

CloudNativeLives

云原生技术的前世今生

CNCF社区主要贡献者倾心打造 & 华为云容器团队核心架构师

大纲

- **CNCF云原生介绍**
- 容器技术发展介绍
- Kubernetes技术架构

CNCF ToC: CNCF Cloud Native Definition v1.0

云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括**容器、服务网格、微服务、不可变基础设施和声明式API**。

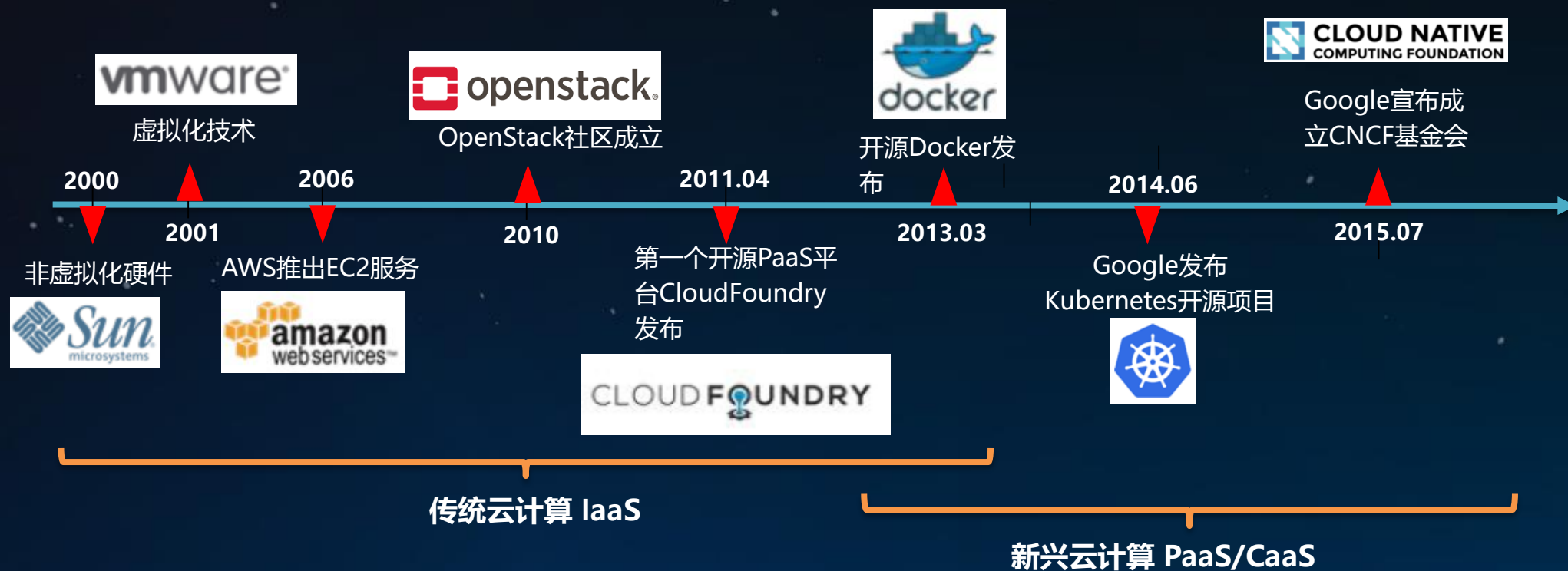
这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统作出频繁和可预测的重大变更。

云原生计算基金会（CNCF）致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术。我们通过将最前沿的模式民主化，让这些创新为大众所用。

云计算的发展历程



“云”中的资源在使用者看来是可以无限扩展的，并且可以**随时获取，按需使用，随时扩展**，按使用付费。这种特性经常被称为像水电一样使用IT基础设施。



云计算伴随云原生技术步入新时代



容器是云原生技术发展的第一波热潮



kubernetes

Kubernetes
成为容器编排
事实标准



Service Mesh
成为微服务
新热点



Serverless在公
有云应用渐宽

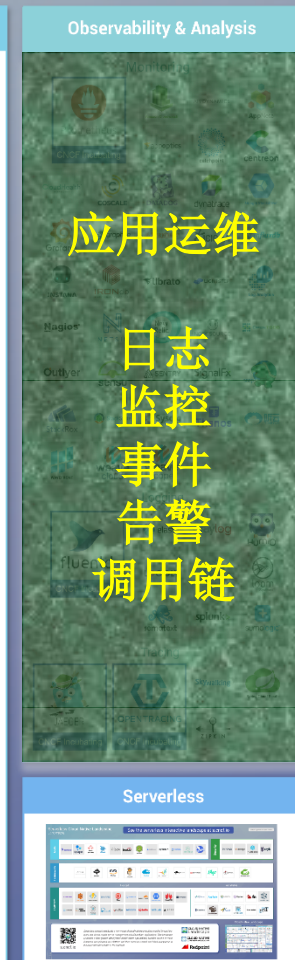
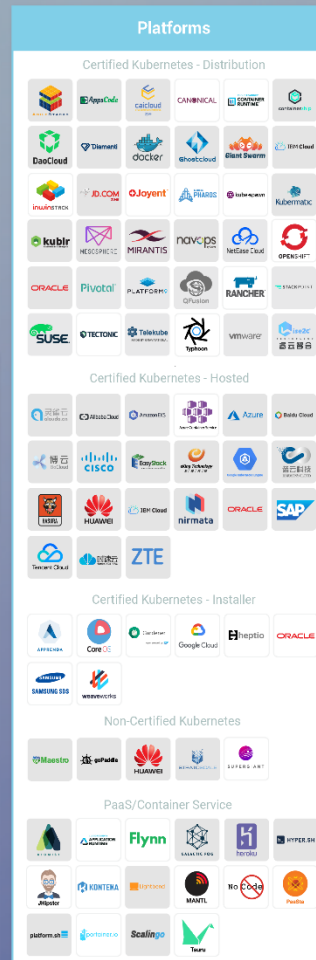
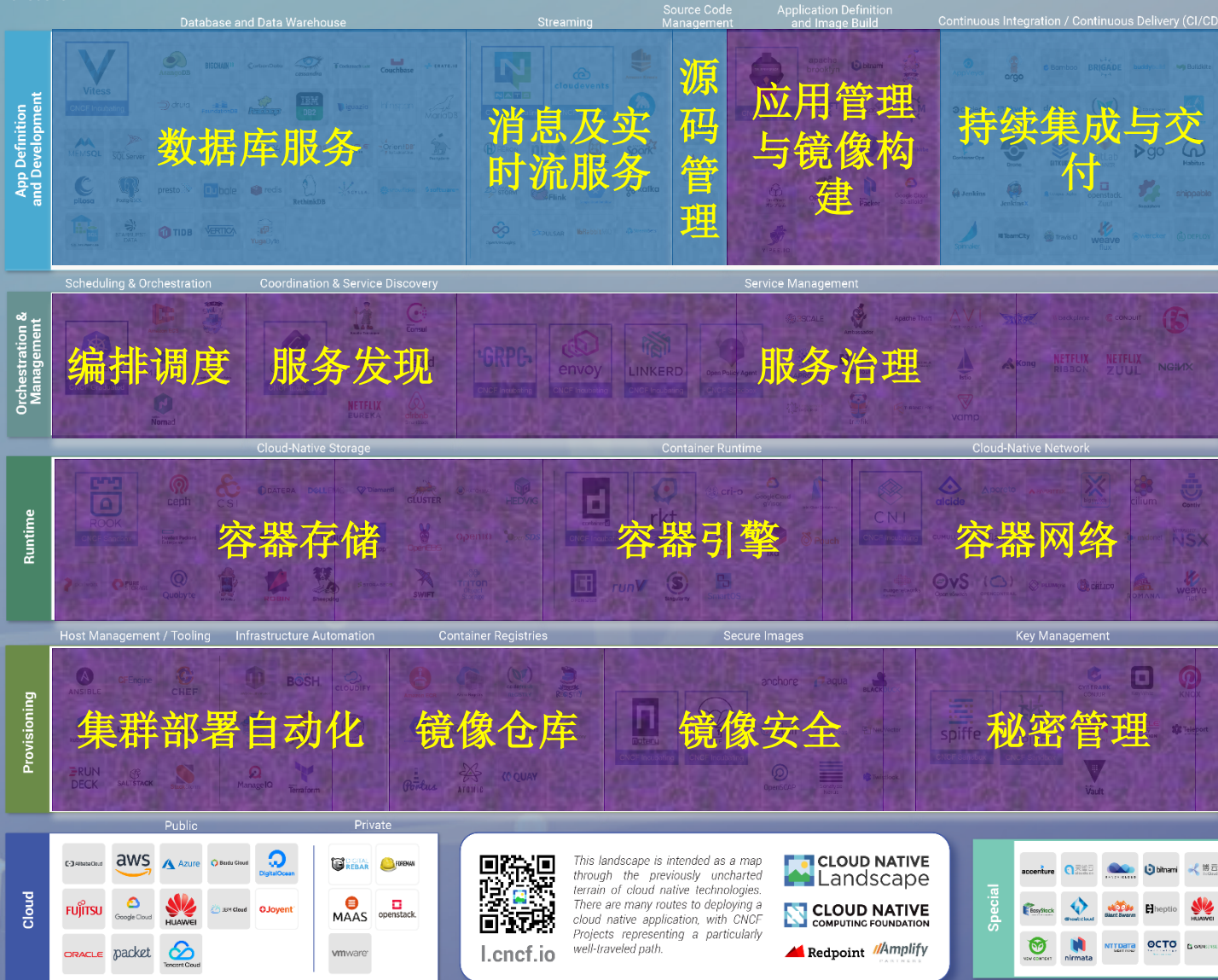
容器服务的目标就是打造Cloud Native应用的运行环境

