernetes 能创建和管理虚拟机资源，virtlet 与 KubeVirt 的目标一致，但实现方式上采用了 Kubernetes CRI 的标准实现，目前处于 pre-alpha 阶段。

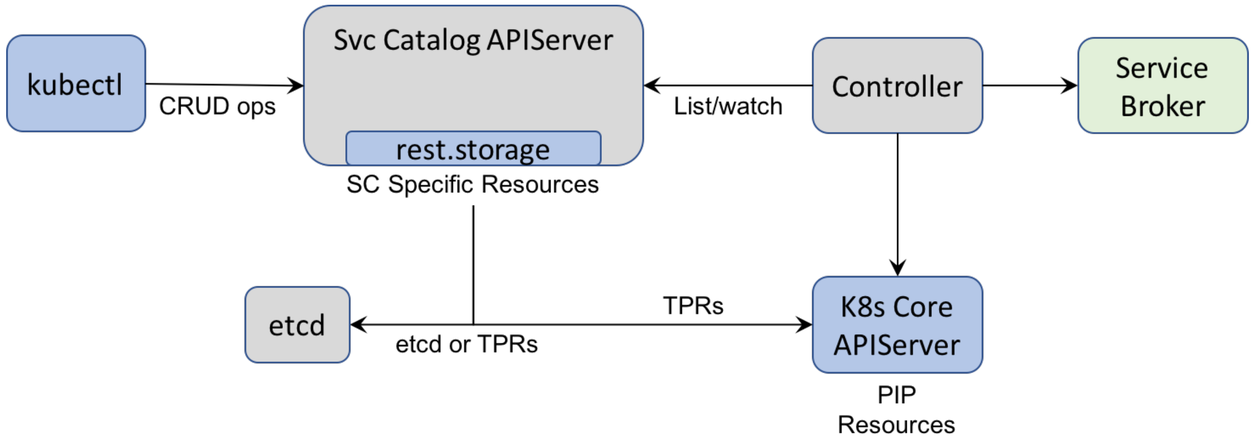
2. Frakti

Frakti 是 HyperContainer 的 Kubernetes CRI 实现，当前已经是 Kubernetes repo 下的正式项目，思路是基于 Hypervisor 的容器化，让 Kubernetes把 pod 和容器直接运行于宿主机的 hypervisor 之上。这是另一种跟 KubeVirt 和 virtlet 不同的容器和虚拟机深度融合的思路。

Open Service Broker API 以及 Kubernetes Service Catalog

Service Broker 可以理解成可管理一系列外部服务的服务端程序，作为云原生平台如 Kubernetes，OpenShift，CloudFoundry中，应用程序跟外部服务之间的桥接，让云平台用户和应用程序能简单地消费服务而不用关注服务创建销毁同步等细节。连接开发者和众多服务生态，让应用程序能简单方便地使用外部服务。





Audit in Kubernetes

审计是按时间顺序记录用户、管理员或其它系统组件的跟安全相关的操作行为，给系统管理员提供对以下几个问题的回答：1.发生了什么？2.什么时候发生的？3.谁发起的？4.对谁发起的？5.在哪里被观察到的？6.在哪里被发起的？7.会影响那些地方？用来发现系统安全违规行为、性能分析以及软件 bug 等。

提醒大家注意的一点是，审计系统并不会给系统带来额外的安全特性。以 Kubernetes Audit 为例子，Audit 是 kube-apiserver 接收到的请求流程中的验证插件之一，工作在 Authentication 认证之后，也就是说，审计只会针对已认证用户的操作行为进行记录。

Kubernetes Audit 审计作为kube-apiserver 的一部分，配置非常简单，在启动 kube-apiserver 时添加如下选项可以打开审计功能：

