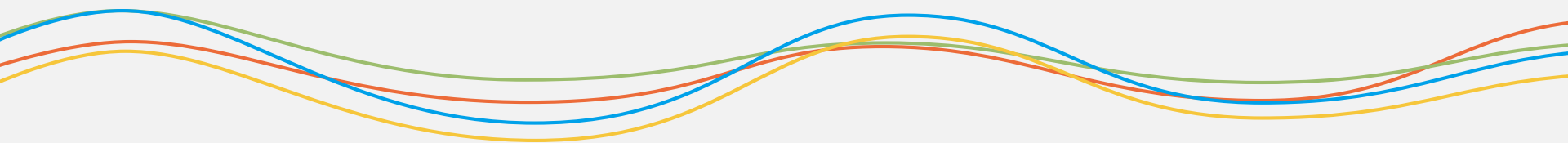


Web服务架构变化及性能优化

尤春



Web架构变化

- Web演变
- PaaS架构

Web性能优化

- 动态调度
- 服务治理
- 日志监控

Web服务趋势

- XaaS
- Serverless

一、Web架构演变

- All in one Arch

- 架构清晰、便于维护、成本很低
- 提升单机负载 和 容灾能力

- Application Clusters Arch

- 负载均衡

keepalived+ipvsadm

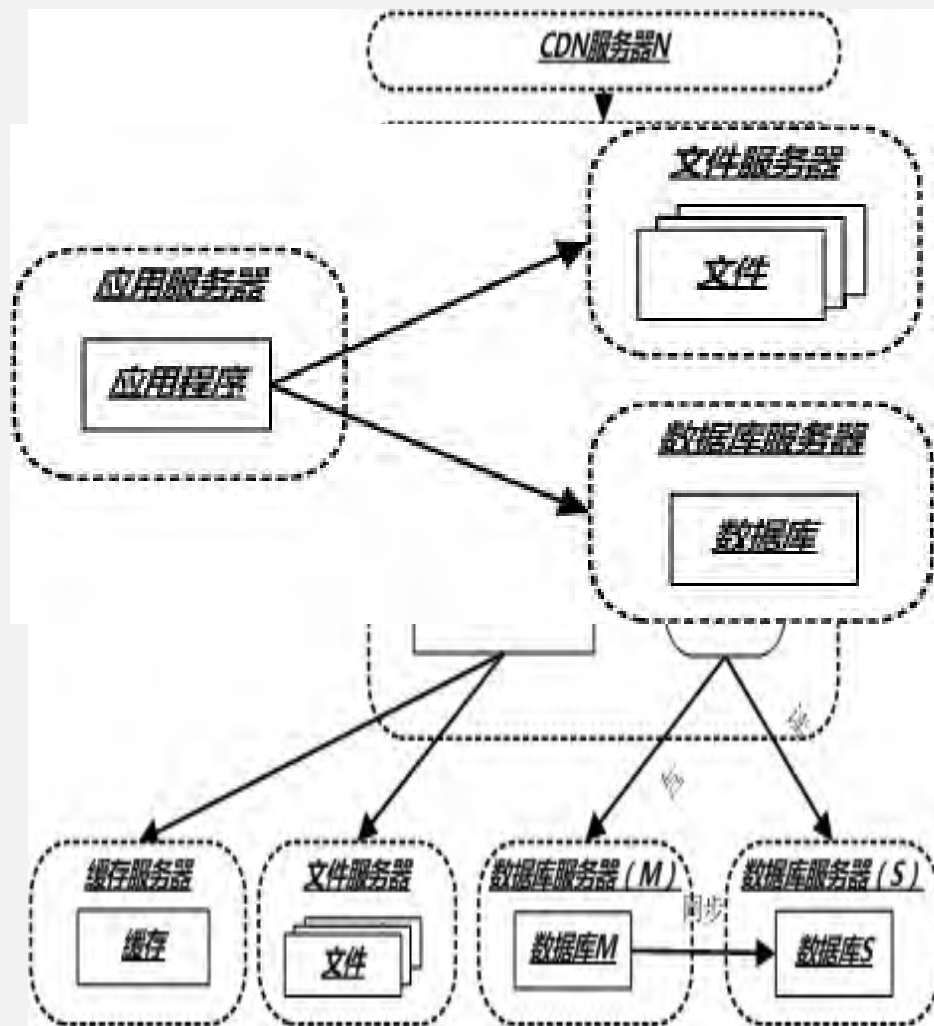
- 反向代理

apache、nginx

- 集群调度算法与模式

算法：rr、wrr、sh、dh 等等

模式：nat、dr、tun



一、Web架构演变

- Distributed Arch

- 状态共享 (Session一致性)

1. Session Sticky
2. Session Replication
3. Session数据集中存储
4. Cookie Base

- 数据库优化

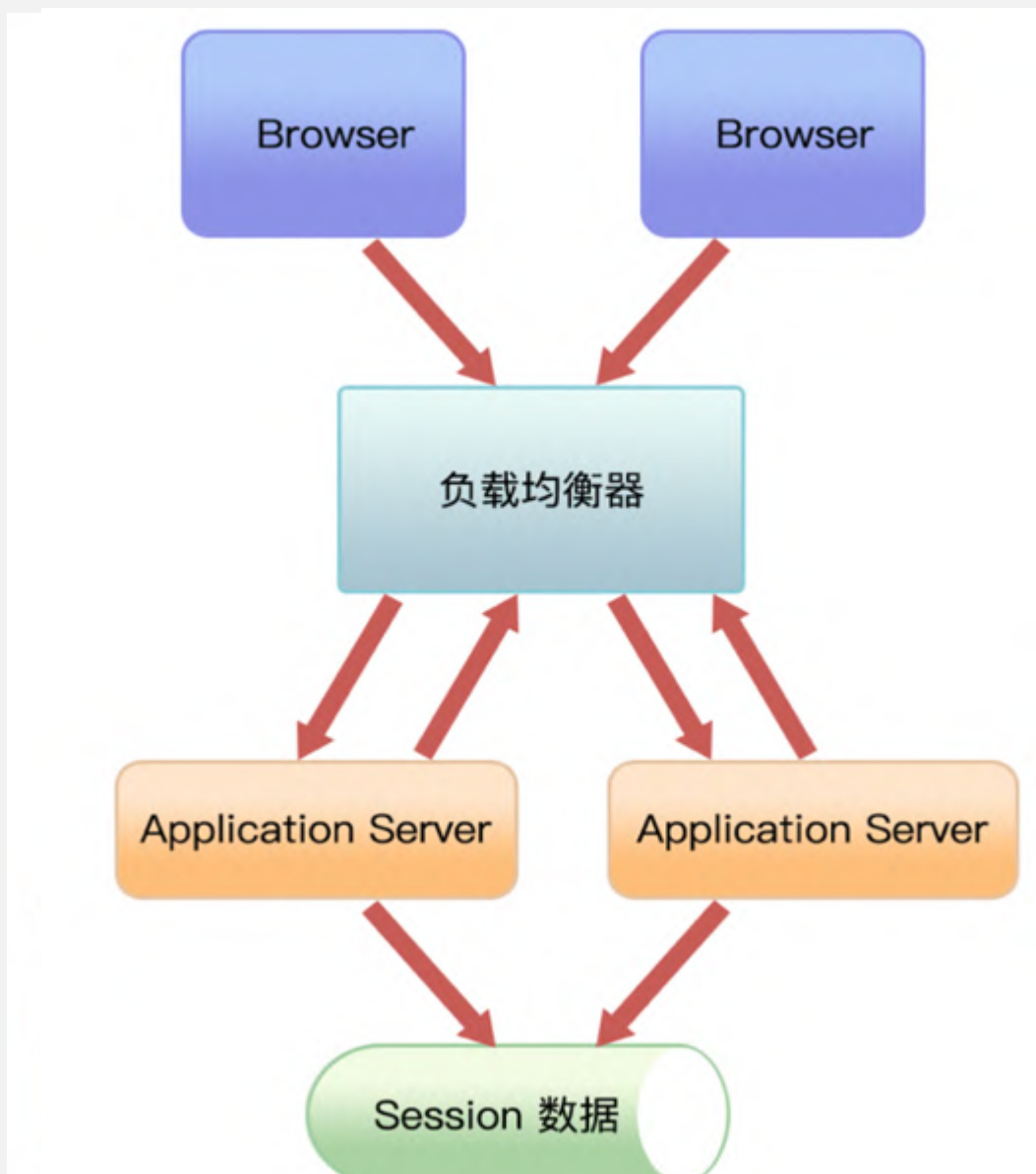
- 1.读写分离 (主从复制)
- 2.分库分表 (跨库事务)

