

2018 | 中国·北京站
DevOps 落地，从这里开始

DevOps 国际峰会

暨 DevOps 金融峰会

指导单位： 云计算开源产业联盟
Open Source Cloud Alliance for Industry (OSCAI)

主办单位： DevOps时代

 高效运维社区
GreatOPS Community

2018年6月29日-30日

地址：北京悠唐皇冠假日酒店

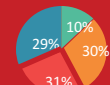
如何落地全球最大 Kubernetes 生产集群

鲍永成 京东技术总监

京东数据中心集群管理发展之路

集群编排

支撑京东组件化。
实现一键建站 组件仓库。



资源调度

京东阿基米德调度项目。
节省数据中心数亿元采购成本。

Container as VM

培养习惯。
用户最小学习成本迁移到容器跑平台。

构建容器生态

重构整个数据中心基础设施与基础软件。
实现基础设施能力可编程 API化。

All In Container

目录

- ➔ **1** Container Eco System
- 2** 化繁为简 K8S重构
- 3** 大规模集群运营
- 4** 巡检与可视化

Container Eco System

DOIS



ContainerDNS



ContainerLB



ContainerFS



ContainerIS

Archimedes



ContainerCI

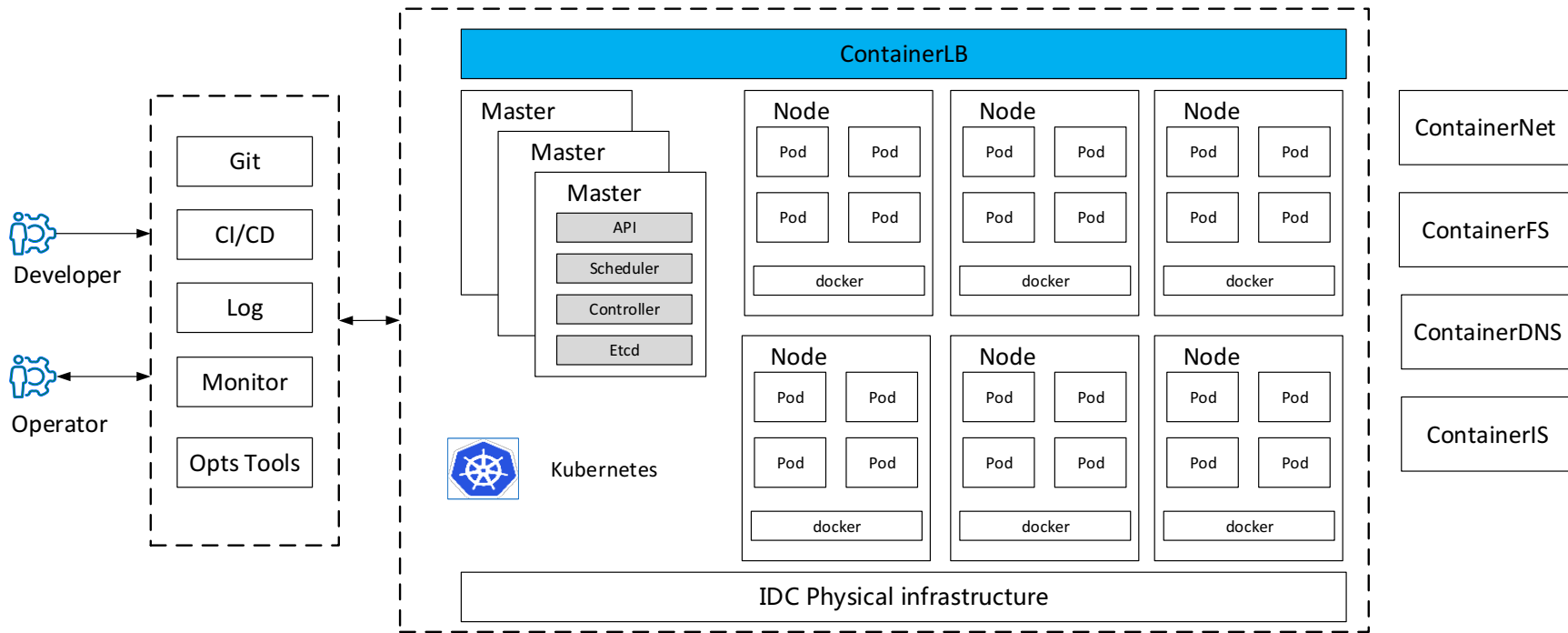


MDC

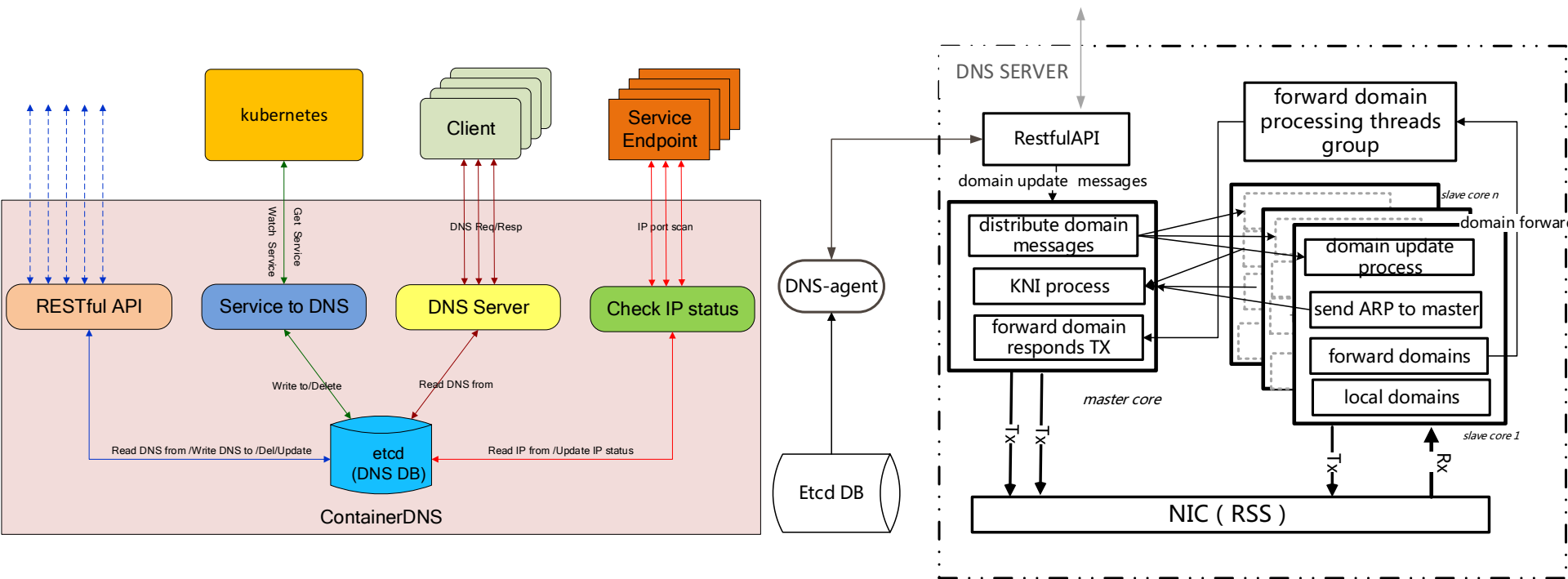


ContainerLog

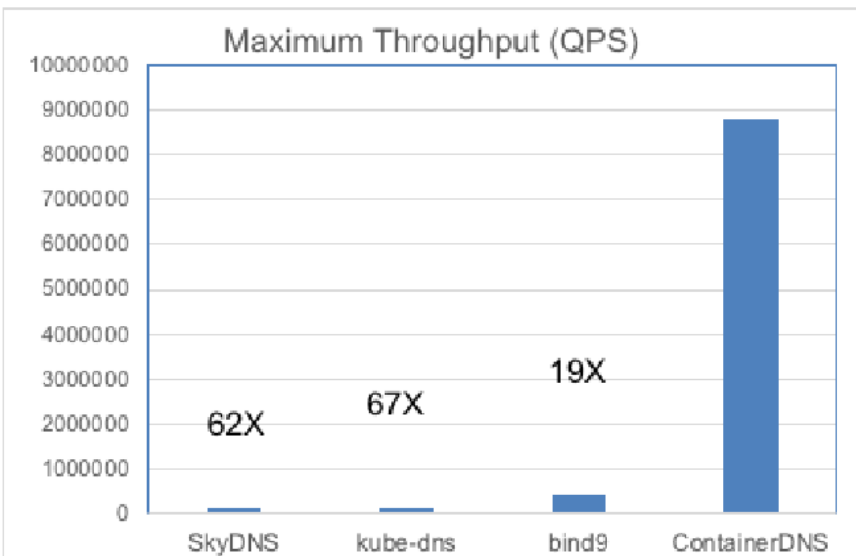
Container Eco System 架构图



ContainerDNS 架构图



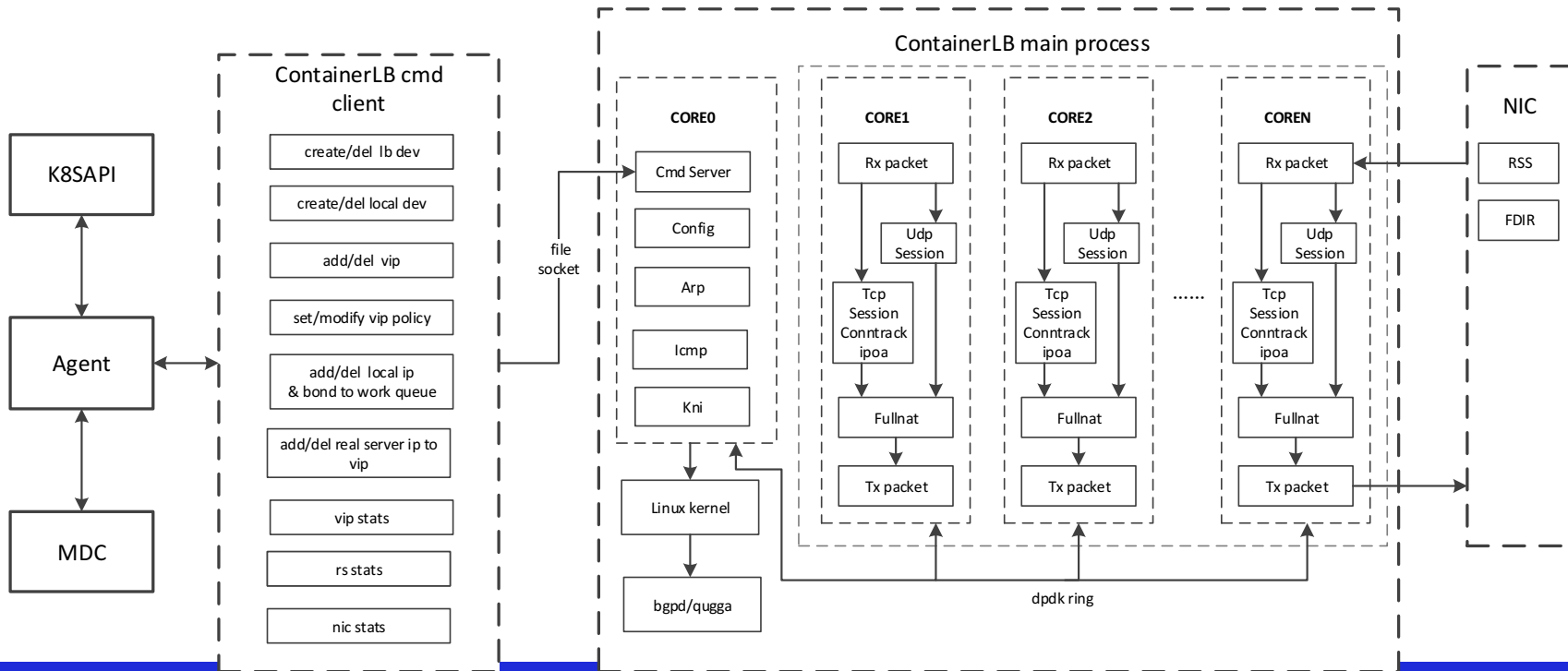
ContainerDNS Performance



Systems	TPS	Threads	CPU	Mem.	Net. (MB/s)
<i>SkyDNS</i>	141,406	60	10.9%	4.5%	25.0
<i>kube-dns</i>	131,911	60	24.5%	4.3%	25.8
<i>bind9</i>	456,608	120	19.7%	5.6%	84.3
<i>Container DNS</i>	880,0085	80,00	16.3%	4.6%	1729.5

Query数量	最慢响应us (bind9/ KDNS)	最快响应us (bind9/ KDNS)	平均响应us (bind9/ KDNS)
1	1140/226	15/16	102/68
3	1138/654	18/17	172/83

Container LB 架构&核心流程



Container LB Performance

调度算法	线程数	Min(ms)	Max(ms)	Avg(ms)	TP99(ms)	QPS(笔/秒)	交易失败数	交易失败率
tcp.IP_port	1000	0	124	0	1	2088316	0	0.00%
tcp.RR	1000	0	226	0	1	2070281	0	0.00%
tcp.LC	1000	0	410	1	6	696950	0	0.00%

调度算法	线程数	Min(ms)	Max(ms)	Avg(ms)	TP99(ms)	QPS(笔/秒)	交易失败数	交易失败率
udp.IP_port	1000	0	58	0	1	4212952	0	0.00%
udp.RR	1000	0	226	0	1	4325008	0	0.00%
udp.LC	1000	0	58	1	3	812356	0	0.00%

目录

1 Container Eco System

➔ 2 化繁为简 K8S重构

3 大规模集群运营

4 巡检与可视化

Opensource

拿来主义与自己造轮子并不矛盾.



化繁为简 K8S重构

严肃负责

生产环境是严肃的环境 任何开源软件必须经过亲自严格测试.

化繁为简

面向生产 keep simple.

规模哲学

大集群降低运维风险与成本 vs 鸡蛋放一个篮子.

可运维性

全面自动化与可视化.

