

# Docker容器云平台中监控 信息的搜集，展现和应用

# 绪论：容器技术的起源与发展

操作系统级别虚拟化



Cgroup Namespace



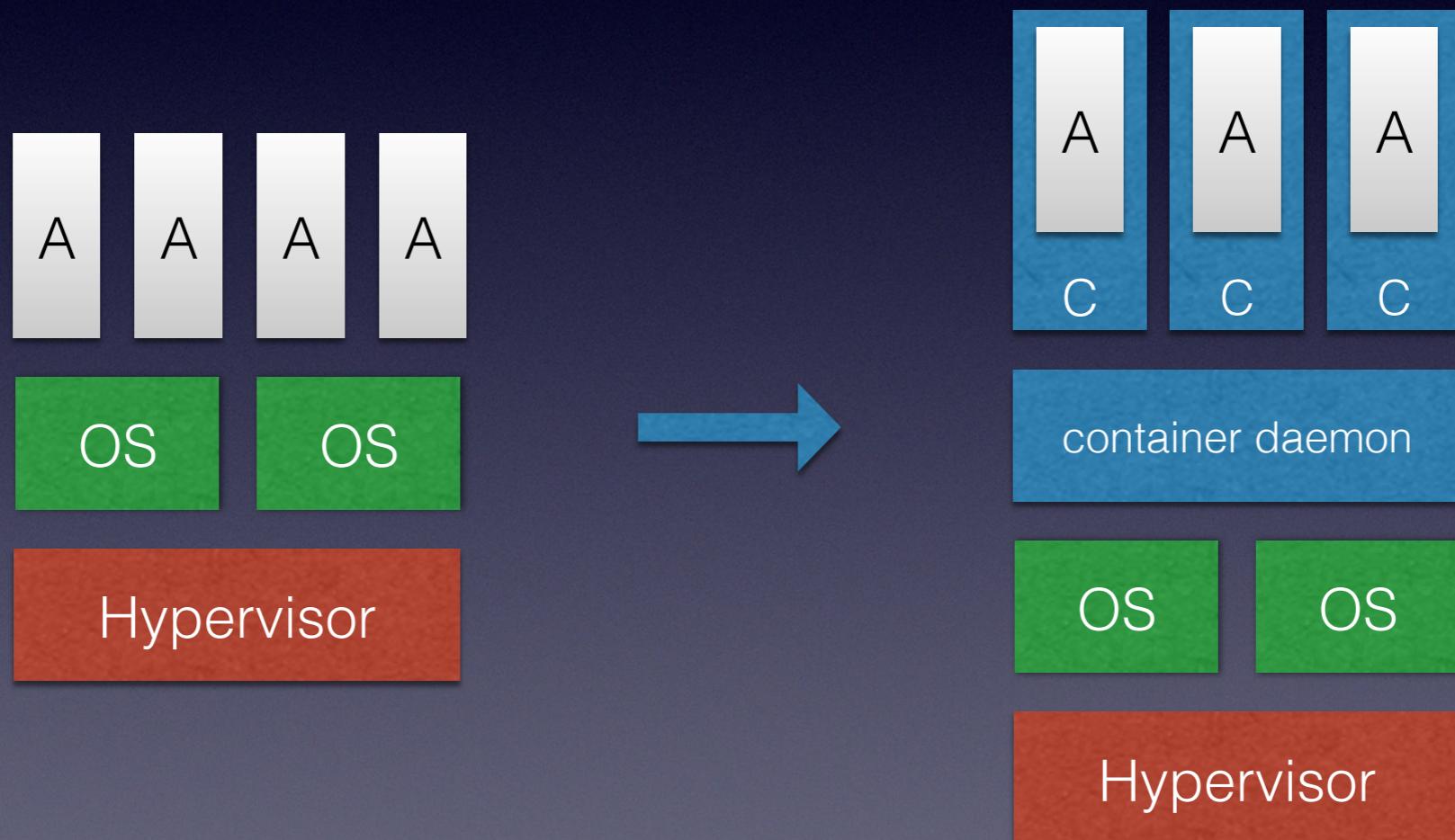
DockerContainer



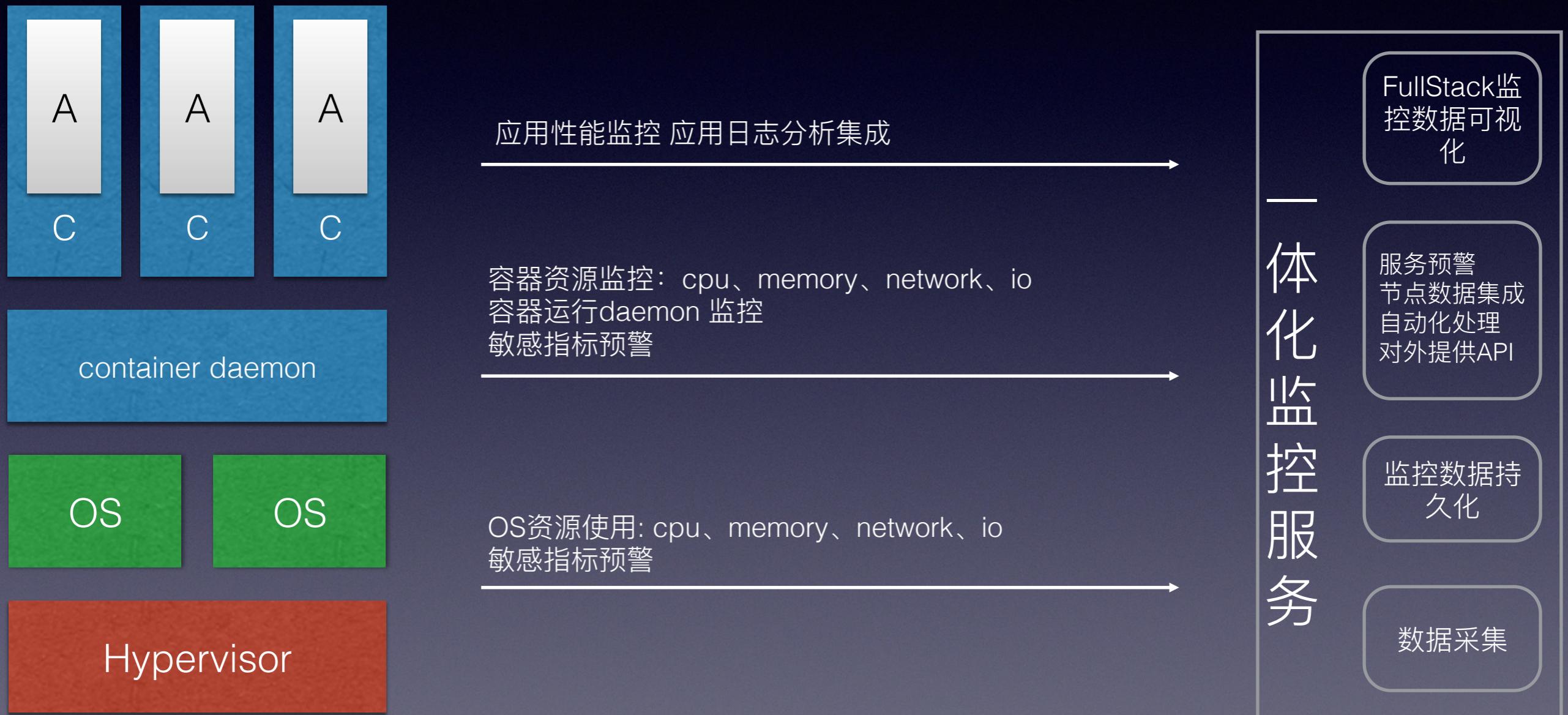
OCI OCF

# 绪论：容器监控技术的演变

A=Application  
C=Container



# 绪论：容器监控技术的演变



# 绪论:本文主要工作

1 容器监控信息的搜集方面:

容器主要资源指标搜集+网络包监听

2 容器监控信息的存储方面:

监控数据携带有特定的label信息上传数据库

基于服务发现的方式将监控信息汇总到key:value类型的数据服务中

3 容器监控信息的存储展示方面:

通过集成可视化工具灵活检索，聚合数据指标

拓扑图展现project-container结构

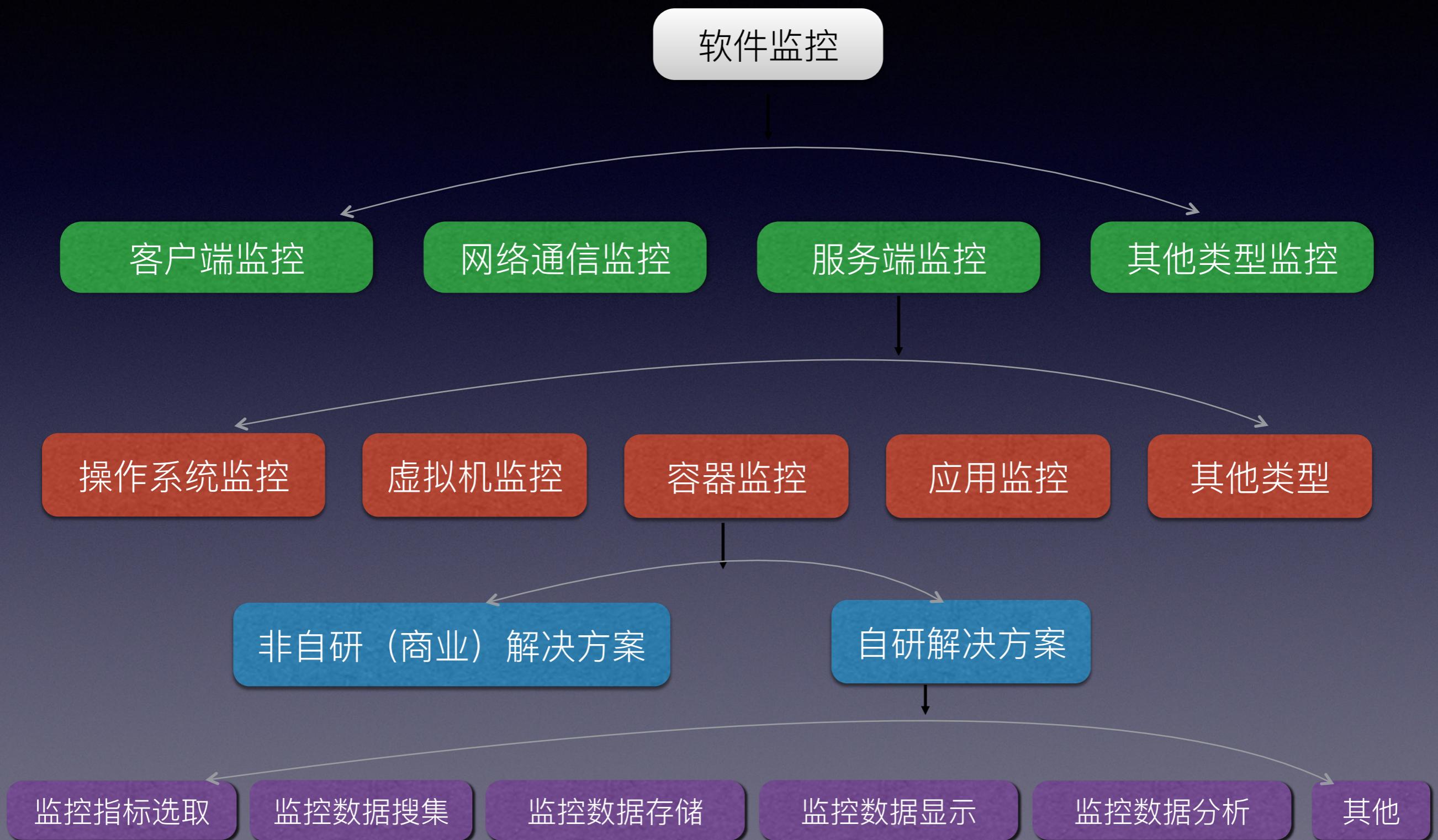
拓扑图展现容器之间网络调用关系

4 容器监控信息的应用方面:

结合报警规则进行自动报警

根据资源使用情况对容器进行分类

# 容器监控相关技术综述



# 容器监控相关技术综述

- 1 宿主机层面的解决方案：Ganglia, Nagios
- 2 容器层面的监控解决方案

商业解决方案： Datadog, Sysdig

自研解决方案：

监控指标的分类和选取

监控数据的搜集

监控数据的存储

监控数据的显示

监控数据的分析和应用

# 容器监控相关技术综述

## 容器层面的监控解决方案的主要难点

- 1 合适的容器监控指标难以确定
- 2 多样的监控信息的来源增加了监控系统的实现难度
- 3 容器平台中应用的层次结构增加了监控信息整合的难度
- 4 容器本身的多状态和增加监控的复杂度
- 5 容器的迁移调度会增加监控信息的搜集难度

# 监控信息的搜集

## 监控指标的目的与类型

了解系统的概况

系统资源占用状况和性能分析

系统资源不足时进行预警

为其他集群管理功能提供数据支撑

应用自定义的告警指标

# 监控信息的搜集

## 宿主机监控指标及其数据来源

指标名称	具体含义	数据来源	指标类别
machine.container.num	宿主机上存在的Docker容器的个数	Docker API	概况, 管理
machine.image.num	宿主机上容器镜像的个数	Docker API	概况, 管理
machine.general	宿主机上cpu, 内存, 网卡带宽, 磁盘IO情况	Docker API	性能、告警

# 监控信息的搜集

## 容器层面监控指标

指标名称	具体含义	数据来源	指标类别
docker.name	Docker容器的名称及其id	Docker API	概况、管理
docker.image	Docker容器启动时所使用镜像的id	Docker API	概况、管理
...	...	...	...
docker.cpu.user	容器内部在运行期间 cpu在用户态下的时间占比	/sys/fs/cgroup	性能、告警、管理
docker.cpu.sys	容器内部在运行期间 cpu在系统态下的时间占比	/sys/fs/cgroup	性能、告警、管理
docker.memory.used	某个容器已经使用的memory总量	/sys/fs/cgroup	性能、告警、管理
docker.net.rbps	某个容器的网卡每秒钟接受到的byte数目	/sys/fs/cgroup	性能、告警、管理
docker.net.tbps	某个容器的网卡每秒钟发出的byte数目	/sys/class/net	性能、告警、管理
docker.diskio.rbps	容器每秒通过cgroup转发到linux设备上的bytes数目	/proc文件系统	性能、告警、管理
docker.diskio.wbps	容器每秒从linux设备上写入容器文件系统的bytes数	/proc文件系统	性能、告警、管理
docker.diskio.delay	容器内的进程在单位时间内进行读写操作所花费的时间	Netlink接口	性能、告警、管理
docker.network.rpdtime	宿主机上某个http请求的相应时间	额外计算	其他
docker.type	容器在一段时间内资源占用的类型	额外计算	其他

