

微博峰值应对系统DCP在混合云方面架构实践

Geekbang>

极客邦科技

整合全球最优质学习资源, 帮助技术人 and 企业成长
Growing Technicians, Growing Companies

InfoQ
ueue

专注中高端技术人员的技术媒体



EGO EXTRA GEEKS' ORGANIZATION
NETWORKS

高端技术人员
学习型社交网络



StuQ
ueue

实践驱动的
IT 职业学习和服务平台



GiT GEEKBANG
INTERNATIONAL
TRAINING
极客邦培训

一线专家驱动的
企业培训服务



旧金山 伦敦 北京 圣保罗 东京 纽约 上海
San Francisco London Beijing Sao Paulo Tokyo New York Shanghai

QCon

全球软件开发大会

2016年4月21-23日 | 北京·国际会议中心

主办方 **Geekbang**  **InfoQ**
极客邦科技

7折 优惠 (截至12月27日)
现在报名, 节省2040元/张, 团购享受更多优惠

www.qconbeijing.com



扫描获取更多大会信息

自我介绍

微博平台技术经理

DCP项目

微博关系架构

Feed高可用架构实现



分享主要内容

一、DCP整体介绍

二、弹性集群

三、弹性调度

四、服务发现





Part 1

DCP整体介绍

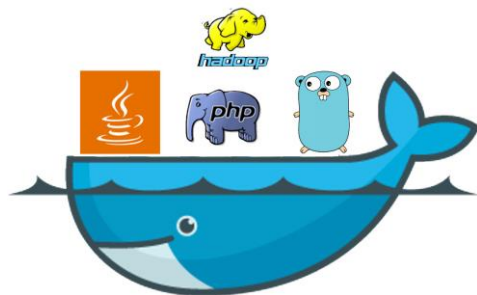
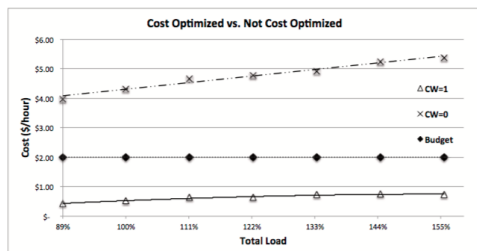
DCP架构演进



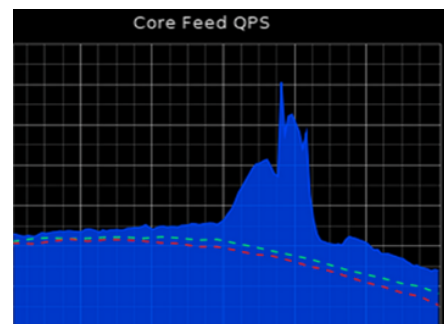
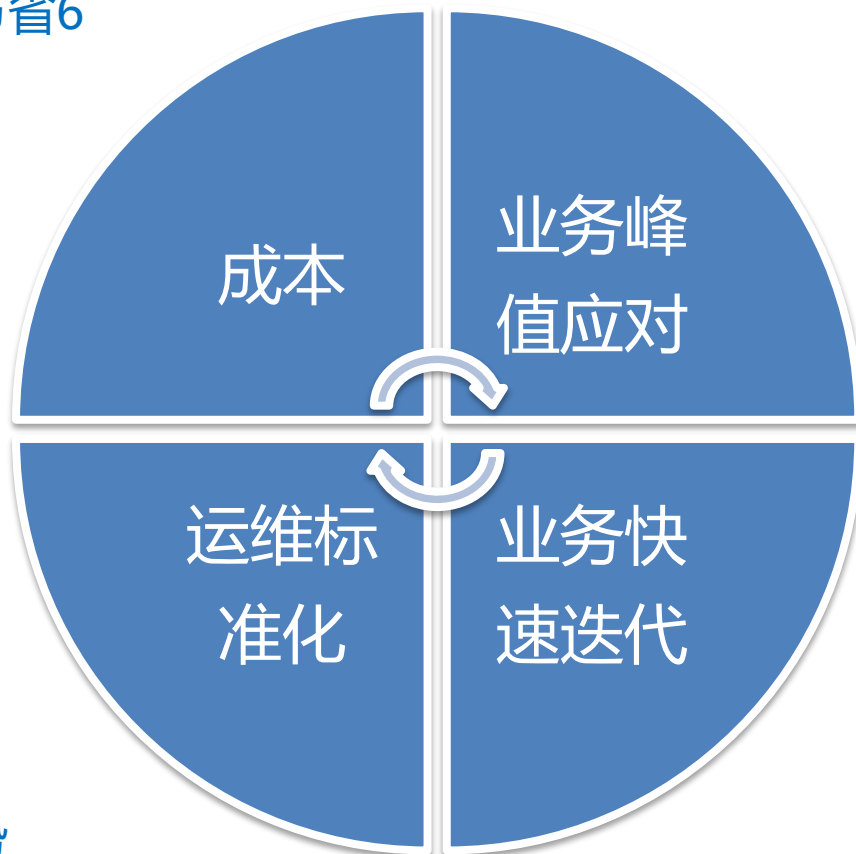
DCP项目出发点