Cloud Native Landscape

v20180525

See the interactive landscape at l.cncf.io

Greyed logos are not open source



Kubernetes Certified Service Provider

Kubernetes Training Partner

Special



l.cncf.io

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.



Redpoint Amplify

CNCF当前项目介绍



容器编排



Kubernetes



Helm

容器引擎

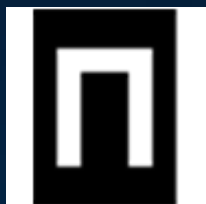


Containerd



Rocket

容器镜像仓库



Notary



TUF

CNCF当前项目介绍

容器网络



CNI

服务网格&服务发现



CoreDNS



Linkerd



Envoy

容器监控运维



Prometheus



Fluentd



Jaeger



OpenTracing



CNCF当前项目介绍

消息通信



GPRC



NATS

数据库



Vitess

Sandbox项目



Rook



SPIFFE



SPIRE



Open Policy Agent



CloudEvents



Telepresence

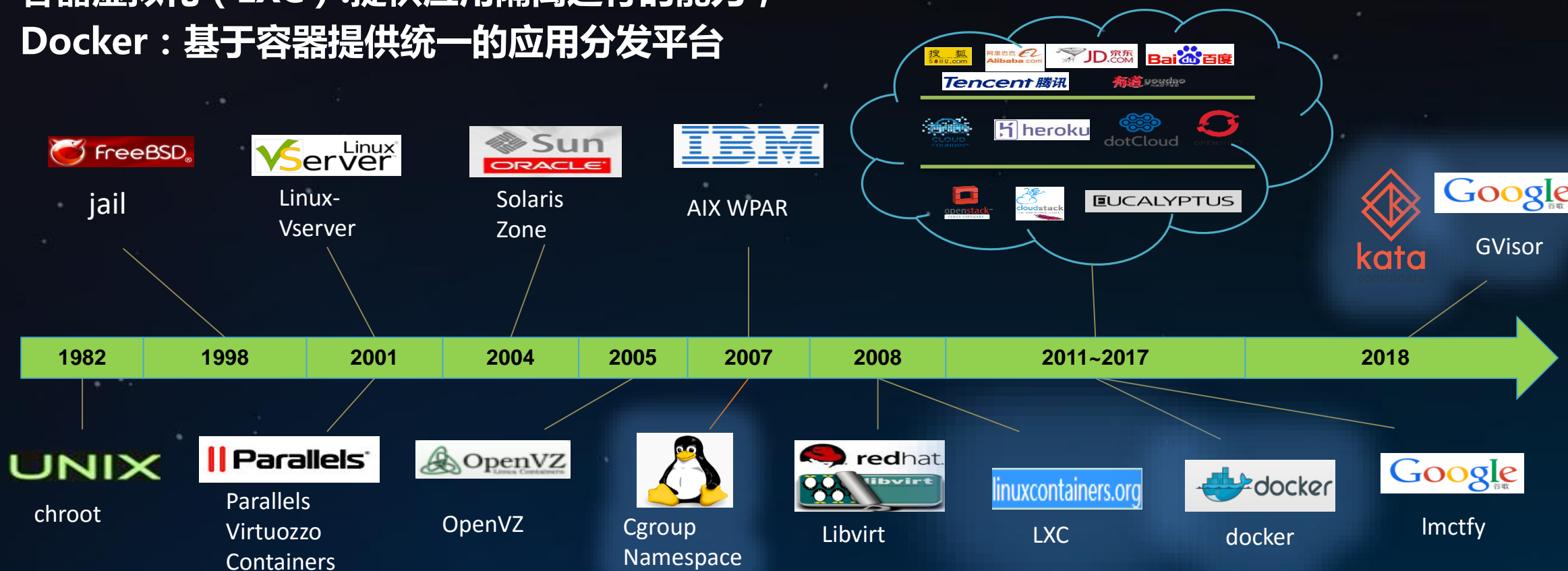
大纲

- CNCF云原生介绍
- **容器技术发展介绍**
- Kubernetes技术架构

容器技术发展历史

容器虚拟化 (LXC) : 提供应用隔离运行的能力 ;

