

China · Beijing

# 容器技术与微服务架构 在跨境电商领域的集成实践

陈天影 敦煌网-容器云平台负责人

  
**ArchSummit**  
全球架构师峰会 2016

[ 北京站 ]

主办方 **Geekbang**  **InfoQ**  
极客邦科技



促进软件开发领域知识与创新的传播



关注InfoQ官方微信  
及时获取ArchSummit  
大会演讲视频信息



全球软件开发大会 [北京站]

2017年4月16-18日 北京·国家会议中心

咨询热线: 010-64738142



全球架构师峰会 2016 [深圳站]

2017年7月7-8日 深圳·华侨城洲际酒店

咨询热线: 010-89880682

# 大纲

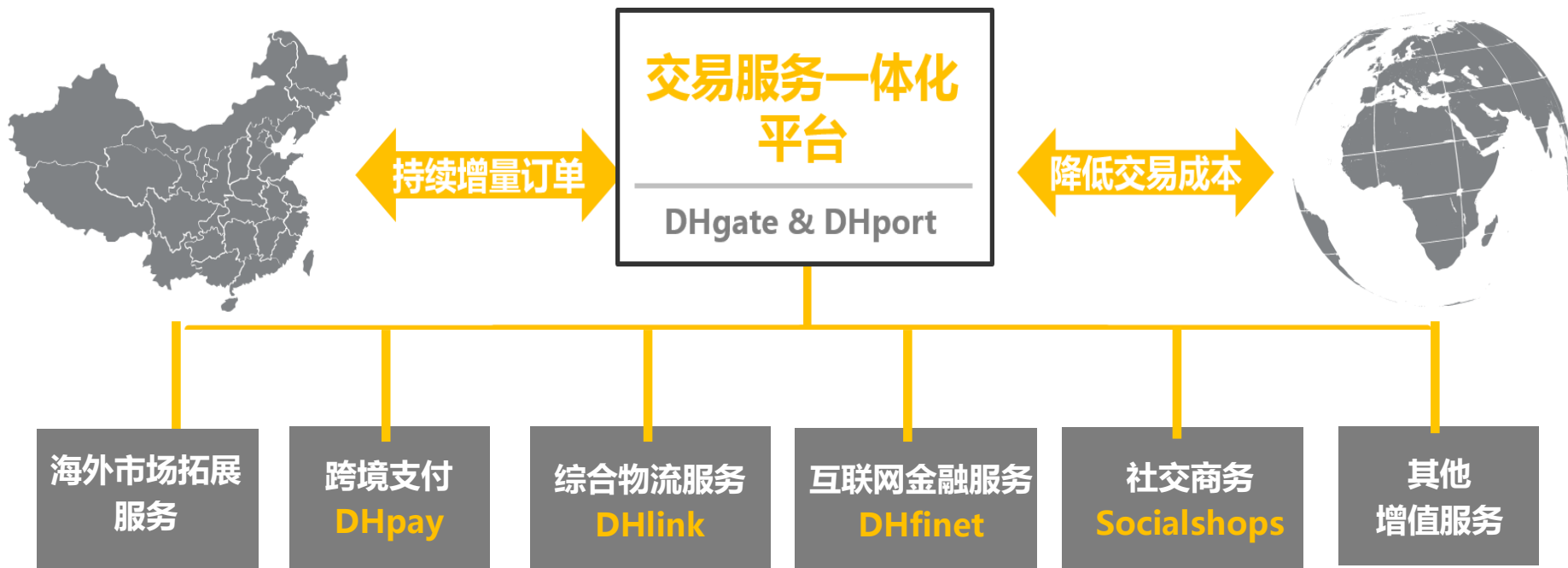
- 一、跨境电商业务特征及技术特点
- 二、微服务架构及面临的困境
- 三、基于Docker的私有容器云平台设计
- 四、Docker实践中遇到的问题及解决方案
- 五、总结

