

网易云 Kubernetes 性能优化实践

赖冬林, 2017-12-22



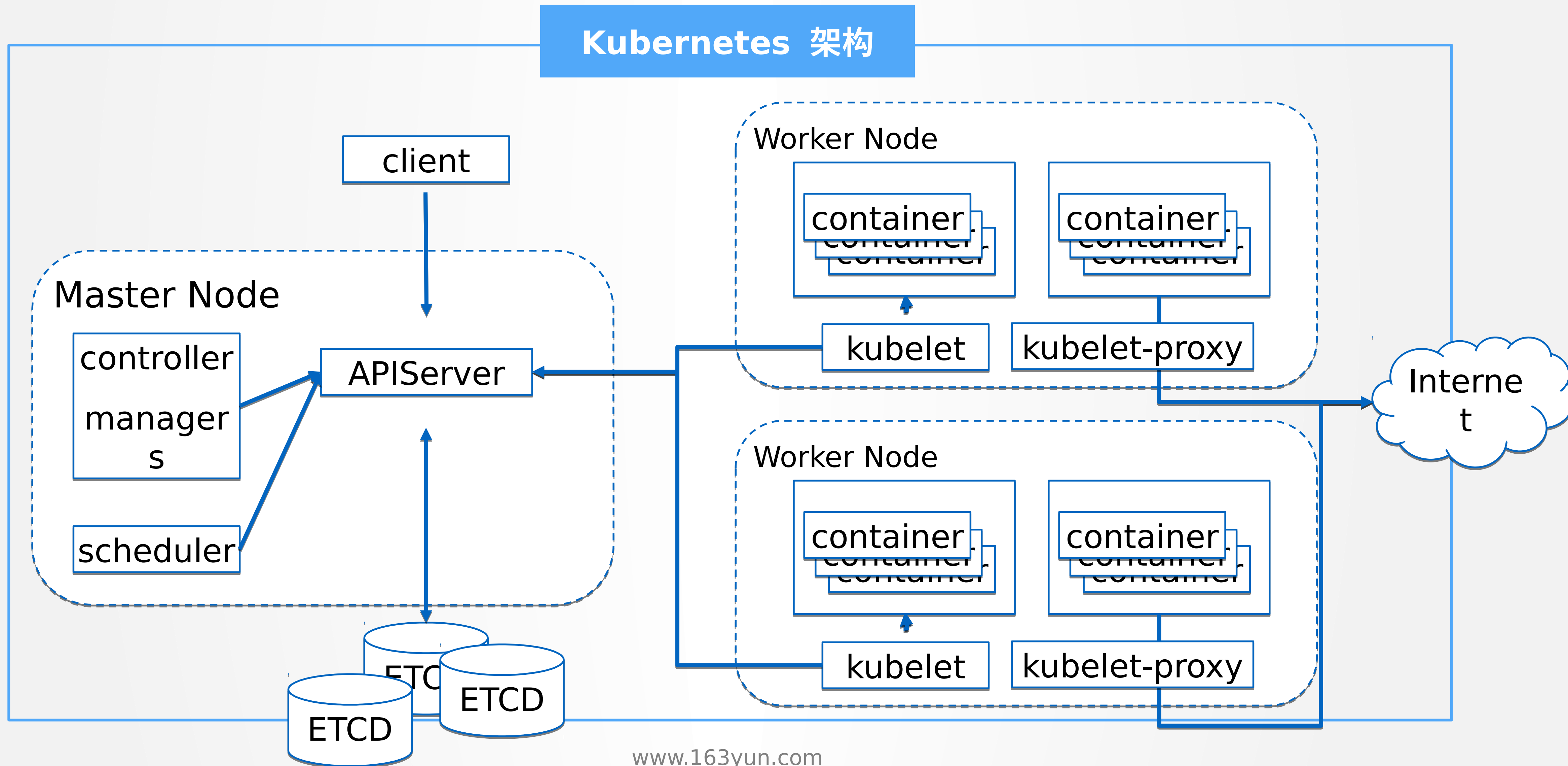
不用很累很麻烦的性能优化实践

赖冬林，2017-12-22

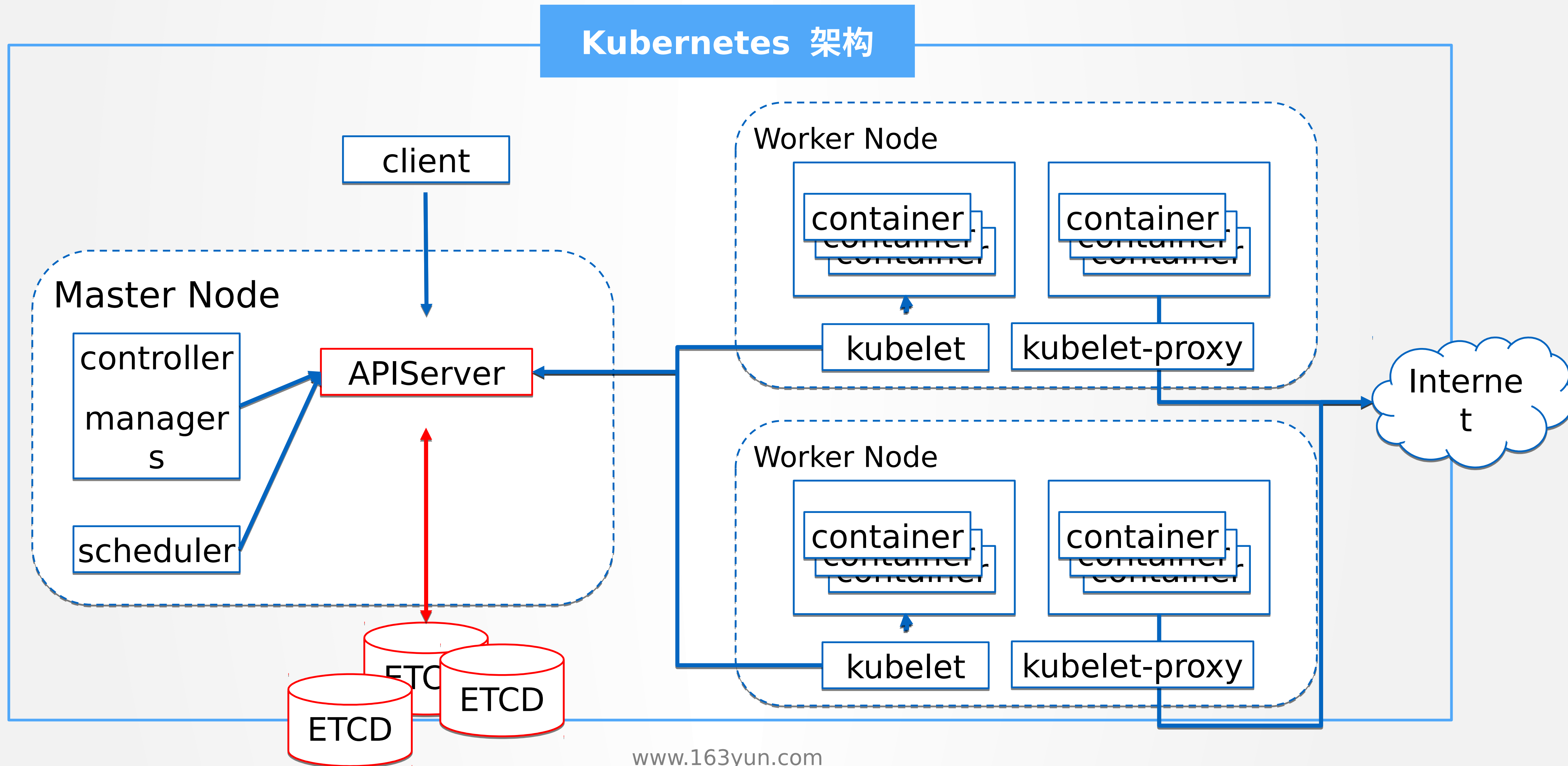


为什么要优化 Kubernetes?