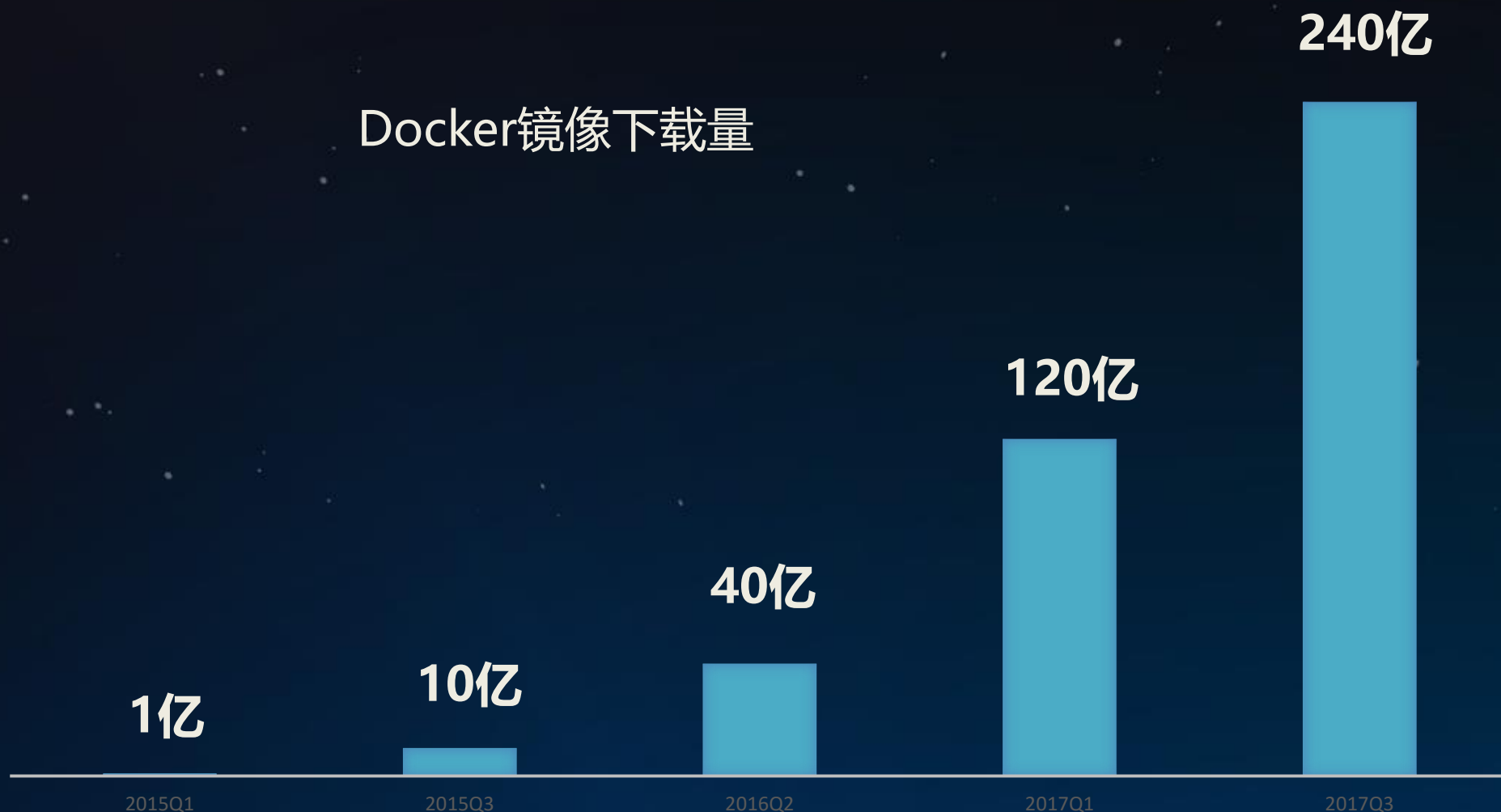
Docker : 基于容器提供统一的应用分发平台



2001年早期轻量级虚拟化产品如OpenVZ、Linux-Vserver当时主要用于VPS，均未进入Linux内核；
2007年随着cgroup以及namespace合入内核，linux容器技术得到Linux内核主线的支持；
2008年LXC开源项目成立，实现用户态容器管理工具；容器技术在业界的使用越来越广泛；
2013年3月份docker出现，针对容器虚拟化更有理想的统一平台出现
2018年，Kata容器合并RunV和Intel Clear Container，Google发布gVisor容器