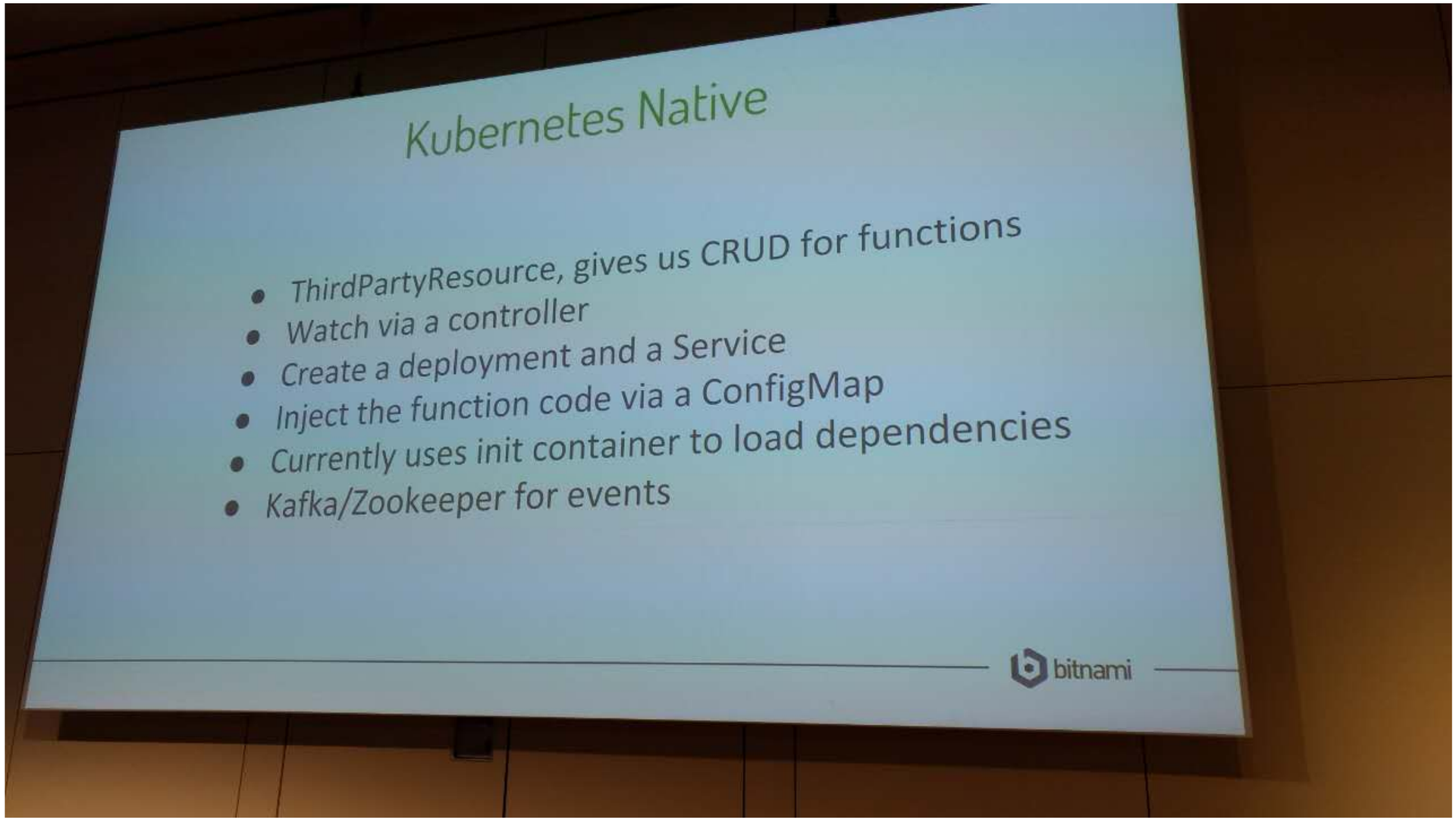
kube-apiserver

…

```
- -audit-log-path          //把审计日志追加到指定文件或新创建出审计文件
- -audit-log-maxage        //审计日志文件最长保留天数，过期后 Kubernetes 自动删除老文件
- -audit-log-maxbackup     //审计日志文件最多保留数量，超过数量后 Kubernetes 自动删除
- -audit-log-maxsize       //单个日志文件的大小，MB 为单位
```