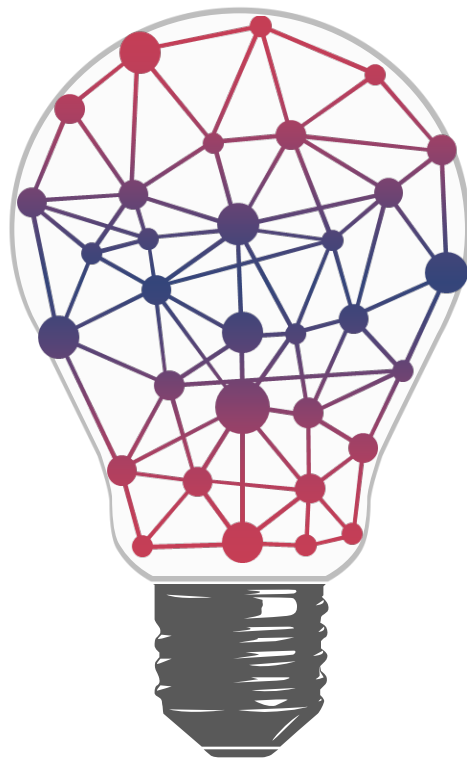


大规模集群，你准备好了吗？

规模估计



稳定加固



瓶颈分析




功能定制




规模预估


Node
1,000



Pod
25,000




Configmap
150,000



以一个1000节点的集群为例

API QPS
8500+



优化从细节着手

api相关参数

各个组件的api qps/burst的设置

节点资源预留

为系统预留cpu和内存

Detail

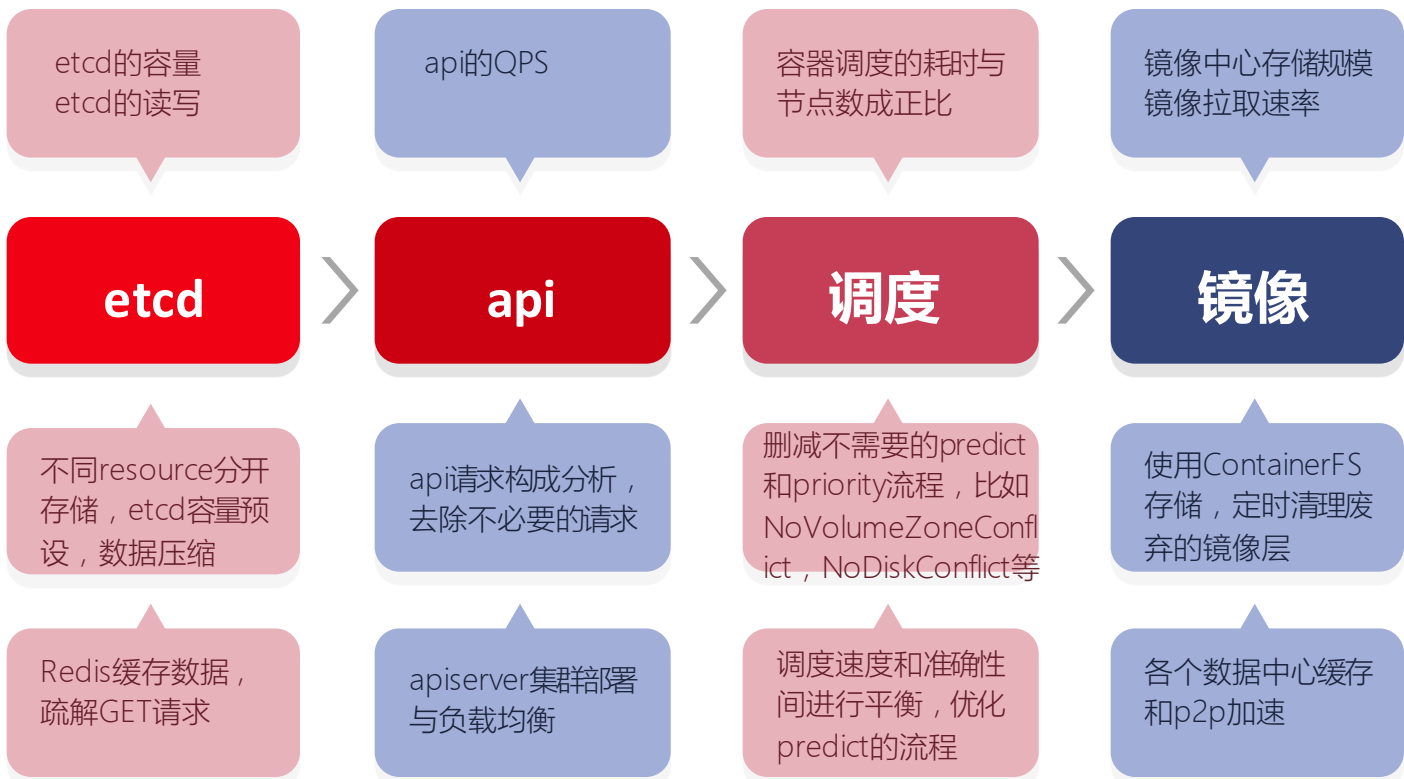
dm尽量使用单独的盘
共用盘时，dm参数需要根据盘容量优化
使用native cgroup driver

Docker的部署优化

系统参数的优化，比如
vm.min_free_kbytes,
net.nf_conntrack_max,
vm.overcommit_memory等

系统参数优化

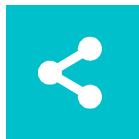
瓶颈分析



etcd/apiserver瓶颈分析与优化

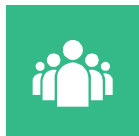
etcd容量监测与压缩

提前做好etcd容量设定，定时对etcd的容量进行巡检，对etcd进行容量压缩以便减少其占用空间



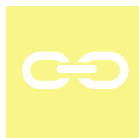
etcd资源分别存储

将请求量和容量较大的event、configmap、pod等资源分别使用不同的etcd进行存储



node心跳摘除

kubelet无需高频上报心跳，使用基于GOSSIP协议的分布式巡检对node状态进行检查，将异常节点直接更新node为notready状态



api请求分析

对apiserver的请求日志进行数据分析，对请求的方法、资源、耗时以及次数进行统计分析。统计结果发现configmap的请求占到了总请求数的99%。其次是node的请求。



GET请求疏解

使用redis进行数据的缓存，将大量的GET请求数据直接从缓存中获取，减少etcd的读压力



apiserver负载均衡

apiserver需要使用LB，并采用最小连接数算法以进行负载均衡。apiserver进行维护时，需要对LB进行停止导流，确定所有apiserver重启成功后，再恢复流量。