# 监控信息的搜集

宿主机文件系统：

/proc 目录下获取宿主机信息

/sys/fs/cgroups 下获取容器cgroup子系统类型：

找到对应子系统下docker的id

根据对应的容器id中读取出指定目录下希望得到监控数据

比如/sys/fs/cgroup/cpuacct/docker/0366…/cpuacct.stat

文件中包含了id为0366…这个容器对于cpu资源的使用情况

# 监控信息的搜集

Docker Daemon API 获取

GET /containers/(id or name)/json

容器的 label 信息

启动该容器的镜像名称

容器本身的状态信息

每个容器的 event 信息（标记容器状态转移）

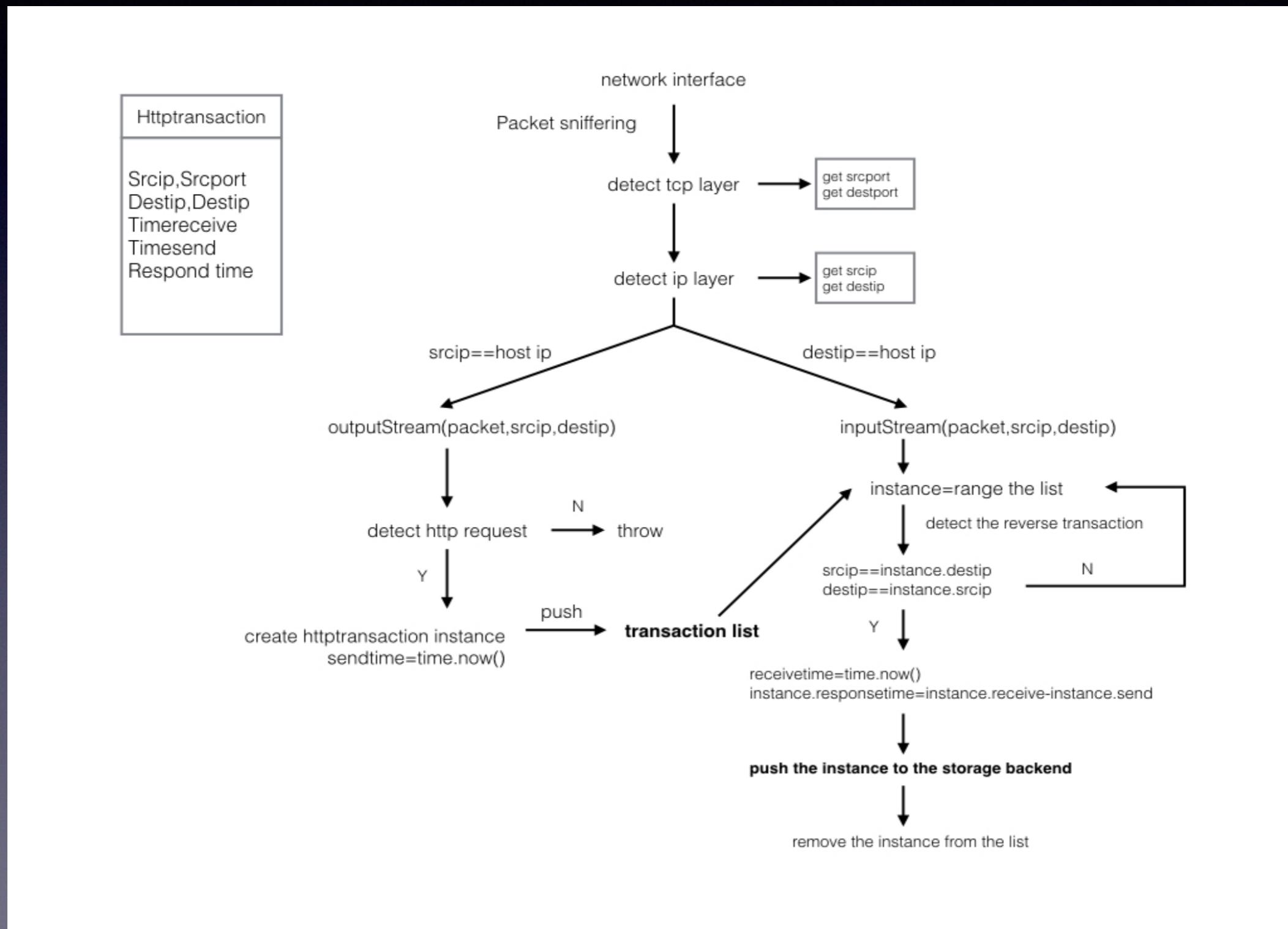
# 监控信息的搜集

其他方式获取信息

通过netlink接口与操作系统内核通信（由于I/O操作导致的CPU时间延迟）

HTTP网络包的返回时间延迟：监听宿主机网卡

# 监控信息的搜集 (HttpResponsetime算法)



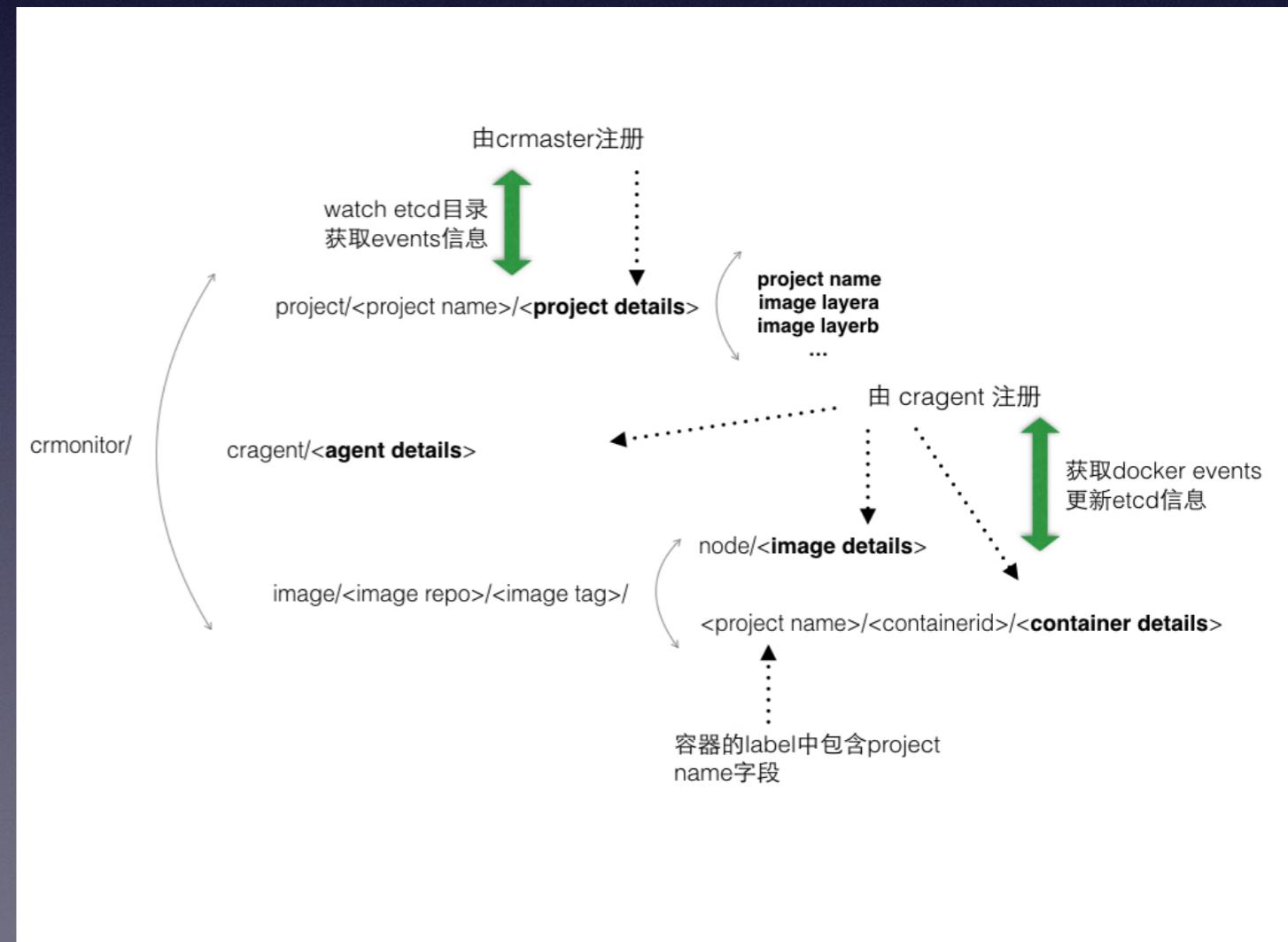
# 监控数据的存储

存储用于图表显示的数据 (influxdb)

存储用于拓扑图展示的数据 (etcd)