容器惊人的发展速度

Docker镜像下载量



数据来源：



容器的三大好处，为应用而生



资源隔离与利用率提升

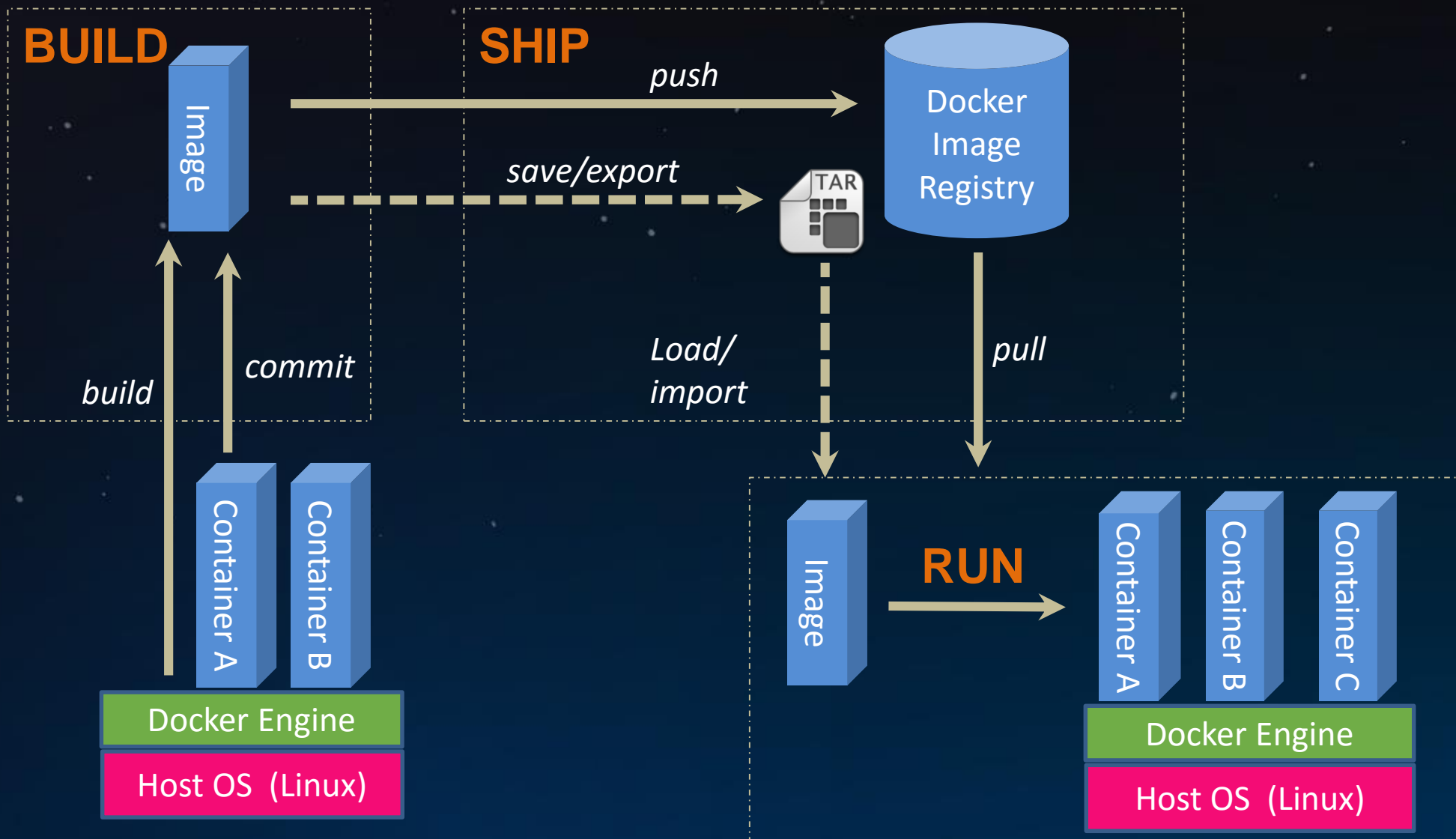


秒级弹性



环境一致性，简化交付

Docker – Build, Ship, and Run Any App, Anywhere



容器底层关键技术 - Linux Cgroup

Docker使用Linux Cgroup技术来实现容器实例的资源管理

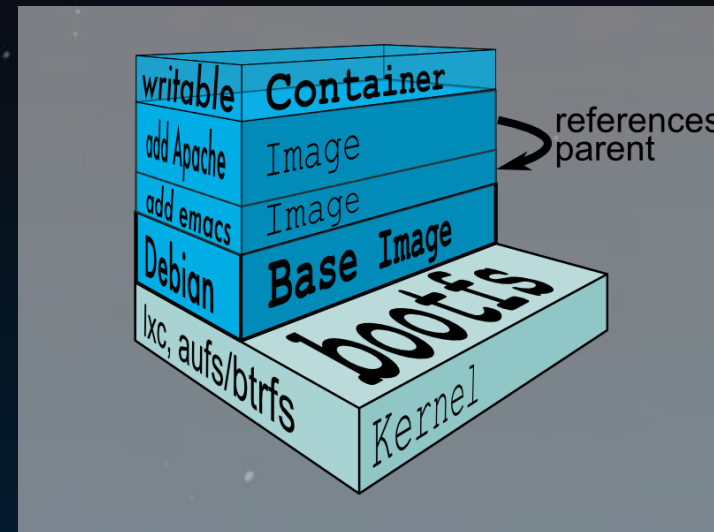
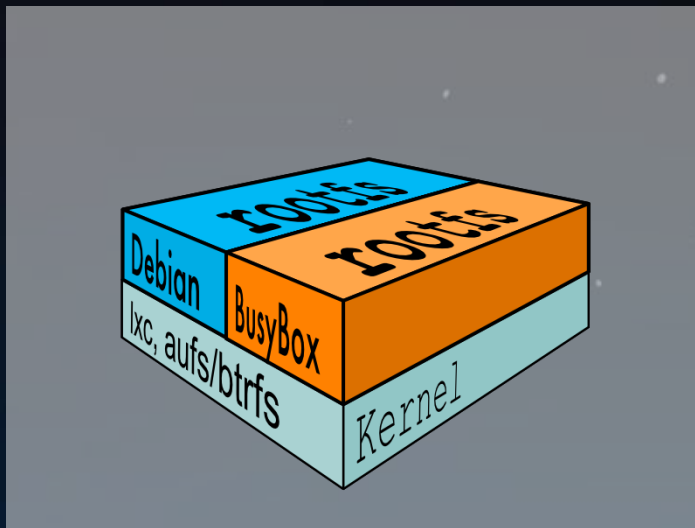
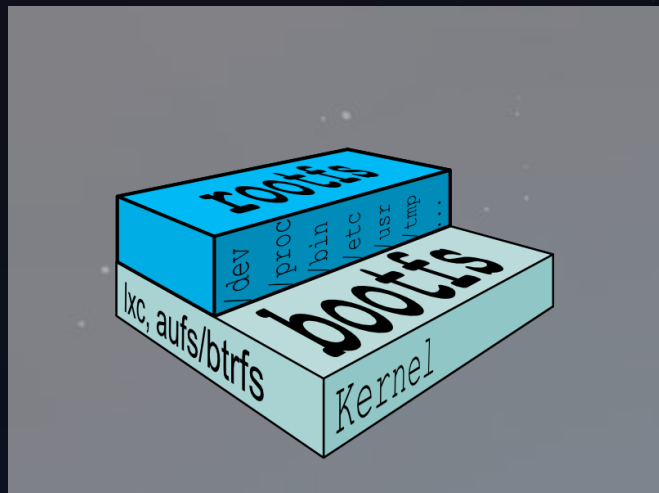
cgroup子系统	资源管理	对应的docker接口
memory	限制cgroup中的任务所能使用的内存上限	-m --memory-swap --memory-reservation --kernal-memory --oom-kill-disable --memory-swappiniss
cpu	使用调度程序提供对CPU的cgoup任务访问	-c --cpu-period --cpu-quota
Cpuset	为cgroup中的任务分配独立CPU和内存节点	--cpuset-cpus --cpuset-mems
Blkio	为块设备设定输入/输出限制	--blkio-weight --blkio-weight-device --device-read-bps --device-write-bps --device-read-iops --device-write-iops

容器底层关键技术 - Linux Namespace

Docker使用linux namespace技术来实现容器实例间的资源隔离

Namespace	系统调用参数	隔离内容
PID namespace	CLONE_NEWPID	隔离不同用户的进程，不同的namespace中可以有相同的pid。 允许嵌套，可以方便实现docker in docker
UTS namespace	CLONE_NEWUTS	允许每个容器拥有独立的hostname和domain name，使其在网络上可以被视为独立的节点。
IPC namespace	CLONE_NEWIPC	保证容器间的进程交互，进行信号量、消息队列和共享内存的隔离。
Network namespace	CLONE_NEWNET	实现网络隔离，每个net namespace有独立的network devices、ip addresses、ip routing tables、/proc/net目录。
Mount namespace	CLONE_NEWNS	隔离不同namespace的进程看到的目录结构，每个namespace的容器在/proc/mounts的信息只包含所在namespace的mount point。
User namespace	CLONE_NEWUSER	允许每个容器可以有不同的user和group id