瓶颈优化效果

单容器调度
耗时减少



90%

etcd
读写速率降低



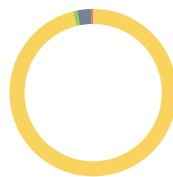
70%

镜像
平均拉取时间缩短



40%

原生
api请求各资源比例



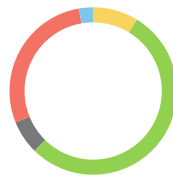
configmaps

endpoints

nodes

events

定制优化后
api请求各资源比例



configmaps

endpoints

nodes

events

原生



定制优化后

1328
116

0 20000 40000 60000 80000 100000

■ 总的请求qps

■ configmap qps

api请求减少

98%

各个节点频繁的remount导致configmap请求数量随着集群规模激增。对apiserver产生直接的压力。

用户根据自己的实际需求，选择使用静态或者动态挂载。静态的configmap不必每次都remount，减轻了api的压力。

稳定性4问

1. 任意组件挂掉/拥塞，会影响已运行的容器么？

挂掉是否会导致现有容器的重建/网络不通等问题
挂掉/拥塞是否会导致自动化的误判或集群雪崩
各个组件是否进行了严格的测试

3. 任意组件异常，都有告警和相应的处理方式么？

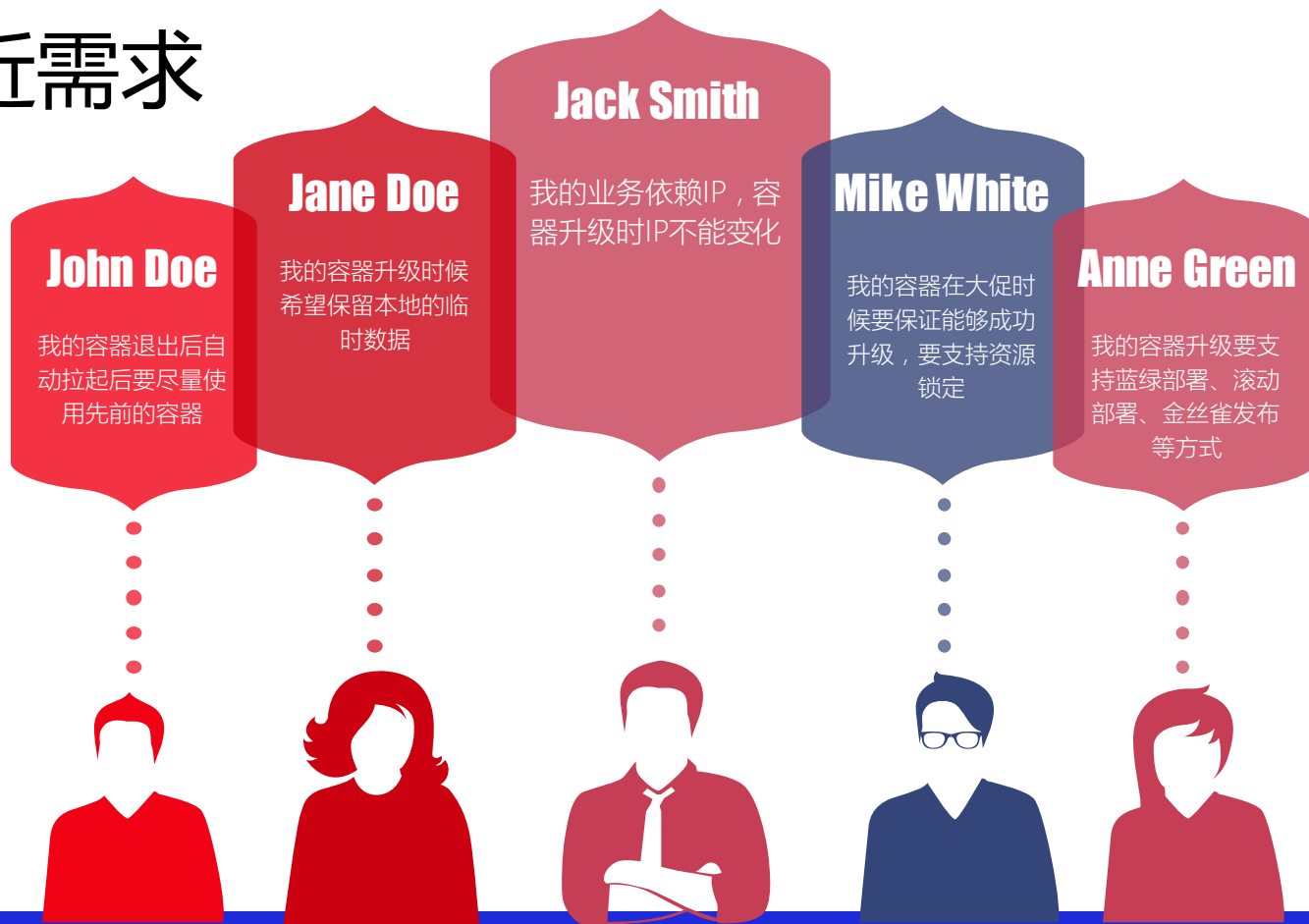
对于组件如何进行监控
如何定义告警规则和方式
如何进行自动化的异常处理

2. 任意组件损坏，集群都能恢复么？

各个组件的高可用与灾备方式
各个组件的故障预案设计与恢复演练

4. 资源隔离问题

贴近需求



功能定制



容器reuse策略

容器优先自动拉起先前退出的容器，而非总是新创容器



IP保持不变

设计IP保留池，以应用为单位进行IP保留。容器删除则IP回池，该应用的容器创建则使用该IP池中的IP



支持容器rebuild

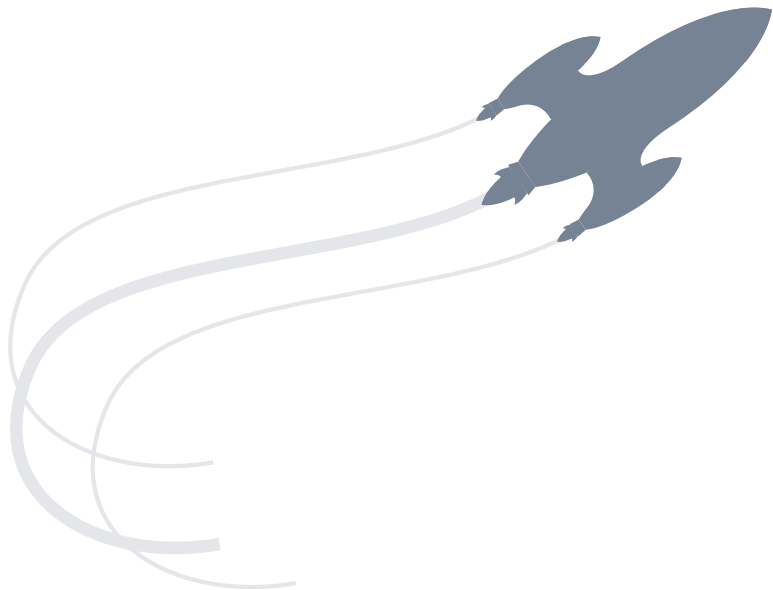
容器修改了镜像、配置文件、存储、环境变量等时，则在当前容器所在节点新启动一个容器，而不用重新调度。并使用原来的数据卷



定制deployment

容器升级可以控制暂停在某个状态，两个版本的容器可以同时存在。并可以在此基础上继续升级或者回滚

Rebuild 创建容器



Local Rebuild Create Pod First

是创建Pod最快 最可靠的方式.

继承之前local数据盘数据.

一些应用的最佳状态路由.