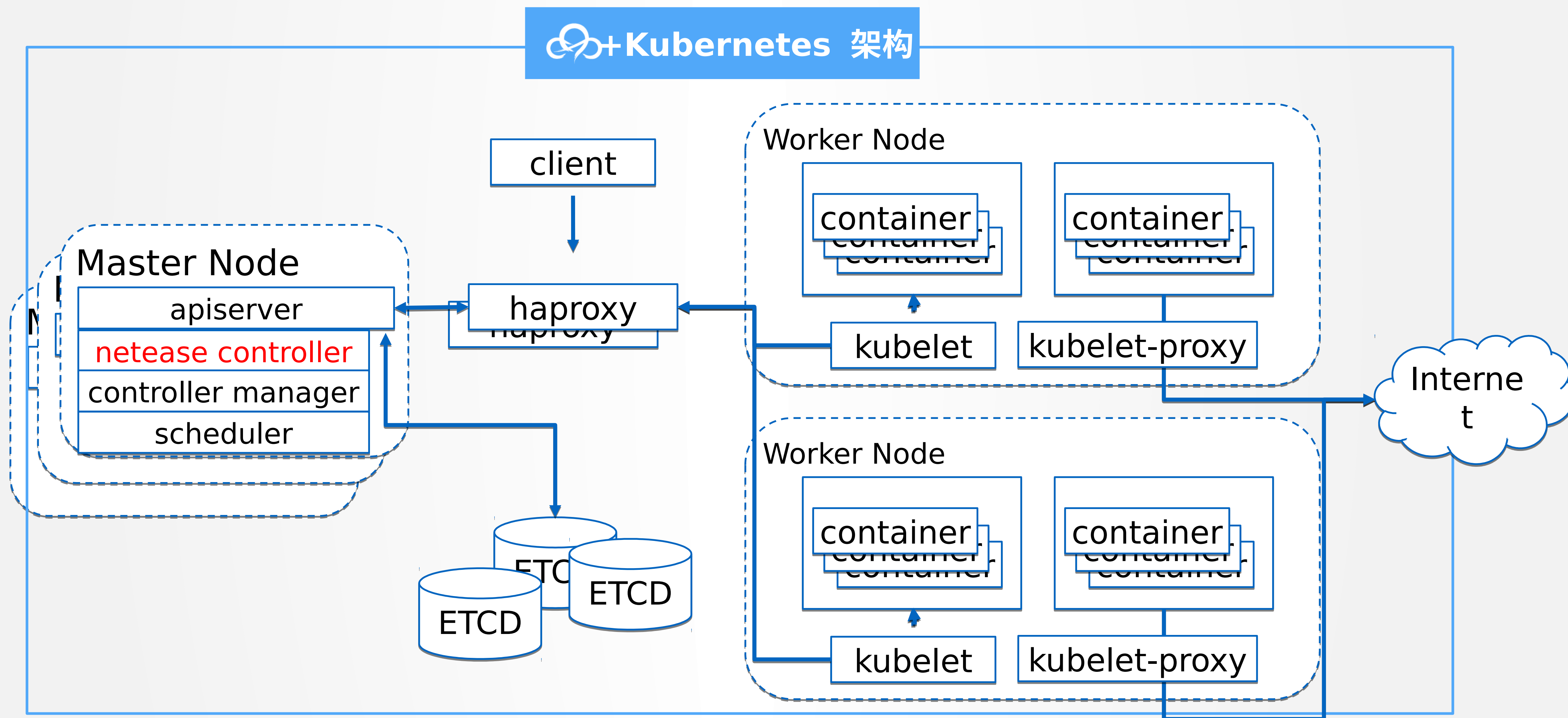
为什么要优化 Kubernetes ?



为什么要优化 Kubernetes ?



为什么要优化 Kubernetes ?



为什么要优化 Kubernetes ?

业务压力

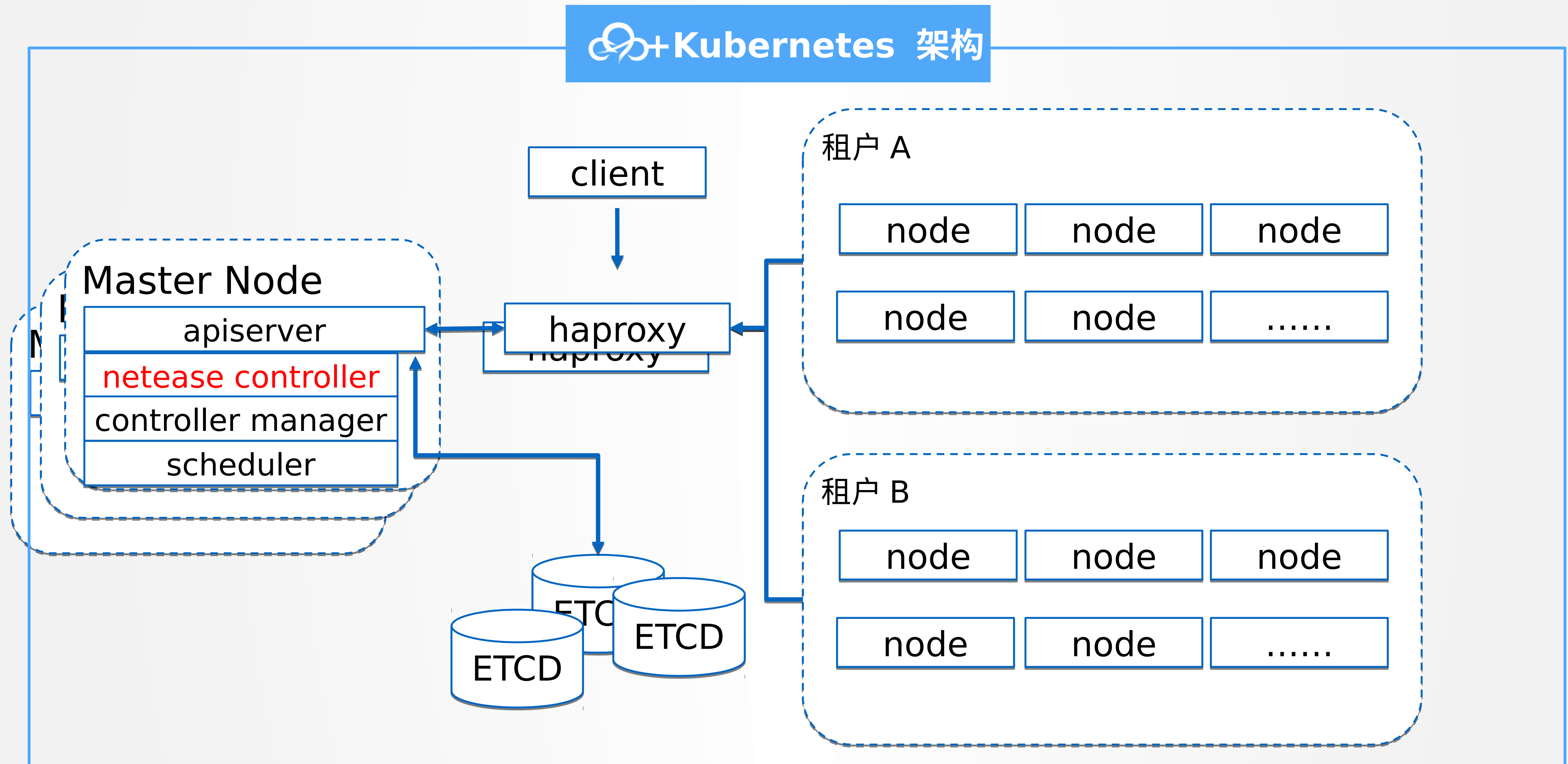
- 服务于网易互联网产品
- 云音乐
- 考拉
- 网易新闻
- 云阅读
- 云课堂
-