当前 Kubernetes 审计的好处是轻量级实现、日志格式简单；但同时，它目前只对HTTP 请求做了审计、噪音数据多、基于日志文件。

Kubeless - Kubernetes 无服务器框架



跟 Kubeless 类似，来自 Platform 9 的 Fission、来自 Fabric 8 的 Funktion 都是建立于 Kubernetes 平台上的无服务器计算框架。其它的无服务器计算开源项目还有 Iron.io、webtask、OpenWhisk等等。

容器内哪个进程的 pid 应该为1？

在 Linux 内核启动的时候会先启动一个 init 进程，这个进程的 PID 始终为 1并且有着特殊的使命，需要做好的父进程的角色，比如传递信号，确保子进程完全退出；等待子进程退出等。容器使用 PID namespace 来隔离不同容器内的进程 PID 号，意味着不同的 PID namespace 里可以有同样的 PID。在容器环境中每个 PID namespace 里的第一个进程PID也为1，也需要处理系统信号，接管子进程，避免容器无法停止、产生僵尸进程等问题。

3种容器初始化系统

1. dumb-init 或 Tini
这两者的功能类似，都是PID 为1的轻量级进程守护程序，承担容器内 init 系统的作用。以 dumb-init 为例，我们如果要使用 dumb-init ，只需要在容器镜像内安装上 dumb-init 程序，然后在启动应用时，在命令行前加上 dumb-init 就可以了。这样 dumb-init 以 PID 1运行，启动一个进程并且把它接收到的所有信号量路由到一个 session 里并且分发给子进程，因为这时应用的实际进程PID 已经不是1了，当它从 dumb-init 收到信号时，可以跟预期的一样正常响应信号量，跟不在容器内运行有同样的行为，不会导致容器无法停止的问题。

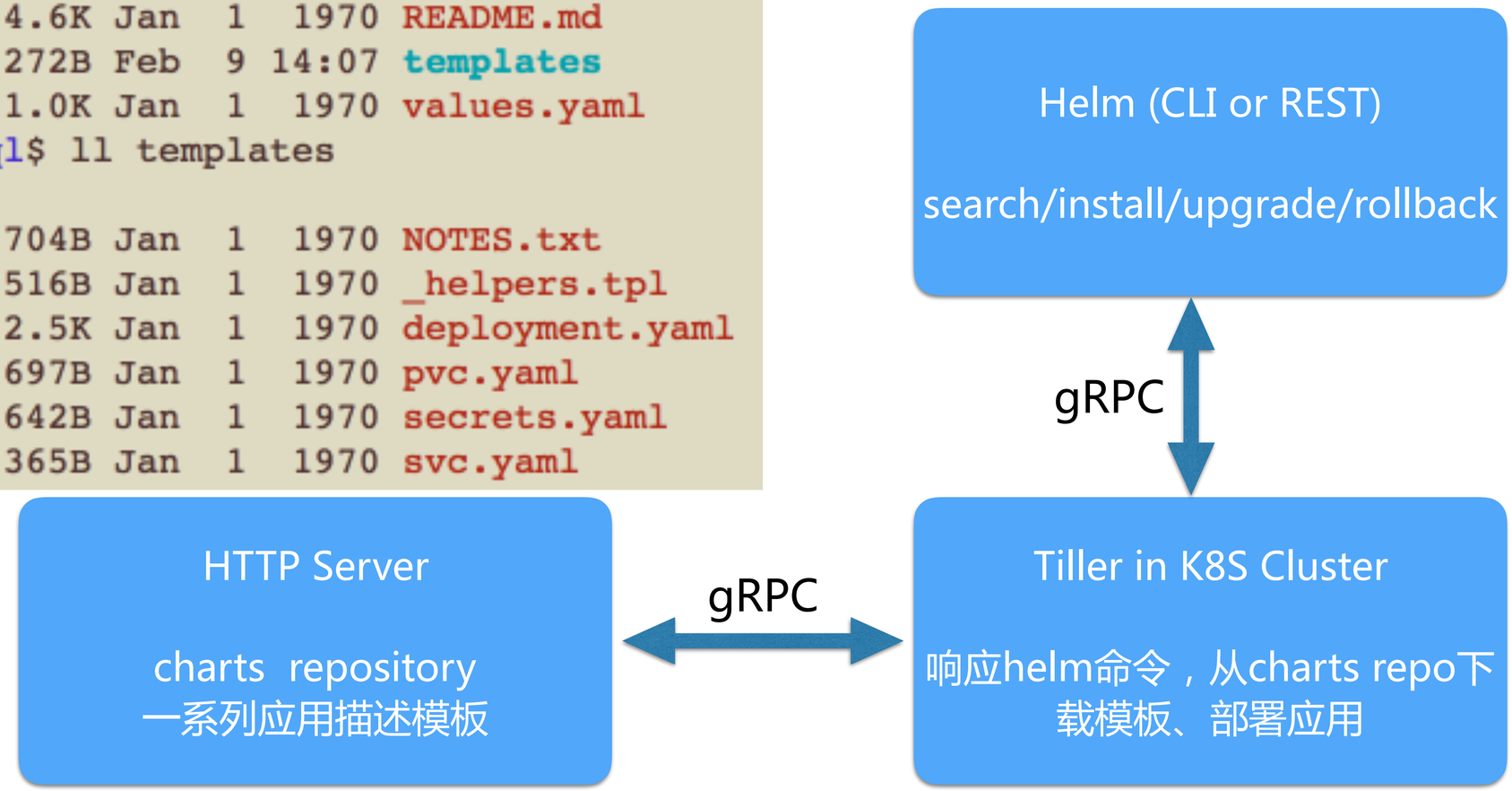
2. docker-init
Docker 1.13版本为 dockerd 进程添加了- - init 标记，如果在进程时添加了这个标记，那后续的容器启动时，docker 会自动为容器启动 PID 为1的僵尸收割进程充当 init 系统。同时，我们可以通过- -shutdown-timeout 来解决dockerd 退出时容器优雅退出，通过- -stop-timeout 解决单个容器优雅退出的问题。
1.13或更新的版本，可以通过docker info 可以看到 docker-init 相关信息。