- 可伸缩的业务利用公有云
- 混合云调度理论能节省6倍成本

- 快速扩容
- 及时回收



- 拉通多语言环境
- 发挥运维规模化优势

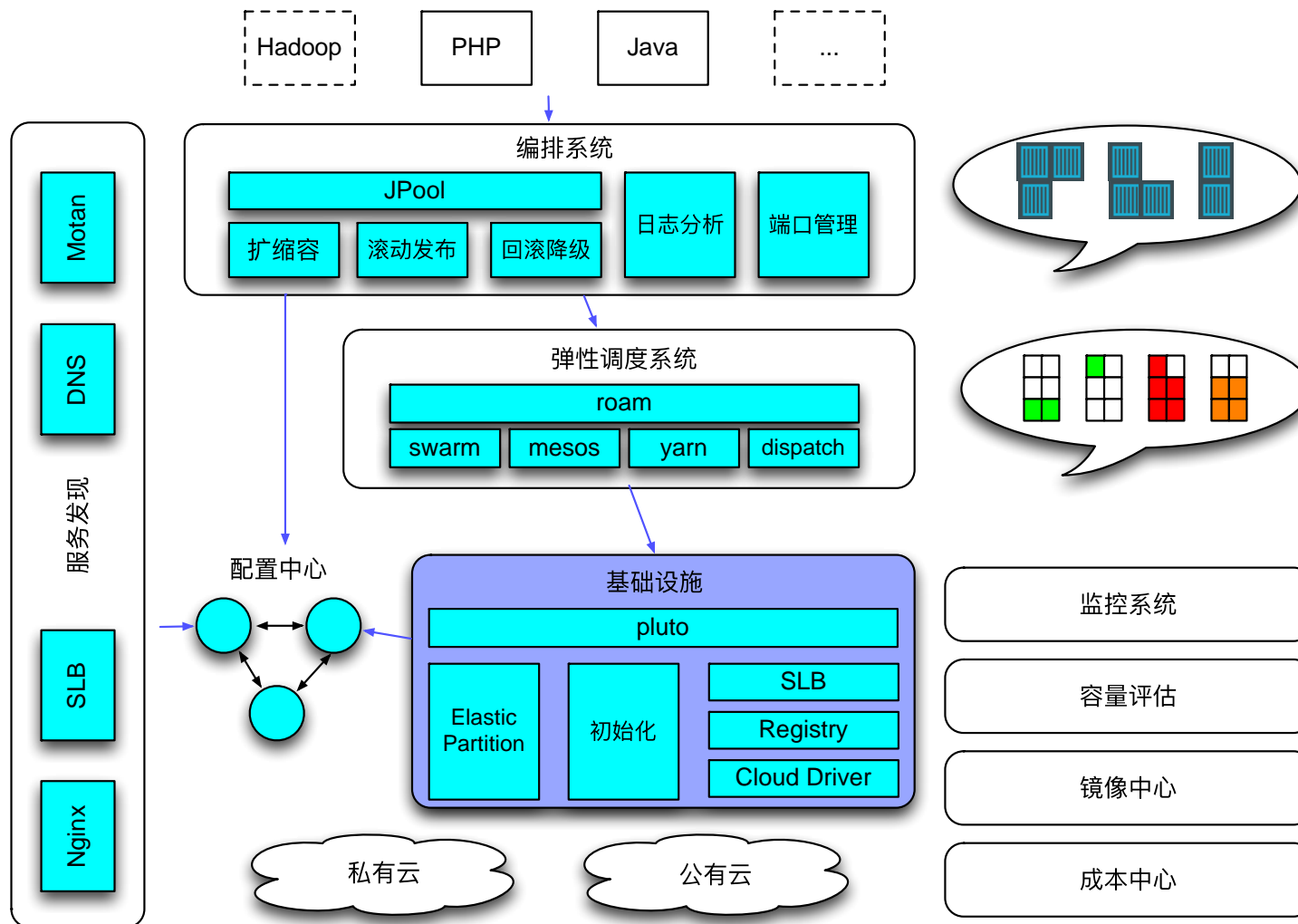


- 享受标准化基础设施红利

DCP主要思路



DCP整体结构

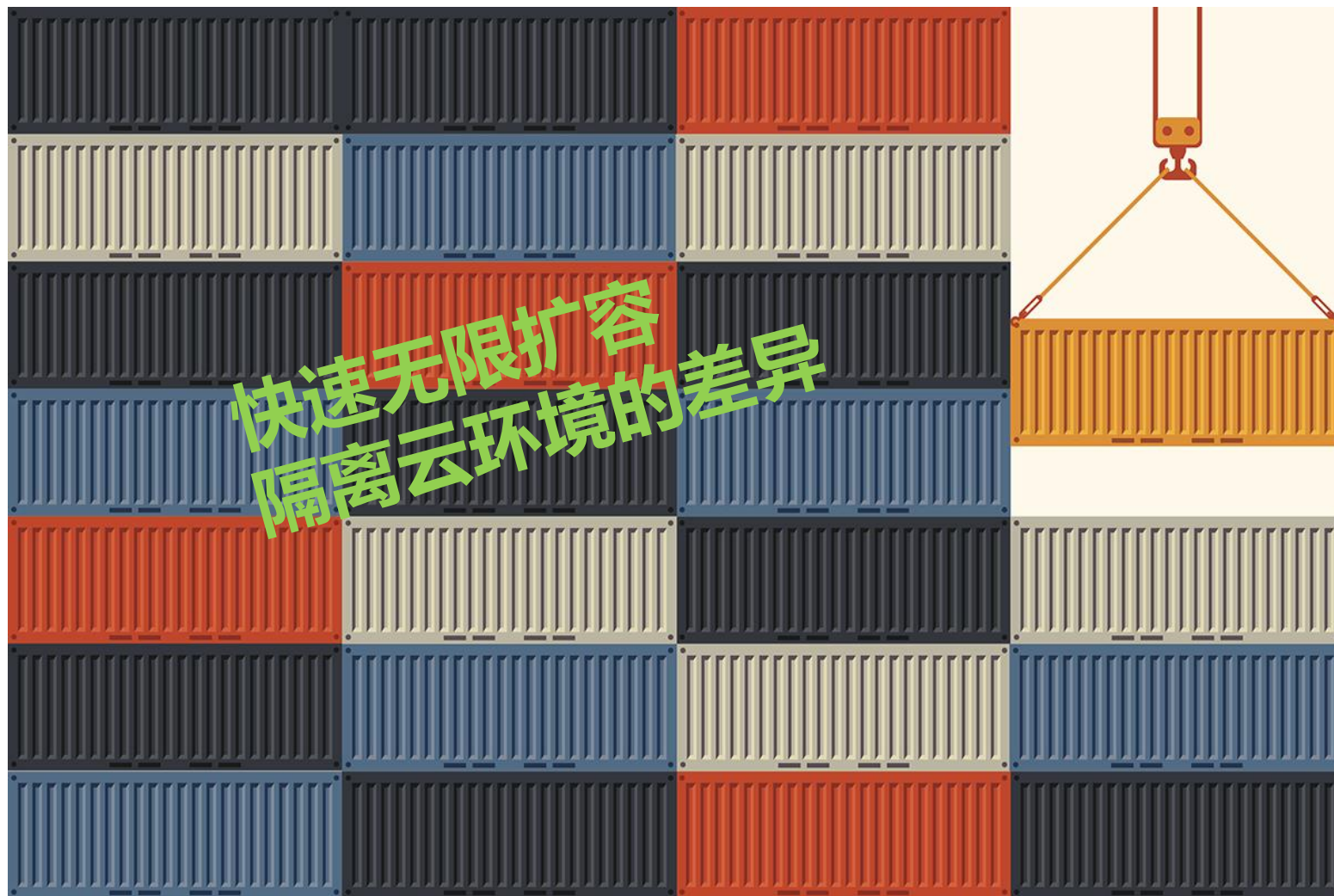




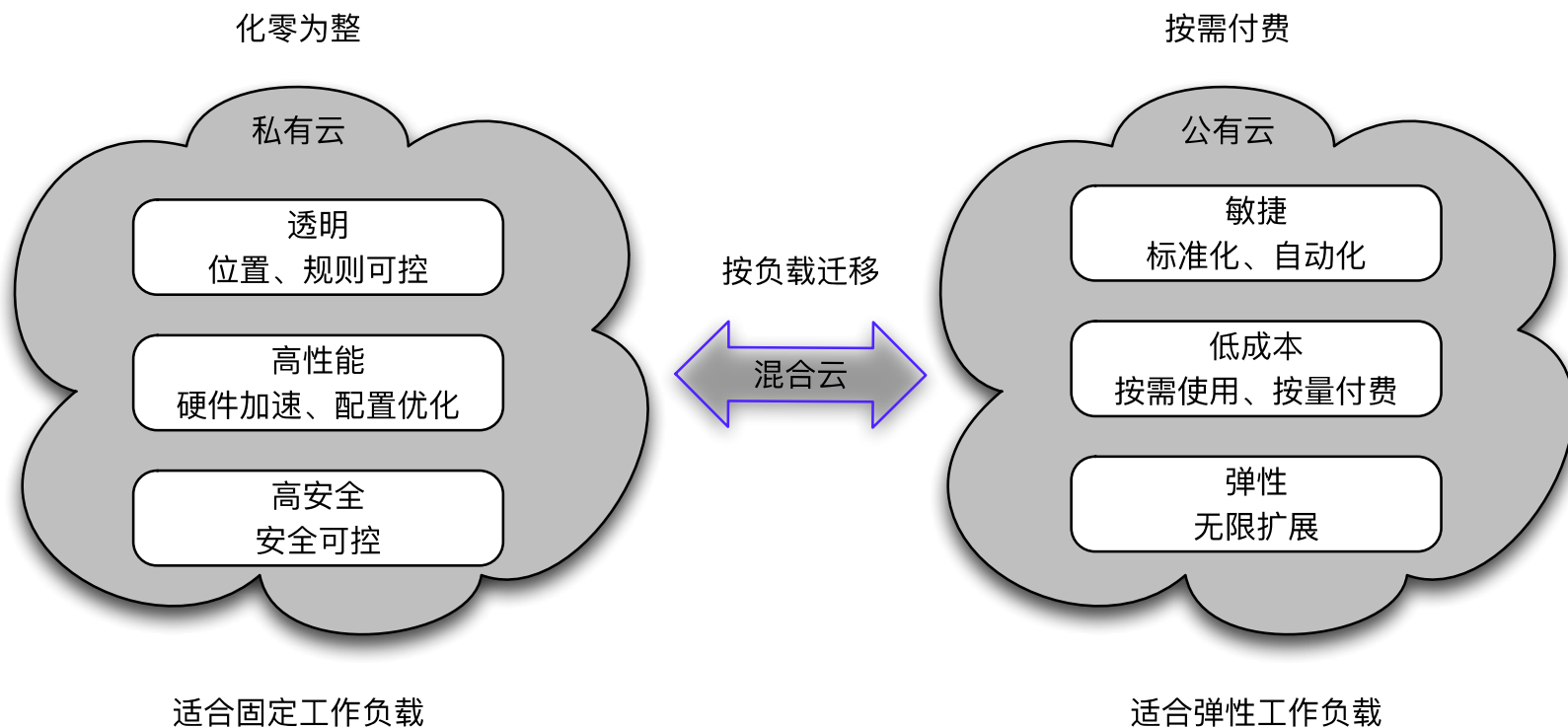
Part 2

弹性集群

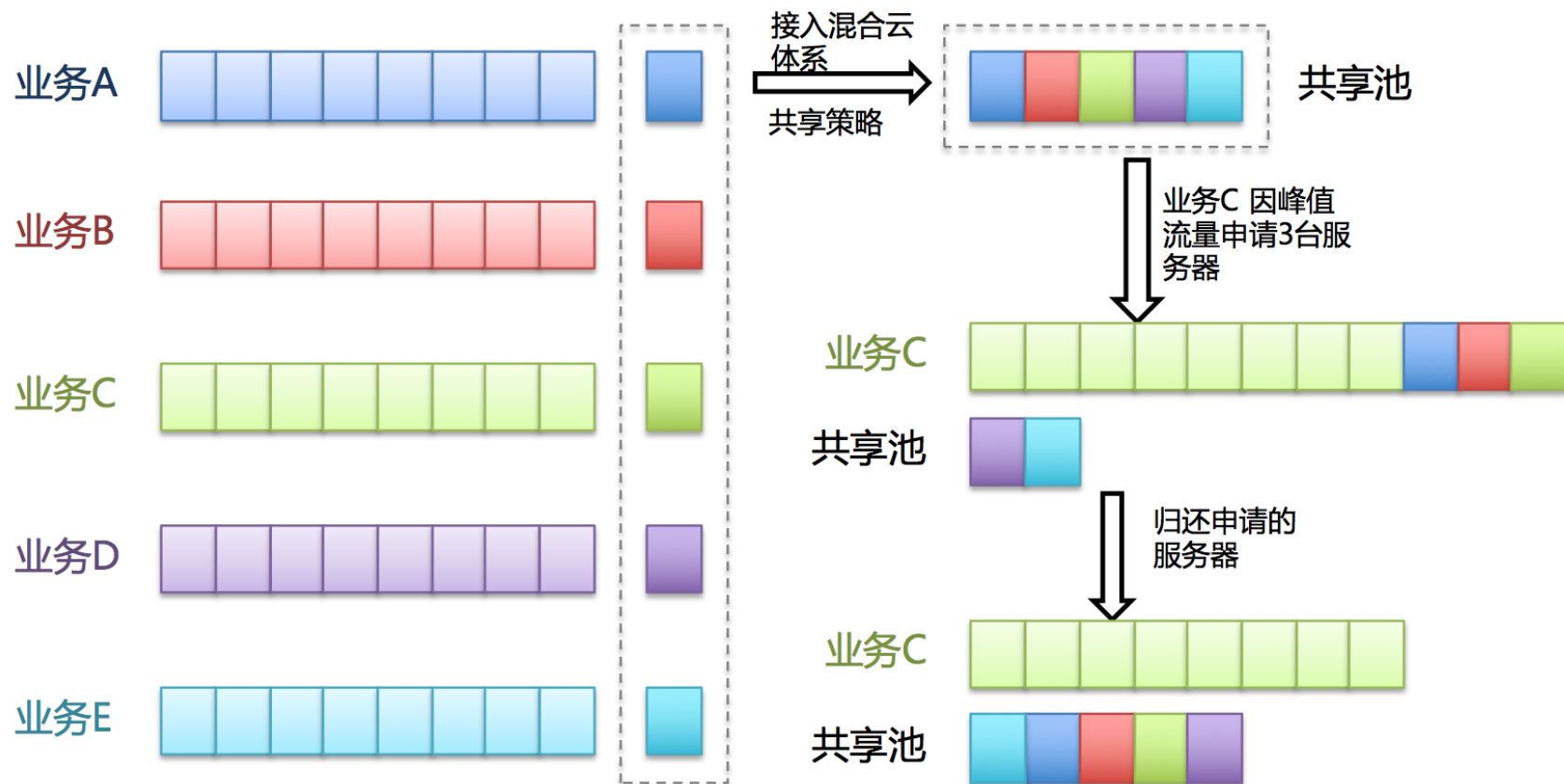
扩！扩！扩！



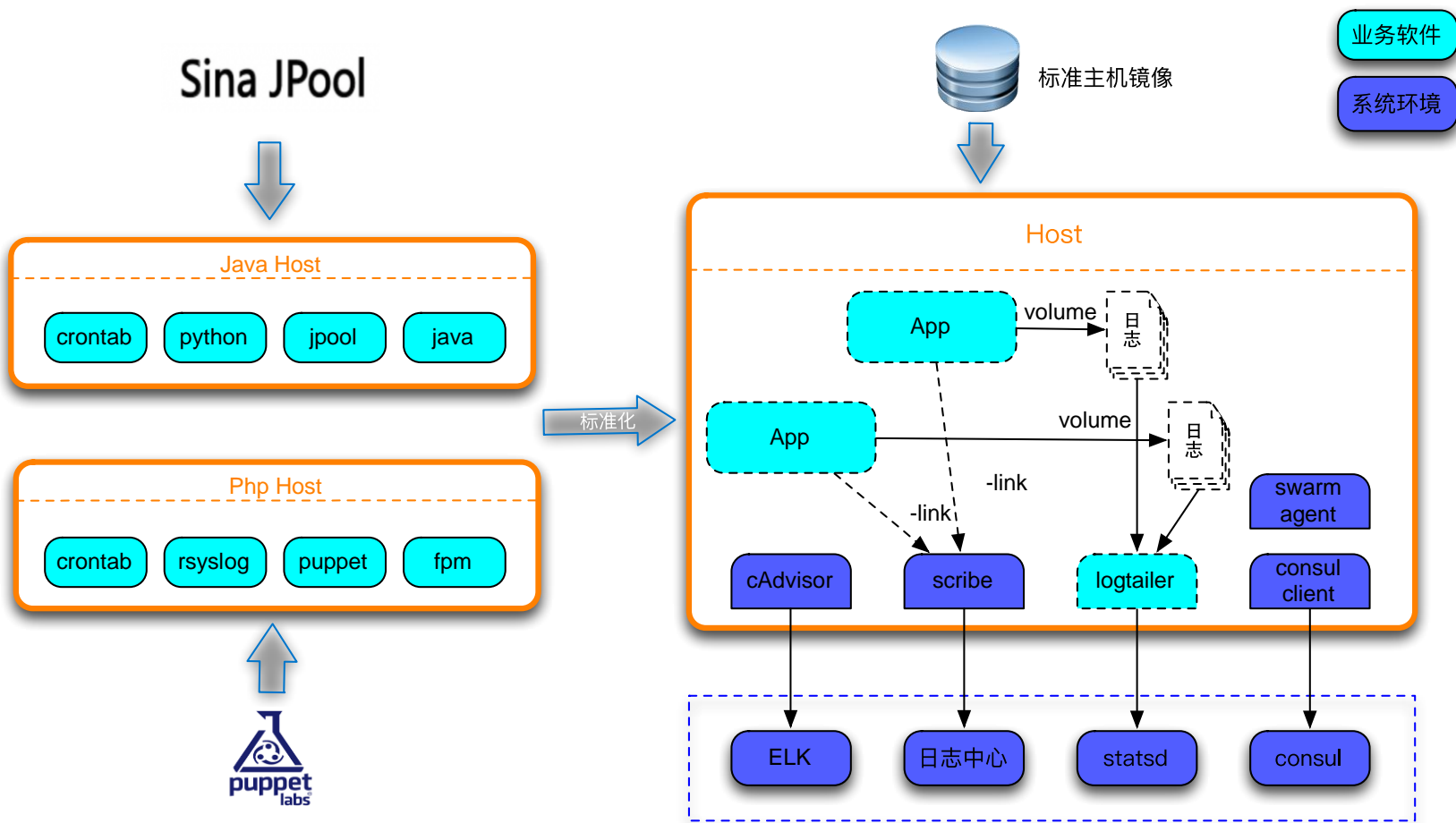
DCP基础设施弹性策略



私有“云” 化零为整



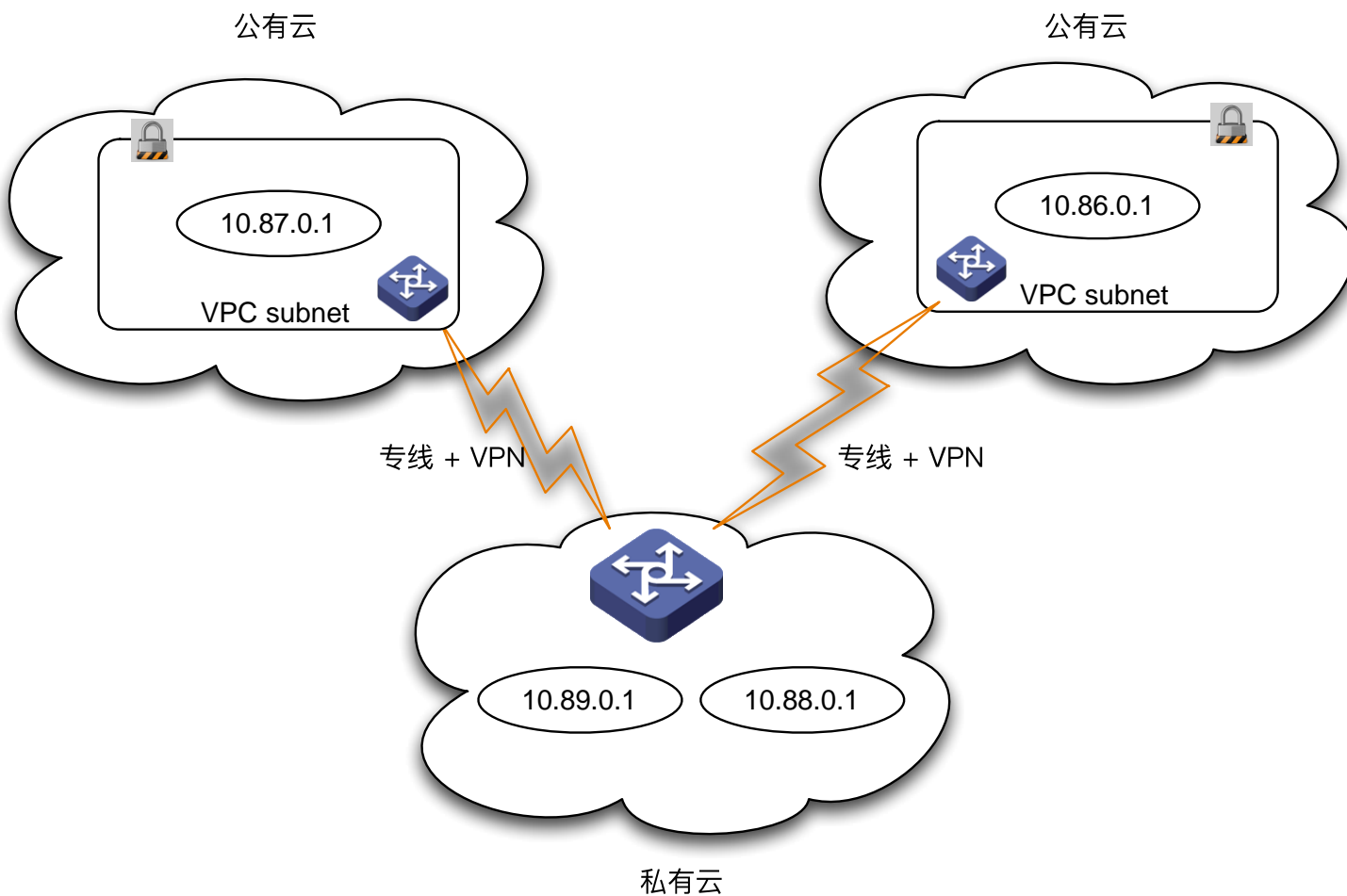
打破差异 - 标准化运行环境



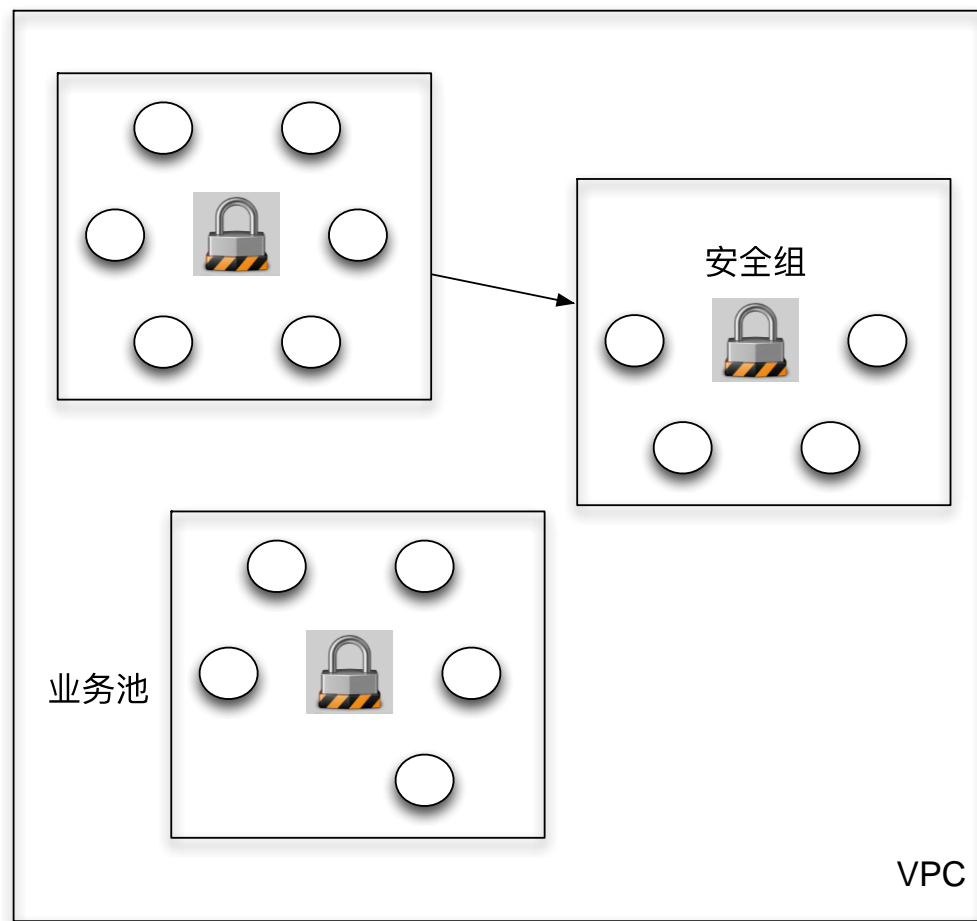
基础环境软件版本

Java		php	
cAdvisor 0.7.1.fix			
Mesos 0.25			
Swarm 1.0.0			
Consul 0.5.2			
Docker 1.6.2	Registry v2	Daemon Wrapper	Registry v1
devicemapper-direct-lvm		Docker 1.3.2	
CentOS 7.1.1503/3.10.0-229.el7.x86_64		devicemapper-loop-lvm	
		CentOS 6.6/2.6.32	

DCP网络高可用



DCP安全隔离



高可用问题 – 不怕死就怕慢

```
90 90 55 48 89 e5 53 48 89 fb 48 83 ec 08 48 8b 47 08 <4c> 8b 00 4c 39 c7 75 39
48 8b 03 4c 8b 40 08 4c 39 c3 75 4c 48
RIP [<ffffffff81294ee0>] list_del+0x10/0xa0
RSP [<ffff8001e9f87dc8>]
---[ end trace 5a299f957d120f7e ]---
Kernel panic - not syncing: Fatal exception
Pid: 1553, comm: docker Tainted: G      D      2.6.32-431.23.3.el6.x86_64 #1
Call Trace:
[<ffffffff815284fc>] ? panic+0xa7/0x16f
[<ffffffff8152c824>] ? oops_end+0xe4/0x100
[<ffffffff81010e0b>] ? die+0x5b/0x90
[<ffffffff8152c302>] ? do_general_protection+0x152/0x160