容器底层关键技术 – 联合文件系统



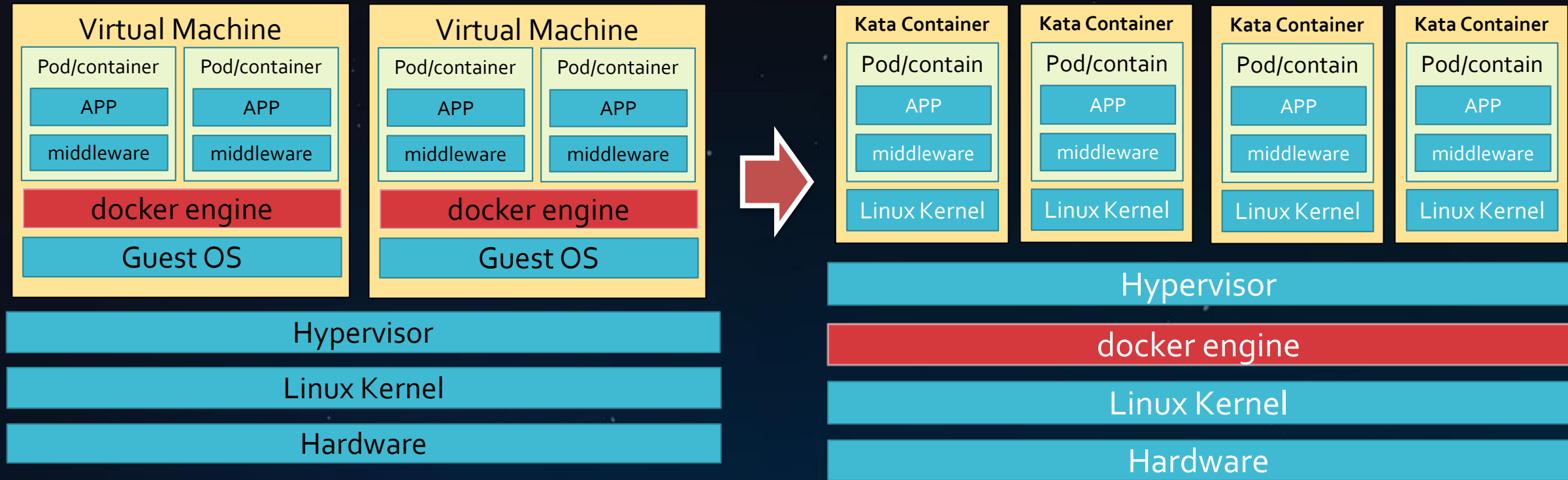
概念：一个基于文件的接口，通过把一组目录交错起来，形成一个单一视图。

优点：

- 1、多个容器可以共享image存储，节省存储空间；
- 2、部署多个容器时，base image可以避免多次拷贝，实现快速部署；

Docker目前支持的联合文件系统种类包括devicemapper、overlay2、aufs、btrfs、vfs和

新一代Kata容器实现多租户容器强隔离



- 虚拟化层的存在，保障了容器在多租户场景下的安全性。
- 高度裁剪和优化过的KVM, guest OS保证了VM启动时间极短，性能损耗极小。
- 接口层支持对接docker引擎或crio，镜像完全兼容docker镜像，无需修改。完美融入K8S容器生态。

大纲

- CNCF云原生介绍
- 容器技术发展介绍
- **Kubernetes技术架构**

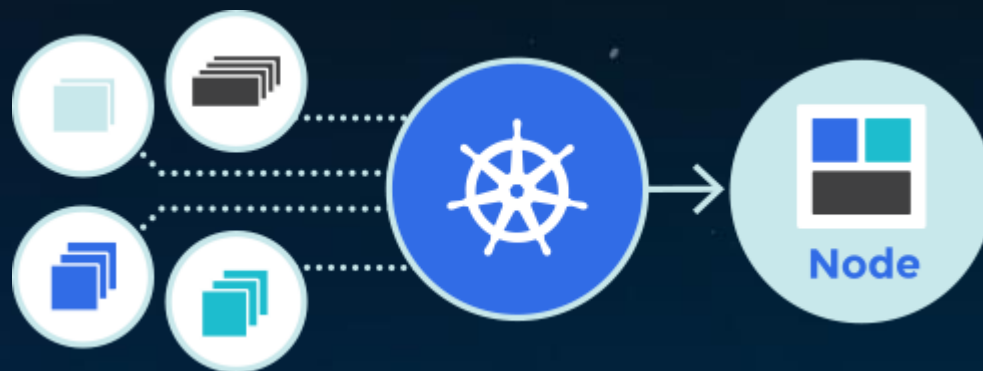
什么是Kubernetes ?