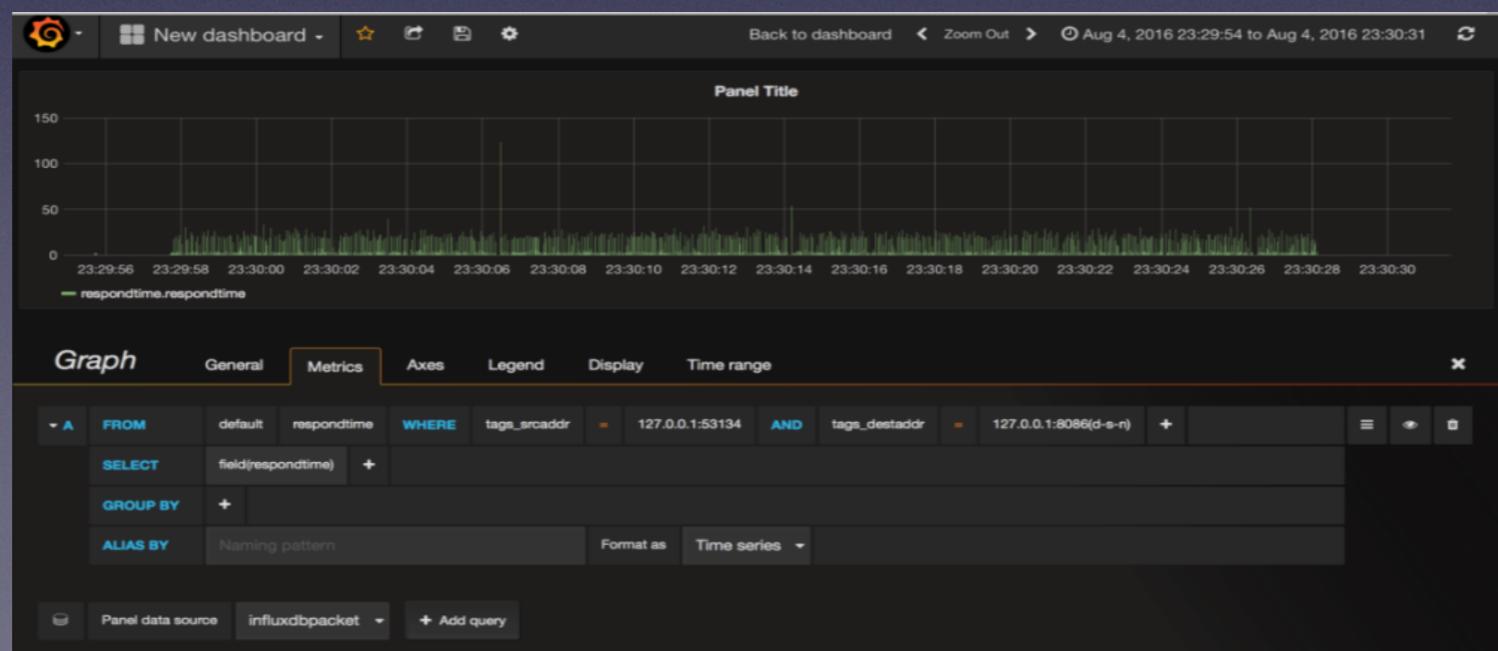
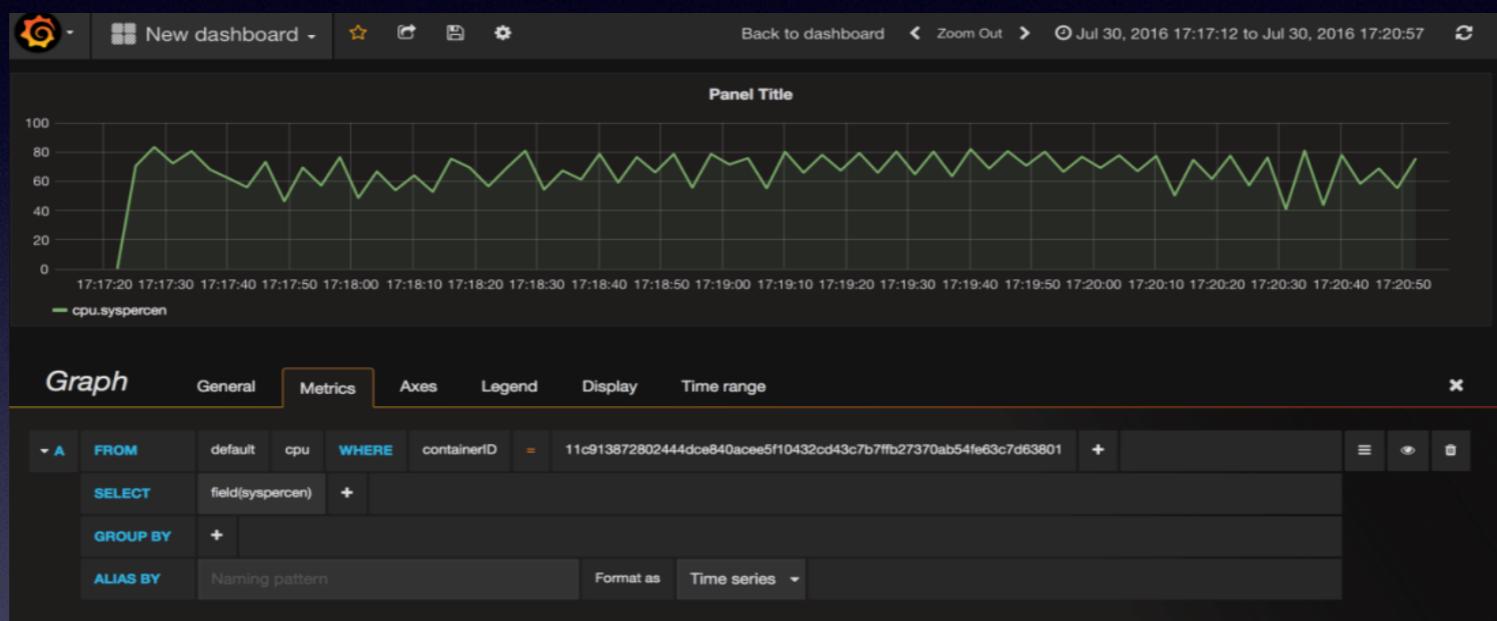
## 服务注册与服务发现理论