[<ffffffff8152bad5>] ? general_protection+0x25/0x30
[<ffffffff81294ee0>] ? list_del+0x10/0xa0
[<ffffffff810c9e12>] ? cgroup_event_wake+0x42/0x70
[<ffffffff810546b9>] ? __wake_up_common+0x59/0x90
[<ffffffff81058bc8>] ? __wake_up+0x48/0x70
[<ffffffff811d56ed>] ? eventfd_release+0x2d/0x40
[<ffffffff8118a715>] ? __fput+0xf5/0x210
[<ffffffff8118a855>] ? fput+0x25/0x30
[<ffffffff81185a9d>] ? filp_close+0x5d/0x90
[<ffffffff81185b75>] ? sys_close+0xa5/0x100
[<ffffffff8100b072>] ? system_call_fastpath+0x16/0x1b
```

慢了！怎么办？

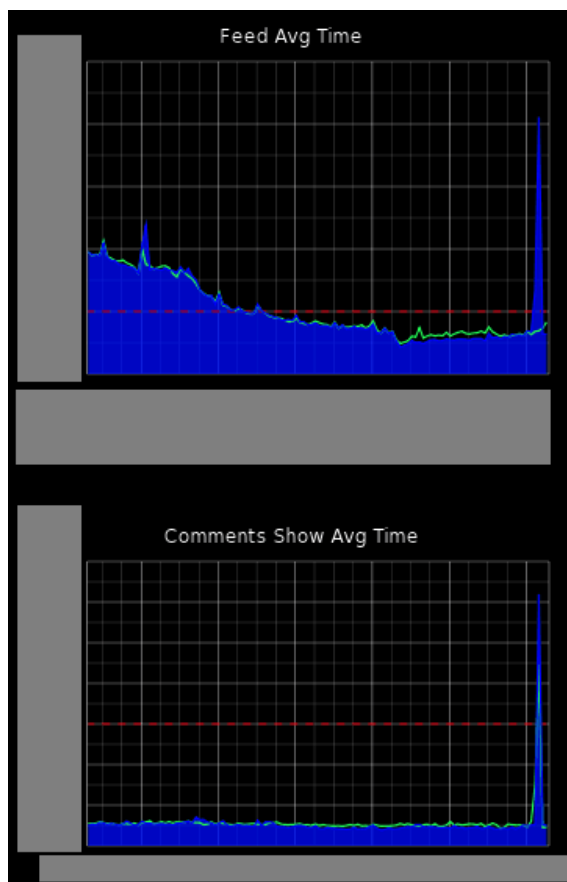
- 有可能是业务慢
- 有可能是机器慢
- 有可能是网络慢

死了，可自动补充：

1. 根据镜像新建设备
2. 调度系统重建业务容器

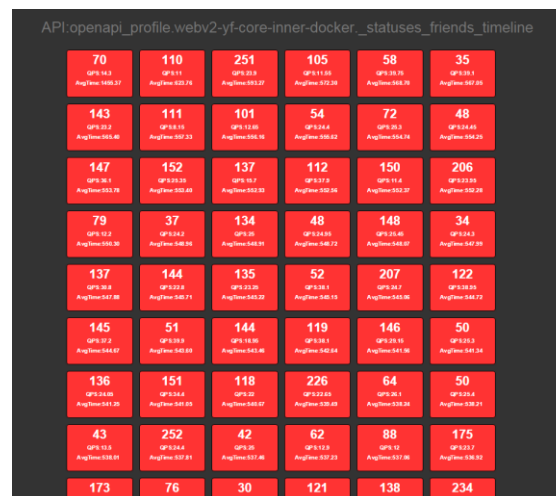


监控从全网平均到单机全覆盖



往往性能恶化是由于单机问题引起
如何快速定位有问题的机器？

单机性能恶化



服务所在位置:/data1/friendship

System Load:0.01

Tomcat Load:0.01

Swap使用比例:0.42%

非堆使用比例:70.28%

10秒内HTTP请求次数:508

10s内错误响应次数:332

10s内超过500ms的响应次数:1

Tomcat繁忙线程比例:0.0%

系统当前停顿时间:0ms

系统10分钟内停顿次数:0

系统10分钟内YGC次数:1

GC后Old区使用比例:27.66%

GC后Old区减少比例:-0.56%

DirectBuffer数量:1263

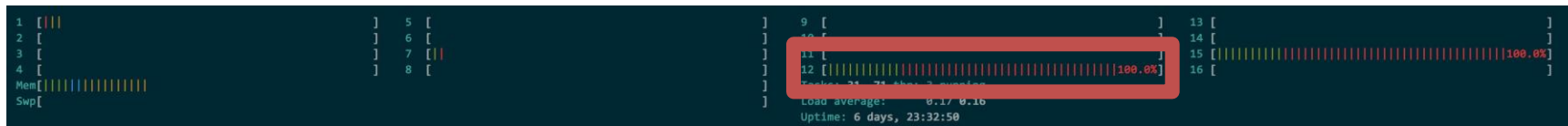
DirectBuffer池总的容量:52.589467MB

DirectBuffer池使用的容量:52.589467MB

业务容器级报警

高可用问题- 公有云单机性能瓶颈

单机性能问题



测试CPU：

`sysbench --test=cpu --cpu-max-prime=10000 run`

`sysbench --test=threads --num-threads=64 --thread-yields=2000 --thread-locks=2 run`