- 缓存优化

1. 页面缓存
2. 数据缓存

- 应用拆分

- 1.业务解耦
- 2.故障隔离



一、Web架构演变

- Service Arch

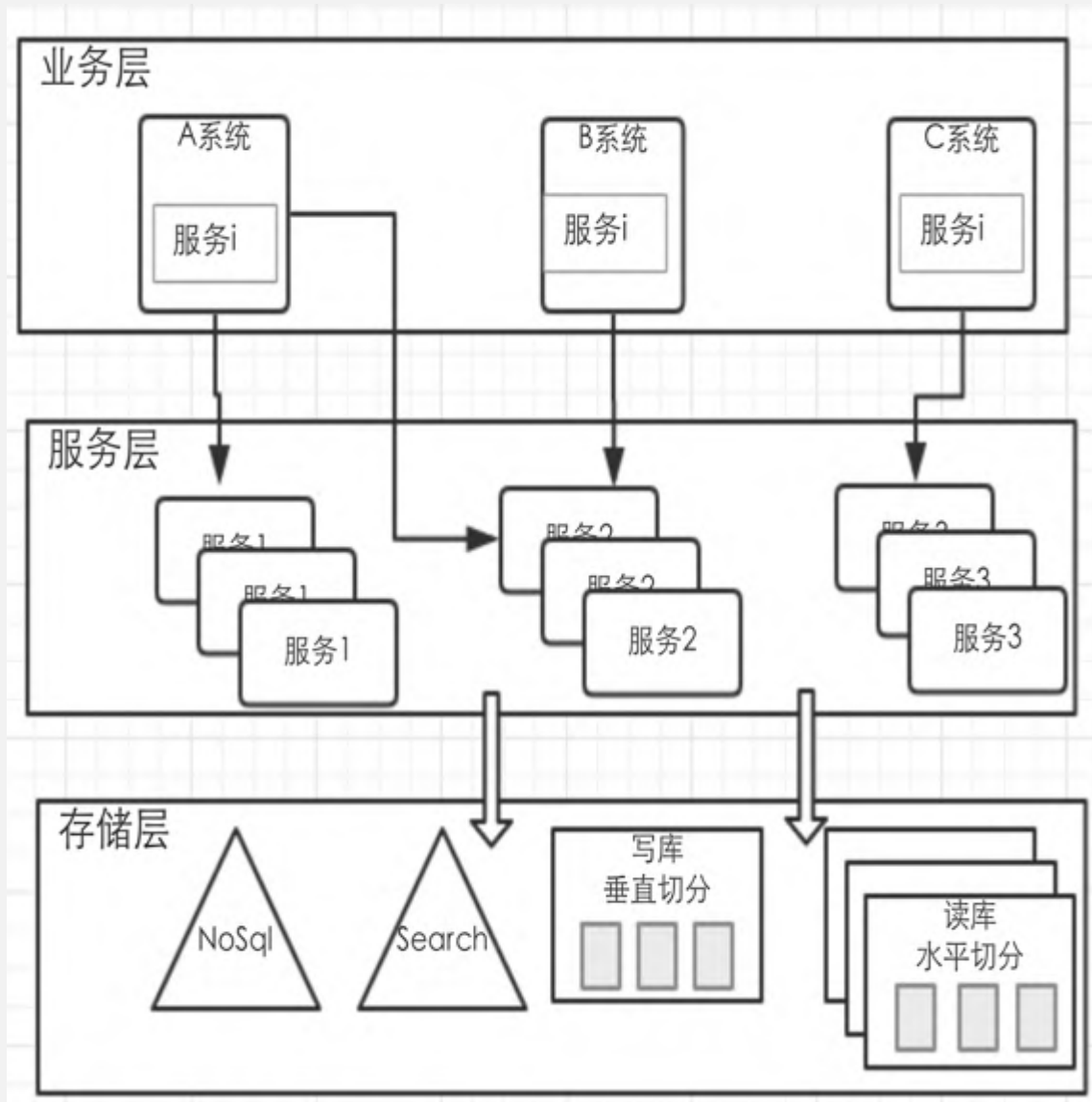
- 代码模块复用
- 分层，便于集中优化
- 远程调用

异步 (message)

同步 (Rpc / Rest)