Kubernetes 是用于自动部署，扩展和管理容器化应用程序的开源系统。

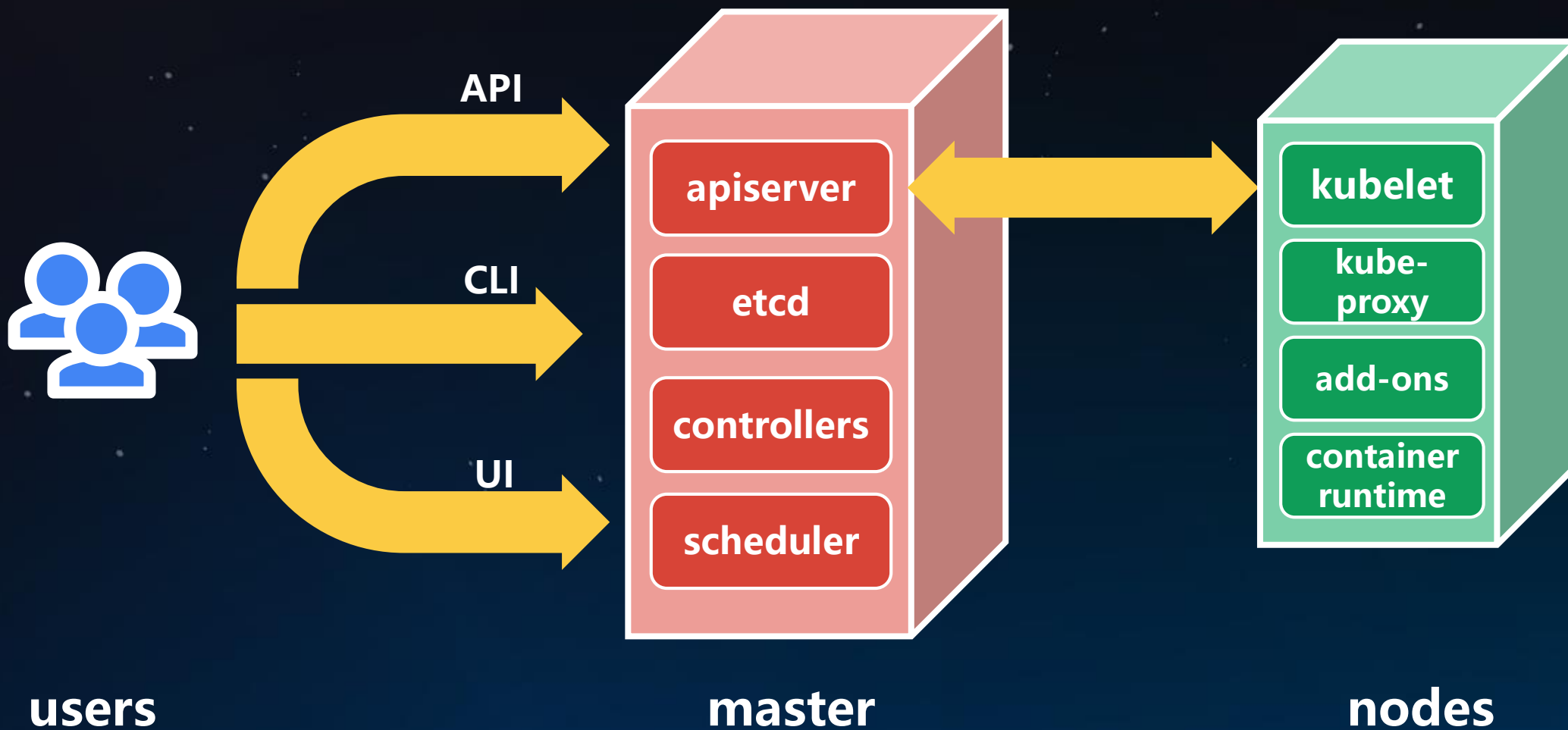
生产级别的容器编排系统

数据中心OS

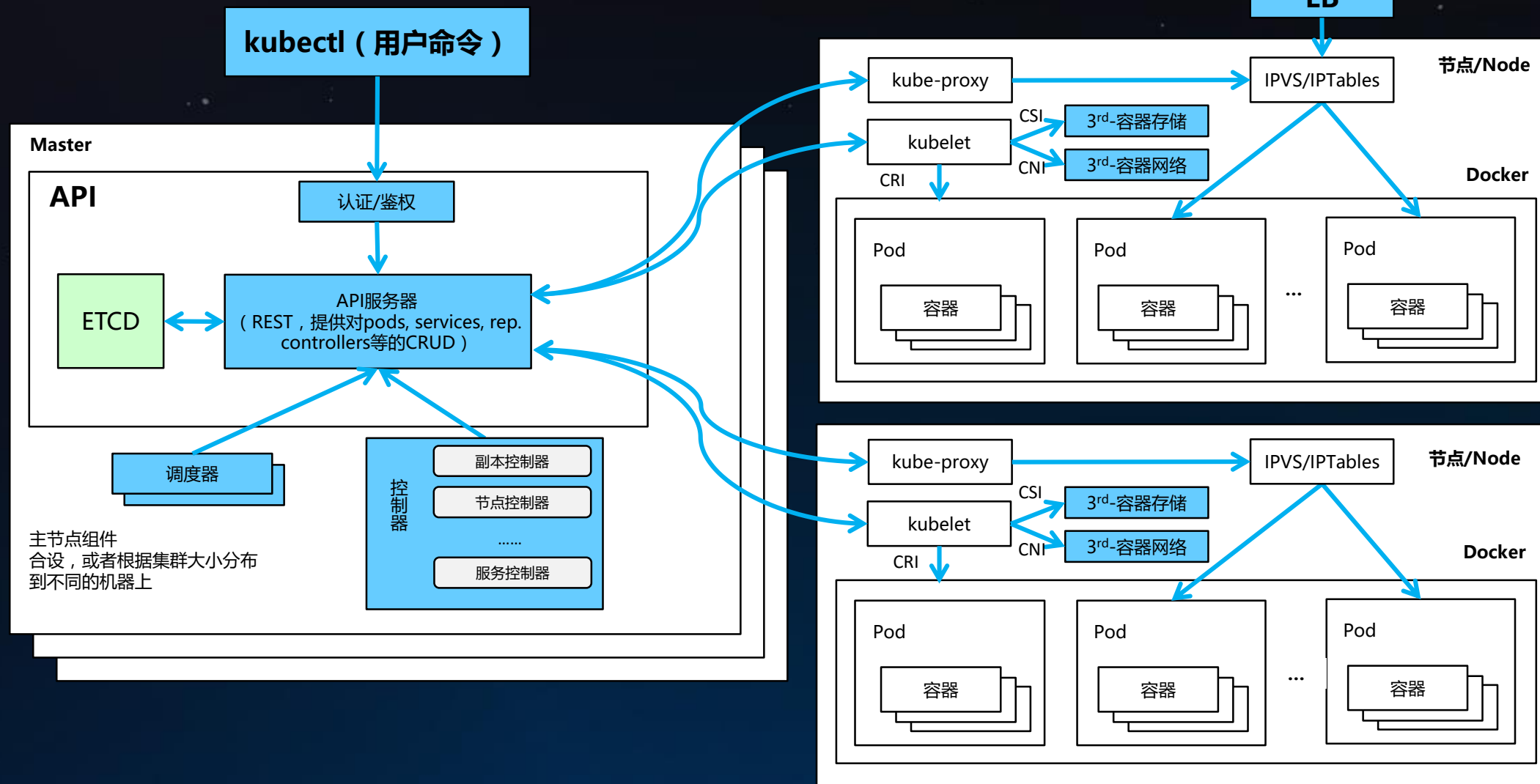
新生代的云平台



Kubernetes系统组件



Kubernetes架构



Declarative (申明式API) vs Imperative (命令式API)



从命令行角度来看：

命令式：kubectl run nginx --image nginx

声明式：kubectl create -f **nginx.yaml**



从Kubernetes API角度来看：

绝大部分的Kubernetes**采用申明式API**进行设计

Kubernetes当前也部分提供命令式的API：如执行Pod Exec动作

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

.....

status:

availableReplicas: 2

unavailableReplicas: 2

.....

多设计API

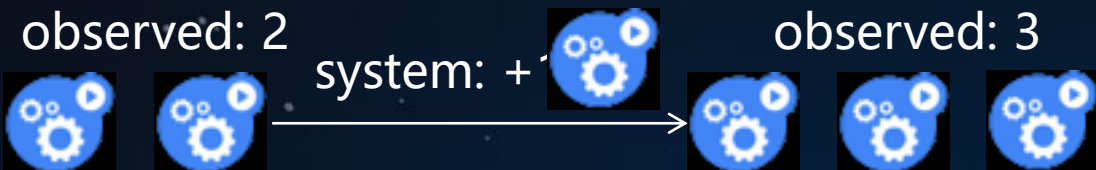
Declarative (申明式API) vs Imperative (命令式API)



Declarative (声明式API) :

Definition: user specifies what the desired state is.
The system will know what to do to move current state to desired state.


user: set desired state to 3




GET
/apis/apps/v1beta2/watch/namespaces/{namespace}/daemonsets

Imperative (命令式API) :

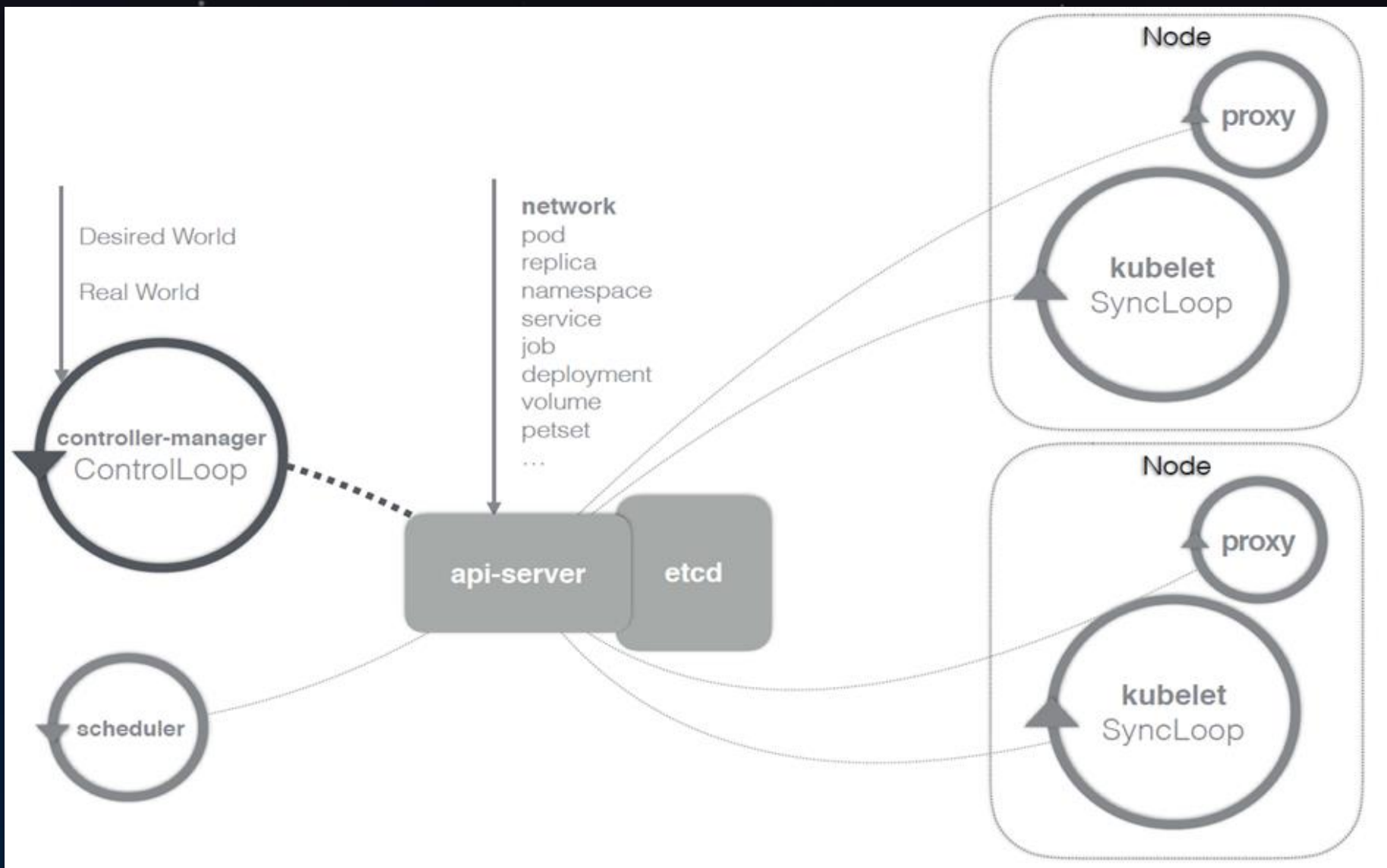
Definition: user tells the system what to do.