# 一、跨境电商的业务特征及技术特点

# 跨境电商的业务特征

外贸商户/工厂

批发商/零售商



约120万家国内供应商；  
1000万买家  
遍布全球230个国家和地区



国际合作伙伴  
物流&支付

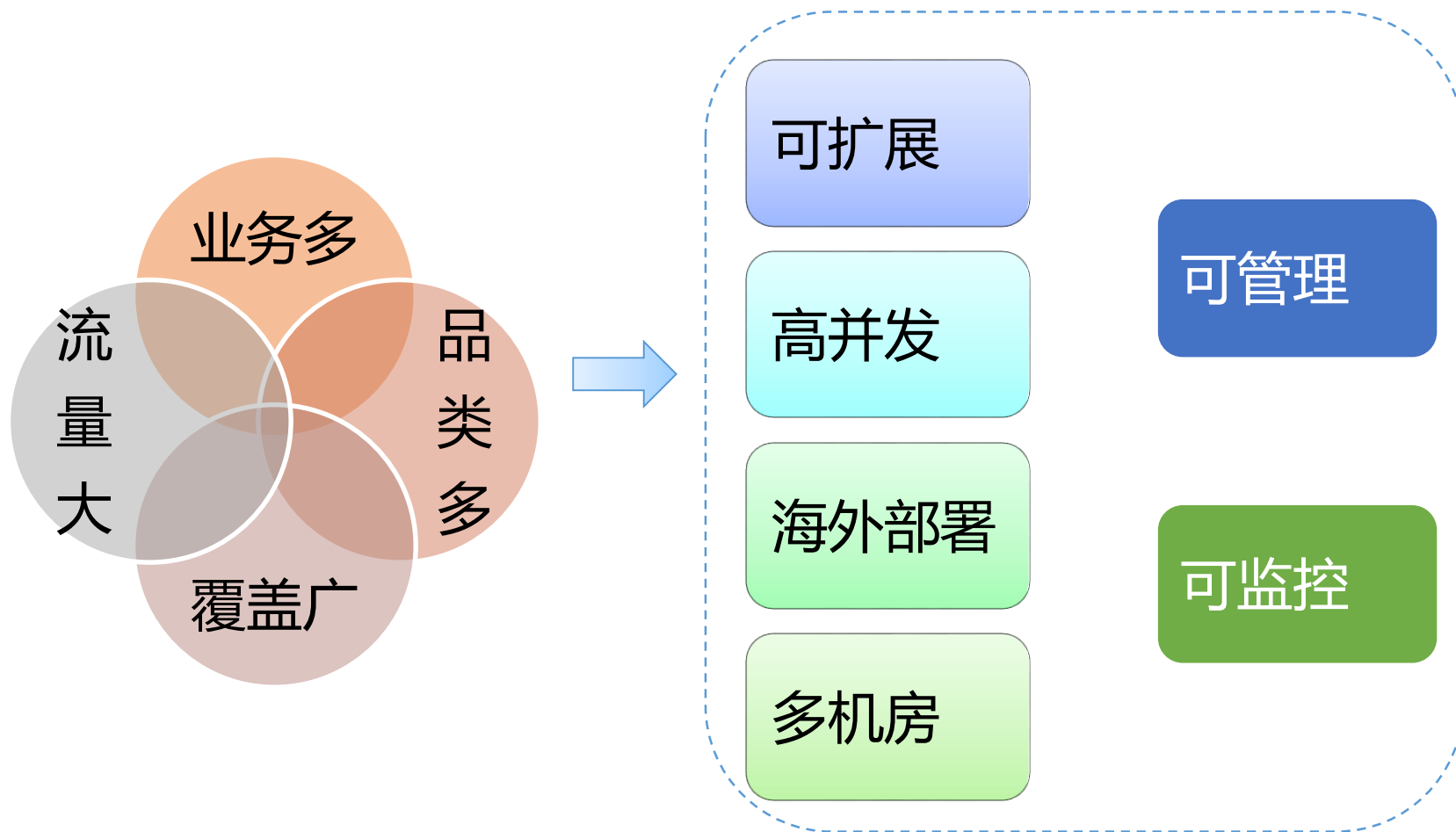


8个多语言  
平台



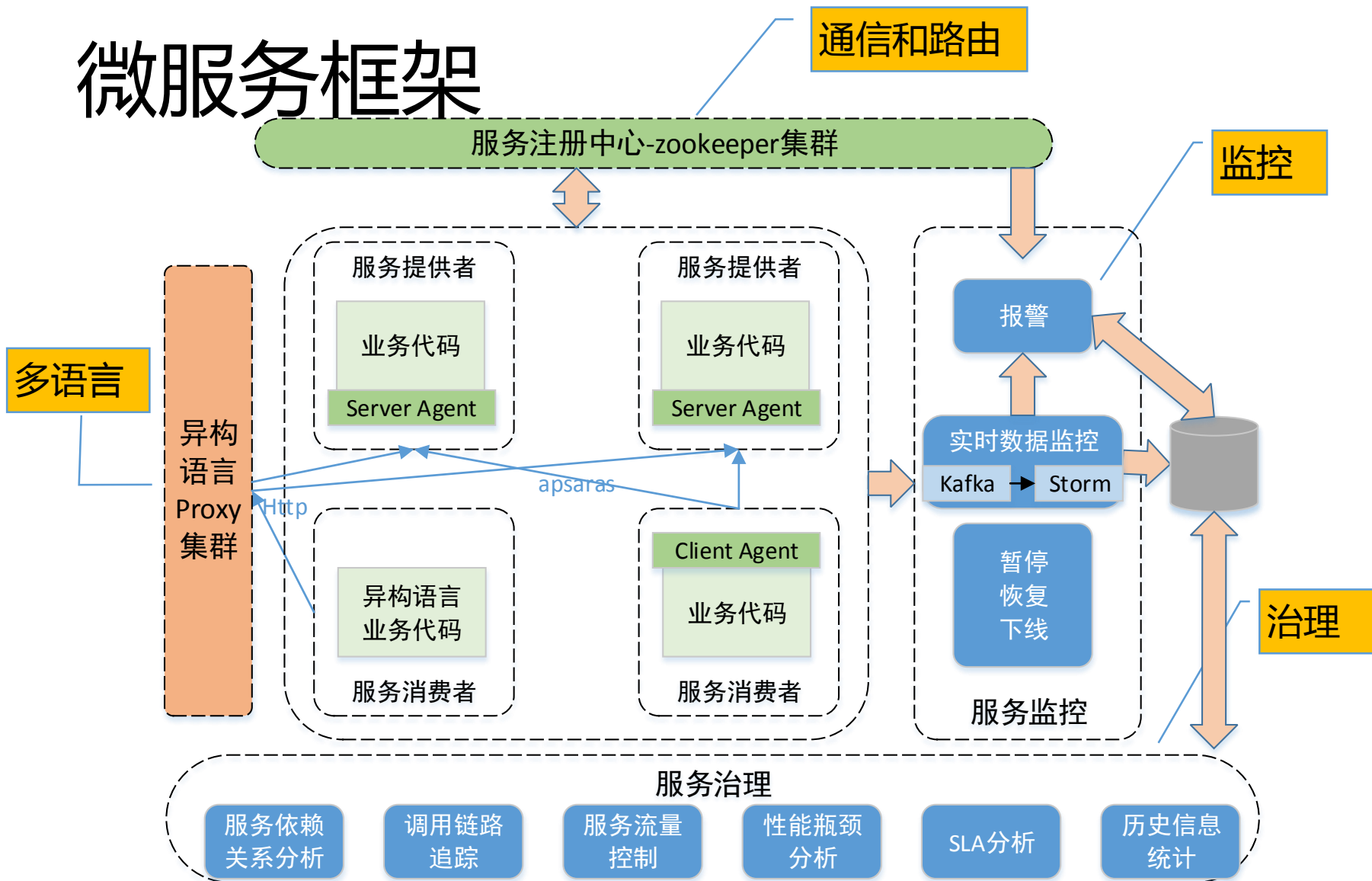
4000万  
在线产品

# 跨境电商的技术特点



## 二、微服务架构及面临的困境

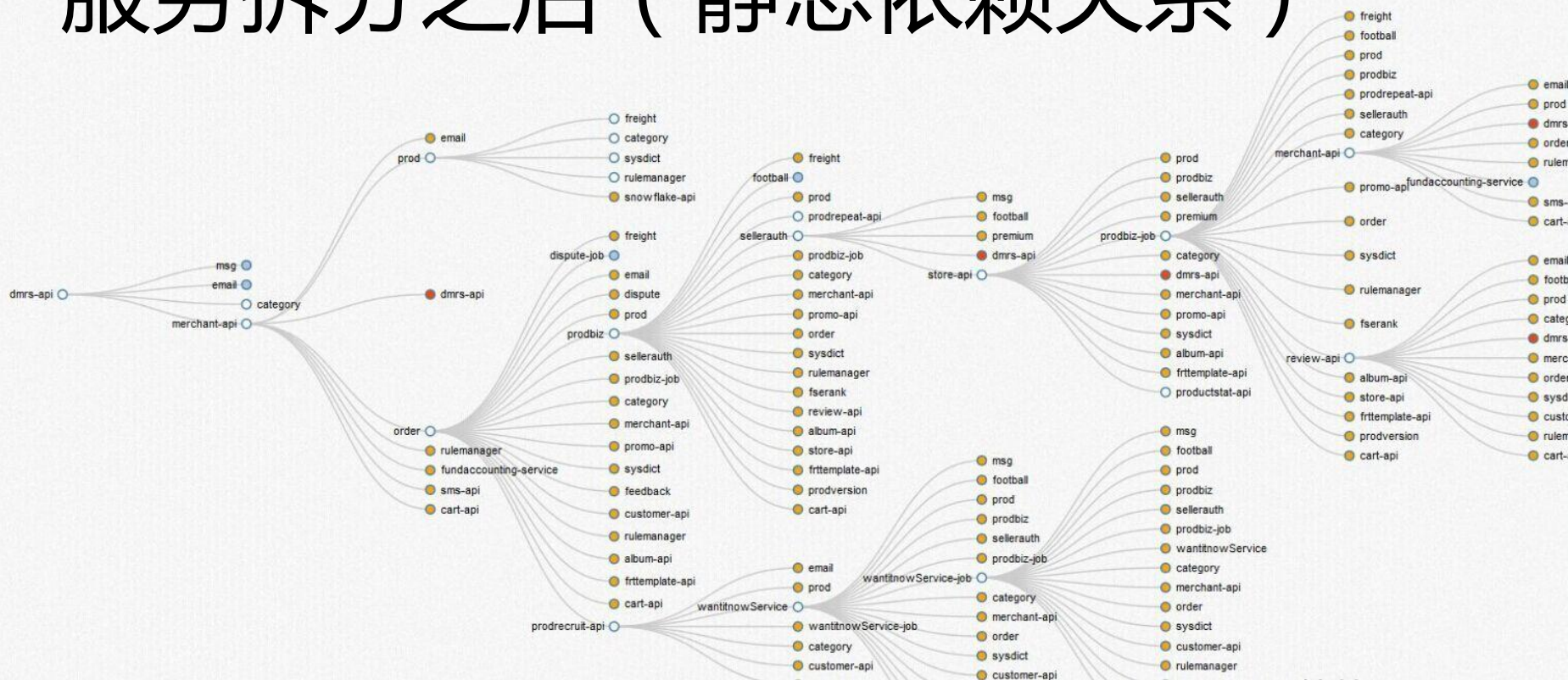
# 微服务框架



微服务架构要解决的问题：服务通信、路由寻址、服务监控、服务治理、多语言



## 服务拆分之后（静态依赖关系）



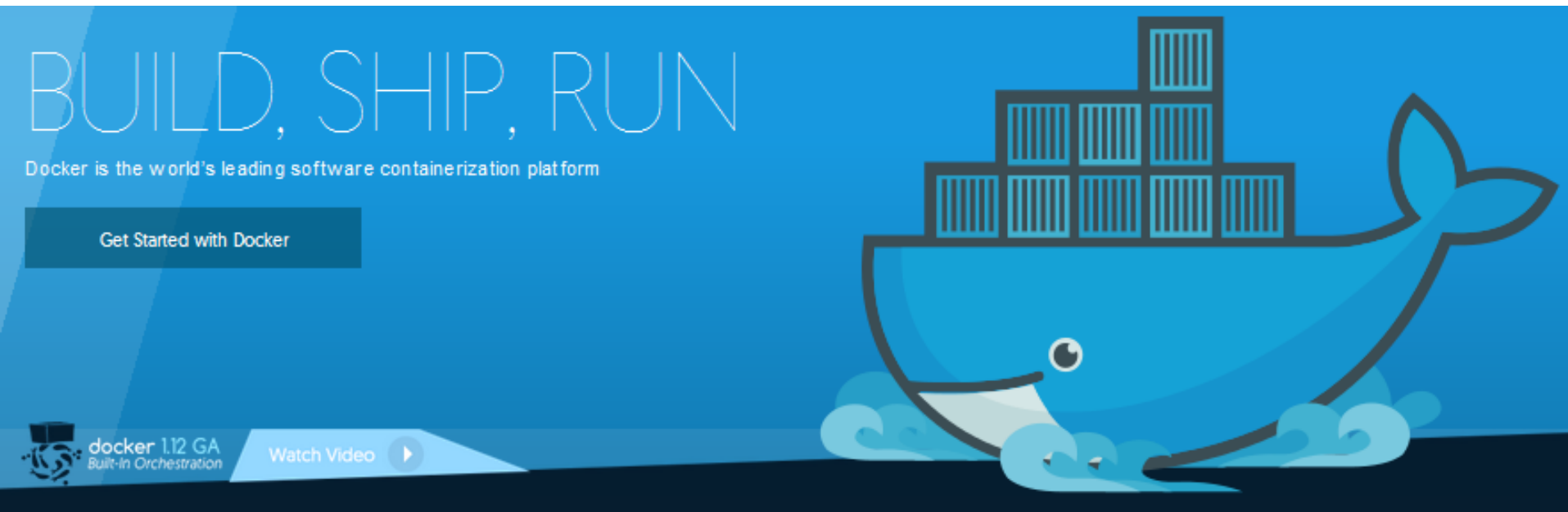
### 优势：

- ❑ 解耦（模块化），易扩展
- ❑ 效率（开发、测试、部署）
- ❑ 高可用（多实例）
- ❑ 弹性伸缩，灵活应对峰值流量（分布式、独立运行）
- ❑ 多机房部署