3. Systemd
让Systemd 以非特权模式在容器内运行，充当容器的 init 的作用，这样带来的额外好处是可以更好处理日志以及服务启动顺序的处理等等。当前有 OCI Systemd Hook 可以支持这种方案。

Helm/Tiller/Charts — 容器应用商店

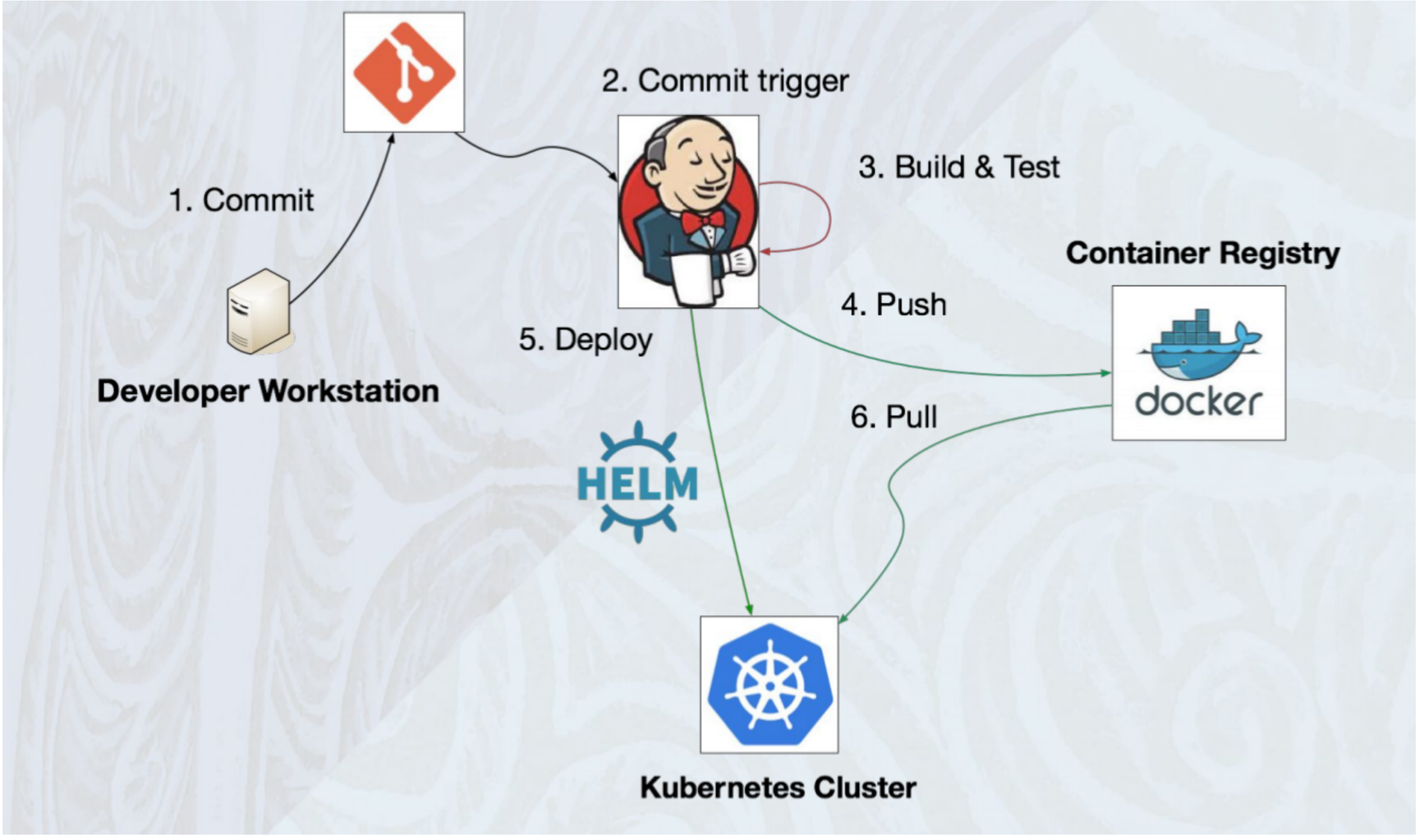
Kubernetes的应用包管理器，类似于Linux里的yum/apt；Helm为客户端、Tiller为服务端、Charts为应用模板描述文件。

```
[houming@rmbp ~/helm_charts/mysql$ ll
total 32
-rwxr-xr-x  1 houming  staff   288B Jan  1  1970 Chart.yaml
-rwxr-xr-x  1 houming  staff   4.6K Jan  1  1970 README.md
drwxr-xr-x  8 houming  staff   272B Feb  9 14:07 templates
-rwxr-xr-x  1 houming  staff   1.0K Jan  1  1970 values.yaml
[houming@rmbp ~/helm_charts/mysql$ ll templates
total 48
-rwxr-xr-x  1 houming  staff   704B Jan  1  1970 NOTES.txt
-rwxr-xr-x  1 houming  staff   516B Jan  1  1970 _helpers.tpl
-rwxr-xr-x  1 houming  staff   2.5K Jan  1  1970 deployment.yaml
-rwxr-xr-x  1 houming  staff   697B Jan  1  1970 pvc.yaml
-rwxr-xr-x  1 houming  staff   642B Jan  1  1970 secrets.yaml
-rwxr-xr-x  1 houming  staff   365B Jan  1  1970 svc.yaml
```