- Elastic Arch

- 服务实例，弹性伸缩



二、为什么选择PaaS平台？

系统架构

多线路/
多机房

HA保障

故障自
动迁移

弹性伸缩

系统可
扩展性

高负载
时自动
扩充

低负载
时自动
收缩

开发效率

统一平
台和架
构

基础
Services

专注于
业务逻
辑

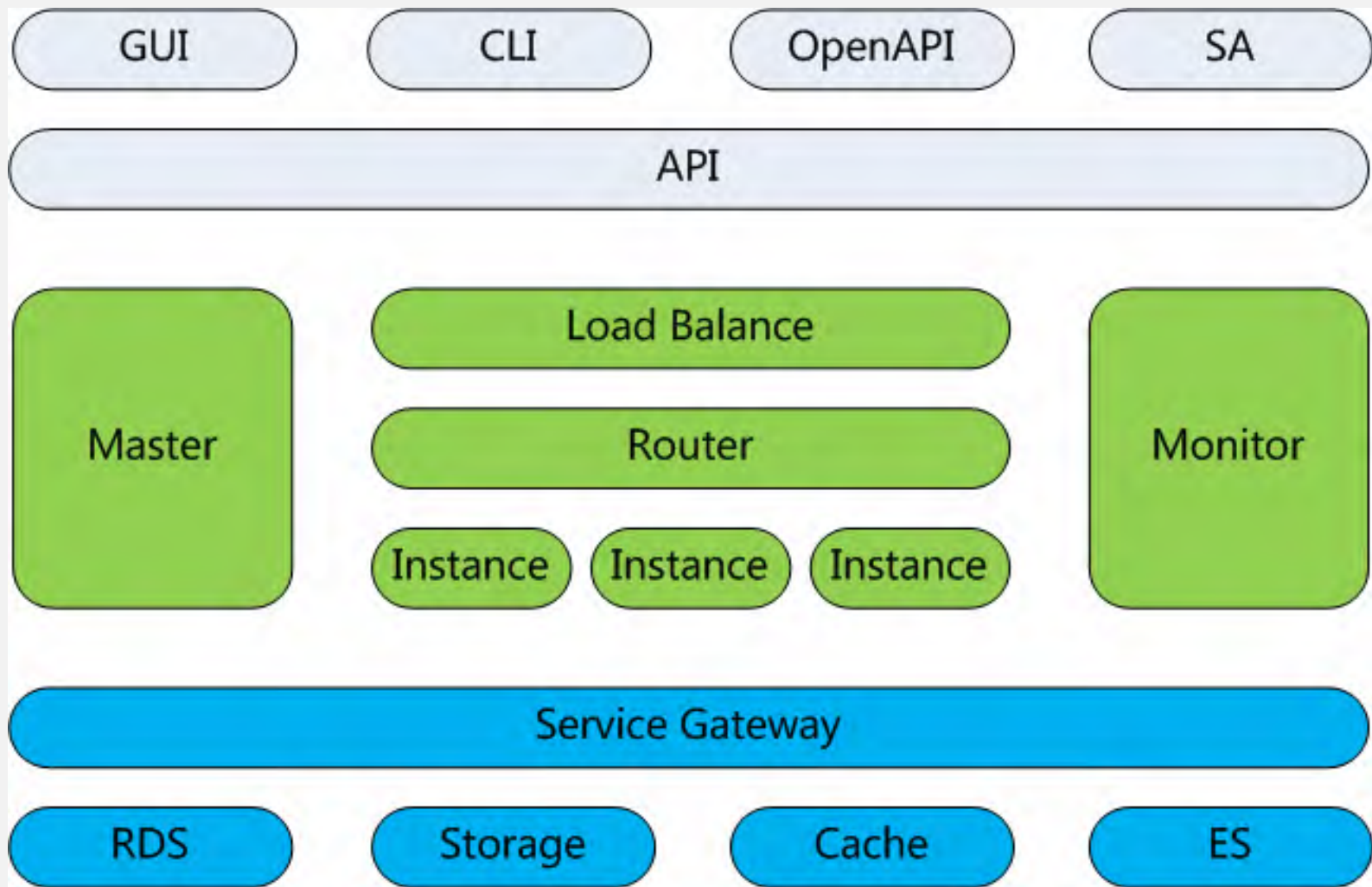
降低成本

提升资
源利用
率

资源成
本

运维成
本

三、PaaS平台架构



▶ **Agent**

1. 部署计算节点
2. 控制容器周期
3. 控制APP生命周期
4. 采集性能数据
5. 交互命令（消息机制）

▶ **Container**

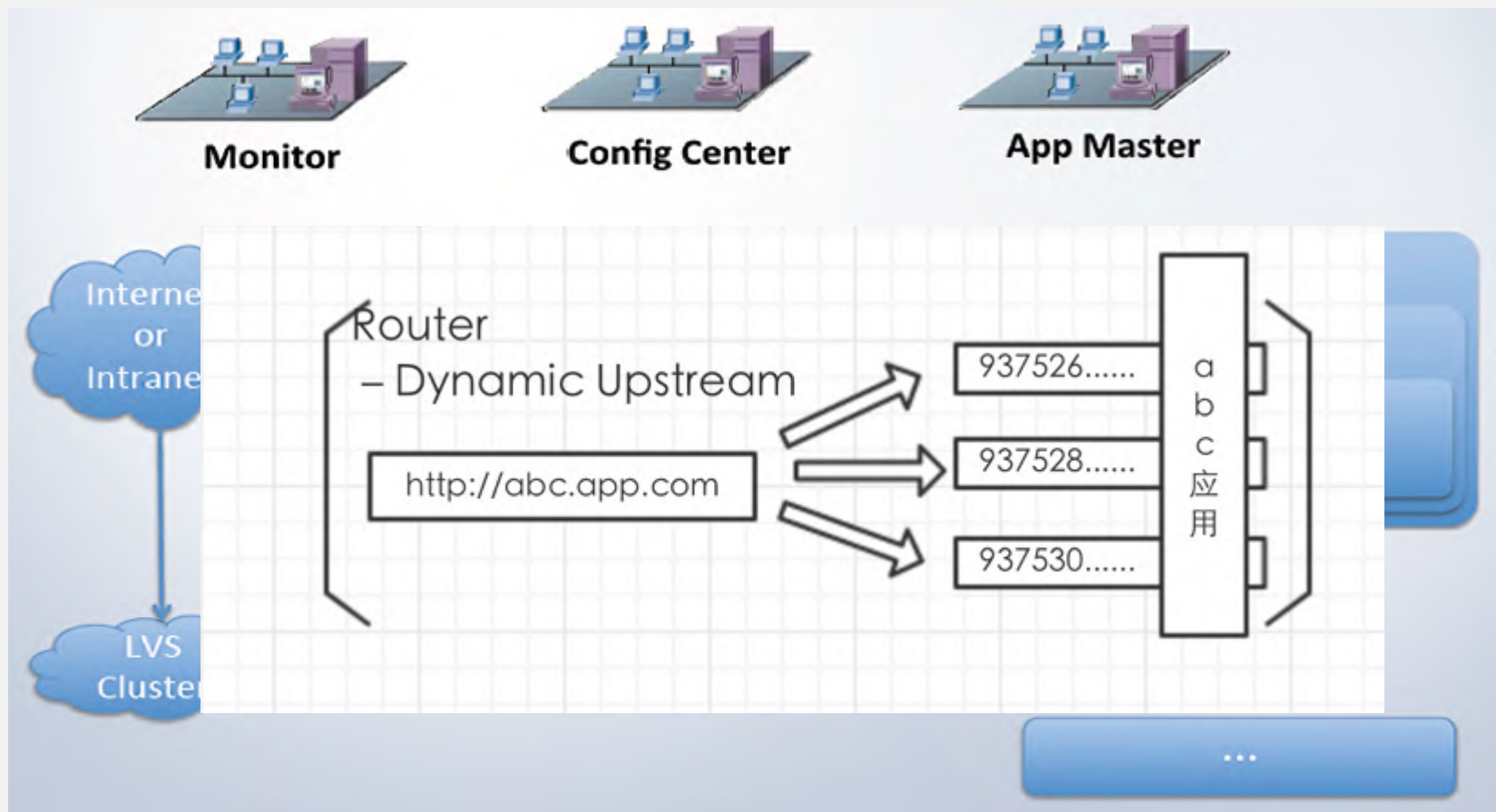
- Lxc
- Docker

▶ **App**

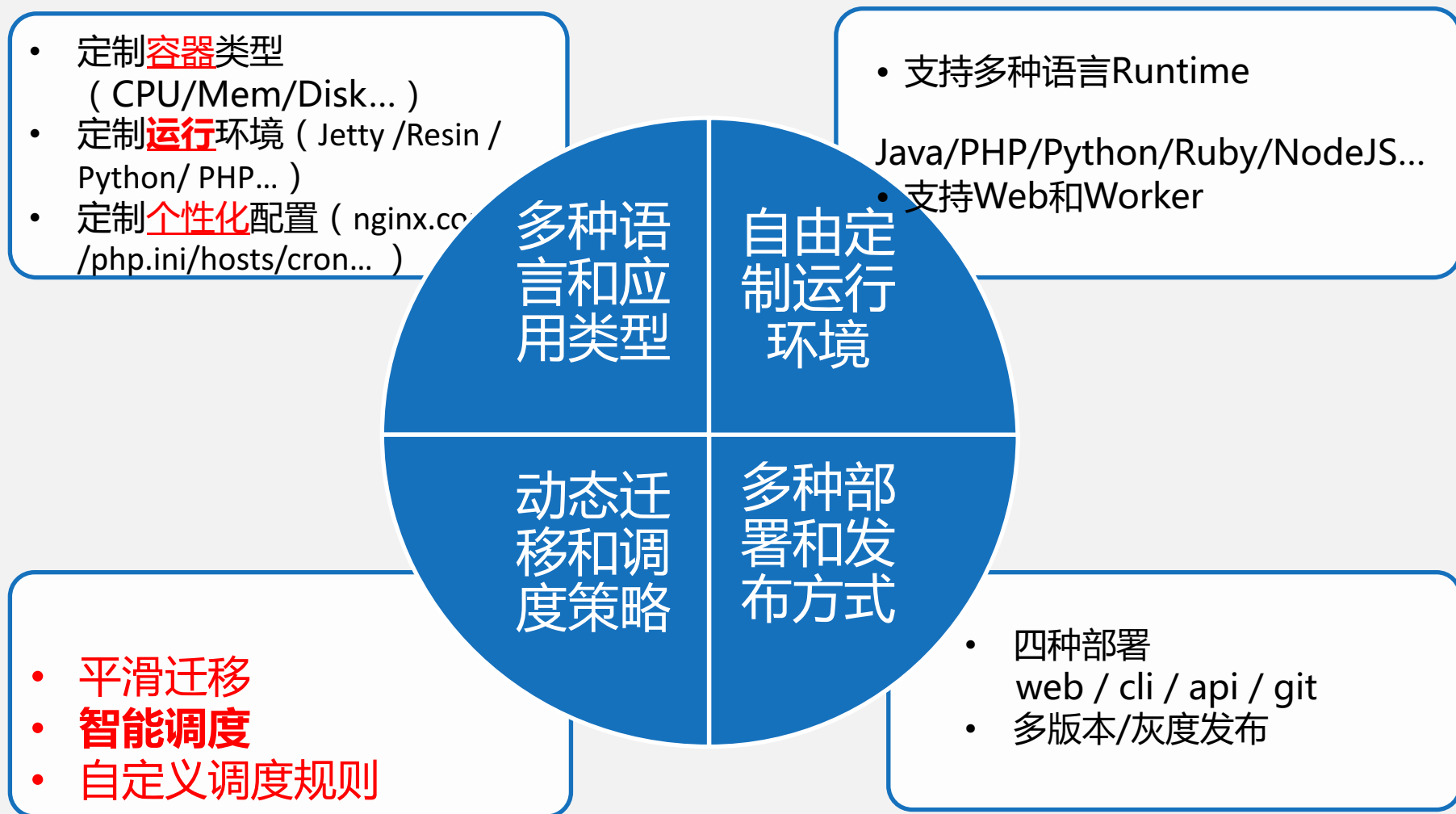
- 支持多种语言Runtime
- 自动化安装软件栈



二、PaaS平台架构 - 应用访问全链路解析