紧急安全上线



- 核心系统紧急上线 速度 是第一位.
- 保留应用之前一些遗留数据 特别是AI具有大量模型文件

集群资源紧张



- 得益于京东阿基米德项目 我们的资源使用率提升明显.
- 有时候buffer也被征用

目录

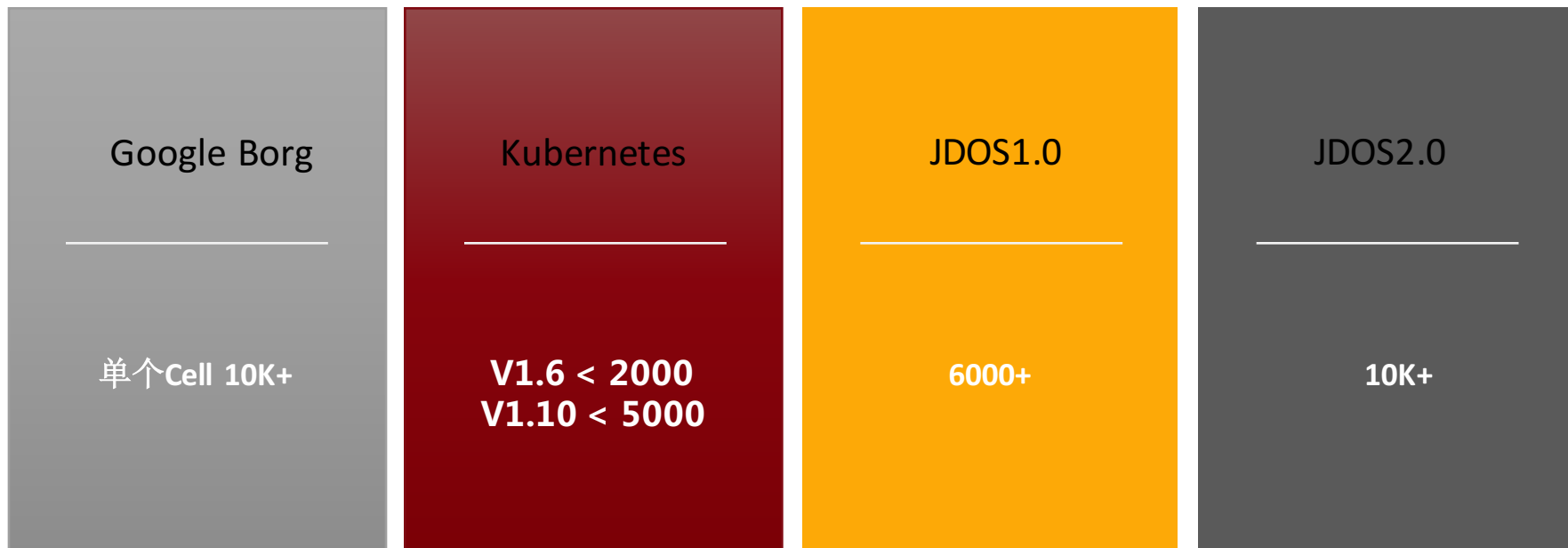
1 Container Eco System

2 化繁为简

➔ 3 大规模集群运营

4 巡检与可视化

集群规模对比



百万级容器网络服务SkyNet

IP分配性能

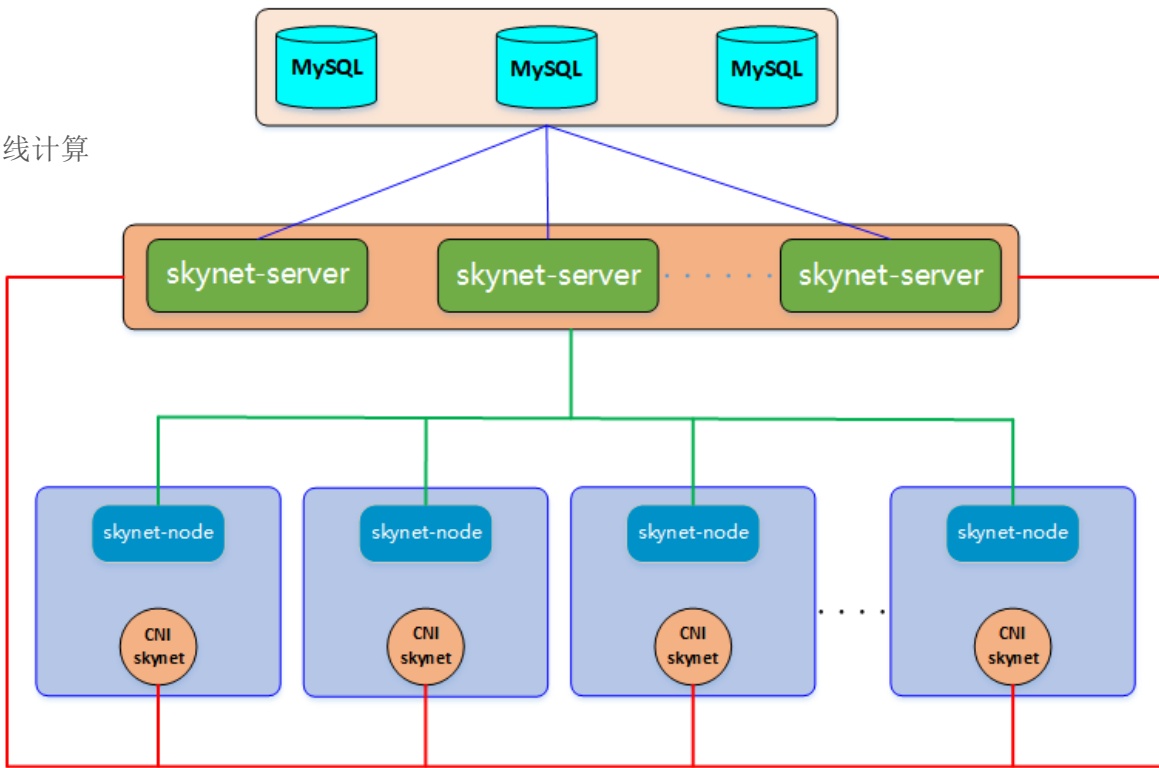
需求：核心应用分组普遍在800容器规模 离线计算

性能：2600 QPS = 15万容器 / 分钟

安全平滑升级

升级网络任何组件没有任何副作用

保持状态一致性



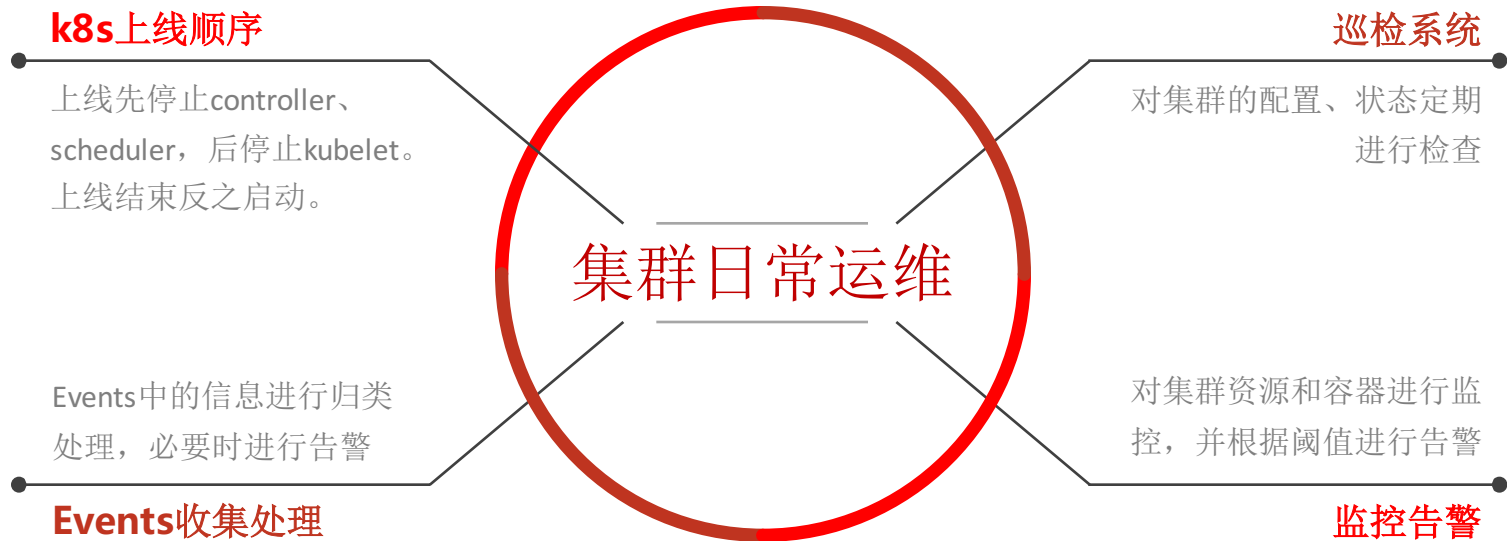
目录

1 Container Eco Sysyem

2 化繁为简

3 大规模集群运营

➔ 4 巡检与可视化



巡检

让生产环境每台Node都是她应该的状态

包含etcd节点的状态检查, 连接数检查,
内存检查, 负载检查等

Etcd
巡检

控制节点
巡检

包含磁盘使用情况检查, 服务检查,
内存检查, 负载检查等

检查点更为详细, 包括了kubelet版本、监控状态
检查, 资源检查, 内核参数检查等

计算节点
巡检

其他服务
巡检

网络服务, 负载均衡服务, 监控服务等
也均纳入统一巡检中

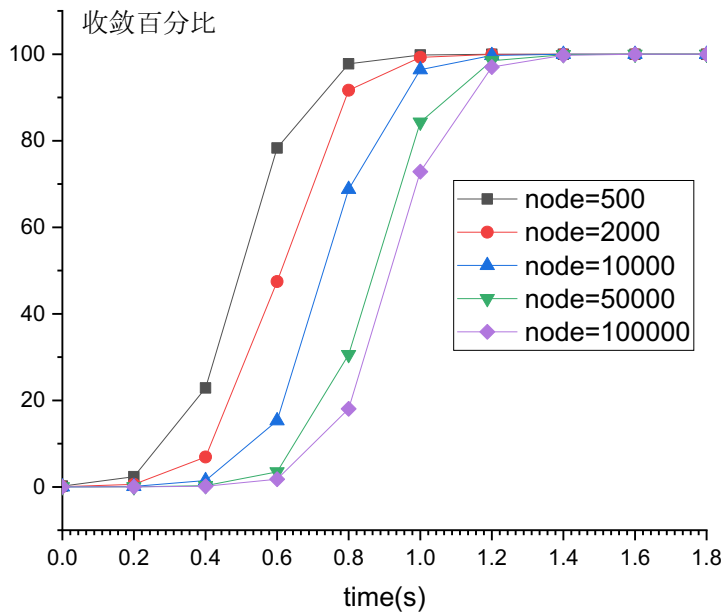