测试磁盘：

`fio -direct=1 -iodepth=64 -rw=randwrite -ioengine=libaio -bs=16k -size=10G -
numjobs=1 -runtime=1000 -group_reporting -name=/path/testfile`

测试内存：

`mbw -q -n 10 256`

测试带宽：

`netperf -H host -l second -t [TCP_STREAM|UDP_STREAM|TCP_RR|TCP_CRR|UDP_RR] -s
localBufferSize -S remoteBufferSize -m localPackageSize -M remotePackageSize -D
TCP_NODELAY`

测试业务

高可用问题 - 自由也是责任

<input type="checkbox"/> 实例ID/名称	监控	所在可用区	IP地址	状态(全部) ▼	网络类型(全部) ▼	配置	标签	专有网络属性
<input type="checkbox"/> [REDACTED]		北京可用区A	[REDACTED]	● 运行中	专有网络	CPU: 4核 内存: 8192 MB		
<input type="checkbox"/> [REDACTED] 06		北京可用区A	[REDACTED]	● 运行中	专有网络	CPU: 1核 内存: 2048 MB		
<input type="checkbox"/> [REDACTED]		北京可用区A	[REDACTED]	● 运行中	专有网络	CPU: 1核 内存: 2048 MB		
<input type="checkbox"/> [REDACTED] 09		北京可用区A	[REDACTED]	● 运行中	专有网络	CPU: 1核 内存: 2048 MB		
<input type="checkbox"/> [REDACTED] 08		北京可用区A	[REDACTED]	● 运行中	专有网络	CPU: 1核 内存: 2048 MB		