### etcd中的数据结构



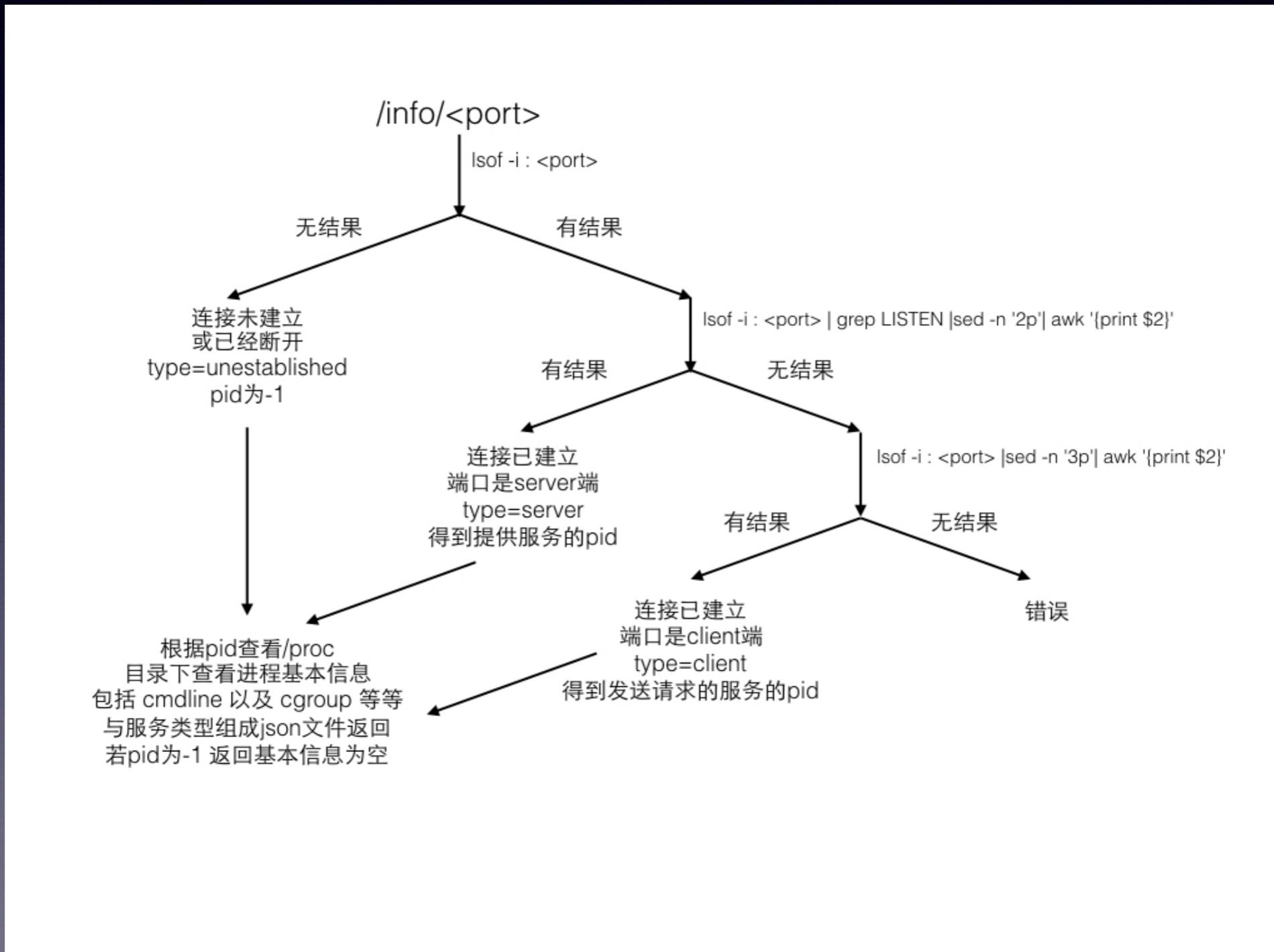
# 监控信息的展示

## 容器基本监控信息的图表展示



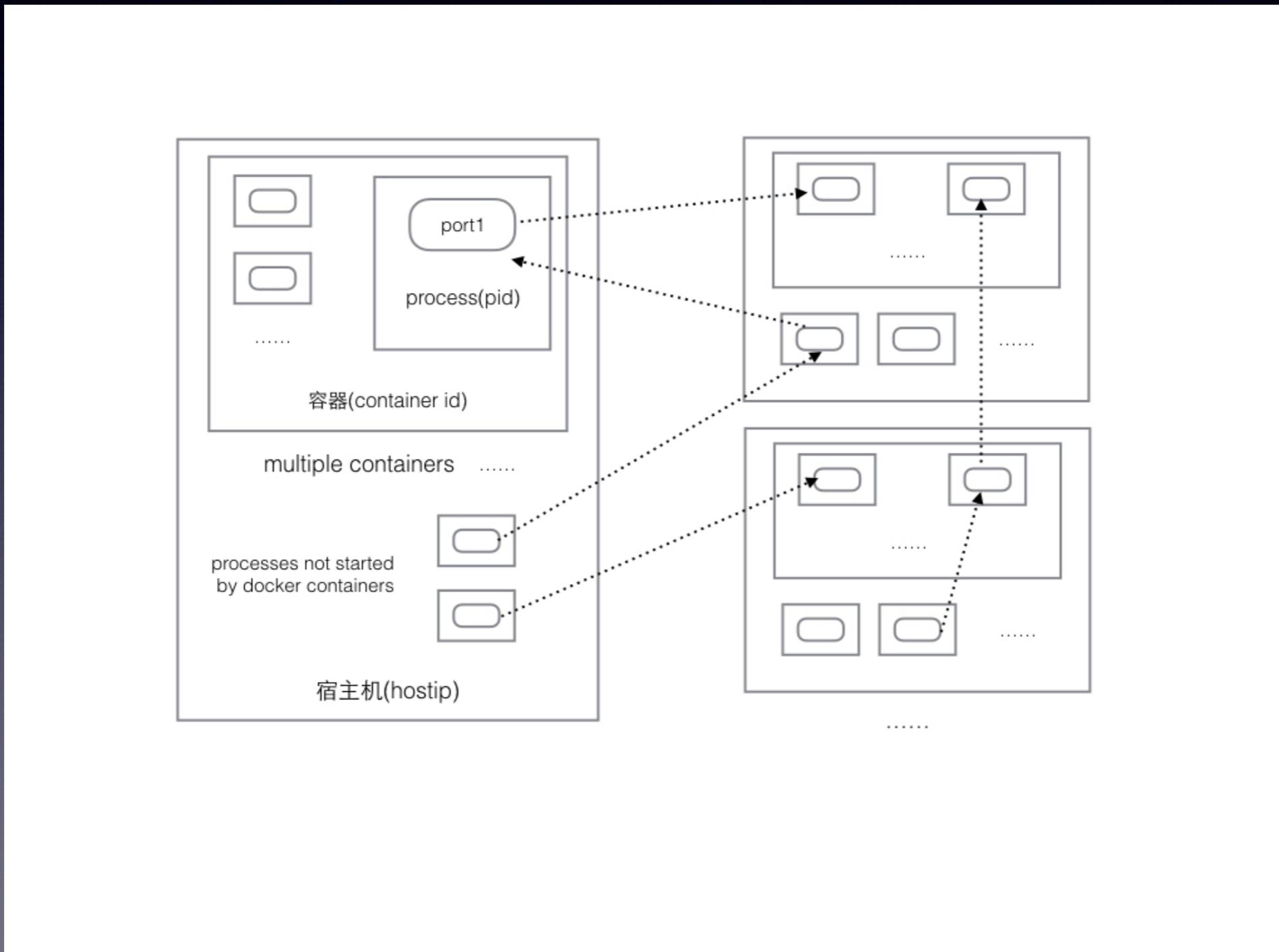
# 监控信息的展示

## 容器网络调用拓扑可视化（通过端口获取进程元信息）



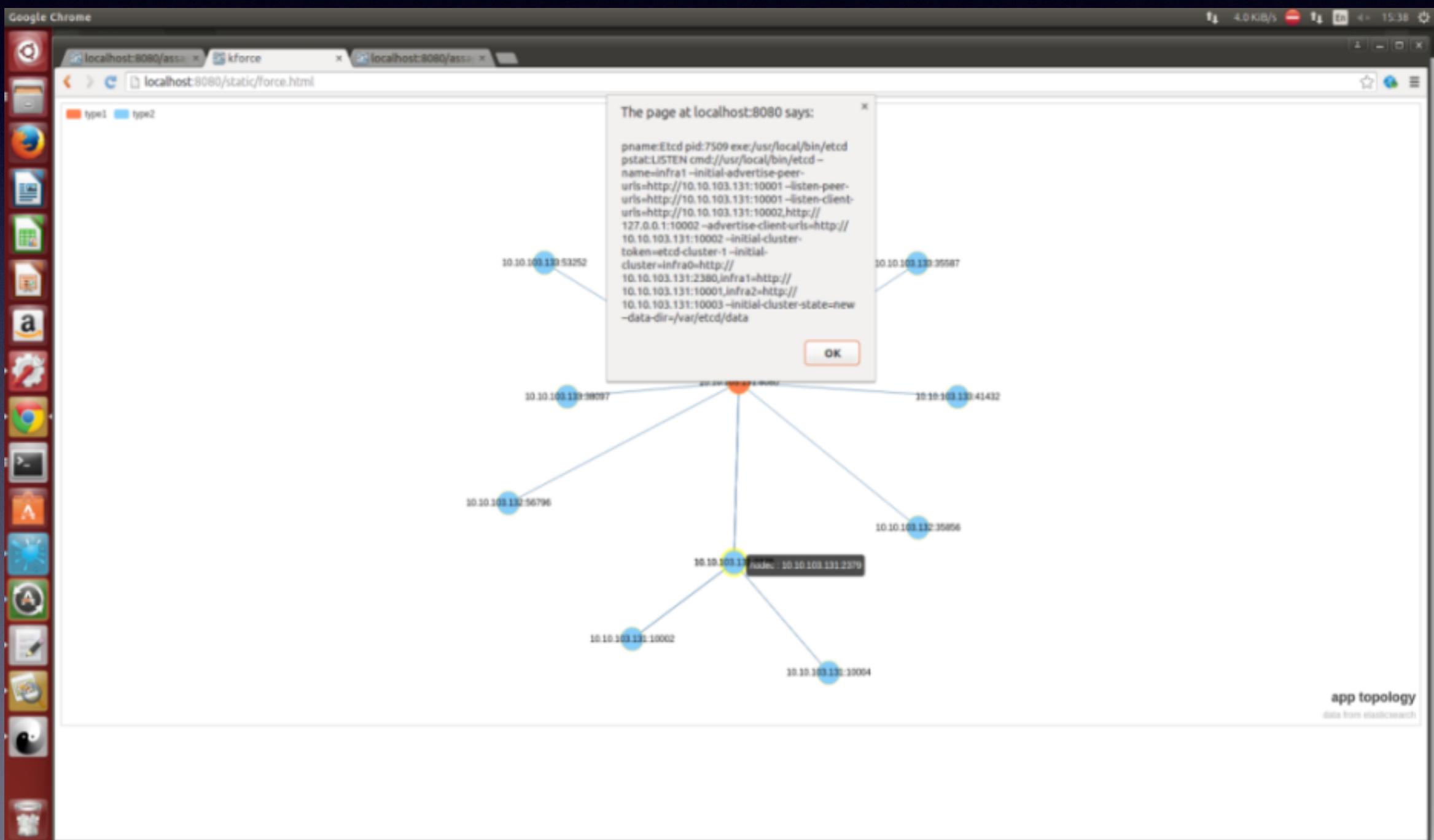
# 监控信息的展示

## 容器网络调用拓扑可视化



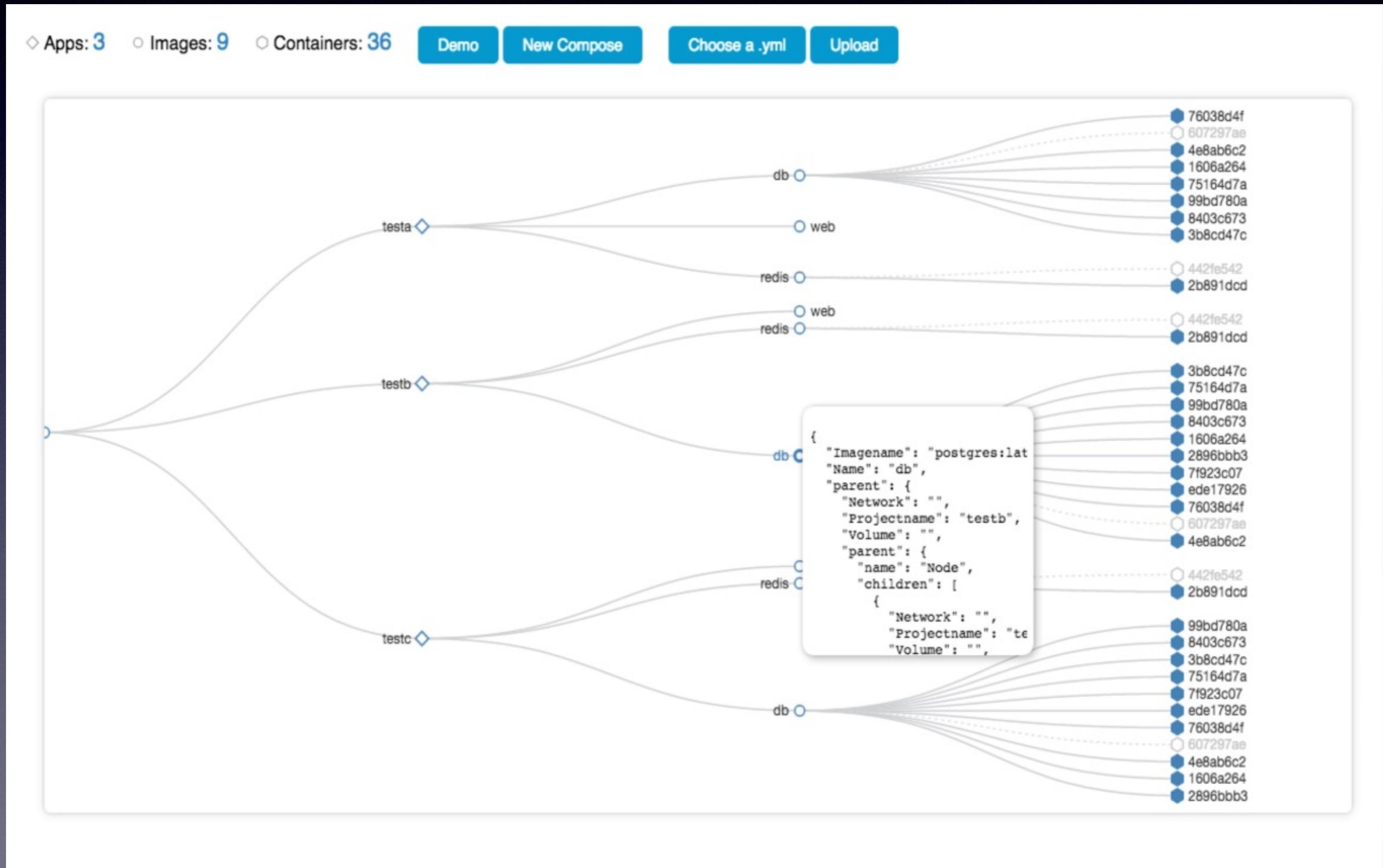
# 监控信息的展示

## 容器网络调用拓扑可视化



# 监控信息的展示

## 应用层级拓扑可视化展示



# 监控信息的应用

监控信息用于资源告警

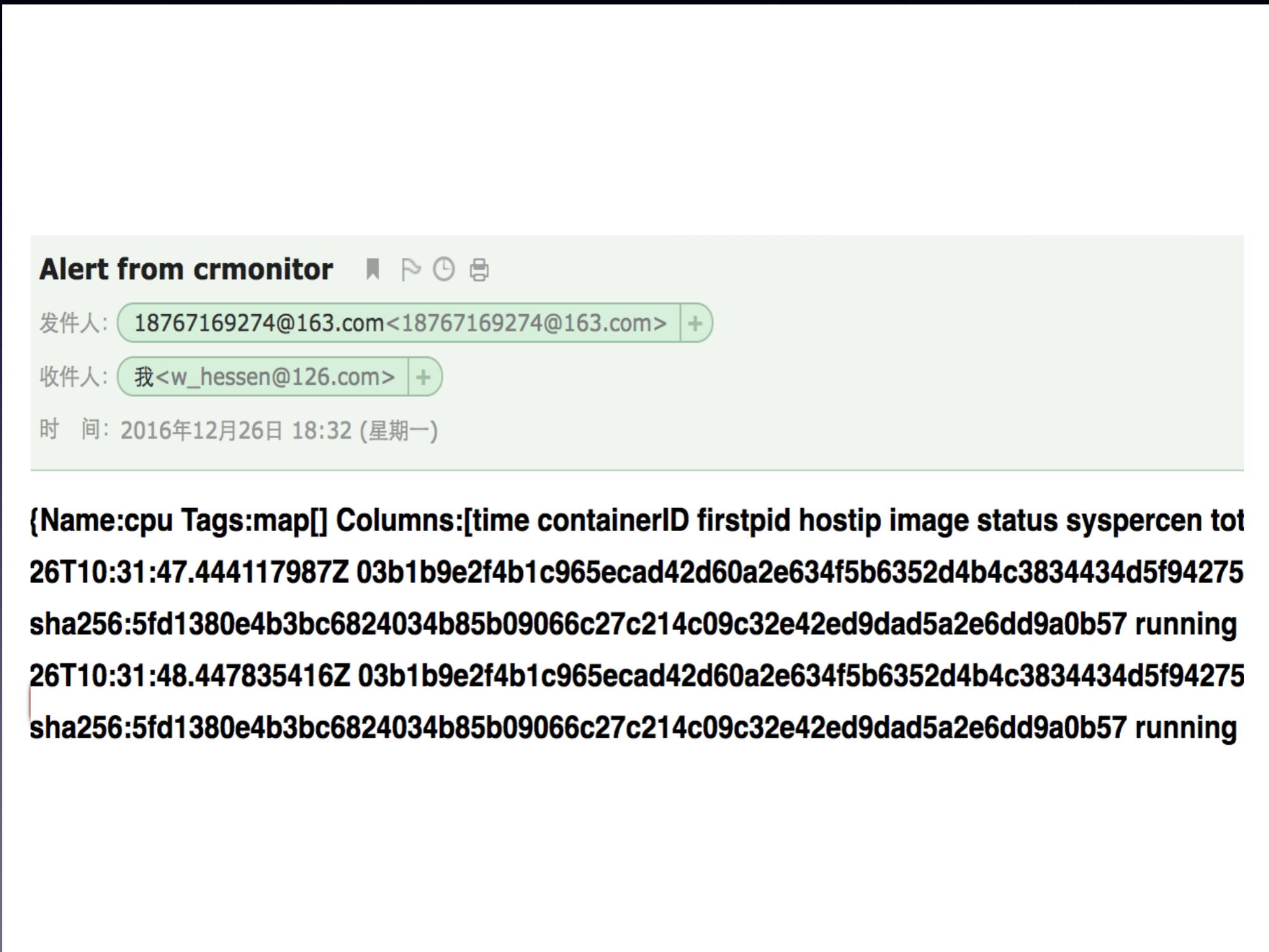
监控信息用于对容器进行分类

模糊分析中的关键概念

分类模型的确定

容器具体类型的判别

# 监控信息用于资源告警

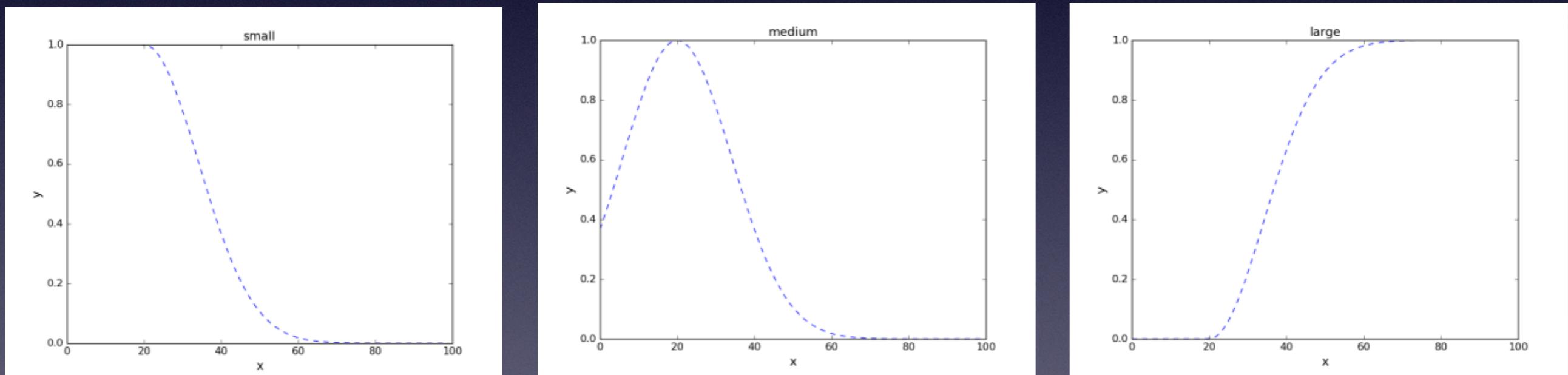