二、PaaS平台架构 - App Engine





三、智能调度

➤ 负载调度

采集容器负载，动态迁移实例

➤ 性能调度

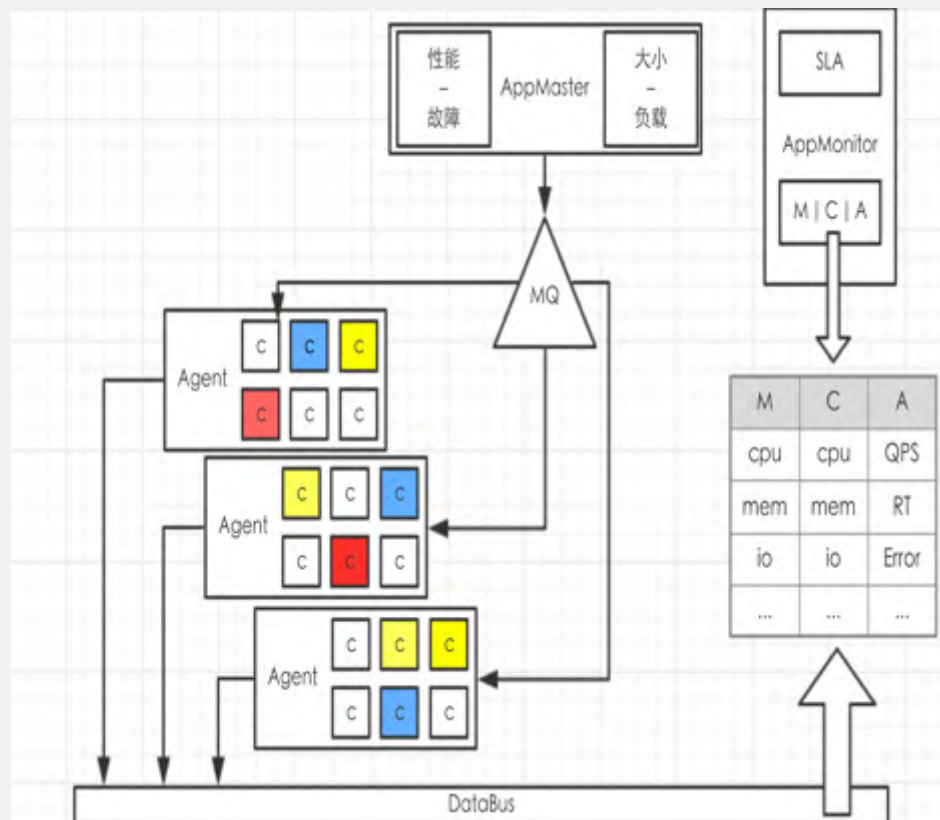
采集访问请求，动态调度实例

➤ 故障调度

健康检查状态，动态迁移实例

➤ 最大最小调度

分时压力预估，实时缩扩容实例





四、服务治理 - 微服务架构范式解析

➤ 微服务架构范式

聚合、代理、链式、分支、异步

➤ 服务间通信方式

Rest&Rpc、同步&异步

➤ 服务依赖复杂性

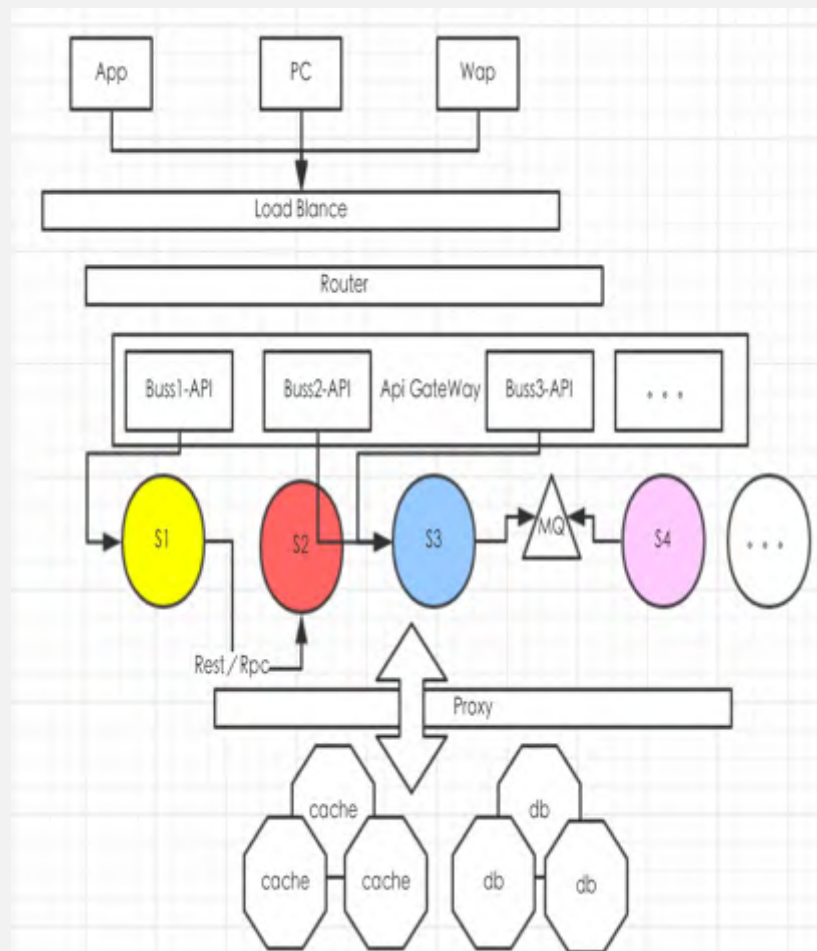
URL管理

循环依赖

服务扩容

访问授权

等等





➤ 等待超时

主动轮询式： $o(\log N) + o(1)$

维护一个有序队列(根据距离过期时间最近做升序排序)，监控线程间隔固定频取出头节点，进行关闭处理。

阻塞通知式： 插入 $o(\log N)$ + 调度 $o(1)$

维护一个二叉树(根据距离过期时间最近做升序排序)，监控阻塞于二叉树队列，获取头节点，通过signal方式唤醒。（数据结构，选择DelayQueue）