产品需求

- 面向公有云
- 多租户
- 专属云
-

最重要的是：节省成本！为老板省钱！

为什么要优化 Kubernetes ?



为什么要优化 Kubernetes ?

原生版本性能

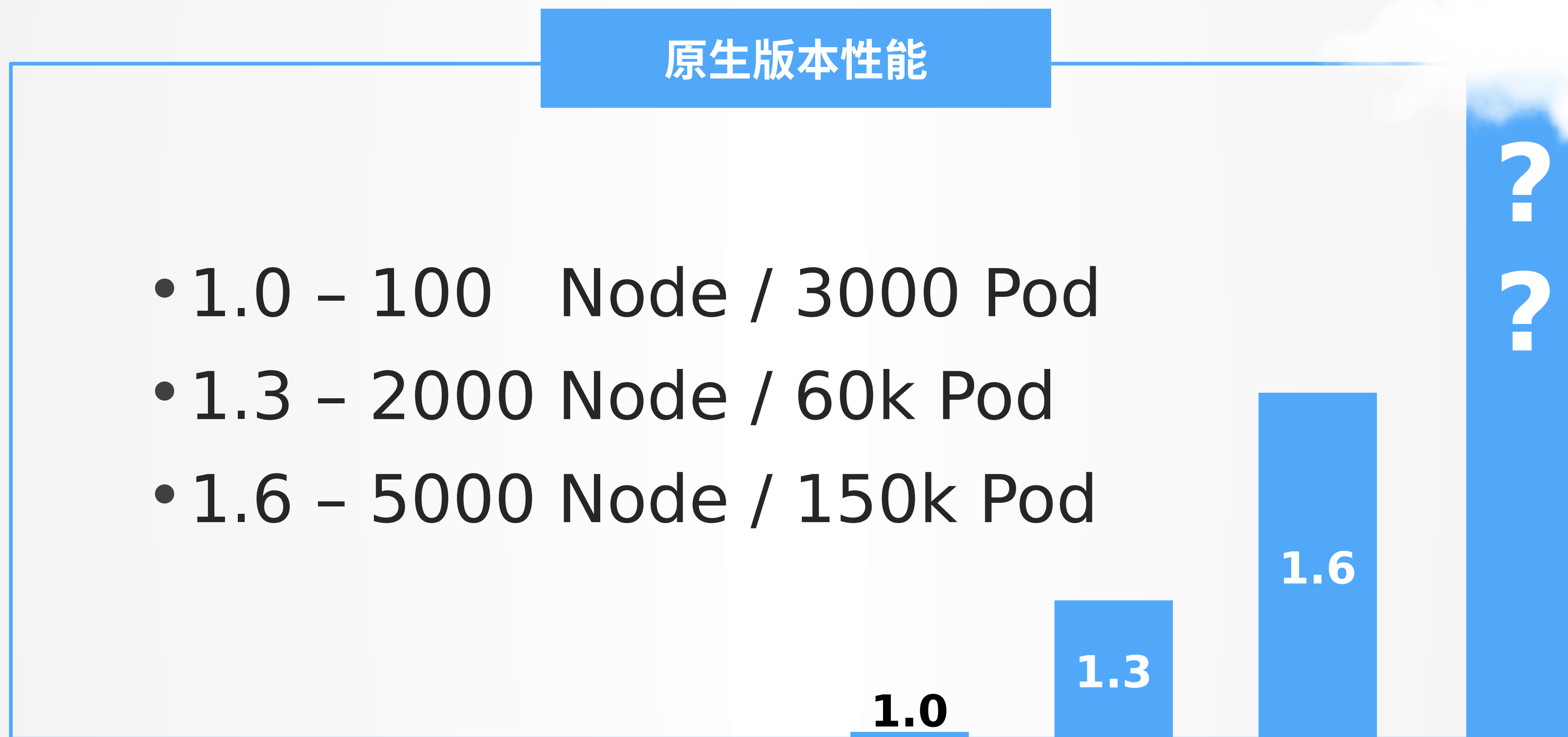
- 1.0 - 100 Node / 3000 Pod
- 1.3 - 2000 Node / 60k Pod
- 1.6 - 5000 Node / 150k Pod