# 困境

- 微服务的独立性使得系统具备弹性伸缩的能力，但仍需人为介入
- 开发效率提高、交付速度有所提升，新业务上线仍受限于资源申请流程
- 微服务架构导致模块数量快速增长，服务粒度与资源粒度的矛盾
- 一台服务器部署多个微服务，产生资源竞争
- 运行环境差异性引发错误
- 部署海外机房周期太长（每个应用——分配资源、配置、部署）
- 降低物力资源成本的需求
- .....

# Docker



- ❑ 标准化：集装箱式的交付方式，快速部署，并避免运行环境差异化
- ❑ 轻量：资源占用小、启动速度快，在一个服务器上可以部署很多容器
- ❑ 便捷：直接部署应用、无需申请资源
- ❑ 隔离：一定的资源隔离性
- ❑ 灵活：可以快速回滚和更新变更
- ❑ 开源：生态系统发展迅速
- ❑ 成本：搭建成本低、学习成本低

# 三、基于Docker的私有容器云平台设计

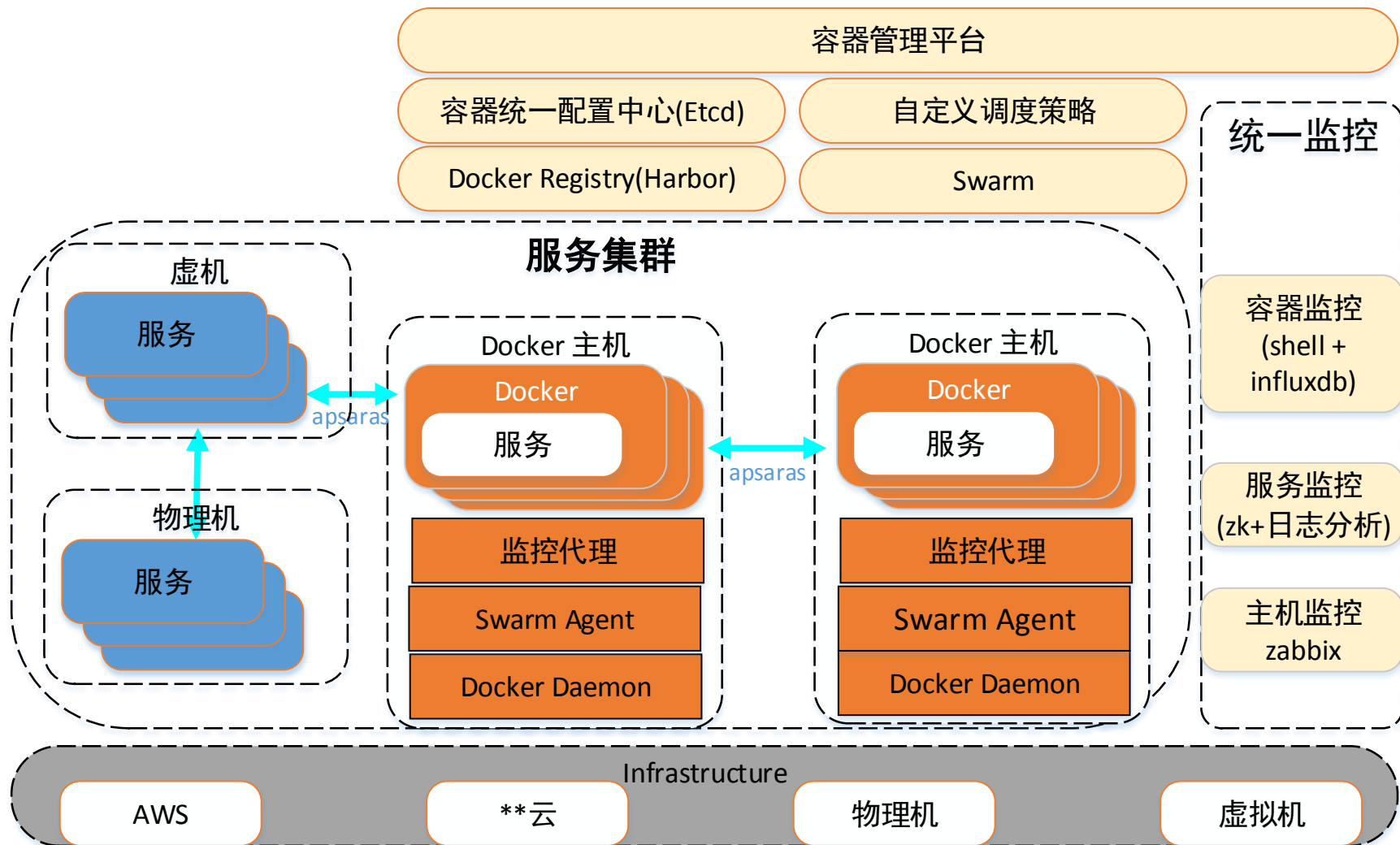
# 设计原则

兼容已有软件架构

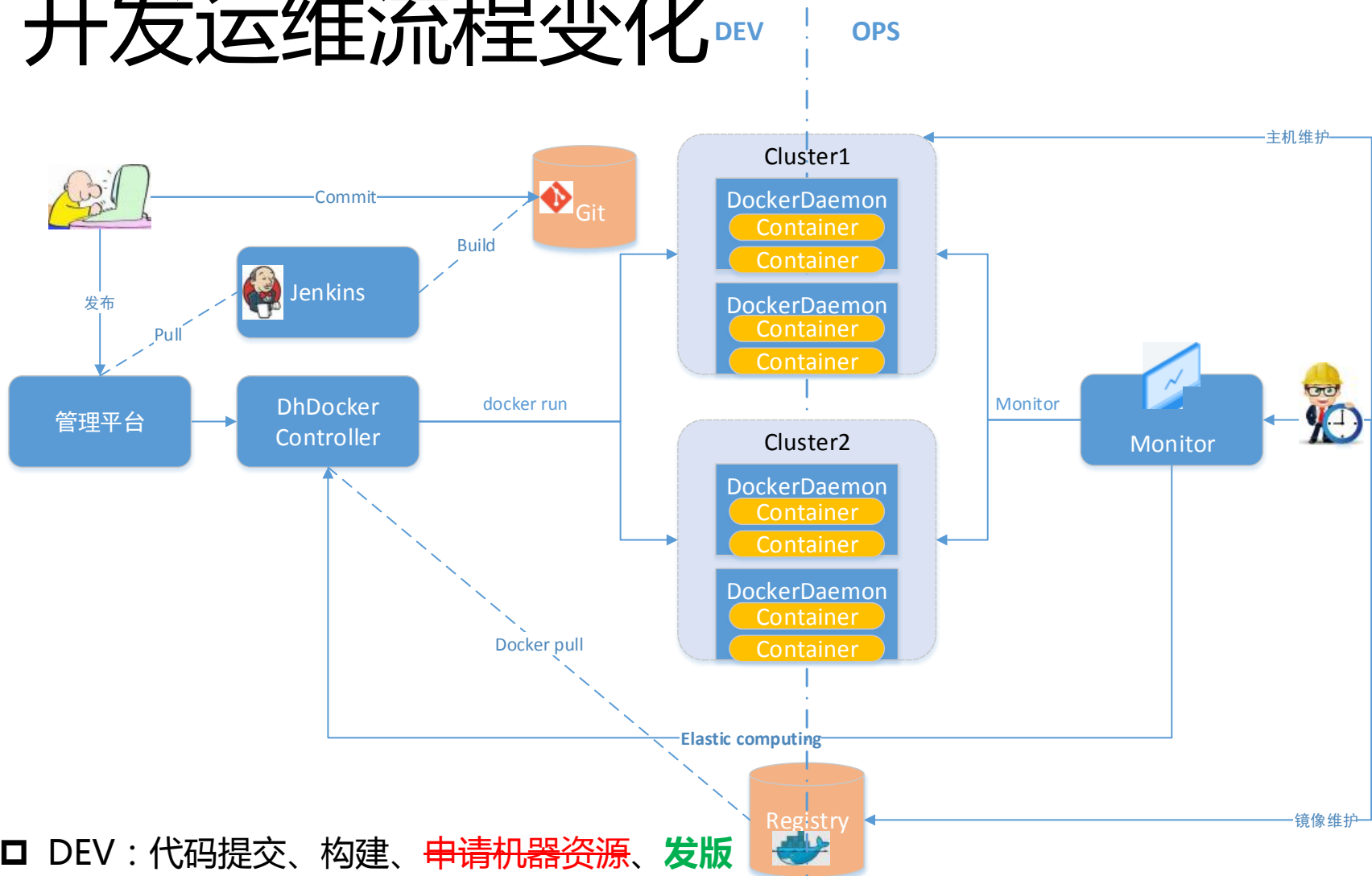
混合部署，历史资产无缝迁移

减少对已有业务开发流程的影响

# 整体架构



# 开发运维流程变化

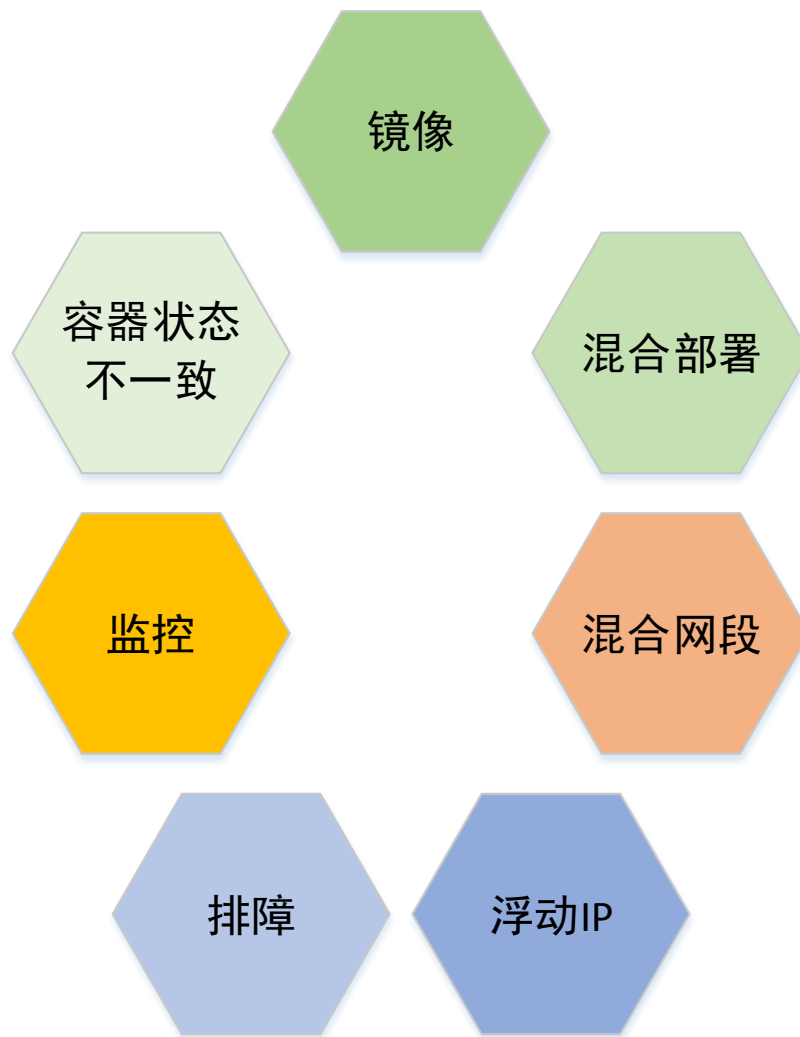


- DEV：代码提交、构建、**申请机器资源**、**发版**
- OPS：集群、主机维护（一键初始化），**镜像维护**，线上监控

## 四、Docker实践中遇到的问题及解决方案



# 遇到的问题



# Problem1- 镜像的制作和维护



## □ 优点：

- 避免mount，应用和镜像一体

## □ 缺点：

- 频繁构建
- 镜像数量猛增
- 改造工作量大