➤ 运行超时

存在于同步情况下，一般设置timeout时间来控制。

四、服务治理 - 限流

限流是预防模式，本质上解决不了依赖问题，通常对各种类型请求设置TPS / QPS 阈值

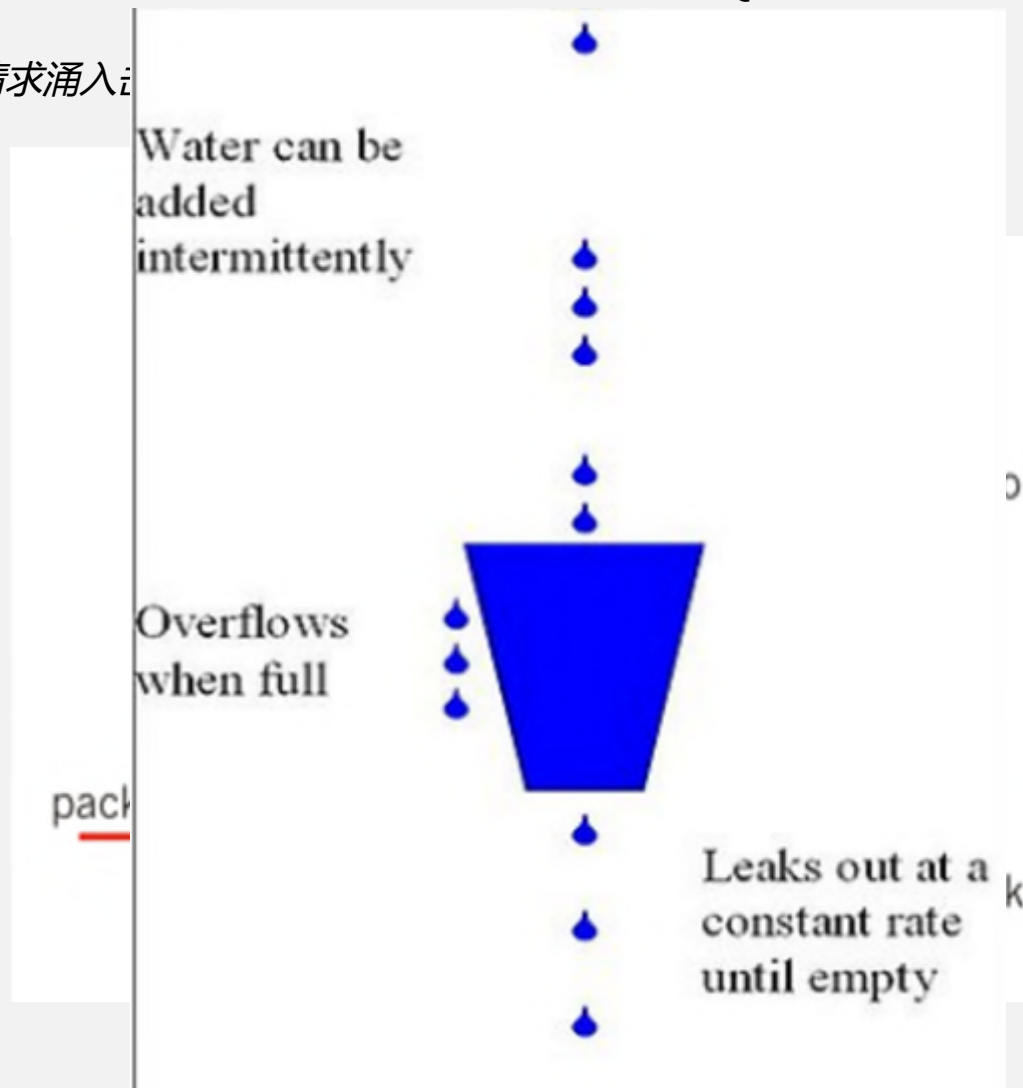
A. 大于阈值，则进行过载保护，防止大量请求涌入；

B. 小于阈值，继续调用服务；

思路1：计数器

思路2：漏桶

思路3：令牌桶



漏桶与令牌桶

相同：两者在运行时，可以控制与调整数据处理速率。

不同：前者，以固定流出速率处理数据；后者，利用令牌，动态调整流出速率。

例子：

1. Guava是一个Google开源项目，其中的RateLimiter提供了令牌桶算法实现：平滑突发限流(SmoothBursty)和平滑预热限流(SmoothWarmingUp)实现。
2. Nginx接入层，利用模块来限制流量，两个：
 - 连接数限流模块ngx_http_limit_conn_module
 - 请求限流模块ngx_http_limit_req_module (漏桶算法)