# 监控信息用于容器分类

确定论域，模糊子集，隶属度函数

论域包括：CPU, MEM, BLK\_IO, BLK\_IN, BLK\_OUT

模糊子集包括：S=“偏小”，M=“中等”，L=“偏大”



论域CPU的三种模糊集的隶属函数

# 监控信息用于容器分类

用于分类的5维数据指标:  $V = (V_{\downarrow \text{cpu}}, V_{\downarrow \text{mem}}, V_{\downarrow \text{blkio}}, V_{\downarrow \text{netin}}, V_{\downarrow \text{netout}})$

数据指标	具体含义
$V_{\downarrow \text{cpu}}$	单位时间内当前Docker容器中所有进程的cpu总时间所占总时间百分比
$V_{\downarrow \text{mem}}$	单位时间内当前Docker容器中所有进程所占服务器节点总内存百分比
$V_{\downarrow \text{blkio}}$	单位时间内当前Docker容器中所有进程由于进行读写操作所花费的时间所占总时间的百分比
$V_{\downarrow \text{netin}}$	单位时间内当前Docker容器所使用的网卡所接收的数据流量超过指定阈值的时间占总时间的百分比
$V_{\downarrow \text{netout}}$	单位时间内当前Docker容器所使用的网卡所发送数据流量超过指定阈值的时间占总时间的百分比

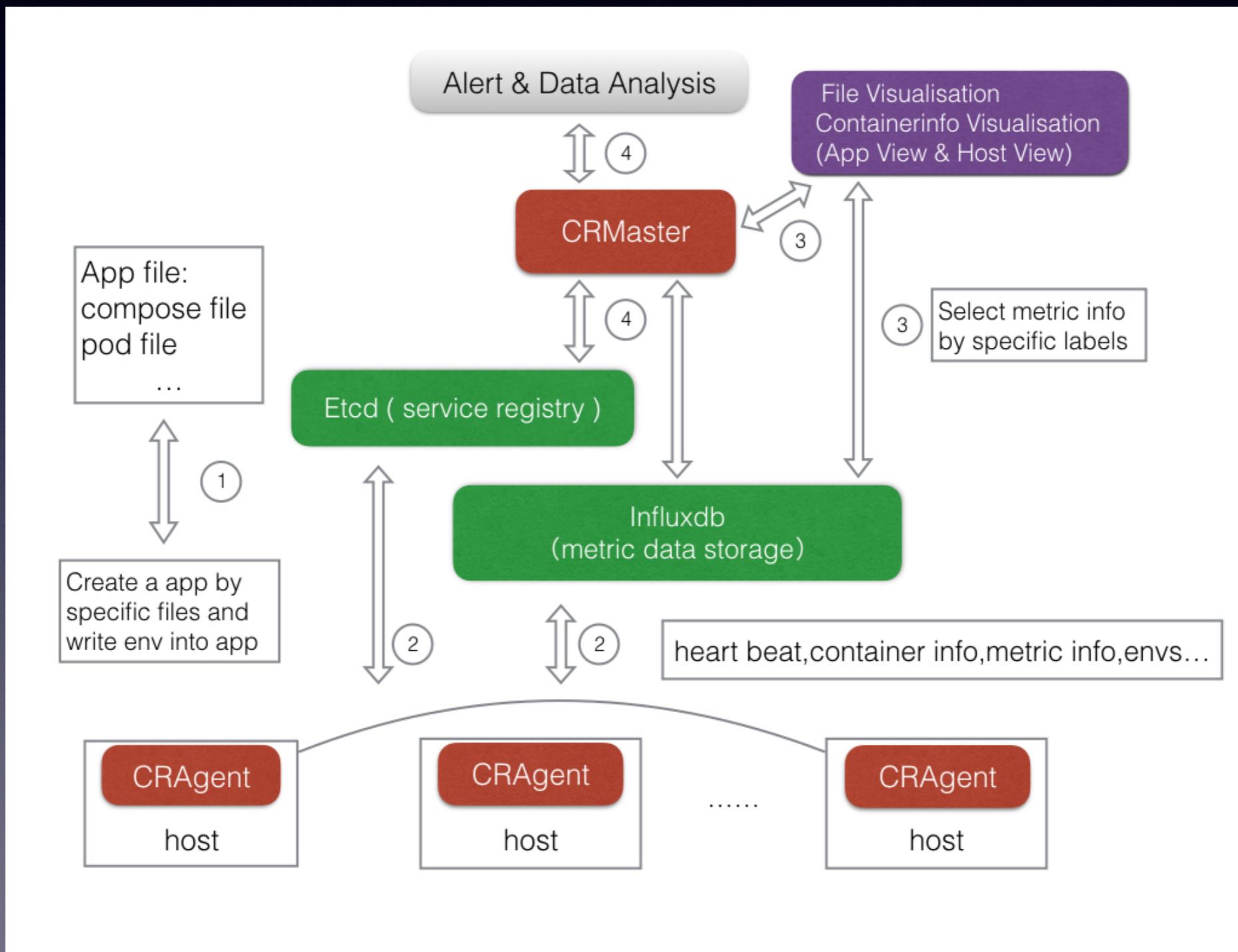
# 监控信息用于容器分类

	CPU	MEM	BLK	NETIN	NETOUT
偏小型	V↓cpu1	V↓mem1	V↓blkio1	V↓netin1	V↓netout1
中等型	V↓cpu2	V↓mem2	V↓blkio2	V↓netin2	V↓netout2
偏大型	V↓cpu3	V↓mem3	V↓blkio3	V↓netin3	V↓netout3

对于列标题为CPU的三个值，它们分别表示当前容器隶属于CPU使用率偏小，中等，偏大的集合的概率是V↓cpu1 , V↓cpu2 , V↓cpu3  
 $V↓type = V↓cpu1 \vee V↓cpu2 \vee V↓cpu3$   
 $V↓type = V↓cpu1$  则在论域CPU上，该容器的模糊子集为偏小型S。