不要向开发、运维开放console权限

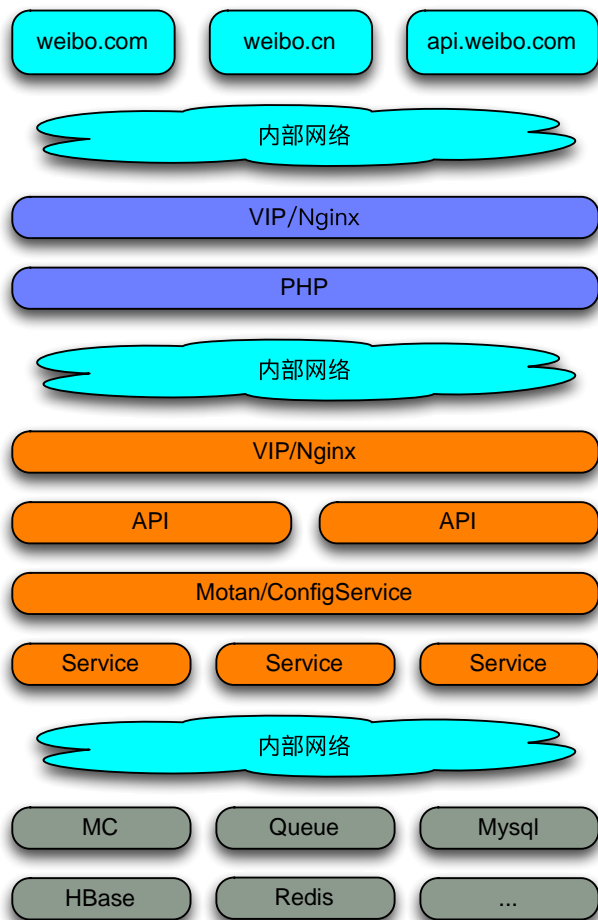




Part 3

弹性调度

弹性调度的挑战 - 重型服务调度难



内网服务架构

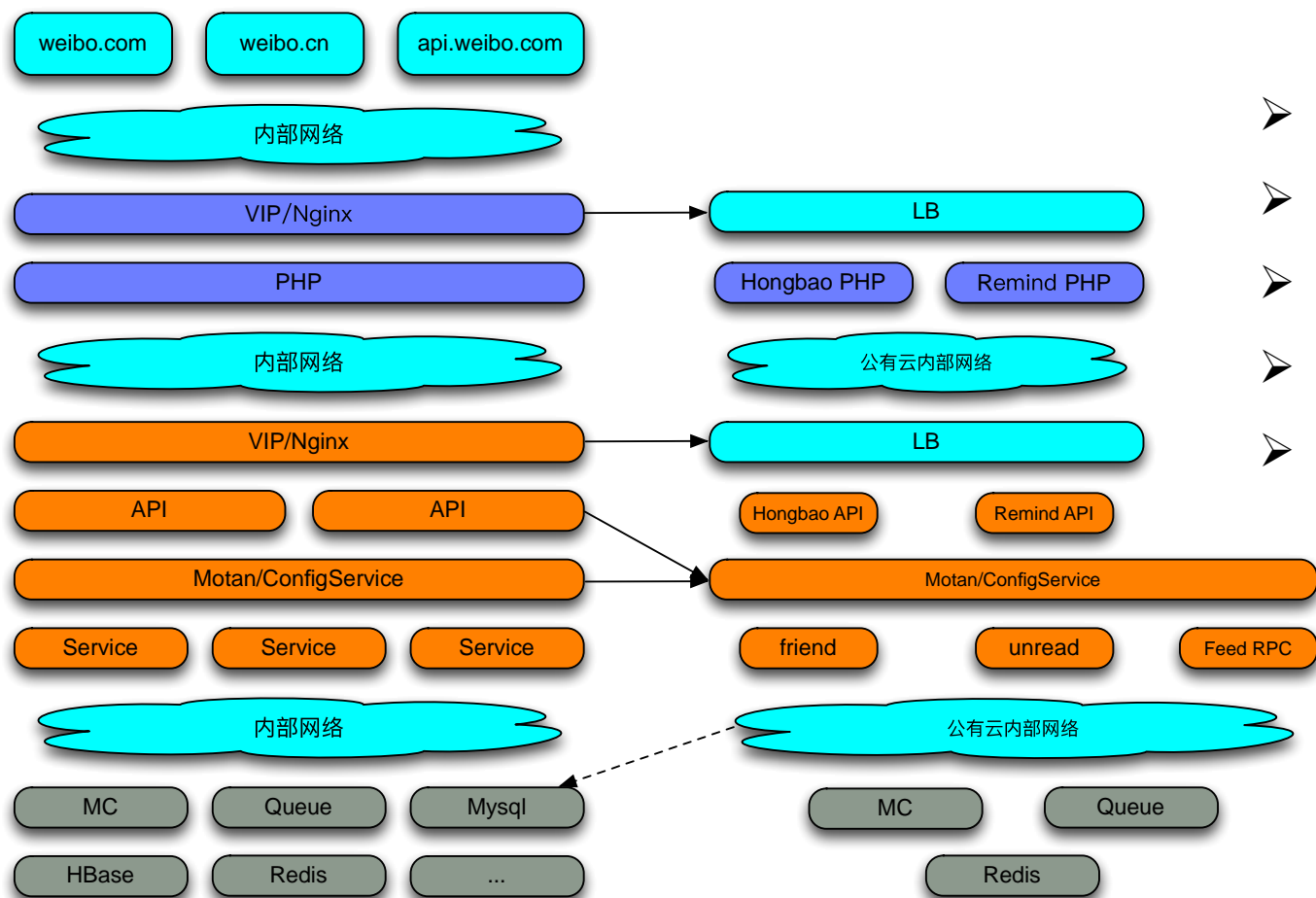


- 依赖关系复杂
- 跨云专线带宽是稀缺资源



- 微服务化，尽量在一个云上完成处理，减少穿透
- 2/8原则，往往一个服务里只有20%的部分需要弹性，而且能解决80%问题

微业务化 - 以最少的资源实现最大的弹性

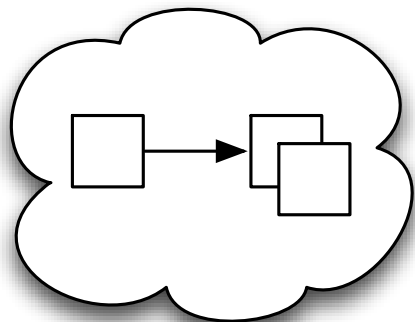


- 红包飞: 依赖聚合
- 提醒: 依赖解耦
- Feed: 算法拆分
- 缓存: 本地化
- 数据库: 仅私有云

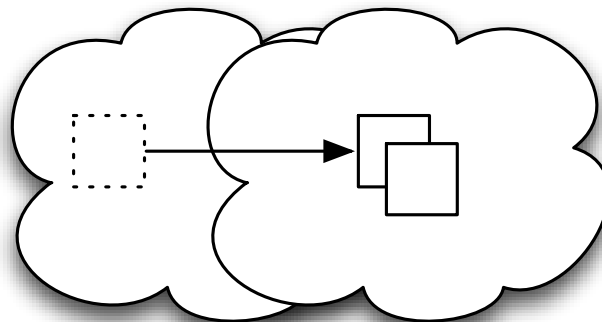


混合云弹性扩容方式

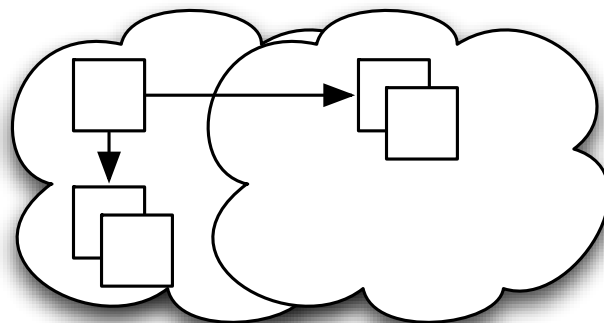
私有内弹性扩容



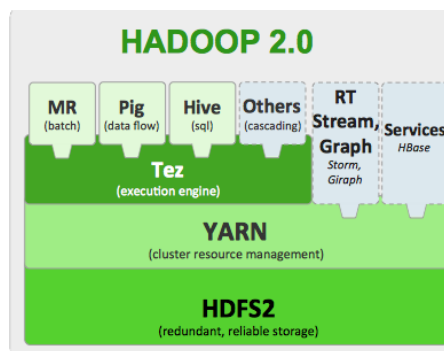
扩容到公有云弹性



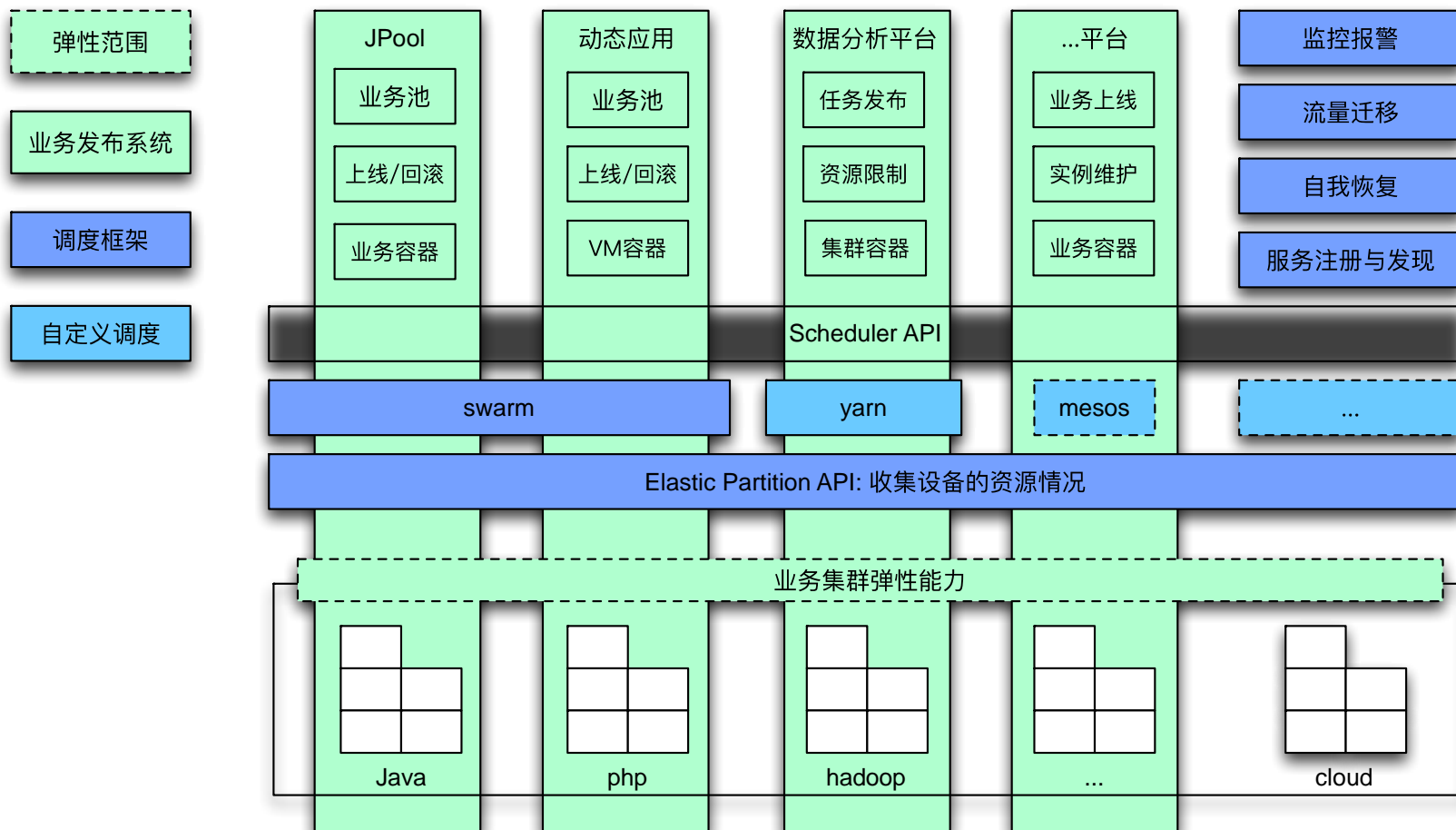
私有云、公有云同时弹性扩容



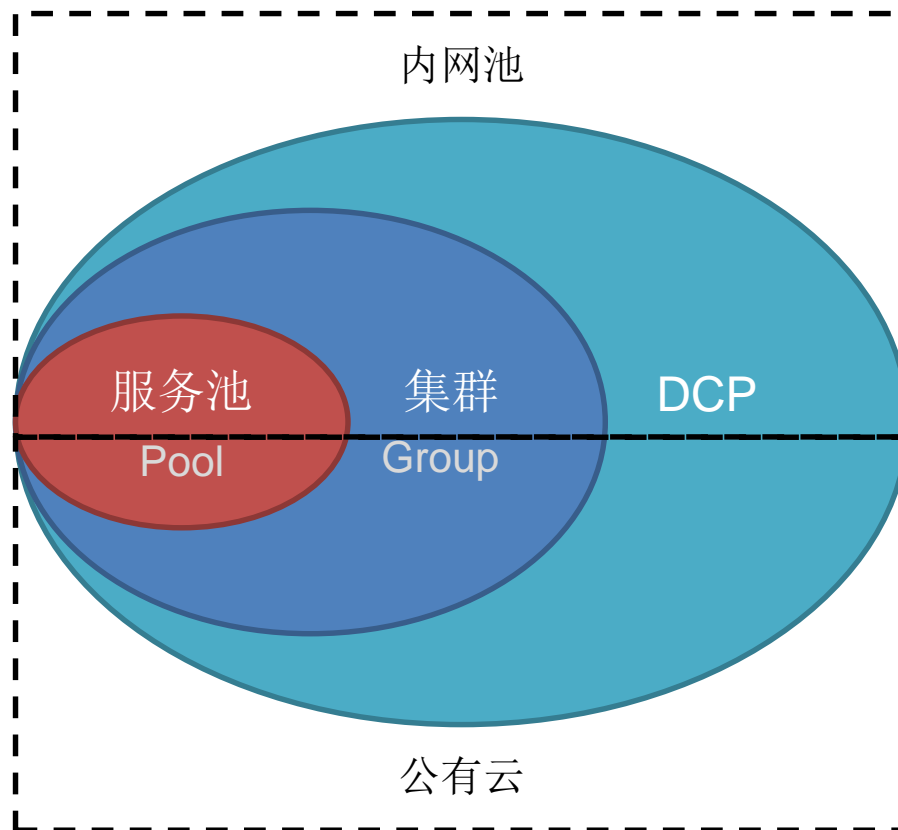
业界：从Static Partitioning到Elastic Sharing



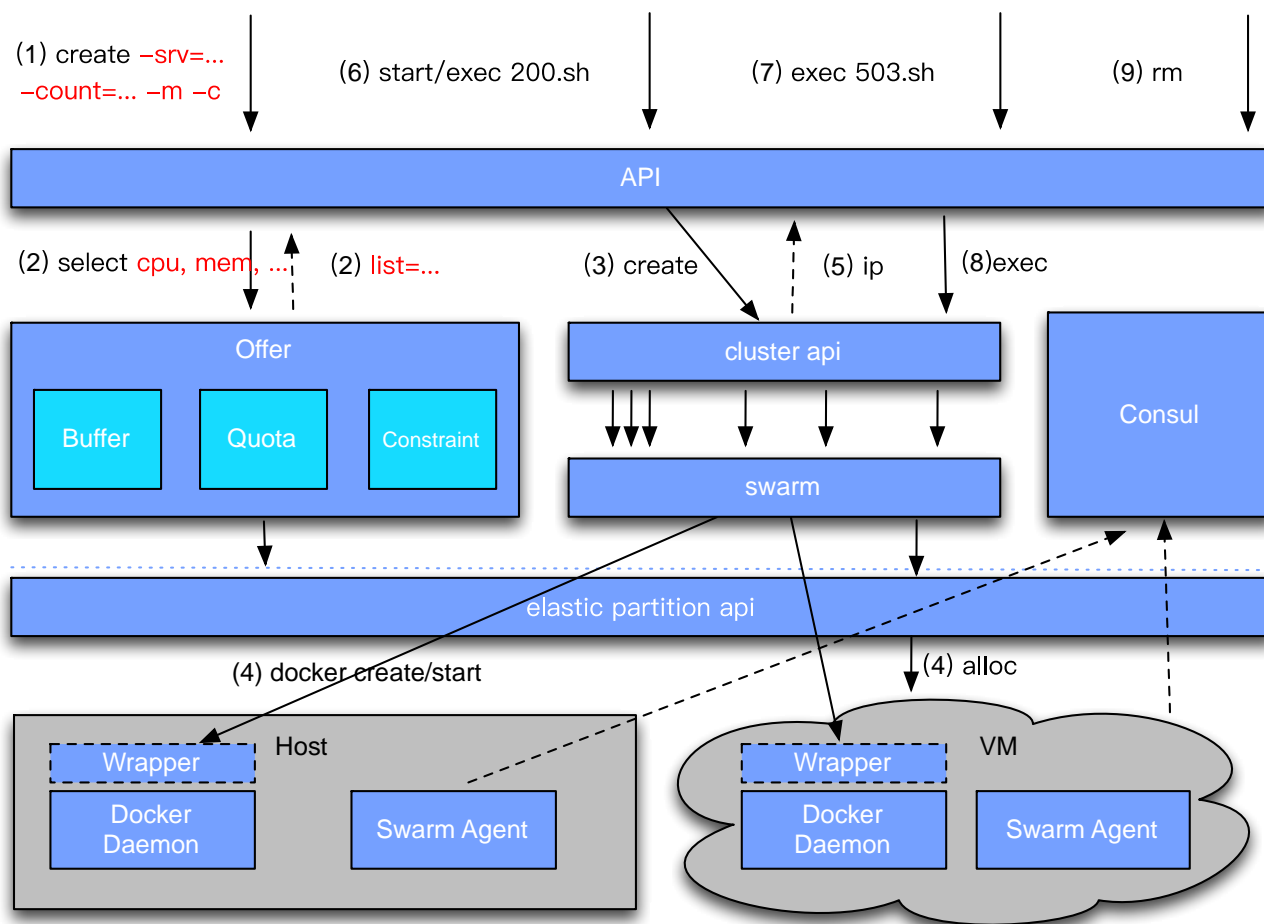
DCP弹性调实现 – Elastic Partition



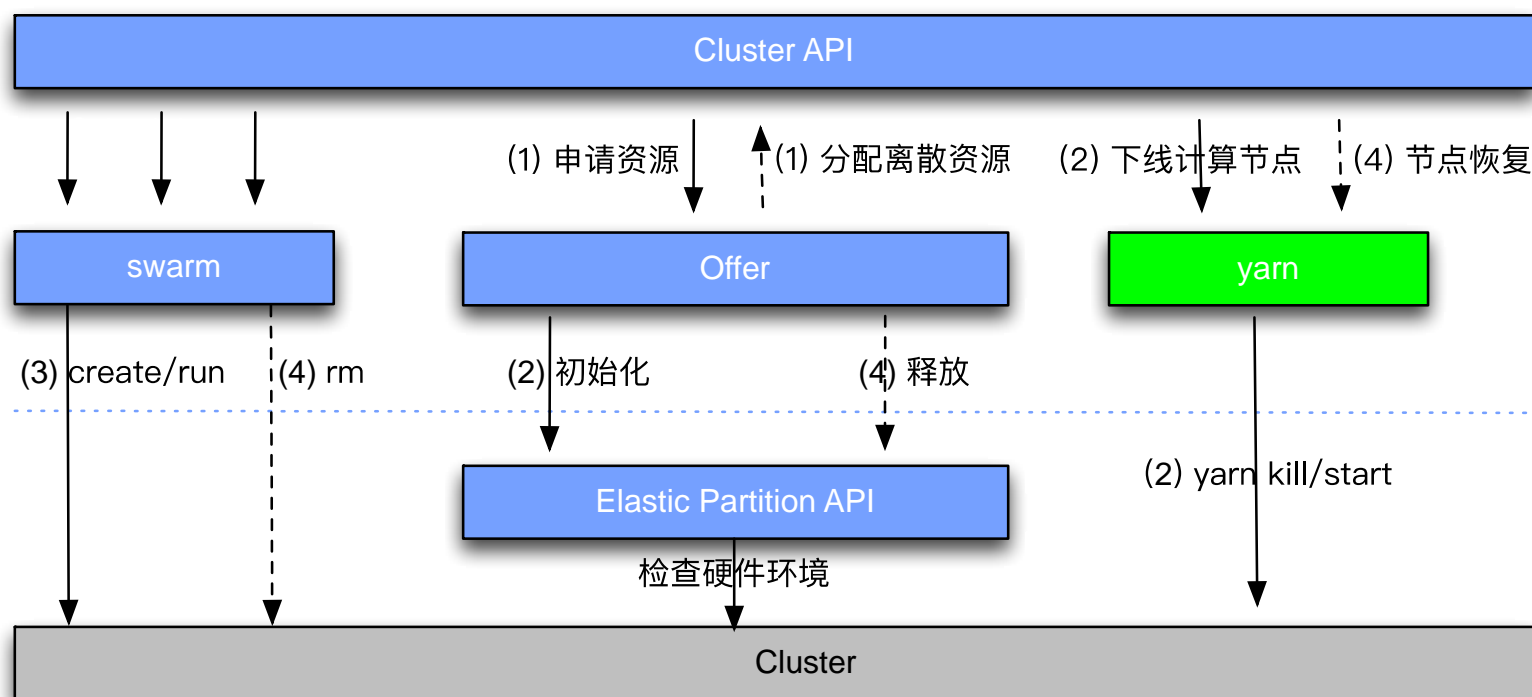
Partition分布 - 业务视角



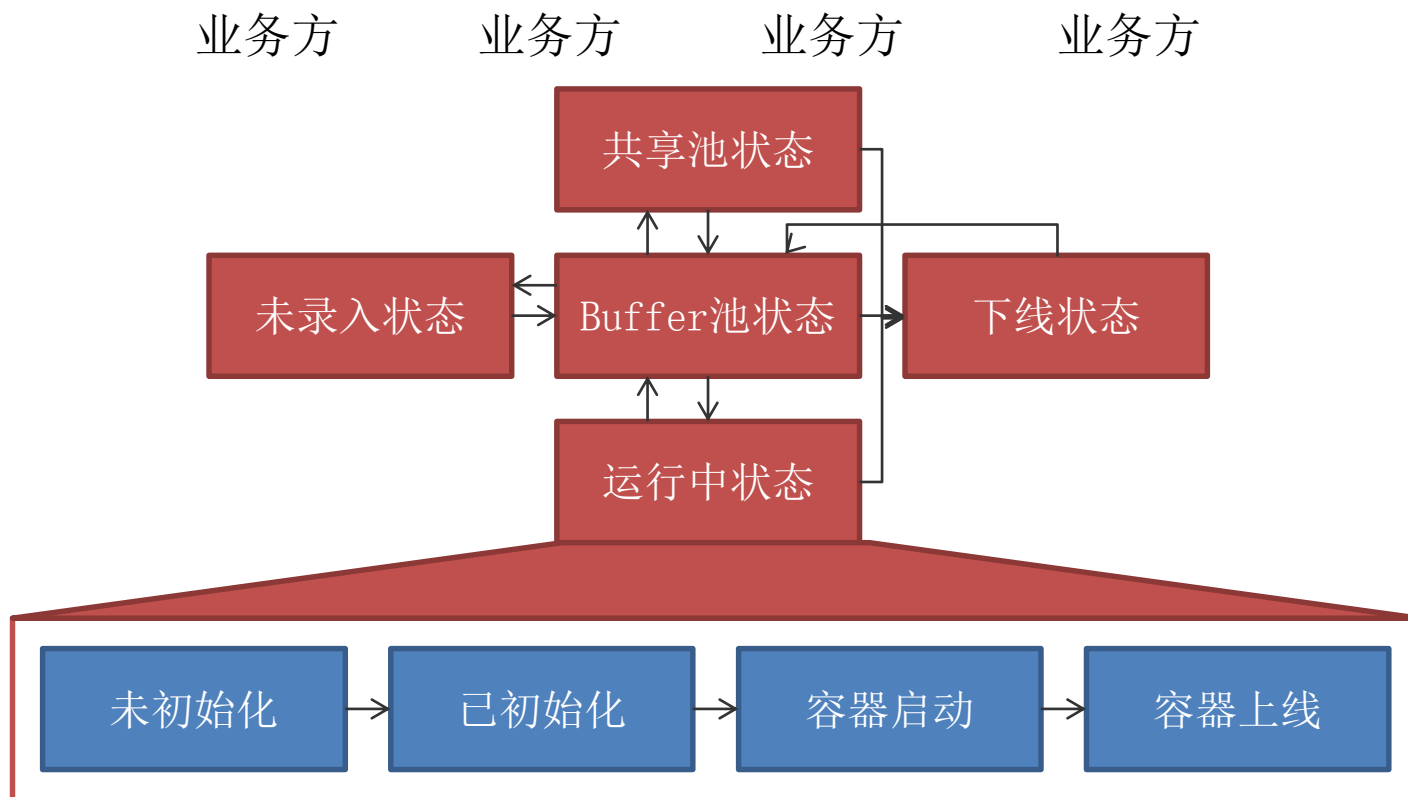
DCP弹性调度过程



DCP弹性调度 - 不同partition间调度



生命周期管理



扩容容易缩容难

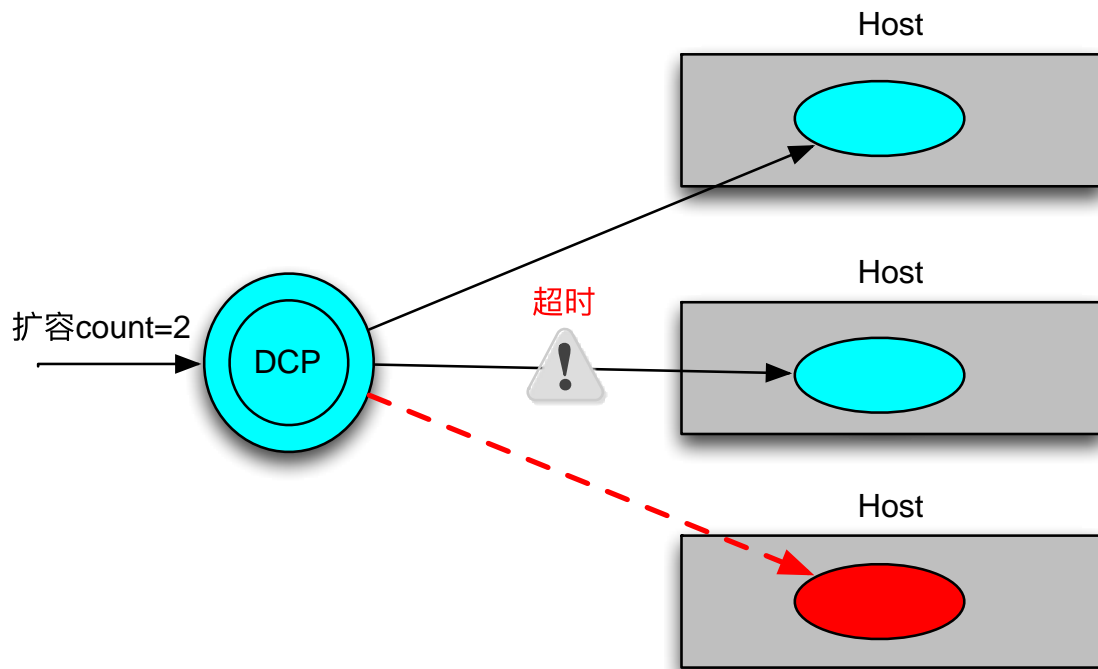
- 流量低峰时如何安全缩容
 - 下线节点，引发**故障**！
 - 下**多少**是安全的？
 - 人是不靠谱的！



容量评估系统



遇到的坑 - 僵尸资源



```
// UsedMemory returns the sum of memory reserved by containers.  