user: +1 

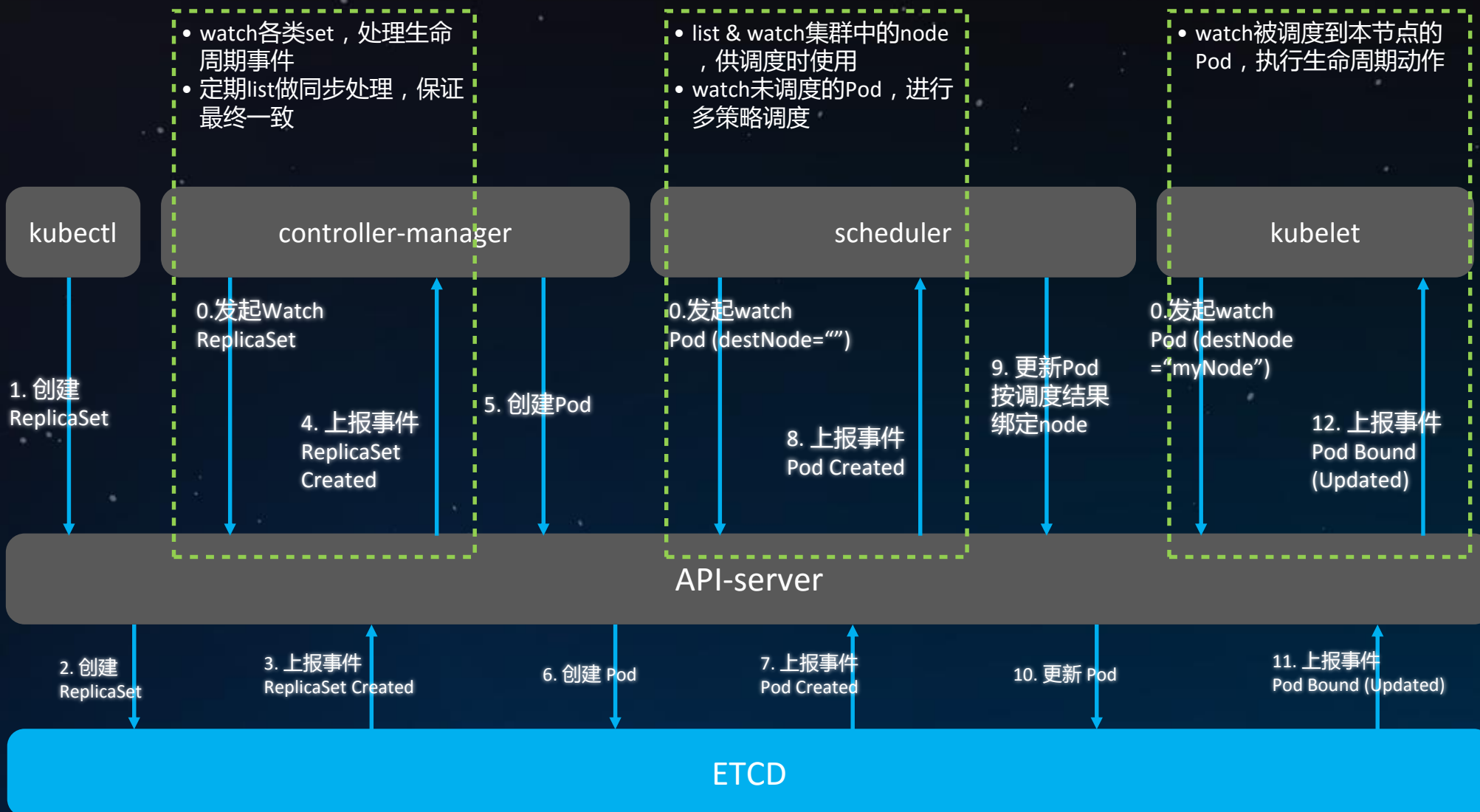
system: +1 

POST
/api/v1/namespaces/{namespace}/pods/{name}/**portforward**

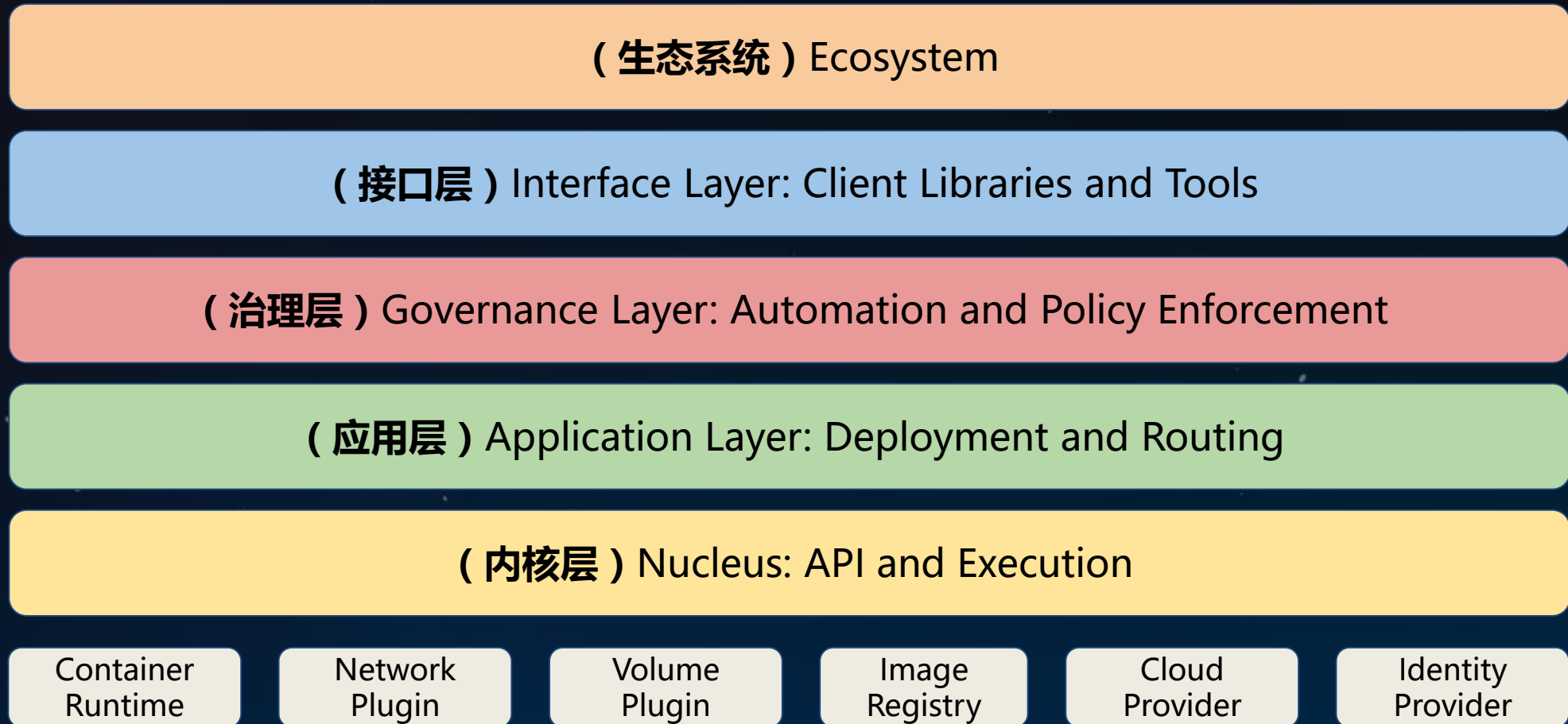
Kubernetes Controllers



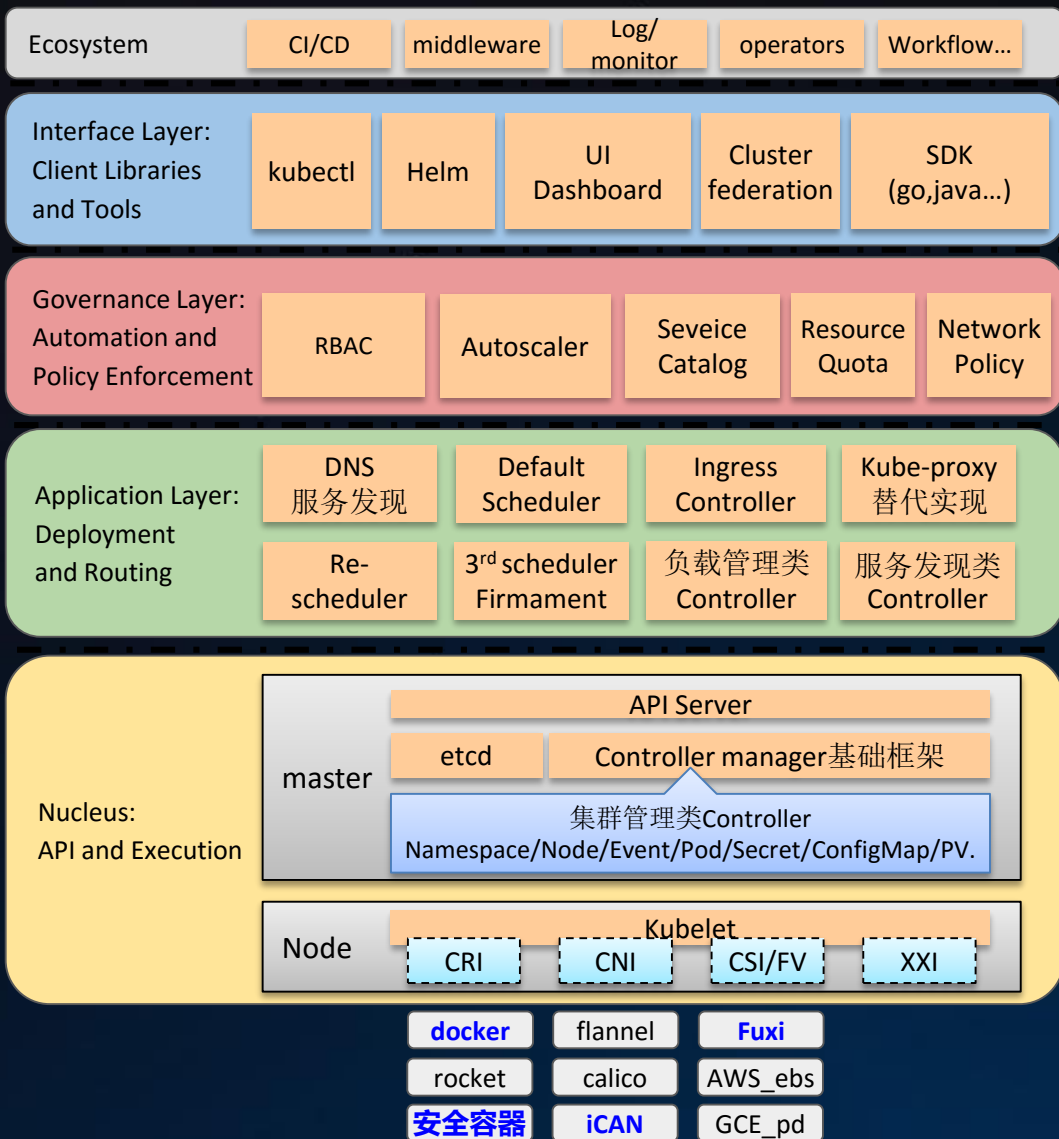
Kubernetes基于list-watch机制的控制器架构



Kubernetes架构分层



K8S社区架构中对各层的详细定义



生态层：不属于K8S范围

接口层（工具、SDK库、UI等）：

- K8S官方的项目会提供库、工具、UI等外围工具
- 外部可提供自有的实现

治理层：策略执行和自动化编排

- 对应用运行的可选层，没有这层功能不影响应用的执行
- 自动化API：水平弹性伸缩、租户管理、集群管理、动态LB等
- 策略API：限速、资源配额、pod可靠性策略、network policy等

应用层：部署（无状态/有状态应用、批处理、集群应用等）和路由（服务发现、DNS解析等）

- K8S发行版必备功能和API，K8S会提供默认的实现，如Scheduler
- Controller和scheduler **可以被替换为各自的实现**，但必须通过一致性测试