1.0

1.3

1.6

为什么要优化 Kubernetes ?



都远达不到我们的目标是：单集群 **30k Node**

为什么要优化 Kubernetes ?

环境与性能标准

- Master 机器：物理机 E5-2650v3 X 2/256G RAM/10G NIC/SSD
- 通过标准
 - 排除 WATCH ，所有 API 响应时间的 99 值最大为 100ms
 - pod 创建成功和 container 成功运行耗时的 99 值在 5s 内
 - deployment 创建时间耗时的 99 值在 15s 内
 - 所有 P0 场景错误率 0%
 - apiserver 进程的 cpu 使用率少于 70%（针对 40 核机器而言， <2800%）
 - 运维操作恢复时间： Master<3min ， 批量 kubelet 运维 < 10min

■ 如何优化 Kubernetes?

如何优化 Kubernetes ?

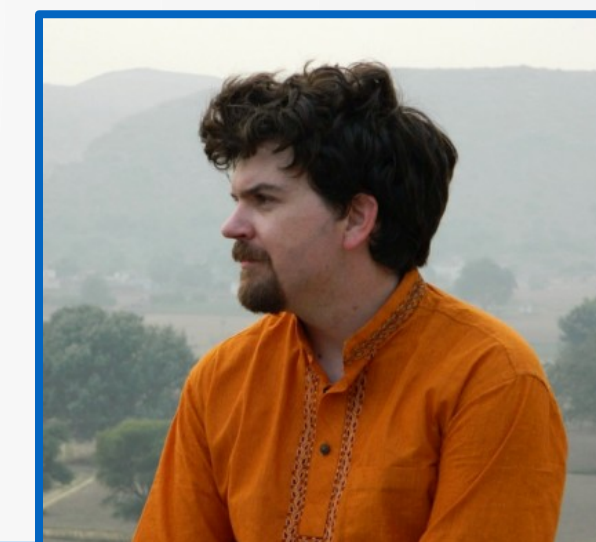
基本原则

- 方法论 USE
- 指标收集
- 性能剖析
- 优化措施

如何优化 Kubernetes ?

方法论 USE

- Utility : CPU / 内存 / 磁盘 / 网络利用率
- Saturation : 各种队列长度 / 饱和度
- Error : 协议栈错误、应用层错误日志