四、服务治理 - 熔断与降级

熔断，是容错处理，对于目标服务调用慢或有大量超时，可以断掉该服务，直接返回；若调用好转后，可以恢复调用目标服务。

降级，是容错处理，系统对服务级别按照业务来划分，形成N级，互不影响。当某个服务的线程资源耗尽，就直接返回预设响应内容，不再调用后续资源，保证重要服务的可用性。

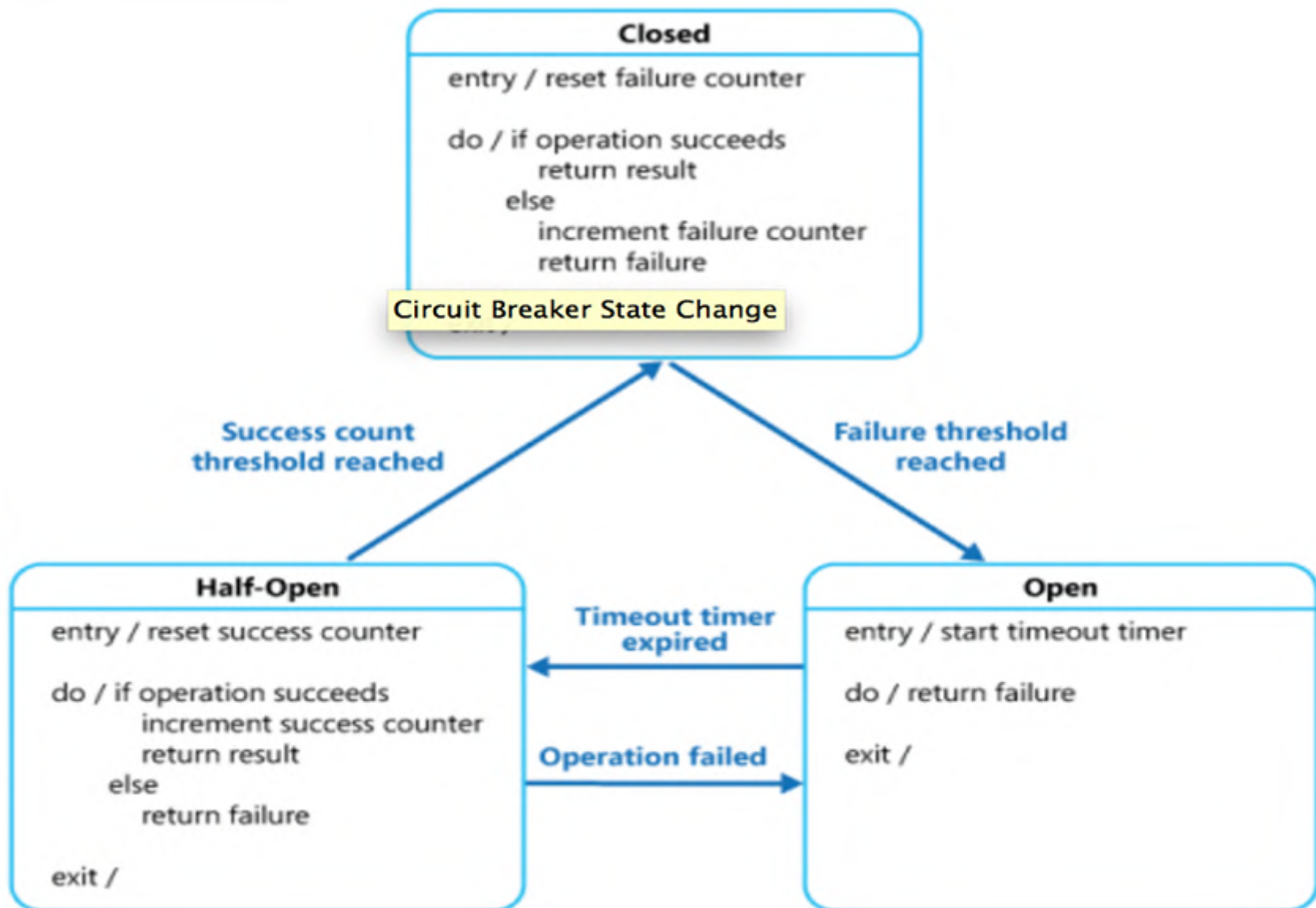
最重要区别：

1.触发原因

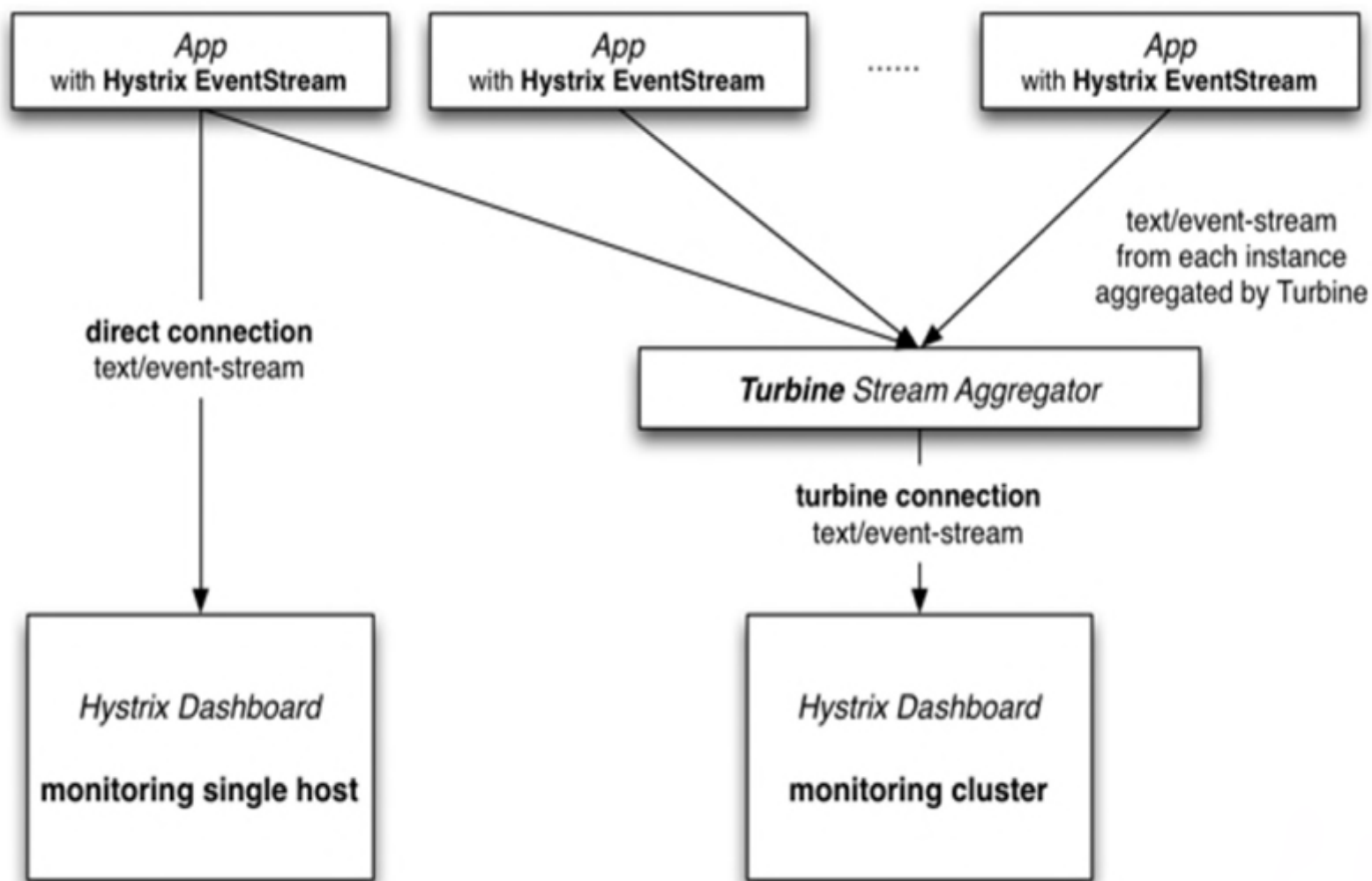
熔断一般在某个服务（下游服务）故障时引起，而降级是对系统整体性能考虑的；

2.层次不同

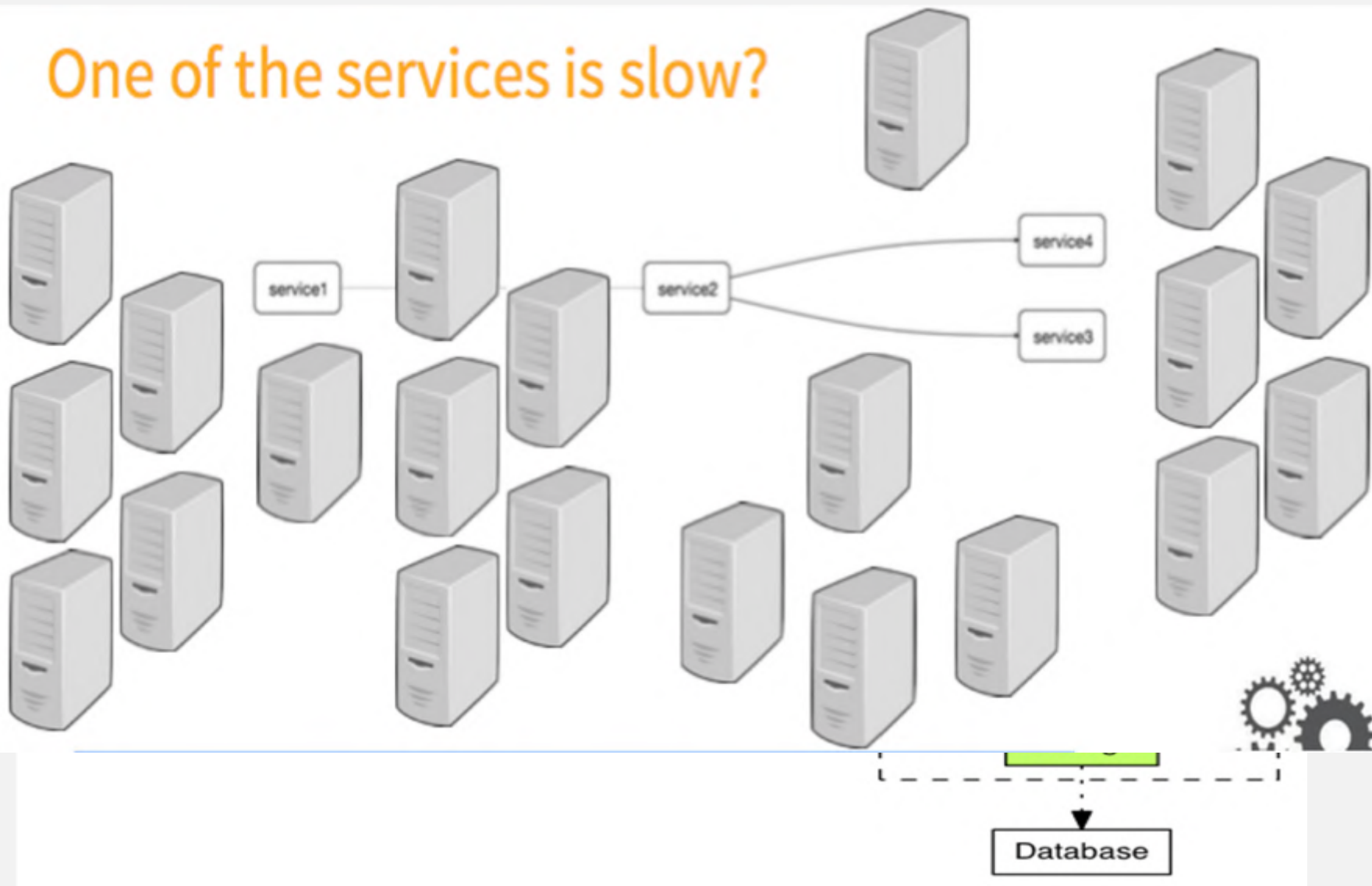
熔断是一个框架级的处理，而降级是针对业务层级而言（一般从外围服务开始）



四、服务治理 - 熔断与降级



One of the services is slow?