- 不同环境配置文件不同

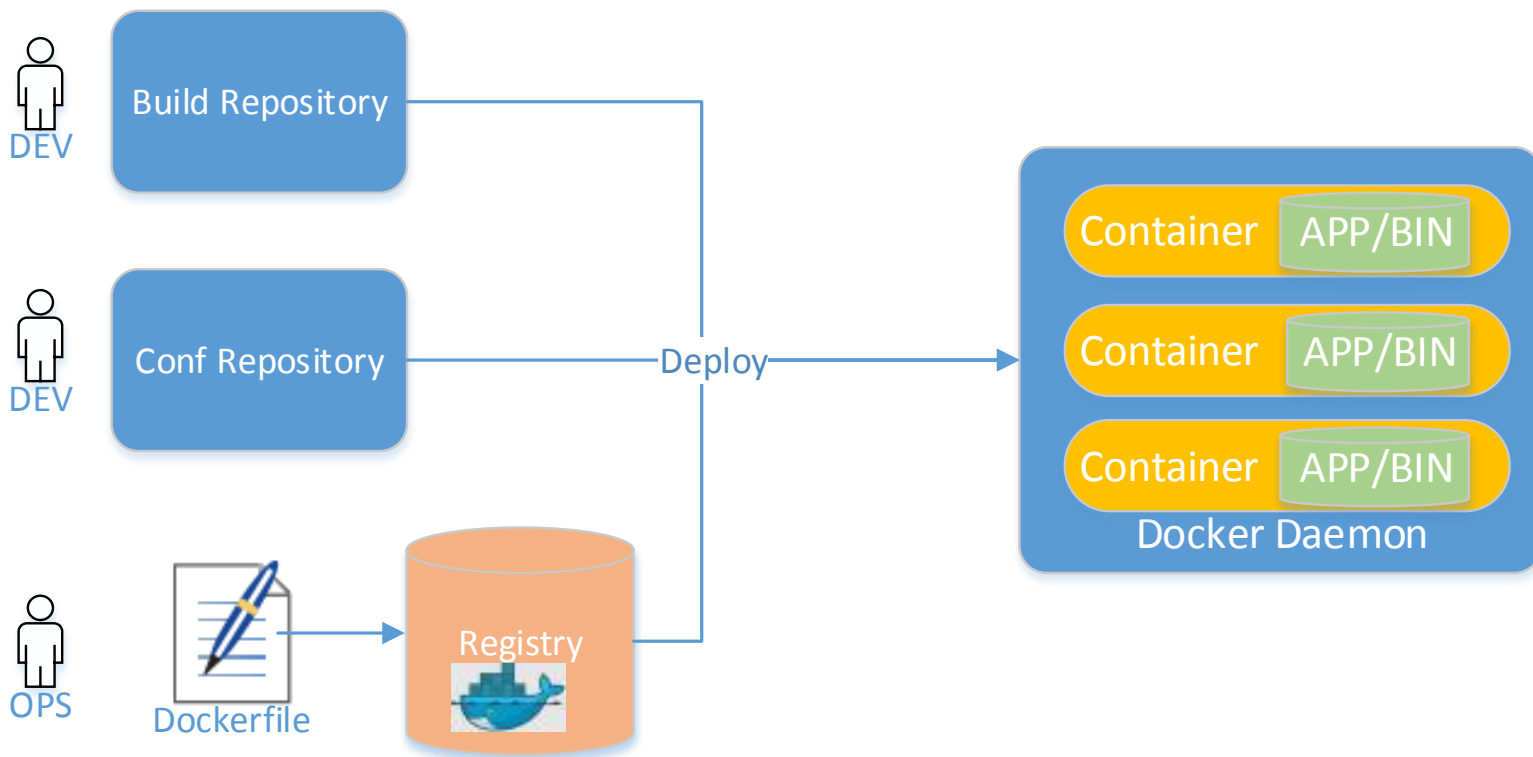
## □ 优点：

- 镜像不用重新构建

## □ 缺点：

- 所有宿主机维护应用副本
- 违背了Docker集装箱原则

# Solution1-基础镜像+应用下载

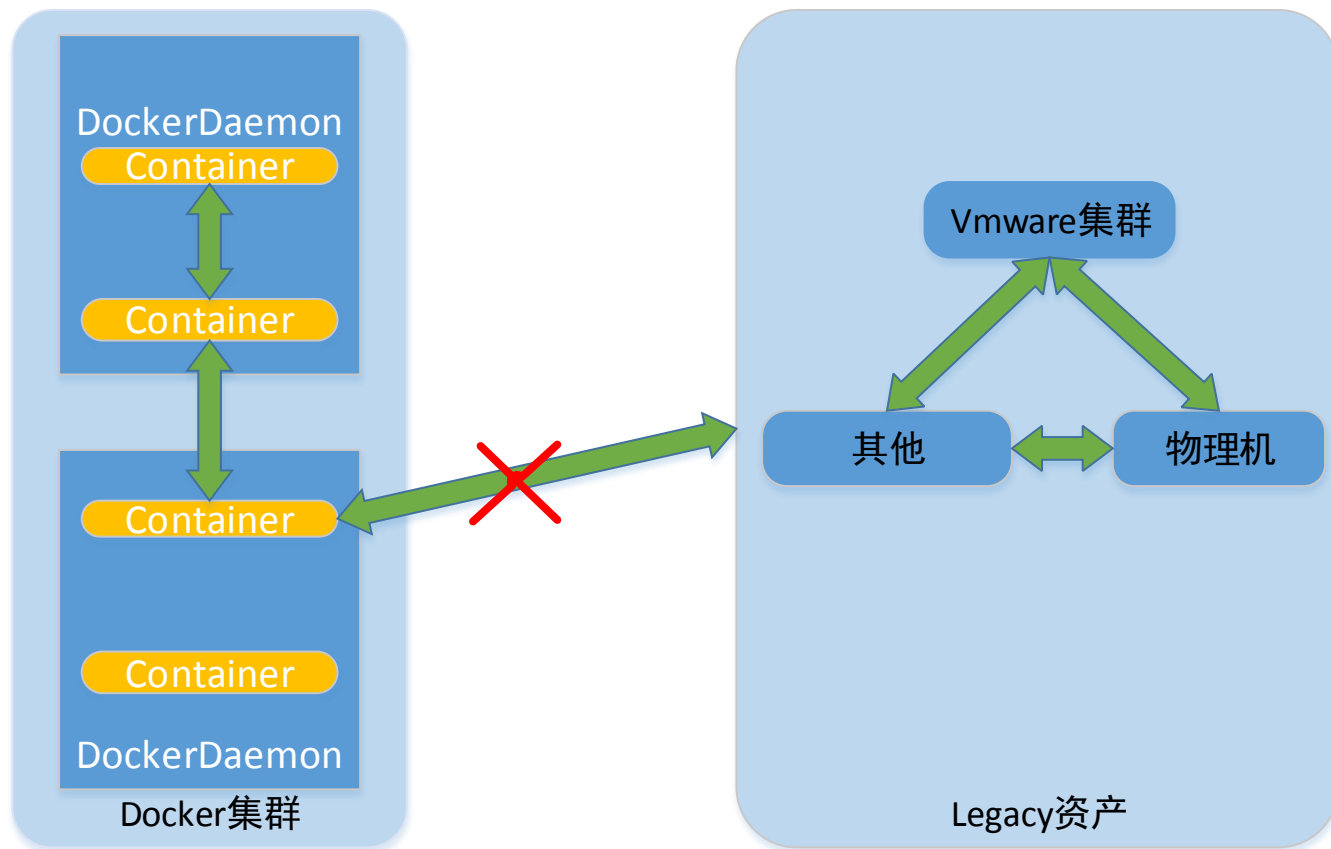


- ❑ 开发时，镜像和应用分离，OPS维护 Dockerfile，DEV维护代码
- ❑ 部署时，镜像和应用合体
- ❑ 多环境不同配置，分布式配置管理中心+配置文件中心

# Problem2-混合部署的网段互通

## 网络连通要求：

- 同一宿主机内的容器互通
- 不同宿主机之间的容器互通
- 容器与其他虚拟机、物理机互通-**混合部署的必要条件**



# Docker的网络方案

- bridge模式：与外界通讯用端口映射，NAT增加通讯复杂性
- container模式：单机的多个容器之间共享网络
- host模式：共享主机网络，端口无法重用，容易冲突
- 自定义

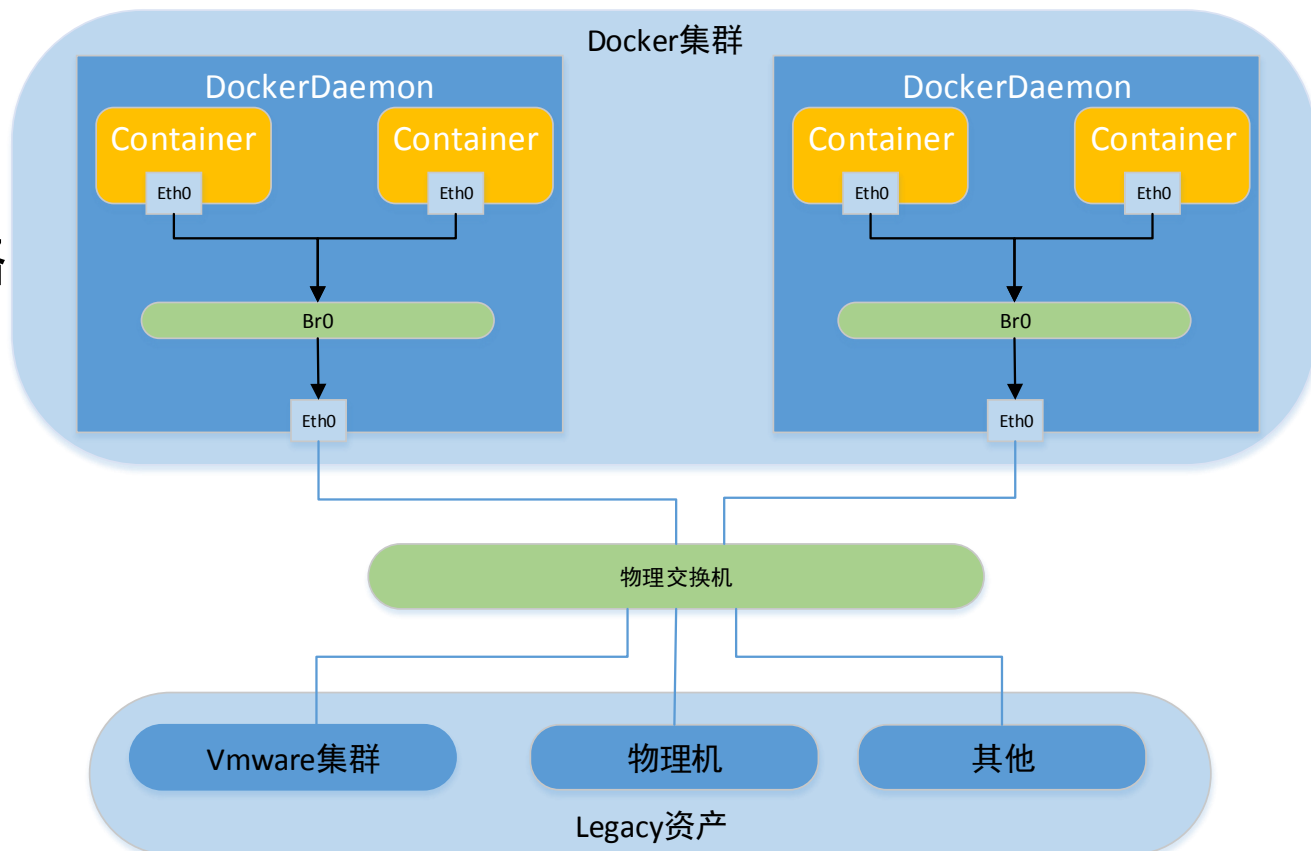
- **Bridge network** – is useful in cases where you want to run a relatively small network on a single host.
- **Overlay network** – multi-host connection , need swarm or a key store.
- **MACVLAN network** – multi-host connection
- **Customized network plugin**