- 业务管理类Controller：daemonset/replicaset/replication/statefulset/cronjob/service/endpoint

内核层：Kubernetes最核心功能，对外提供API构建高层的应用，对内提供插件式应用执行环境

- 由主流K8S codebase实现（主项目），属于K8S的内核、最小特性集。 **等同于Linux Kernel**
- 提供必不可少基础Controller
- 集群管理类Controller：Node/gc/podgc/volume/namespace/resourcequota/serviceaccount

- 1、内核层：提供最核心的特性最小集以及API，为必选模块
- 2、内核层之上：以各种Controller插件方式实现内核层API，支持可替换的实现
- 3、内核层之下：是各种适配存储、网络、容器、Cloud Provider等

Kubernetes生态系统范畴



- 容器内 – 多样化工作负载
- K8S南向 – 多样化的环境
 - 容器、网络、存储插件
 - 镜像仓库
 - 多云插件
 - 集群管理配置
 - 认证系统
- K8S北向 – 多样化的服务和工具
 - 日志&监控
 - 配置语言
 - 编排部署
 - 持续集成
 - PaaS
 - 工作流
 - 函数服务
 - 数据处理服务
 - 对象事务服务 (数据库、存储)
 - 互联网应用

Thank You

<http://zhibo.huaweicloud.com/watch/2174406>

直播 每周四 晚20:00