Helm 应用场景更加丰富

1. CoreOS Quay 集成 Helm 推出第一个 Kubernetes Application Registry
2. 使用 Helm 作为 DevOps 流程中的重要部分



如何让 Kubernetes 支持5万+个 Services？ - IPVS

```
#    Iptables –t nat –L –n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
KUBE-SERVICES  all  --  anywhere             anywhere             /* kubernetes service portals */ ← 1
DOCKER     all  --  anywhere             anywhere             ADDRTYPE match dst-type LOCAL

Chain KUBE-SEP-G3MLSGWVLUPEIMXS (1 references) ← 4
target     prot opt source               destination
MARK       all  --  172.16.16.2          anywhere             /* default/webpod-service: */ MARK set 0x4d415351
DNAT       tcp  --  anywhere             anywhere             /* default/webpod-service: */ tcp to:172.16.16.2:80

Chain KUBE-SEP-OUBP2X5UG3G4CYYB (1 references)
target     prot opt source               destination
MARK       all  --  192.168.190.128      anywhere             /* default/kubernetes: */ MARK set 0x4d415351
DNAT       tcp  --  anywhere             anywhere             /* default/kubernetes: */ tcp to:192.168.190.128:6443

Chain KUBE-SEP-PXEMGP3B44XONJEO (1 references) ← 4
target     prot opt source               destination
MARK       all  --  172.16.91.2          anywhere             /* default/webpod-service: */ MARK set 0x4d415351
DNAT       tcp  --  anywhere             anywhere             /* default/webpod-service: */ tcp to:172.16.91.2:80

Chain KUBE-SERVICES (2 references) ← 2
target     prot opt source               destination
KUBE-SVC-N4RX4VPNP4ATLCGG  tcp  --  anywhere             192.168.3.237        /* default/webpod-service: cluster IP */ tcp dpt:http
KUBE-SVC-6N4SJQIF3IX3FORG  tcp  --  anywhere             192.168.3.1          /* default/kubernetes: cluster IP */ tcp dpt:https
KUBE-NODEPORTS  all  --  anywhere             anywhere             /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */ ADDRTYPE match dst-type
            LOCAL

Chain KUBE-SVC-6N4SJQIF3IX3FORG (1 references)
target     prot opt source               destination
KUBE-SEP-OUBP2X5UG3G4CYYB  all  --  anywhere             anywhere             /* default/kubernetes: */

Chain KUBE-SVC-N4RX4VPNP4ATLCGG (1 references) ← 3
target     prot opt source               destination
KUBE-SEP-G3MLSGWVLUPEIMXS  all  --  anywhere             anywhere             /* default/webpod-service: */ statistic mode random probability 0.50000000000
KUBE-SEP-PXEMGP3B44XONJEO  all  --  anywhere             anywhere             /* default/webpod-service: */
```

5万+个 Services - IPTables 作为 LB 存在的问题

对数据平面来说，当前集群内部的 Cluster 类型 Service 是使用 IPTables 作为服务的负载均衡器，但因为 IPTables天生不是被设计用来作为 LB 来使用的，随着服务的数量增长，IPTables 规则则会成倍增长，这样带来的问题是路由延迟带来的服务访问延迟，同时添加或删除一条规则也有较大延迟。