By Brendan
Gregg

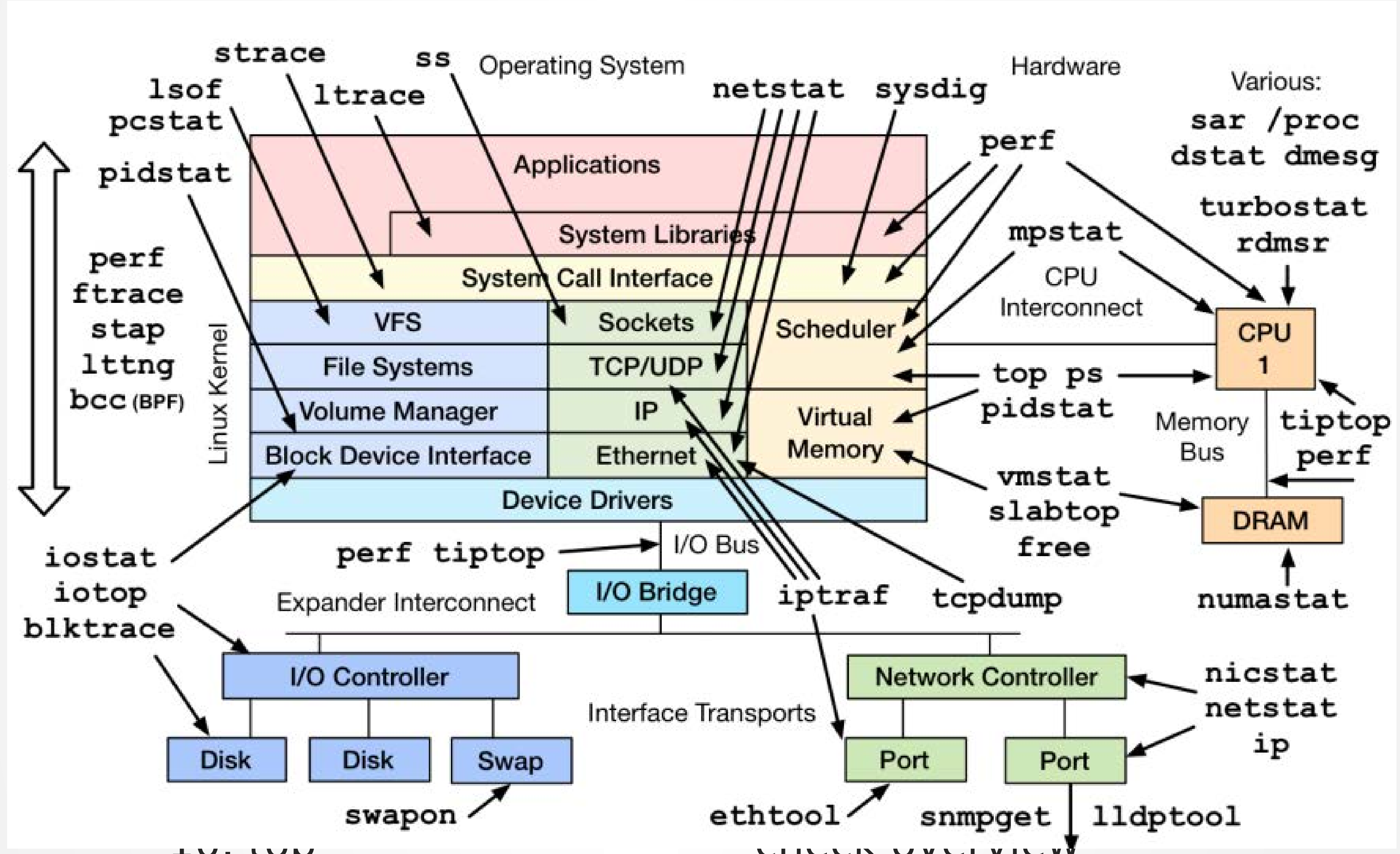
<http://www.brendangregg.com/usemethod.html>

如何优化 Kubernetes ?

指标收集

- 系统指标: top/sysstat 包等 (debian)
- 应用指标: prometheus + grafana

如何优化 Kubernetes ?



如何优化 Kubernetes ?

Perf Analysis in 60s

- 1. uptime
- 2. dmesg | tail
- 3. vmstat 1
- 4. mpstat -P ALL 1
- 5. pidstat 1
- 6. iostat -xz 1
- 7. free -m
- 8. sar -n DEV 1
- 9. sar -n TCP,ETCP 1
- 10. top
- load averages
- kernel errors
- overall stats by time
- CPU balance
- process usage
- disk I/O
- memory usage
- network I/O
- TCP stats
- check overview

如何优化 Kubernetes ?