4. 延时指标

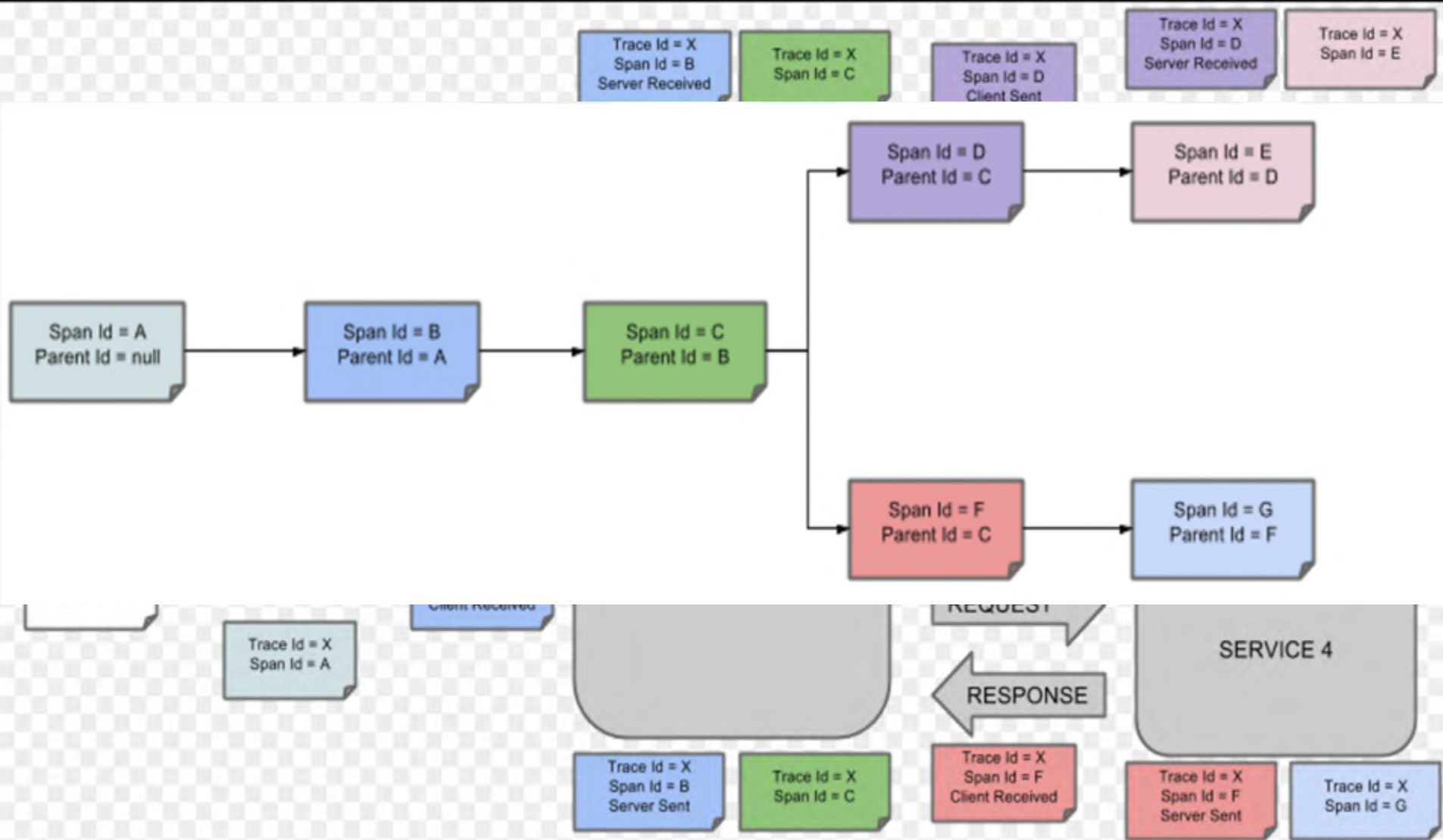


SR-CS = 请求发出延迟时间

SS-SR = 服务端处理延迟时间

CR-CS = 整个链路完成延迟时间

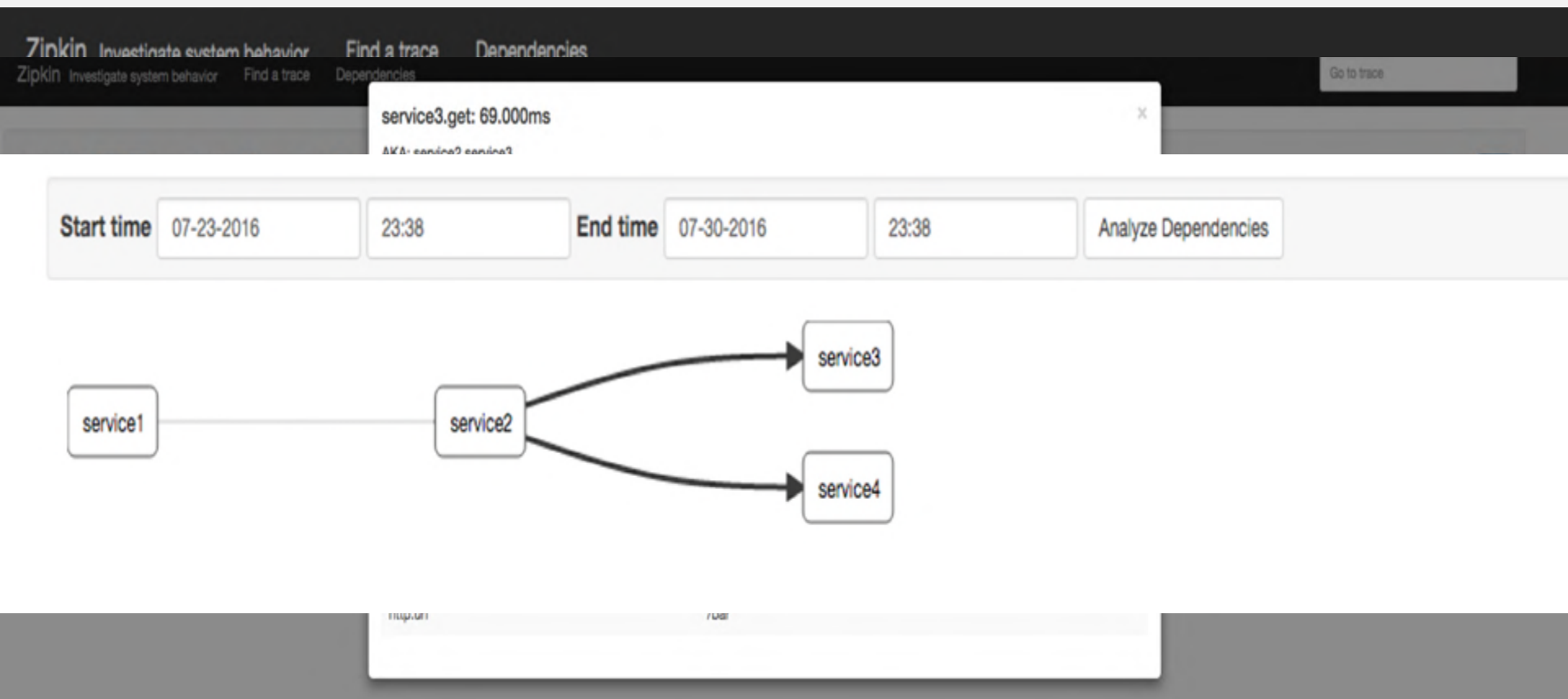
5. Trace数据模型



6. 链路可视化

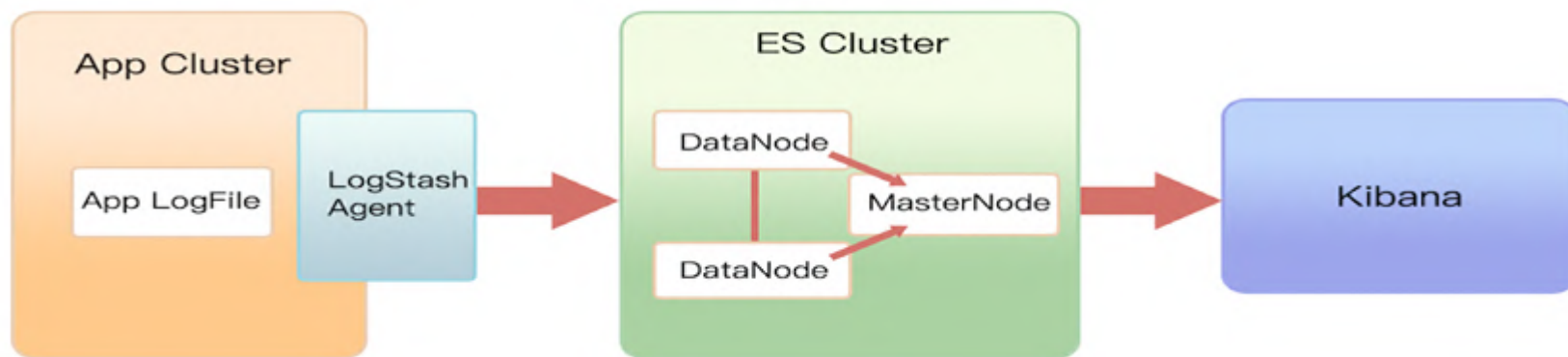
访问<http://dev.auto.ifeng.com/zipkin/start> 后查看zipkin的web UI

a. 依赖视图 b. 节点视图 c. 调用视图



五、日志收集方案 - elk方案迭代

ELK Stack 1



➤ 优点

搭建简单、易于上手。

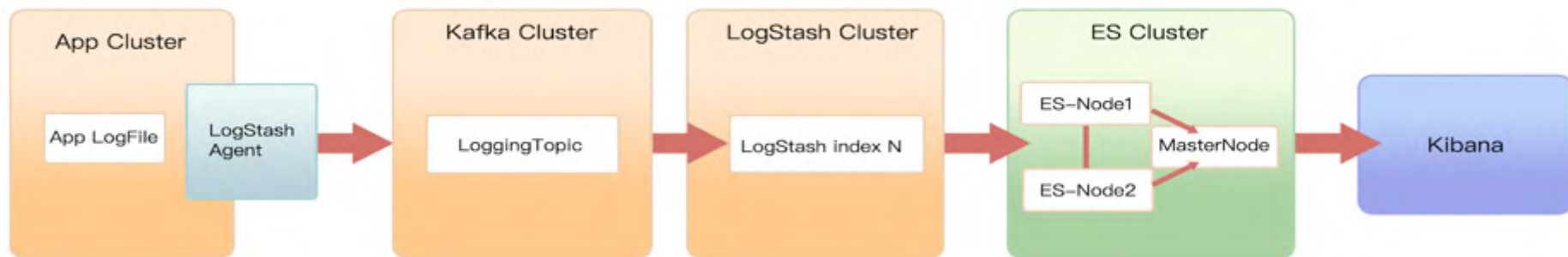
➤ 缺点

Logstash耗资源较大，运行占用CPU和内存高。

没有消息队列缓存，存在数据丢失隐患。

五、日志收集方案 - elk方案迭代

ELK Stack 2



➤ 优点

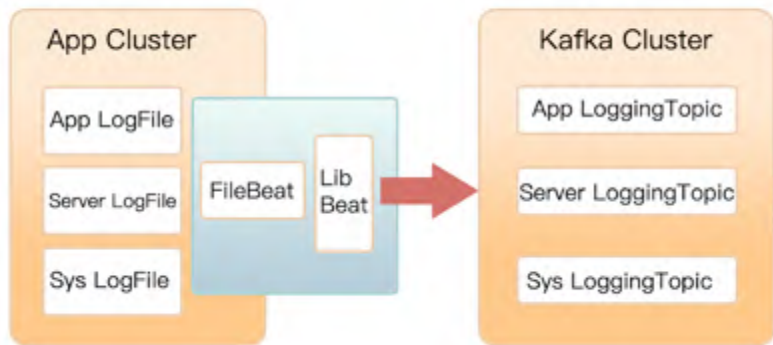
引入kafka消息队列机制，均衡了网络传输，从而降低了网络闭塞，尤其是丢失数据的可能性。