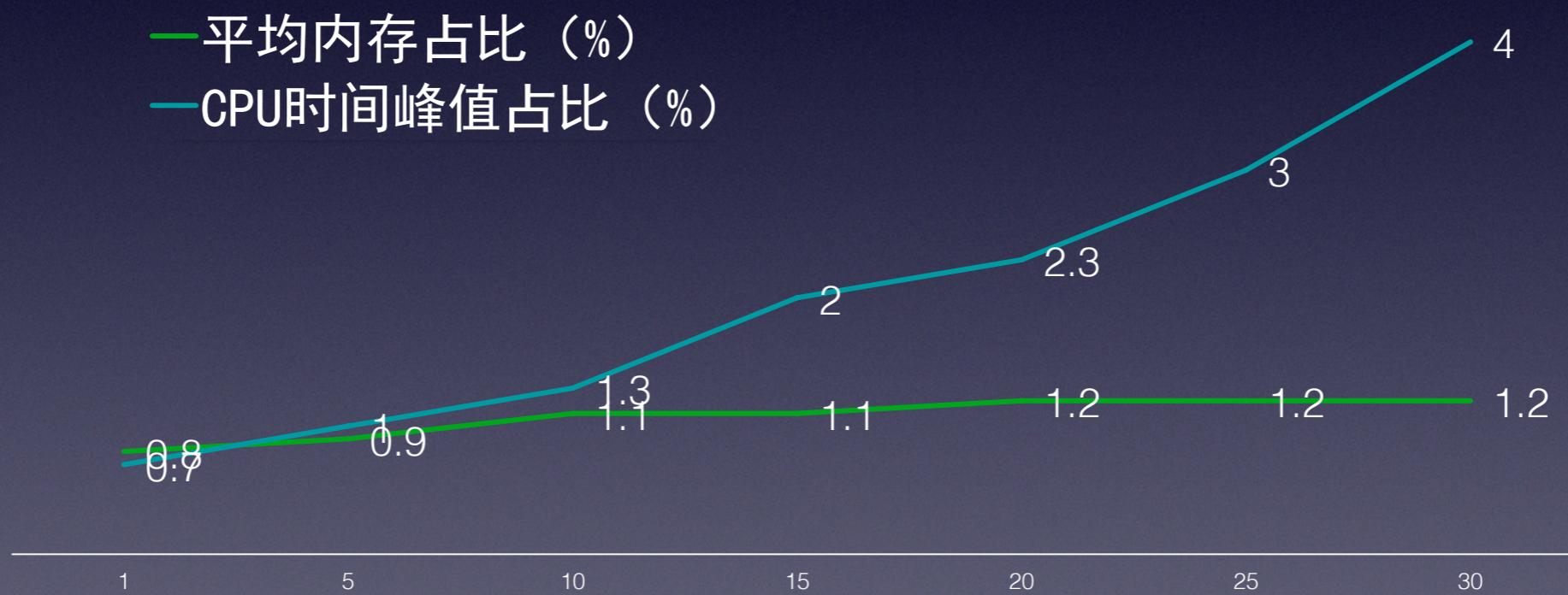
# 监控功能的实验与分析

## CRMonitor 监控系统架构图



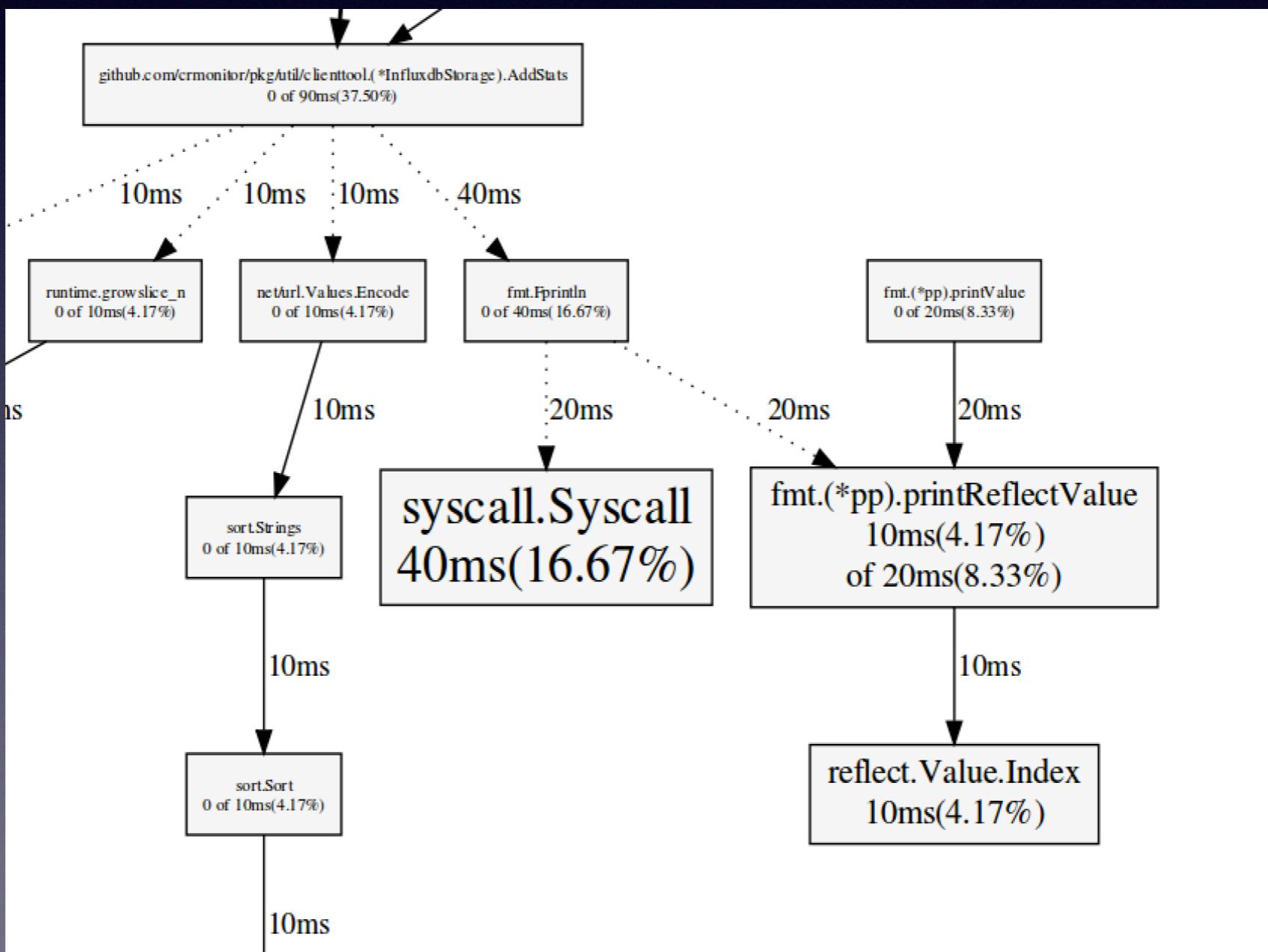
# 监控功能的实验与分析

## 资源消耗随容器个数变化



# 监控功能的实验与分析

## 组件代码的优化



# 监控功能的实验与分析

## 优化之前的函数cpu占用时间

```
(pprof) top
170ms of 240ms total (70.83%)
Showing top 10 nodes out of 105 (cum >= 20ms)
      flat  flat%  sum%          cum  cum%
        40ms 16.67% 16.67%      40ms 16.67%  runtime.usleep
        40ms 16.67% 33.33%      40ms 16.67%  syscall.Syscall
        20ms  8.33% 41.67%      20ms  8.33%  runtime.(*mcentral).grow
        10ms  4.17% 45.83%      20ms  8.33%  fmt.(*pp).printReflectValue
        10ms  4.17% 50.00%      10ms  4.17%  reflect.Value.Index
        10ms  4.17% 54.17%      10ms  4.17%  runtime.acquirep
        10ms  4.17% 58.33%      10ms  4.17%  runtime.cgocall
        10ms  4.17% 62.50%      10ms  4.17%  runtime.cmpbody
        10ms  4.17% 66.67%      10ms  4.17%  runtime.futex
        10ms  4.17% 70.83%      20ms  8.33%  runtime.growslice
```

## 优化之后的函数cpu占用时间

```
(pprof) top
870ms of 1640ms total (53.05%)
Showing top 10 nodes out of 253 (cum >= 170ms)
      flat  flat%  sum%          cum  cum%
       190ms 11.59% 11.59%      210ms 12.80%  runtime.cgocall
       170ms 10.37% 21.95%      170ms 10.37%  runtime.futex
       120ms  7.32% 29.27%      240ms 14.63%  runtime.mallocgc
        90ms  5.49% 34.76%      90ms  5.49%  syscall.Syscall
        80ms  4.88% 39.63%      80ms  4.88%  runtime.memmove
        60ms  3.66% 43.29%      90ms  5.49%  runtime.scanobject
        60ms  3.66% 46.95%      60ms  3.66%  runtime.usleep
        40ms  2.44% 49.39%      40ms  2.44%  runtime.heapBitsForObject
        30ms  1.83% 51.22%      30ms  1.83%  runtime.(*mspan).sweep.func1
        30ms  1.83% 53.05%      170ms 10.37%  runtime.newobject
```

# 总结与展望

- 1 在前端的显示上界面的易用性和界面的美观性上还可以进行进一步的优化。
- 2 报警规则方面以及报警通知手段的多样性上还可以进行进一步的优化。
- 3 优化http网络包解析的算法提升组件的运行效率。
- 4 指标搜集的方式提供对不同操作系统的支撑。
- 5 本文中对容器的分类算法仅仅进行了理论探讨