# Solution2-桥接网络

打通容器与局域网网络

✓ Docker Deamon

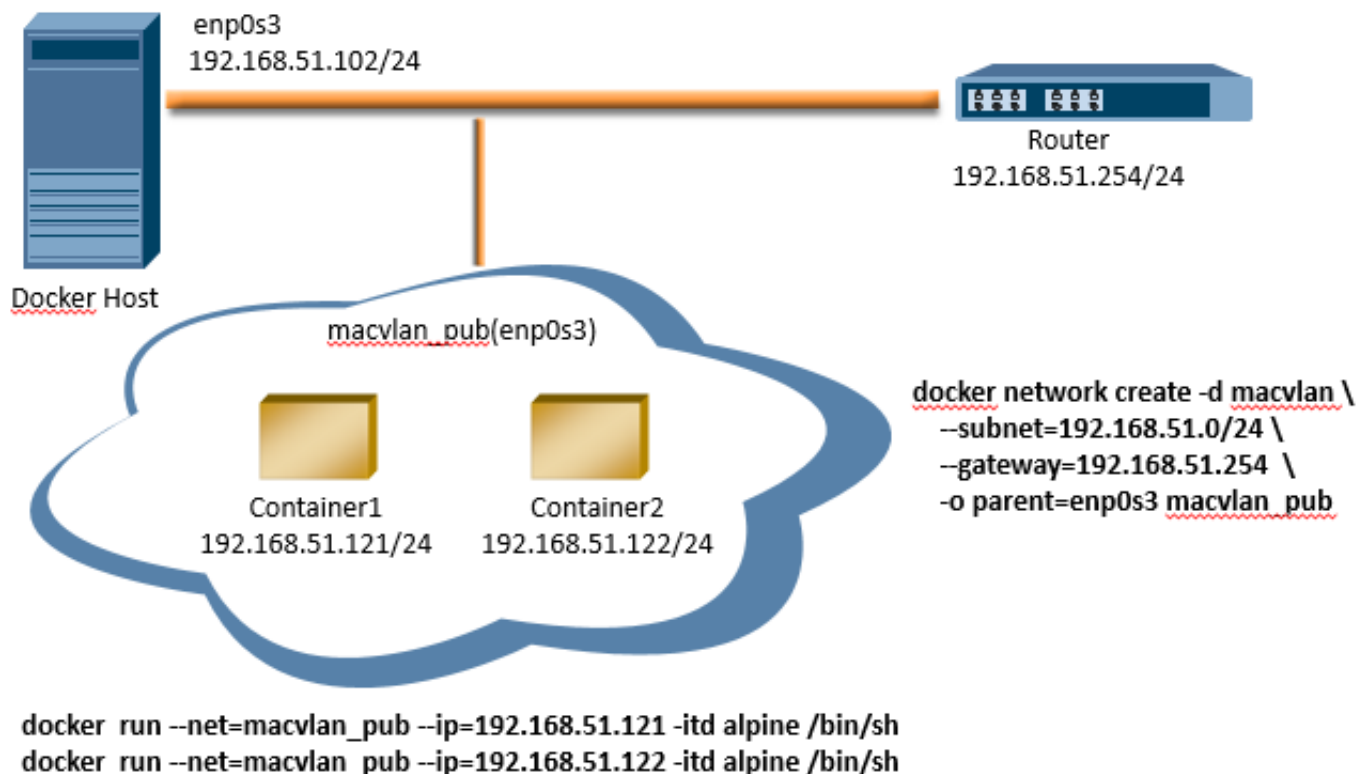
- --iptables=false
- --ip-forward=false



✓ docker network create --driver=bridge -o com.docker.network.bridge.name=br0  
--gateway=192.168.2.14 --aux-address "DefaultGatewayIPv4=192.168.2.254"  
--subnet=192.168.2.0/24 dockernet

✓ docker run -d --net=dockernet tomcat:7.0

# MacVlan



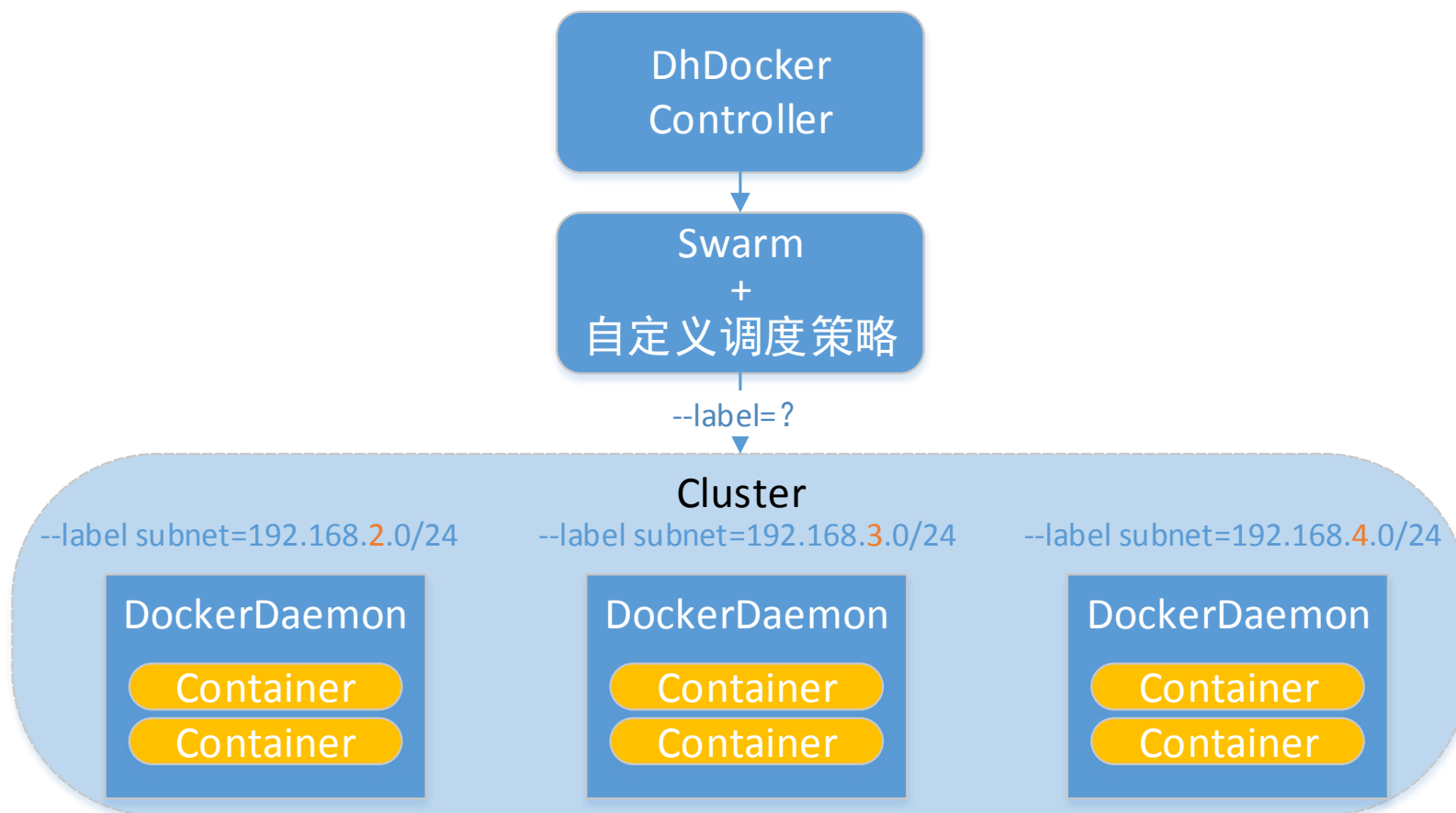
- **优点**：性能较好；可以在一台物理机上部署多个不同网段的容器
- **限制**：Docker 1.11 版本的MacVlan仍然是experimental; 在1.12版本已标注为：MacVlan driver is out of experimental [#23524](#)