➤ 缺点

存在Logstash占用系统资源过多的问题。

五、日志收集方案 - elk方案

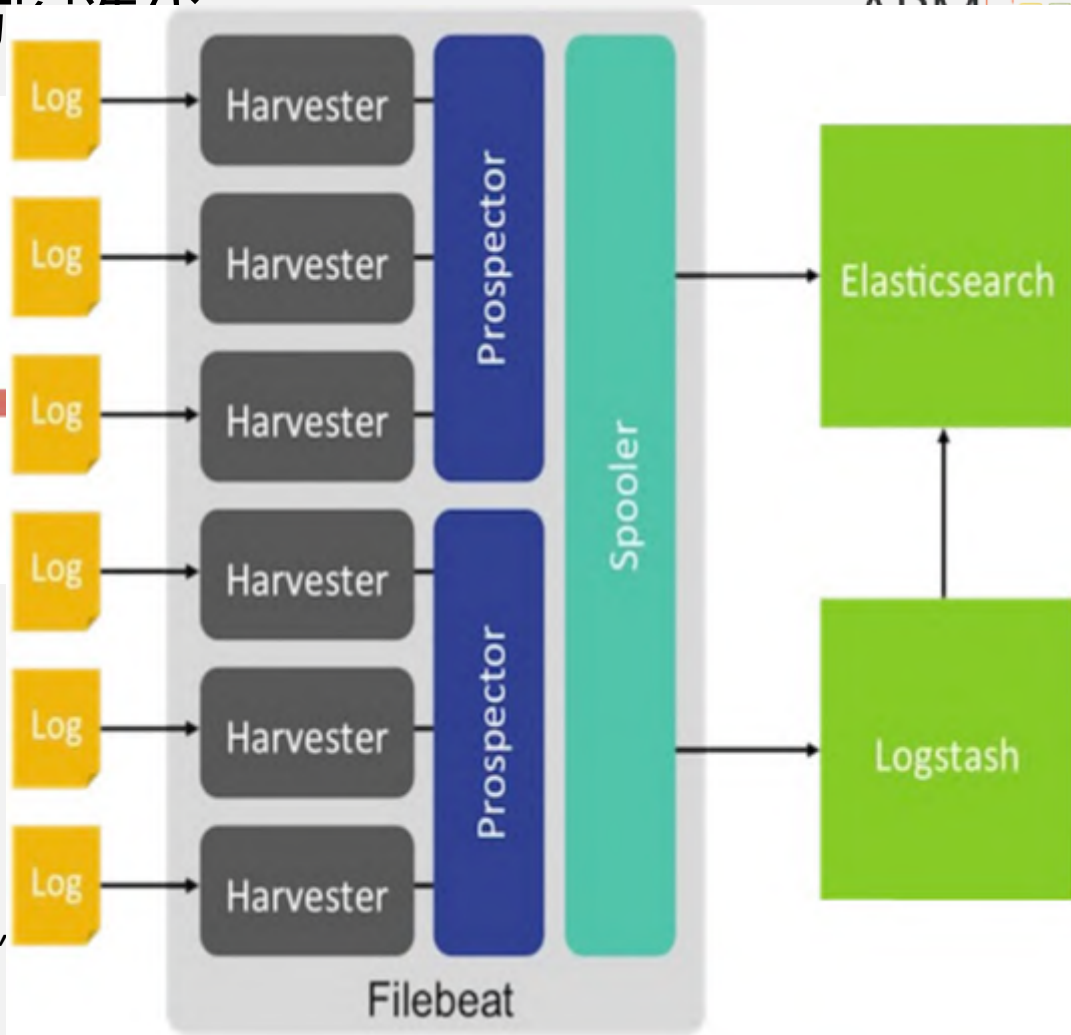
ELK Stack 3



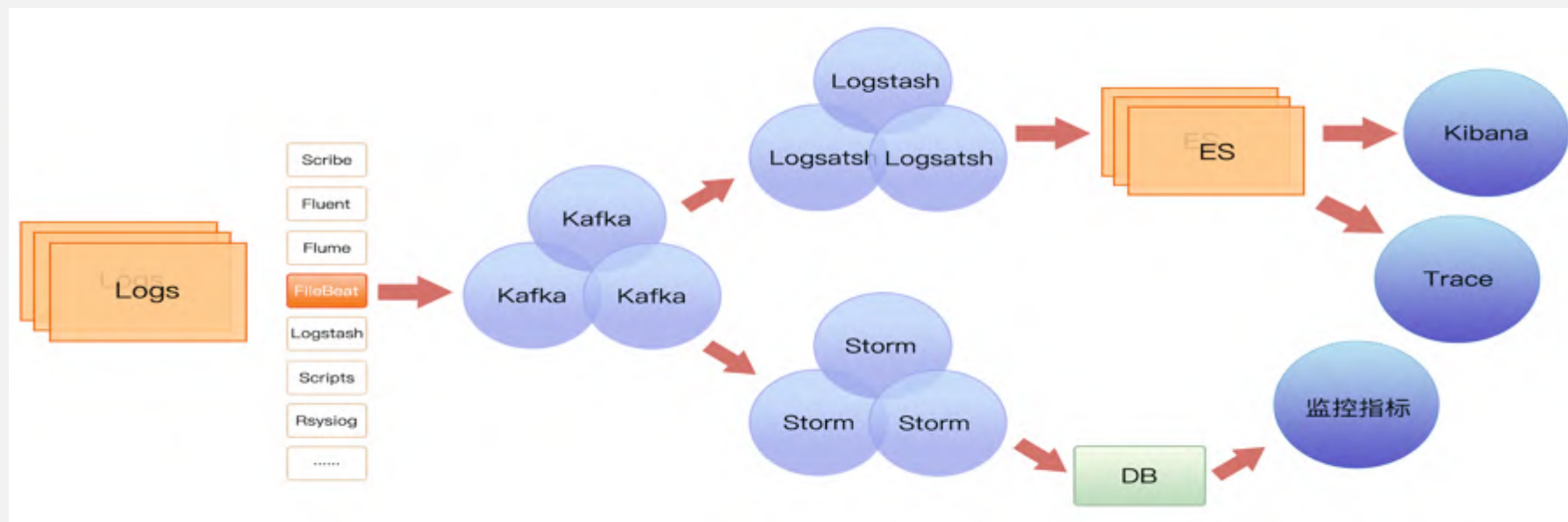
➤ FileBeat架构解析

filebeat是用来收集日志的，那么在filebeat.yml中会配置指定的监听文件，每个log的目录是在prospectors中设置，对于prospectors定位每个日志文件，

Filebeat启动harvester。每个harvester读取新的内容一个日志文件，新的日志数据发送到spooler（后台处理程序），它汇集的事件和聚合数据发送到你已经配置了Filebeat输出。



五、日志收集方案 - elk方案迭代



- 扩容容易，多份备份

- 检索困难。

➤ 范式统一阶段（ELK）

- 日志丢失，引入kafka隔离。

- 应用服务器性能问题，引入Filebeat。

- 模型复杂，统一三个流程（服务查询&检索 / 计算（实时 / 离线） / 追踪）。

1.XaaS

“一切皆服务”，出现IaaS、PaaS、SaaS、DaaS、BaaS等等。

2.Serverless

“无服务器”云计算技术将消除应用开发的复杂性并降低成本，更加专注业务。

主要包括两种类型：

BaaS - 旨在为移动和Web应用提供后端云服务，实现对逻辑和状态进行管理，包括云端数据/文件存储（例如Parse、Firebase）、消息推送（例如极光推送、个推）、应用数据分析等等。

FaaS - 部分服务逻辑由应用实现，运行于无状态的容器中，可以由事件触发，短暂的，完全被第三方管理，不需要关心后台服务器或者应用服务，只需关心自己的代码即可。

经历所有不可能成就可能！

THANK YOU