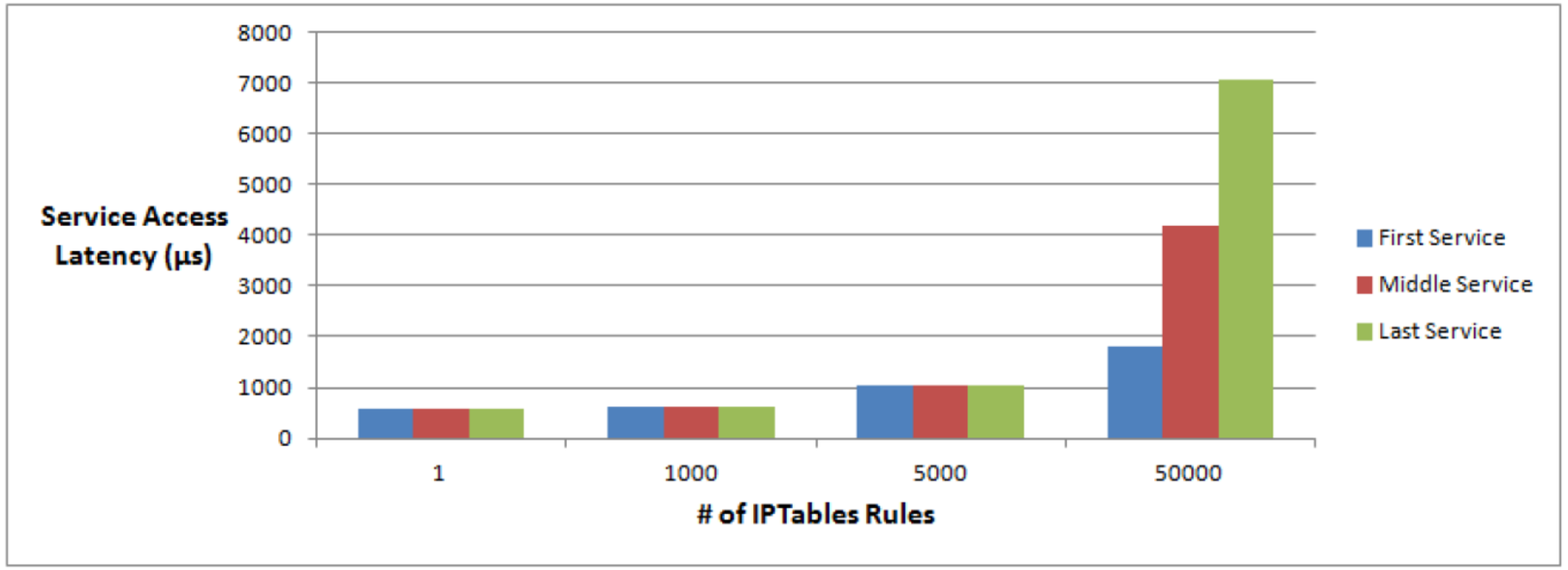
添加规则不是增量的，而是先把当前所有规则都拷贝出来，再做修改然后再把修改后的规则保存回去，这样一个过程的结果就是 IPTables 在更新一条规则时会把 IPTables 锁住，这样的后果在服务数量达到一定量级的时候，性能基本不可接受：在有5千个服务（4万条规则）时，添加一条规则耗时11分钟；在右2万个服务（16万条规则）时，添加一条规则需要5个小时。

5万+个 Services - IPTables 服务路由性能

Where is latency generated?

- Long list of rules in a chain
- Enumerate through the list to find a service and pod

In this test, there is one entry per service in KUBE-SERVICES chain.



	1 Service (µs)	1000 Services (µs)	10000 Services (µs)	50000 Services (µs)
First Service	575	614	1023	1821
Middle Service	575	602	1048	4174
Last Service	575	631	1050	7077

5万+个 Services - 什么是 IPVS

使用 “IP Virtual Server”（IPVS)来替换当前 kube-proxy 中的 IPTables 实现，这样能带来显著的性能提升以及更智能的负载均衡功能如支持权重、支持重试等等。
那什么是 “IP Virtual Server”（IPVS ）呢？作为 Linux Virtual Server(LVS) 项目的一部分，IPVS 是建立于 Netfilter之上的高效四层负载均衡器，支持 TCP 和 UDP 协议，支持3种负载均衡模式：NAT、直接路由（通过 MAC 重写实现二层路由）和IP 隧道。
IPVS 负载均衡模式在 kube-proxy 处于测试阶段还未正式发布，完全兼容当前 Kubernetes 的行为，通过修改 kube-proxy 启动参数，在 mode=userspace 和 mode=iptables 的基础上，增加 mode=IPVS 即可启用该功能。由下图测试数据可见：使用 IPVS 后，添加规则的时间跟 IPTables 相比有极大的提升，5千个服务时规则添加时间由 11分钟降到了2毫秒，2万个服务时规则添加时间由5个小时降到了2毫秒，同时在5千数量级以上个服务存在时，访问服务的网络带宽也有巨大的提升，可见 IPVS完全可以支持大规模服务的场景。
https://github.com/kubernetes/kubernetes/issues/44063

5万+个 Services - IPTables vs IPVS 添加规则时间

Measured by iptables and ipvsadm, observations:

- In IPTables mode, latency to add rule increases significantly when # of service increases
- In IPVS mode, latency to add VIP and backend IPs does not increase when # of service increases

# of Services	1	5,000	20,000
# of Rules	8	40,000	160,000
IPTables	2 ms	11 min	5 hours
IPVS	2 ms	2 ms	2 ms

THANKS

http://easystack.cn