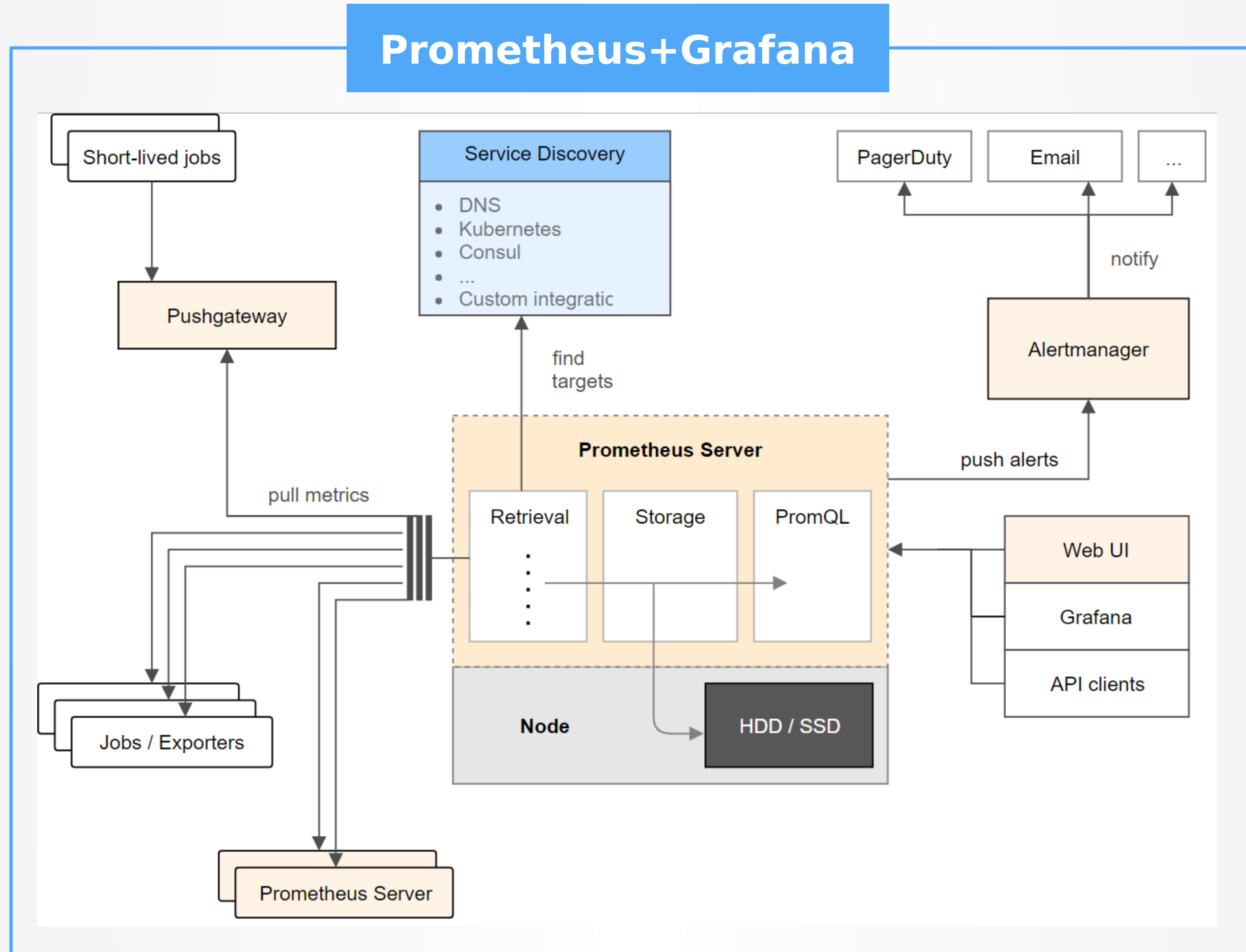
举个栗子

- netstat/sar 网络监控

```
TcpExt:
  867 SYN cookies sent
 1063 SYN cookies received
 2470 TCP sockets finished time wait in fast timer
  25 delayed acks sent
Quick ack mode was activated 150 times