# Problem3-混合网段的容器部署

□Problem： 网段过大导致广播风暴，混合网段如何  
权衡资源调度和IP分配



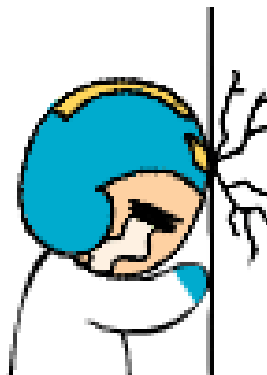
# Solution3-自定义IP资源调度策略



# Problem4-浮动ip

## □Problem：浮动ip

- 重启一下docker daemon/容器，ip就变了
- Ip混乱，不利于定位问题



# Solution4-使用固定ip

## □使用固定ip

➤ `docker run -d --ip=192.168.11.23 --net=dockernet tomcat`

□引入IPAM模块，负责IP池的创建和维护，IP资源的占用和释放

# Problem5-排障

## □持久化日志

- 将日志mount到主机，主机上运行轻量级Agent进行日志采集，集中分析

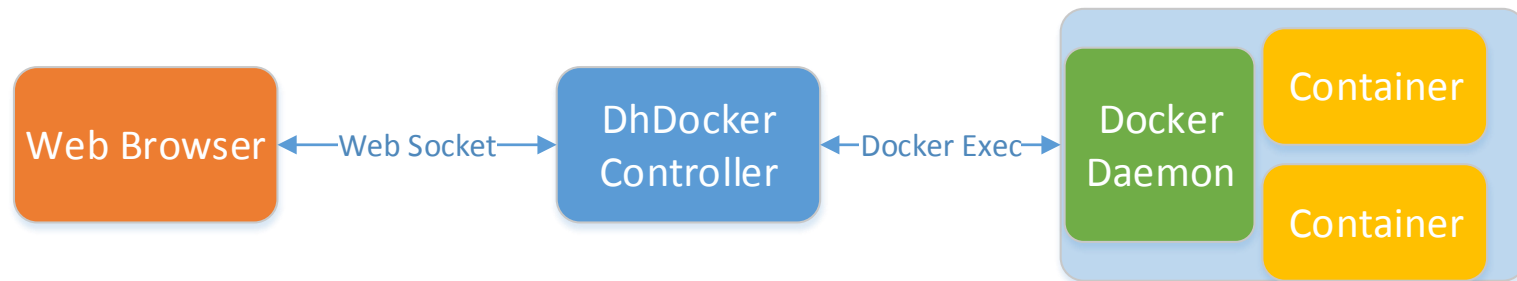
## □实时查看：检查日志，调整配置，重启应用，查看资源使用率

- 装一个SSH Server vs Docker Exec

别把容器当虚拟机使！！

# Solution5-Docker Web Shell

Docker Web Shell实现从Web浏览器以类似SSH的方式登录并操作Docker容器



- ◆ **Web浏览器**负责界面呈现。运行JS脚本，通过Web Socket与Docker Controller建立通信链路。
- ◆ **DhDocker Controller**是Docker容器应用的控制中心，作为桥梁，负责消息的转发。通过Docker HTTP API与Docker Daemon建立通信链路，利用Exec Start返回的数据流承载Docker Controller和Docker Daemon之间的交互数据。
- ◆ Docker Daemon提供HTTP API接口给外部系统调用以访问容器内部。这里用到的API包括：Exec Create、Exec Start、Exec Resize