JDOS2.0 etcd节点巡检(错误:无 告警:无 警告:无)				
发件人:				
收件人:				
时间:				
大小:	30 KB			
巡检简报				
检查点	Error统计	Alarm统计	Warn统计	Info统计
etcd_status检查	0	0	0	3
etcd_健康状态检查	0	0	0	3
etcd_连接数检查	0	0	0	3
系统负载检查	0	0	0	3
内存使用率检查	0	0	0	3
巡检详情				

JDOS2.0 控制节点巡检(错误:无 告警:无 警告:无)				
发件人:				
收件人:				
时间:				
大小:				
巡检简报				
检查点	Error统计	Alarm统计	Warn统计	Info统计
/export使用率检查	0	0	0	3
etcd连接数检查	0	0	0	3
系统负载检查	0	0	0	3
内存使用率检查	0	0	0	3

JDOS2.0 计算节点巡检(错误:有 告警:有 警告:无)				
发件人:				
收件人:				
时间:				
大小:	47 KB			
巡检简报				
检查点	Error统计	Alarm统计	Warn统计	Info统计
kubelet_进程检查	0	0	0	1832
kubelet_健康检查	0	0	0	1832
/export使用率检查	0	0	0	1832
进程数检查	0	0	0	1832
cpu时间误差检查	1	0	0	1831
系统日志级别检查	0	0	0	1832
系统内核版本检查	0	0	0	1832
数据网卡cpu错误检查	0	0	0	1832
cadvisor_进程检查	0	3	0	1829
文件句柄数检查	0	0	0	1832