func (e *Engine) UsedMemory() int64 {  
    var r int64  
    e.Lock()  
    for _, c := range e.containers {  
        r += c.Config.Memory  
    }  
    e.RUnlock()  
    return r  
}
```

➤ 僵尸容器会影响下次调度

遇到的坑 - 开源软件的缺陷

全局锁导致批量创建容器效率极低

```
func (c *Cluster) createContainer(config *cluster.ContainerConfig, name string) error {
    c.scheduler.Lock()
    defer c.scheduler.Unlock()
```

V0.4.0

```
func (c *Cluster) createContainer(config *cluster.ContainerConfig, name string) error {
    c.scheduler.Lock()

    container, err := engine.Create(config, name, true)
```

V1.0.0.rc1



遇到的坑 - 版本兼容陷阱

```
const (  
    // Force-refresh the state of the engine this often.  
    stateRefreshPeriod = 30 * time.Second  
  
    // Timeout for requests sent out to the engine.  
    requestTimeout = 10 * time.Second  
  
    // Minimum docker engine version supported by swarm.  
    minSupportedVersion = version.Version("1.6.0")  
)
```

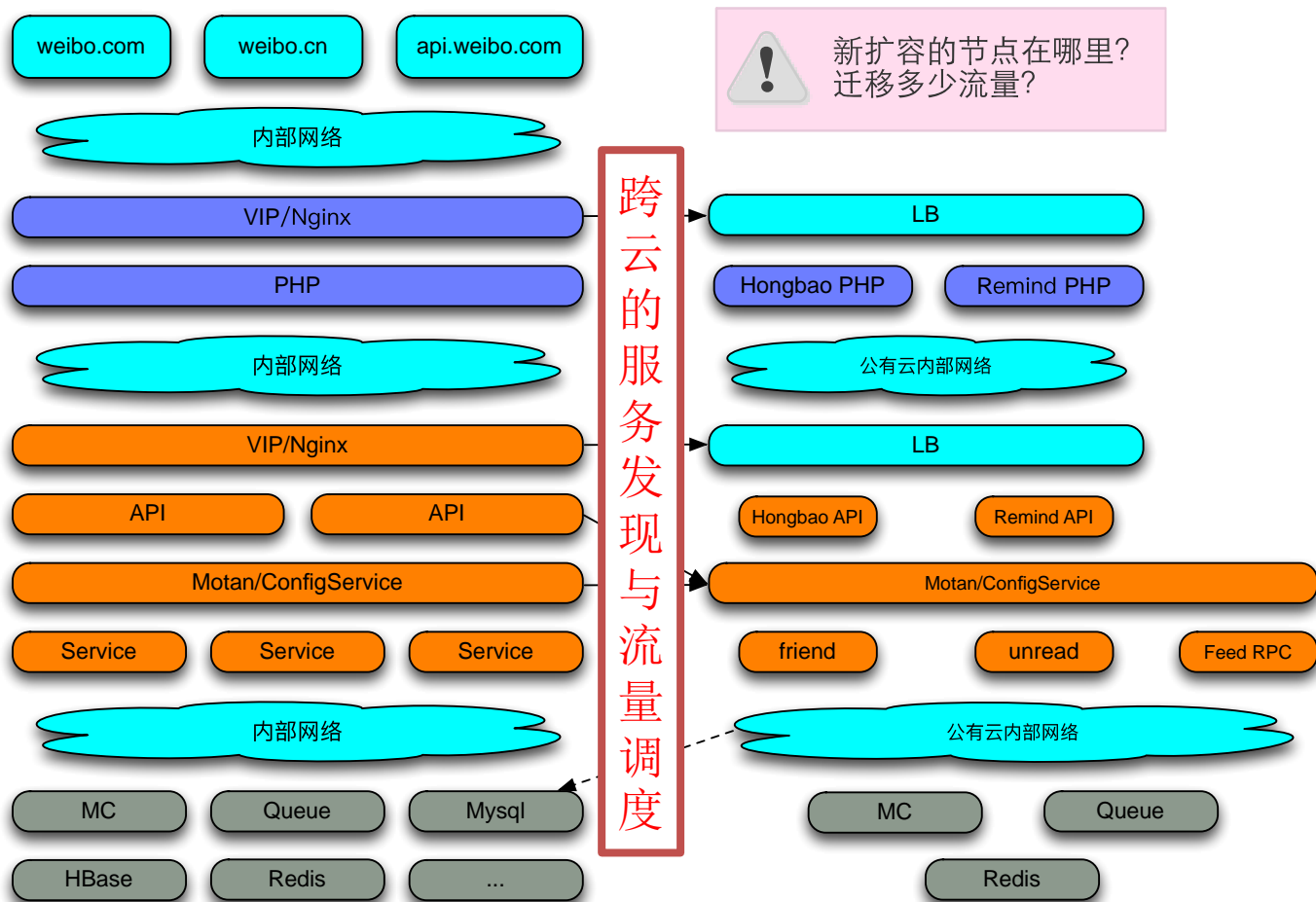
- Working on Docker Daemon Wrapper for 1.3.2
 - 兼容 **Name, Total Memory, CPUs, Labels**