36984 times the listen queue of a socket overflowed
36984 SYNs to LISTEN sockets dropped
 30 packets directly queued to recvmsg prequeue.
```

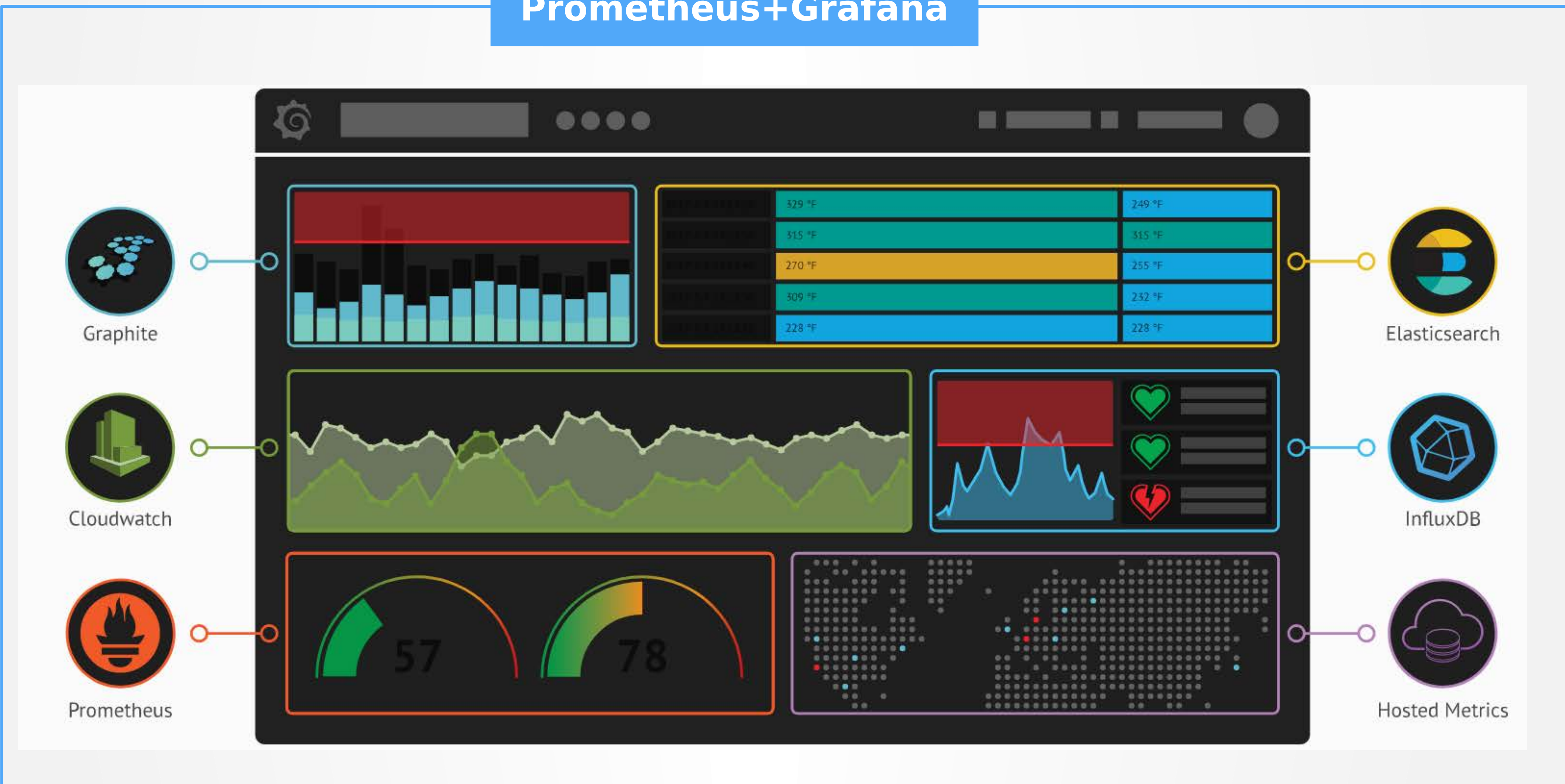
- 重启时发现异常全连接队列太小
- sar -n DEV,ETCP 1 通常会有很高的 retrans