kernel.pid_max值检查	0	0	0	1832
fstab检查	0	0	0	1832
nf_conntrack检查	0	0	0	1832
内存使用率检查	0	0	0	1832
负载检查	0	0	0	1832
kubelet_版本检查	0	0	0	1832
cadvisor_cgroup_memory检查	0	0	0	1832

几十万计算节点如何做信息同步

- Gossip
- Serf



01

状态、数据直观感受

将数据通过图表等形式进行展示，
使得运维人员对集群的状态和性能
有更加直观的感受和体验

可以清晰看到数据变化的趋势，并
结合数据分析集群目前可能存在的
问题和遇到的瓶颈，以及对比优化
前后的对比

瓶颈、问题一目了然

02

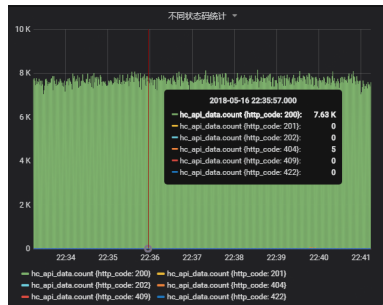
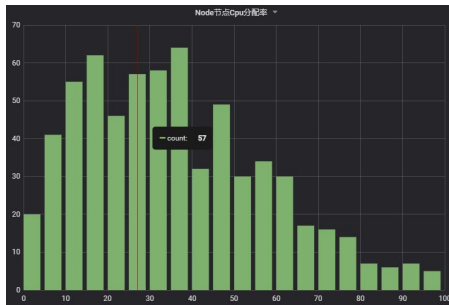
可视化

集群状态实时/历史数据

集群的节点/可用节点/可用资源数量

容器总数随时间的变化

各个节点的不同资源的分配率



集群组件性能数据

API Server的请求数，返回码，以及响应时间

Scheduler调度容器的时间

Kubelet的拉取镜像时间，容器创建时间

Node上资源分配率

通过统计各个节点上的cpu/内存/磁盘分配情况，确定集群的资源瓶颈

API请求统计

统计各个请求、返回码以及响应时间，反映apiserver集群的状态

调度用时

通过展示每个容器调度的平均用时，反映描述调度器的性能



Thanks

DevOps 时代社区 荣誉出品

想第一时间看到高效运维社区的
最新动态吗？