# Docker Web Shell

Dhgate私有云平台 V1.0

```
top - 11:20:43 up 22 days, 23:58,  0 users,  load average: 2.16, 2.31, 2.33
Tasks:  33 total,   1 running,  32 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.1 us, 14.9 sy,   0.0 ni, 76.2 id,   0.0 wa,   0.0 hi,   3.8 si,   0.0 st
KiB Mem : 49246764 total, 269832 free, 15695940 used, 33280992 buff/cache
KiB Swap: 1048572 total,  523364 free,  525208 used. 32767744 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	12148	1972	1244	S	0.0	0.0	3:35.86	bash
91	jboss	20	0	4408708	262084	13304	S	0.0	0.5	34:30.13	java
1937	root	20	0	11768	1636	1380	S	0.0	0.0	0:00.00	bash
4280	root	20	0	11768	1852	1496	S	0.0	0.0	0:00.00	bash
4406	root	20	0	11768	1856	1496	S	0.0	0.0	0:00.00	bash
4419	root	20	0	24860	1696	1380	S	0.0	0.0	0:00.00	vi
4514	root	20	0	11768	1824	1468	S	0.0	0.0	0:00.01	bash
5333	bin	20	0	11768	1700	1360	S	0.0	0.0	0:00.00	bash
5339	bin	20	0	51872	2016	1468	S	0.0	0.0	1:05.26	top
5885	root	20	0	11768	1640	1380	S	0.0	0.0	0:00.00	bash
8938	root	20	0	11768	1636	1380	S	0.0	0.0	0:00.01	bash
8999	root	20	0	11768	1636	1380	S	0.0	0.0	0:00.00	bash
9044	root	20	0	11768	1632	1380	S	0.0	0.0	0:00.01	bash
9114	root	20	0	11768	1636	1380	S	0.0	0.0	0:00.00	bash
10051	root	20	0	11768	1816	1456	S	0.0	0.0	0:00.00	bash
10149	root	20	0	11768	1880	1500	S	0.0	0.0	0:00.02	bash
11588	root	20	0	51872	1952	1420	R	0.0	0.0	0:00.01	top
11604	root	20	0	4312	356	284	S	0.0	0.0	0:00.00	sleep
11958	root	20	0	11768	1888	1500	S	0.0	0.0	0:00.04	bash
15294	root	20	0	11768	1848	1488	S	0.0	0.0	0:00.00	bash