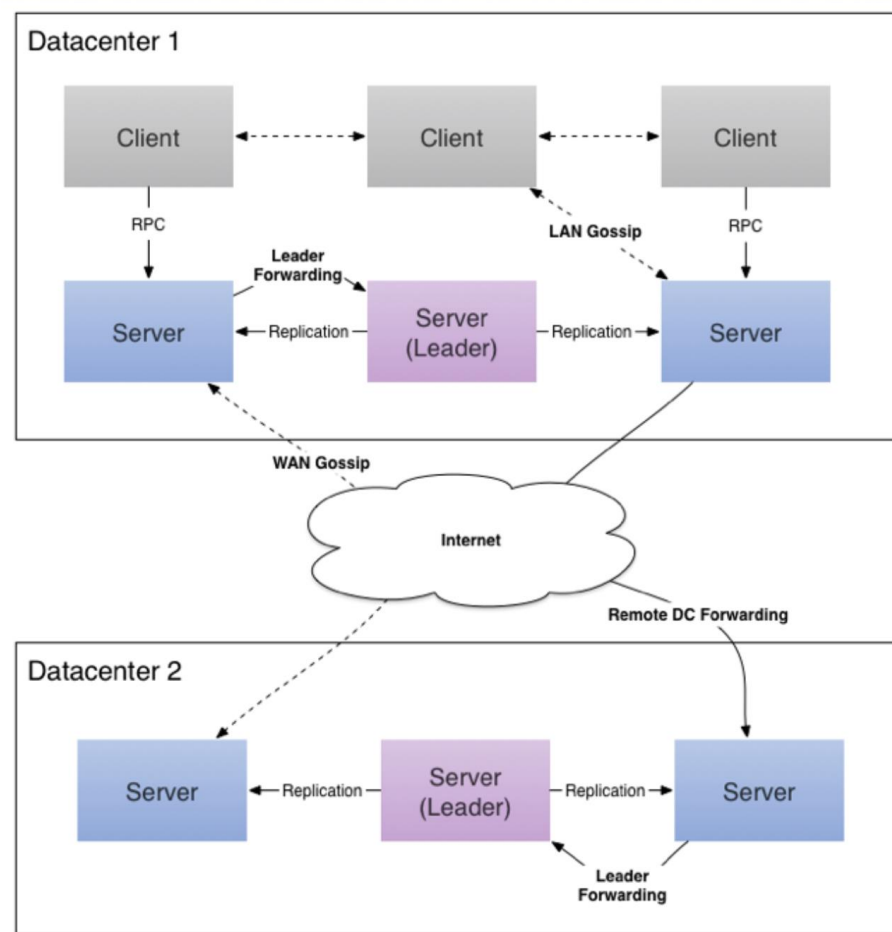
Part 4

服务发现

流量要快速、安全的切到弹性节点

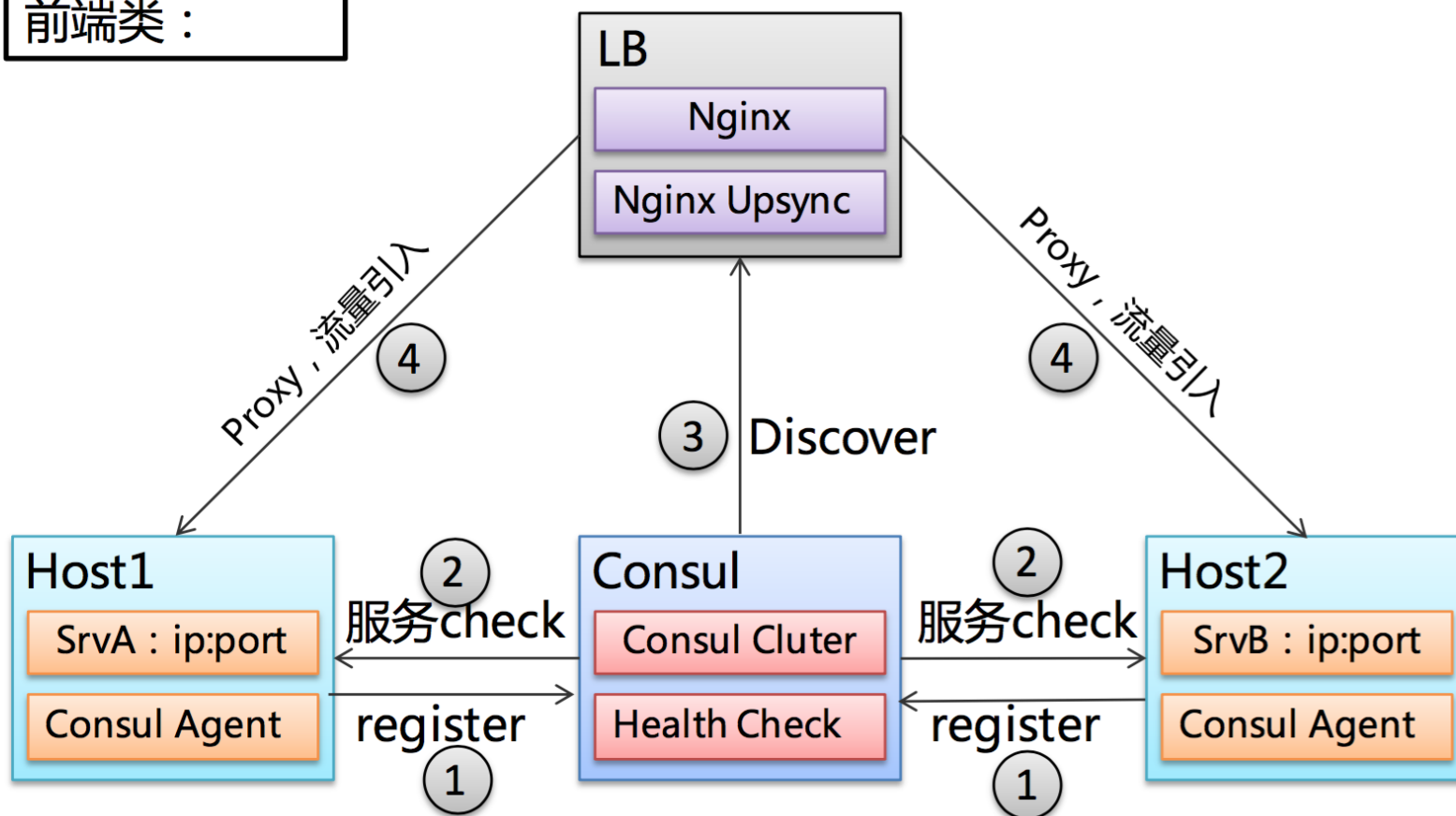


基于Consul实现跨云服务发现



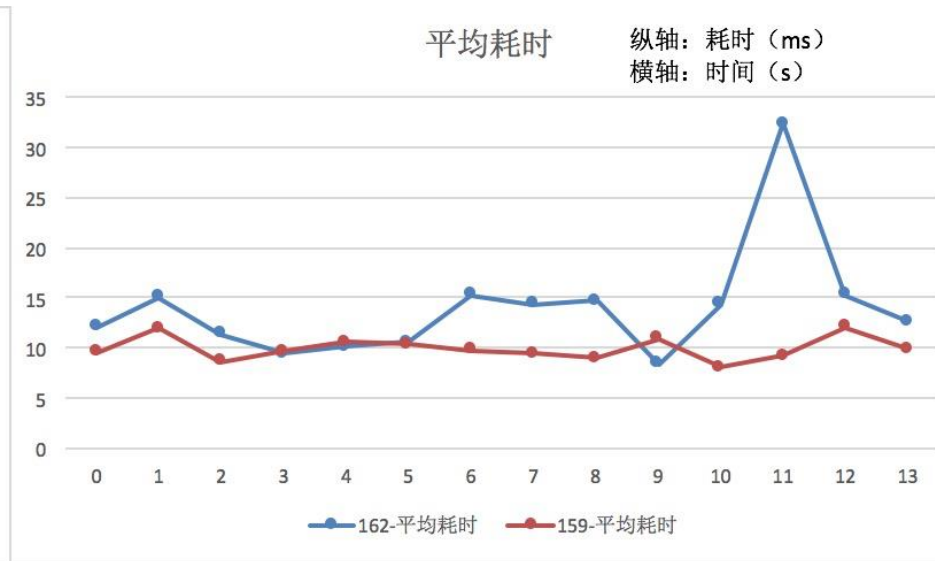
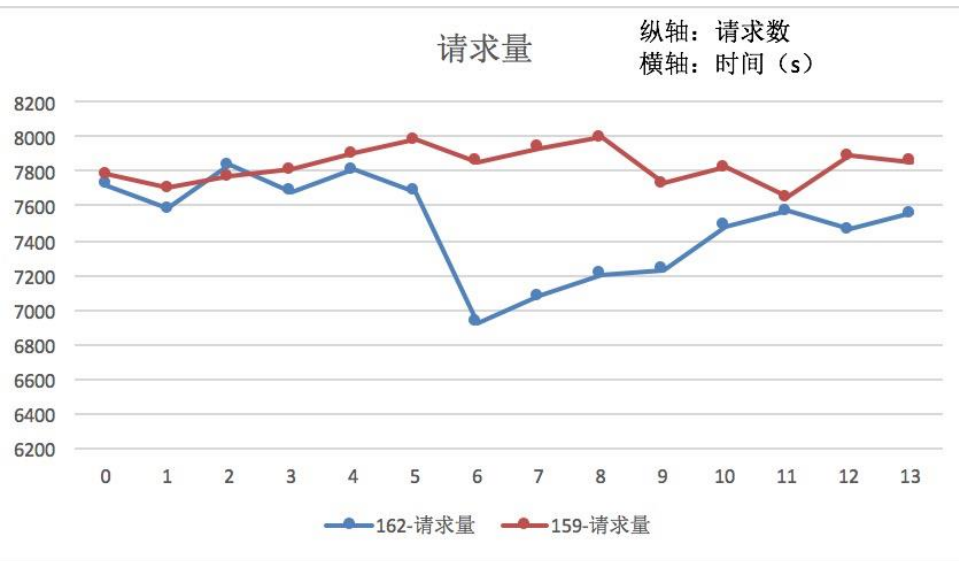
7层服务发现

前端类：



需要注意的问题 – Reload损耗

- 开源解决方案大多利用Nginx的Reload机制
- 性能损耗情况：
 - 请求量：普通reload会导致**吞吐量下降10%**
 - 平均耗时：差异不大，稍有优势

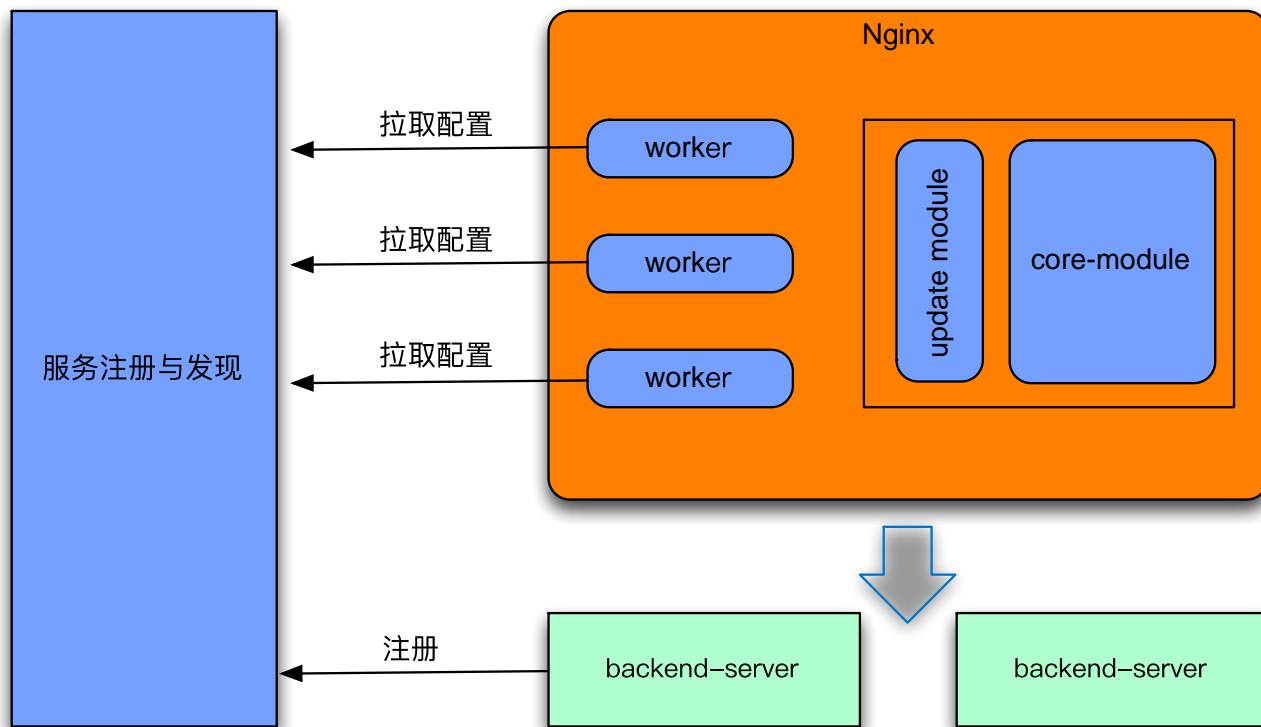


Nginx Reload可能的损耗点

- fork 新的进程的开销（内存，新的work 进程的初始化）；
- 基于client、backend 的所有的长链接需要重新的创建；
- 旧的work 进程需要频繁的检索connection 的链接数据结构，判断是否所有的处理请求已经处理完；
 - 参考：<https://github.com/alibaba/tengine/issues/595>；这是一个普遍问题，1.9.0之后的版本做了一些优化，但是只是优化；
- 新旧work 进程对cpu 的争夺；
- **商业的Nginx Plus提供了API**

微博的方案 - Nginx Upsync

- Nginx Plus的开源版
- 支持基于Consul自动服务发现
- 开源: <https://github.com/weibocom/nginx-upsync-module>