15526	root	20	0	24856	1700	1368	S	0.0	0.0	0:00.00	vi
15561	root	20	0	11772	1864	1500	S	0.0	0.0	0:00.02	bash
15656	root	20	0	11768	1856	1496	S	0.0	0.0	0:00.01	bash
15847	root	20	0	11768	1860	1496	S	0.0	0.0	0:00.00	bash
15860	root	20	0	24860	1696	1380	S	0.0	0.0	0:00.04	vi

# Problem6-监控方案选择

## ❑ docker stats , docker原生

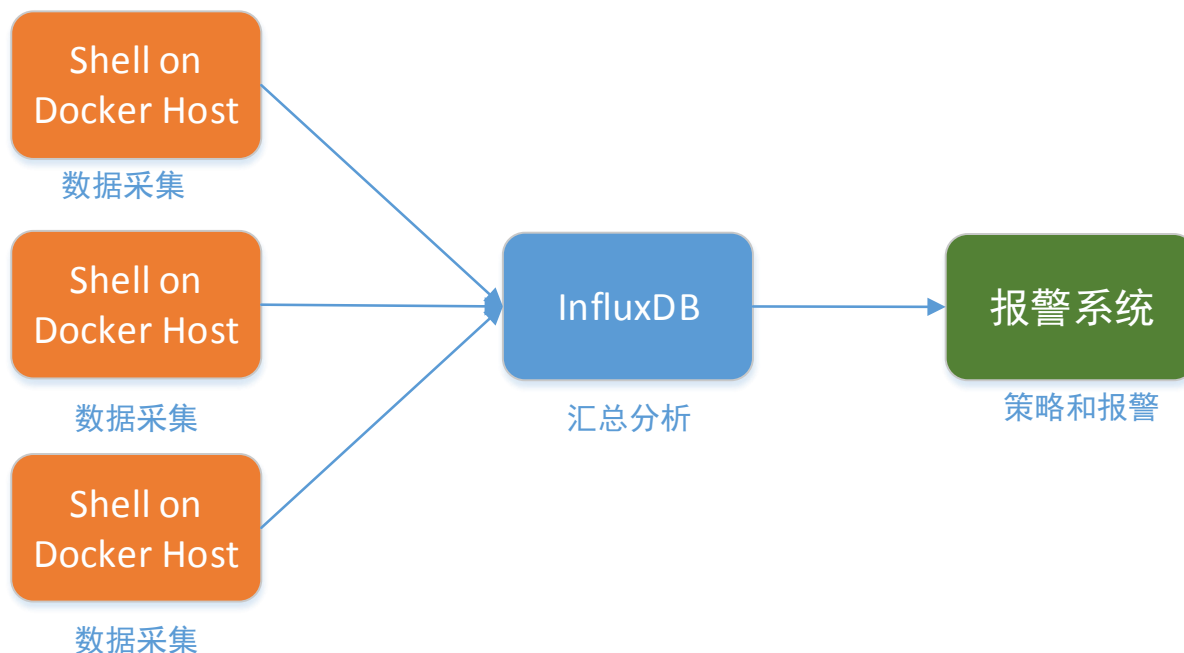
- memory计算争议
- 性能较差

## ❑ cAdvisor , Google开发 , 容器和主机级别监控

- 一定的学习成本
- 与已有监控报警系统集成有难度

# Solution6-自研Shell实现

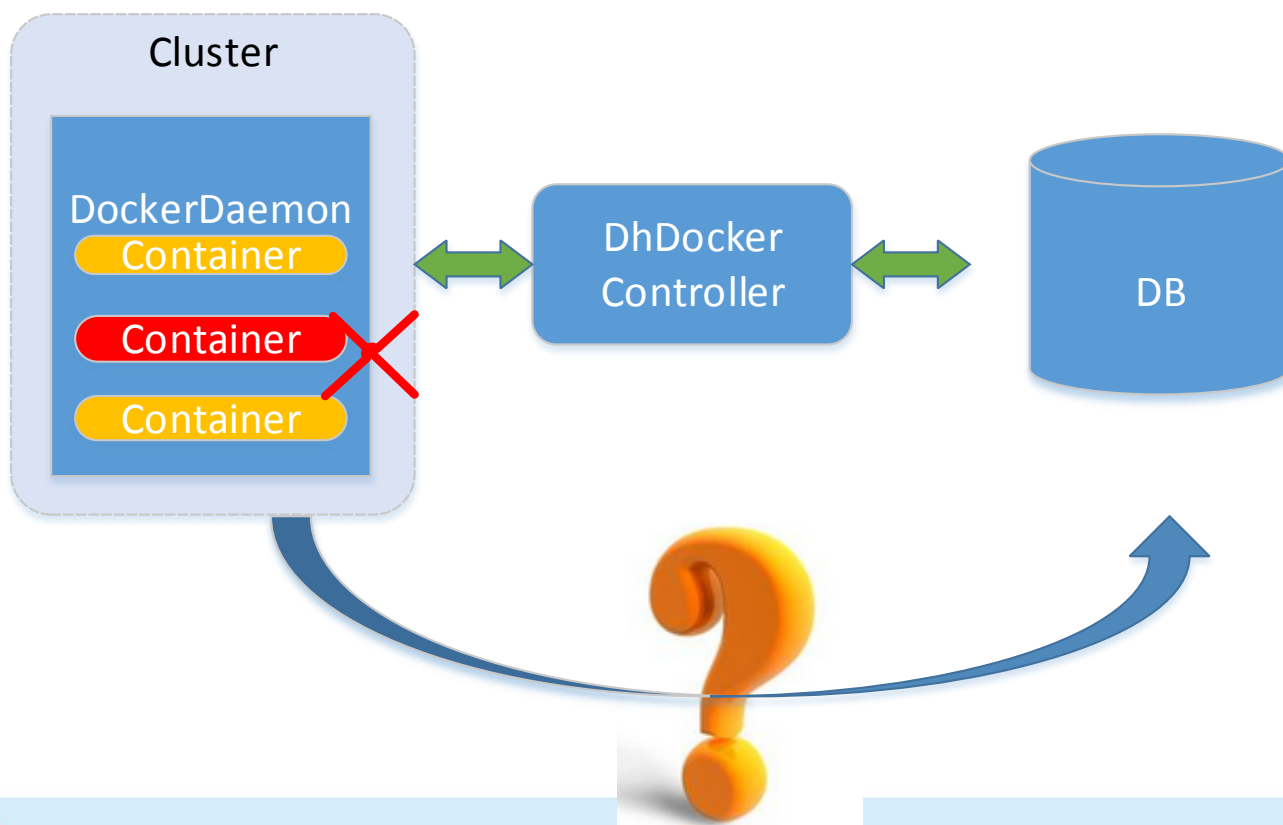
- ❑容器级别数据收集
- ❑应用级别数据统计
- ❑主机级别-容器总数及状态（性能由zabbix监控）



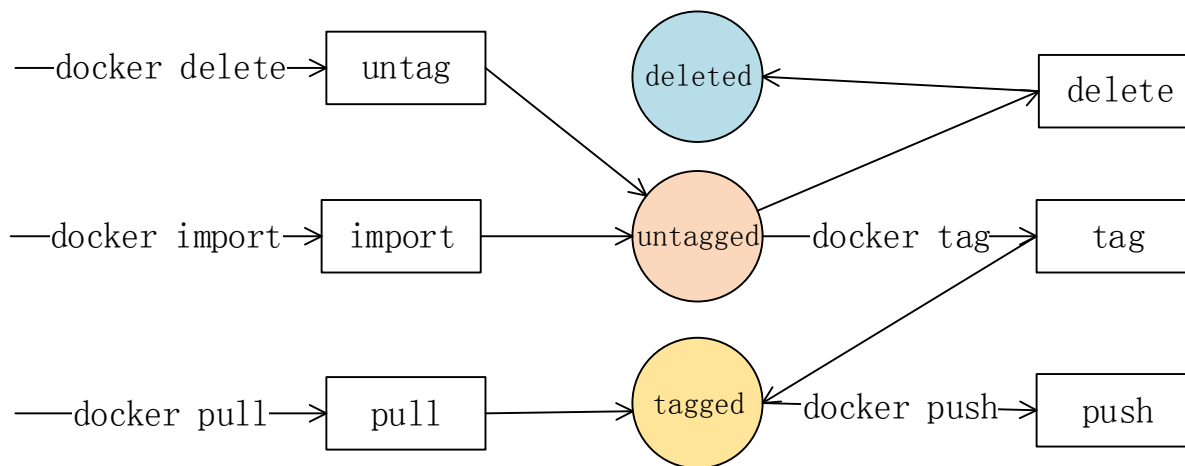


# Problem7-容器状态的同步

- 异常退出
- 命令行创建

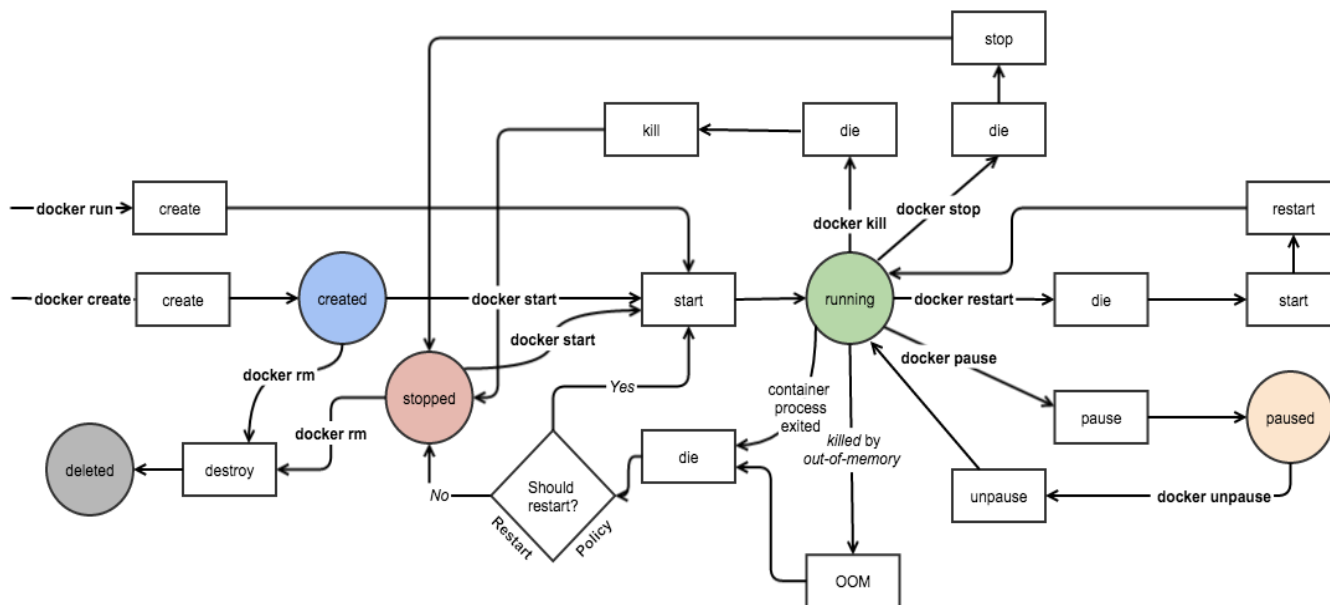


# Docker事件机制



## 镜像事件

## 容器事件

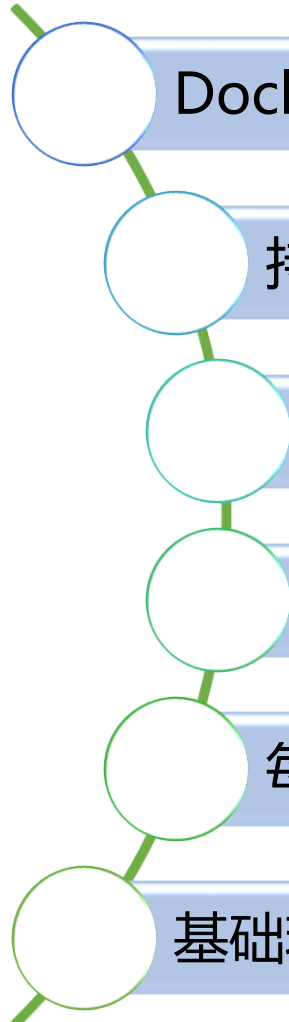


# Solution7-监听容器事件

```
import (  
    "os" ...  
    dockerApi "github.com/fsouza/go-dockerclient"  
)  
  
func main() {  
    ....  
    docker, err := dockerApi.NewClient(*dockerHost)  
    if err != nil {  
        //Add logging here  
        os.Exit(0)  
    }  
    events := make(chan *dockerApi.APIEvents)  
    docker.AddEventListener(events)  
    for msg := range events {  
        if msg.Status == "die" {  
            go inspectContainer(docker, msg.ID, msg.Status) //inspect the container and  
            update info to DB  
        }  
    }  
}
```

# 五、总结

# 总结



Docker+微服务架构，快速部署，弹性伸缩

持续集成，快速迭代，简化上线流程

混合部署，历史资产无缝迁移

基础镜像+应用下载，减小镜像库

每个应用都可以有局域网业务IP；指定静态ip

基础环境标准化，加速海外部署

# THANKS



<http://cloud.dhgate.com>