Nginx Upsync - 自动适配后端处理能力

➤ 弹性节点的处理能力不对等

- `server 10.xx.xx.xx:xxxx max_fails=0 fail_timeout=30s weight=20; #同样的权`

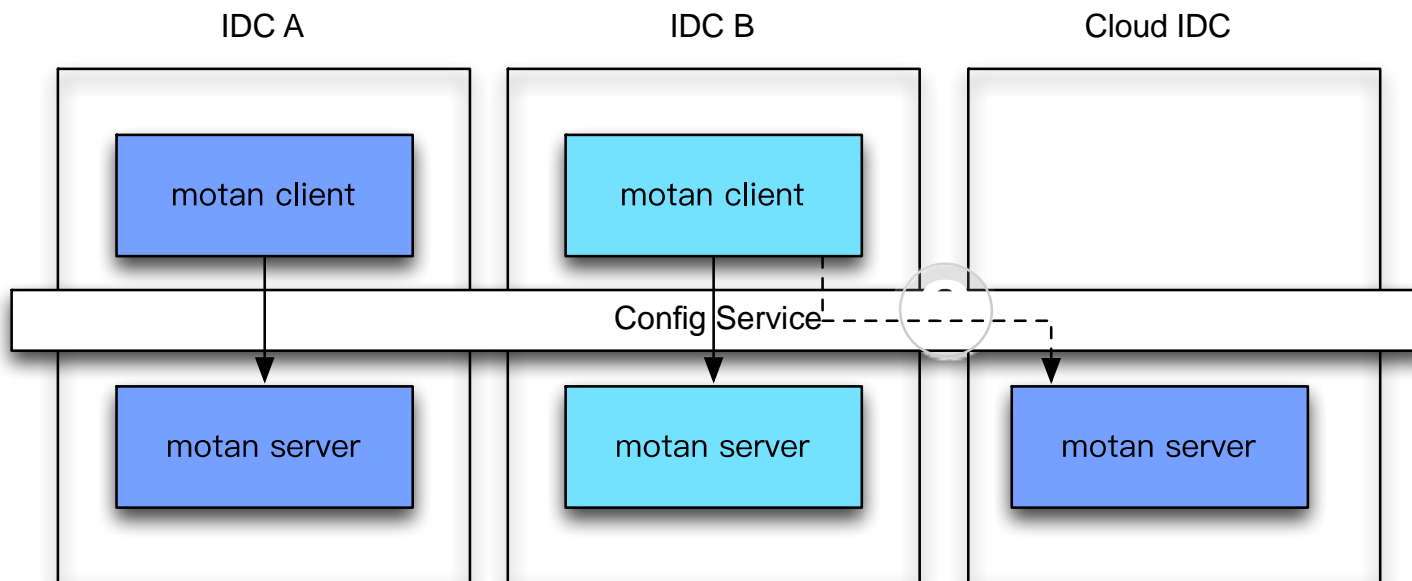
重导致单点性能恶化

➤ 节点注册计算能力

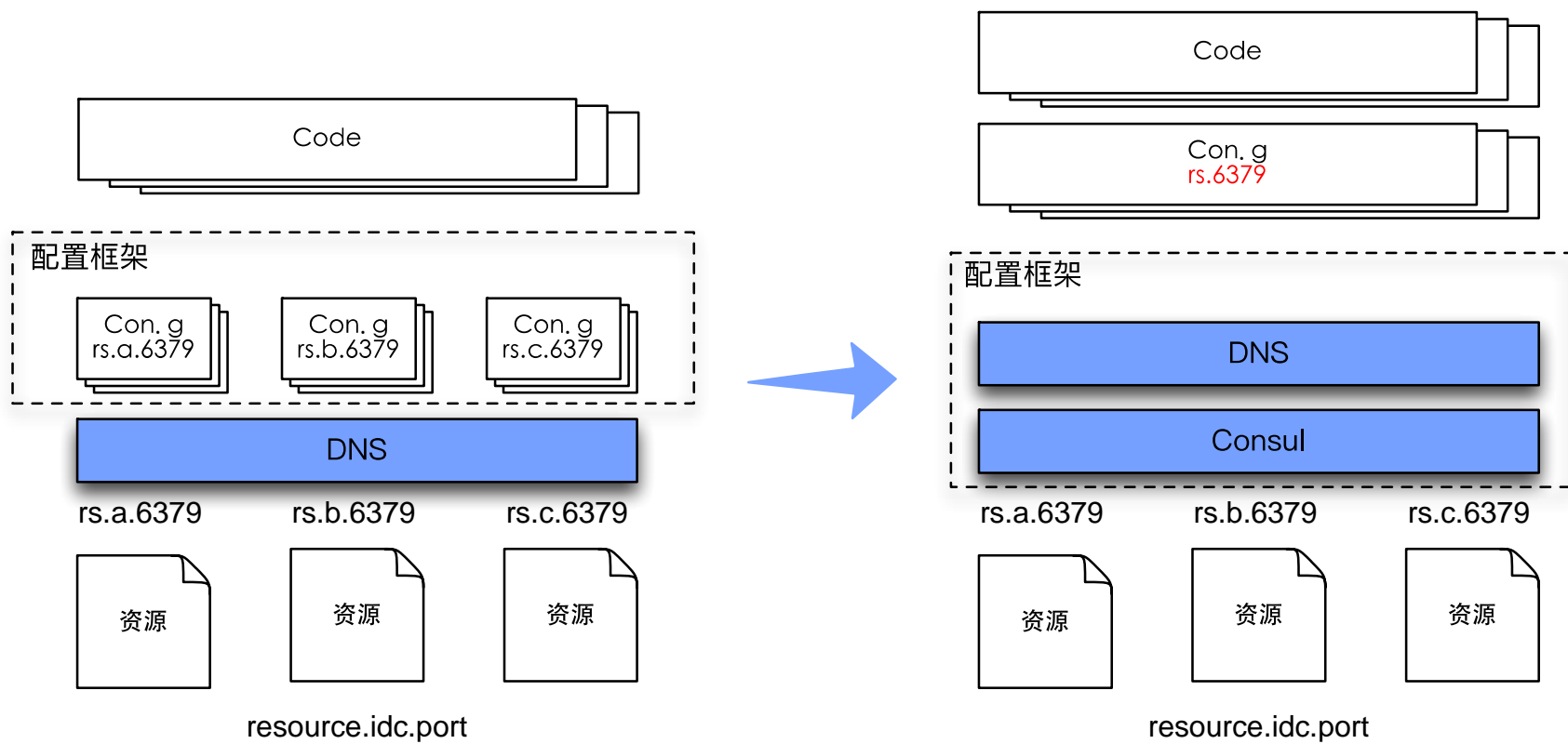
- 所有节点默认权重是20；
- 公有云有20%性能损耗，权重=16；

Motan RPC服务发现

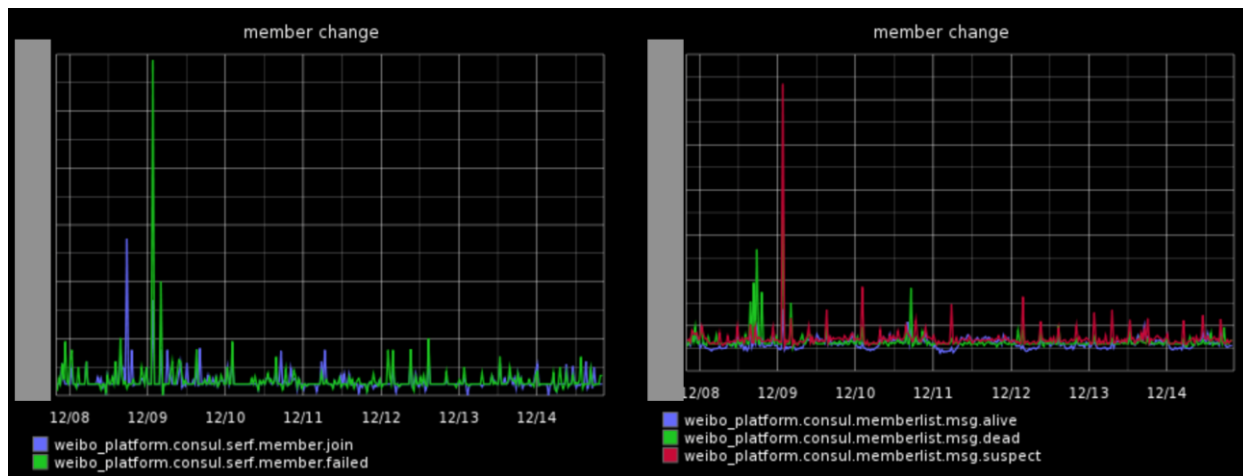
- Motan流量路由模型优先匹配单IDC
 - IDC间切换一般仅在故障处理时采用
- 新版Motan已支持按流量权重配置定向路由



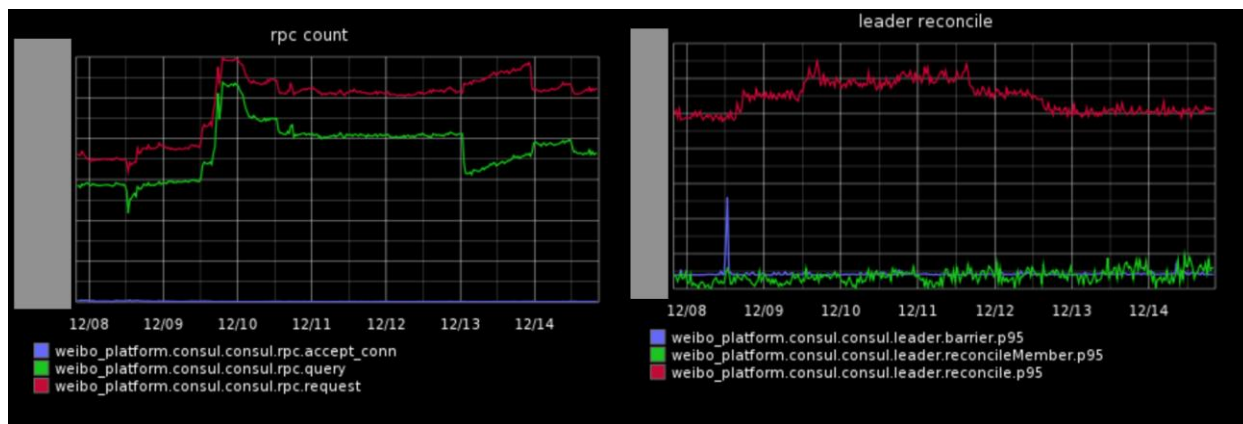
DNS服务发现



Consul集群监控



➤ 节点状态抖动频率



➤ QPS及耗时



Part 5

总结

微博DCP功能点

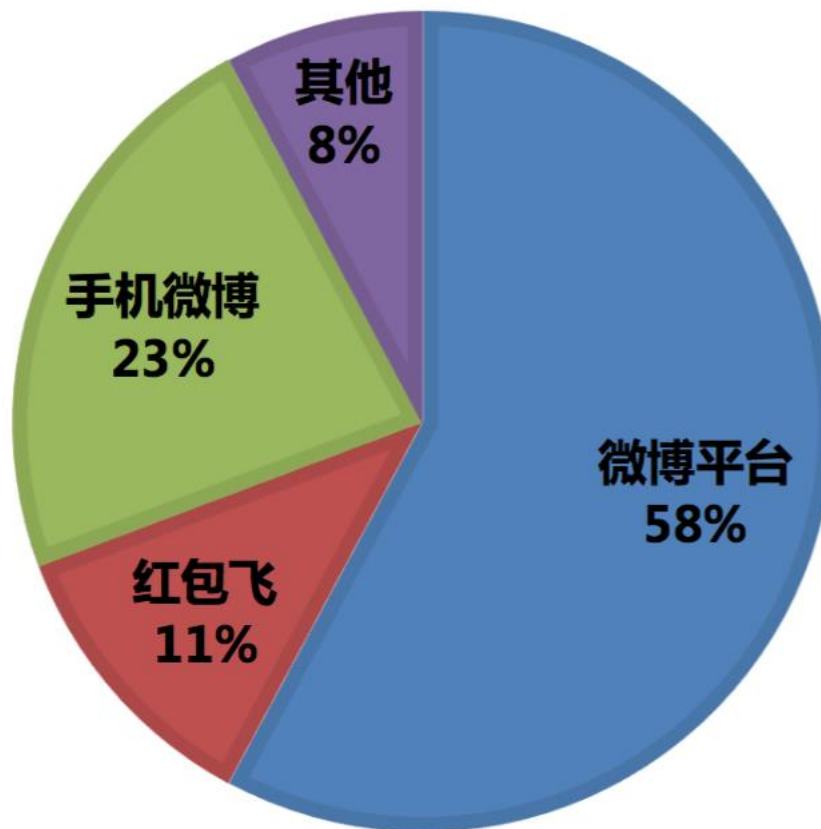


业务方	红包飞	MAPI	广告	有信	Feed
	用户	通讯	平台架构		
P A A S	Swarm	Docker 调度策略	Mesos 调度管理	容量评估	Docker Register
	Java对接		离线对接		Docker 镜像市场
I A A S	Buffer池 调度管理	成本核算	ECS管理	SLB等 管理	Consul 工具管理
	四七层 解决方案	专线保障	公有云 流量管理	审批流程	Docker 版本管理
基础 框架	软件安装	工程框架	安全保障	账户体系	Docker 工具体系
	监控体系	DNS	PPool	公有云 Yum/日志	



微博DCP进展

- 项目进展
 - 项目上线：2015年10月
 - 容器数：3000+
- 双十一
 - 单日10次扩缩容
 - 单次操作时间<5分钟



Thanks!