如何优化 Kubernetes ?



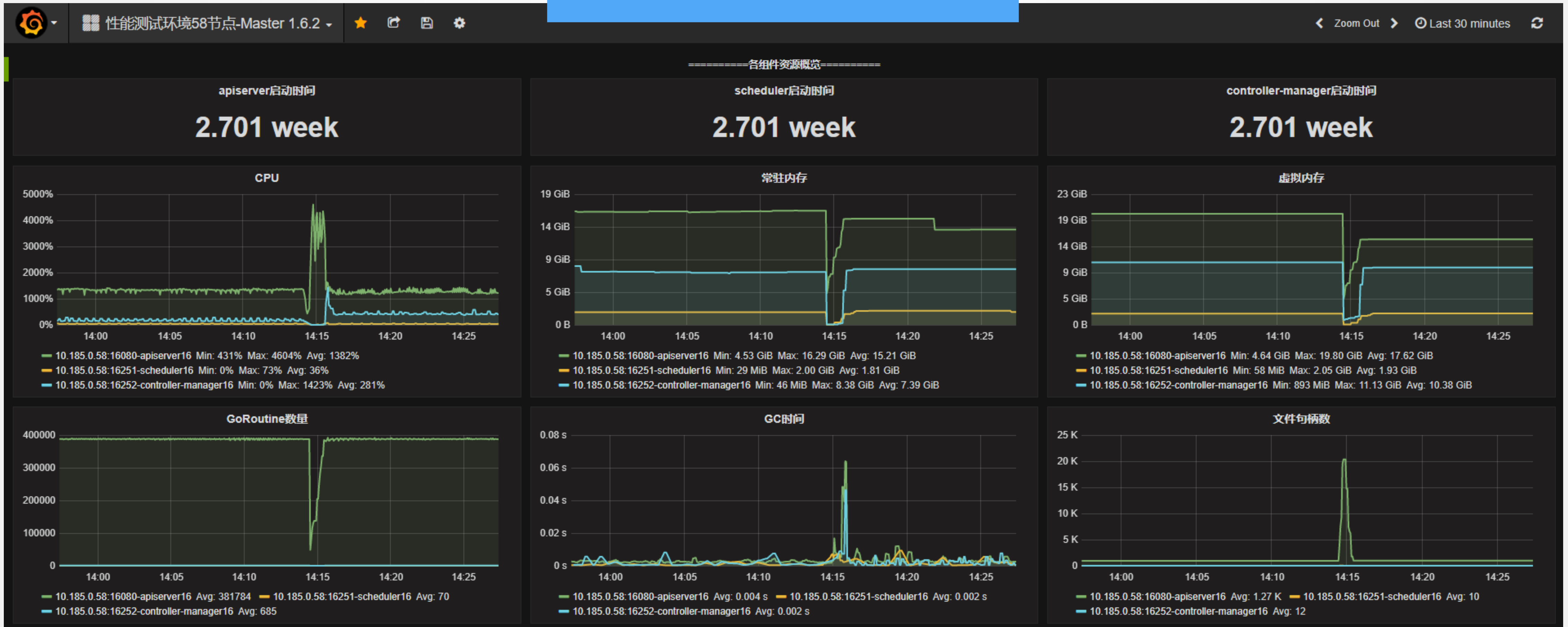
如何优化 Kubernetes ?

Prometheus+Grafana



如何优化 Kubernetes ?

Prometheus+Grafana



如何优化 Kubernetes ?

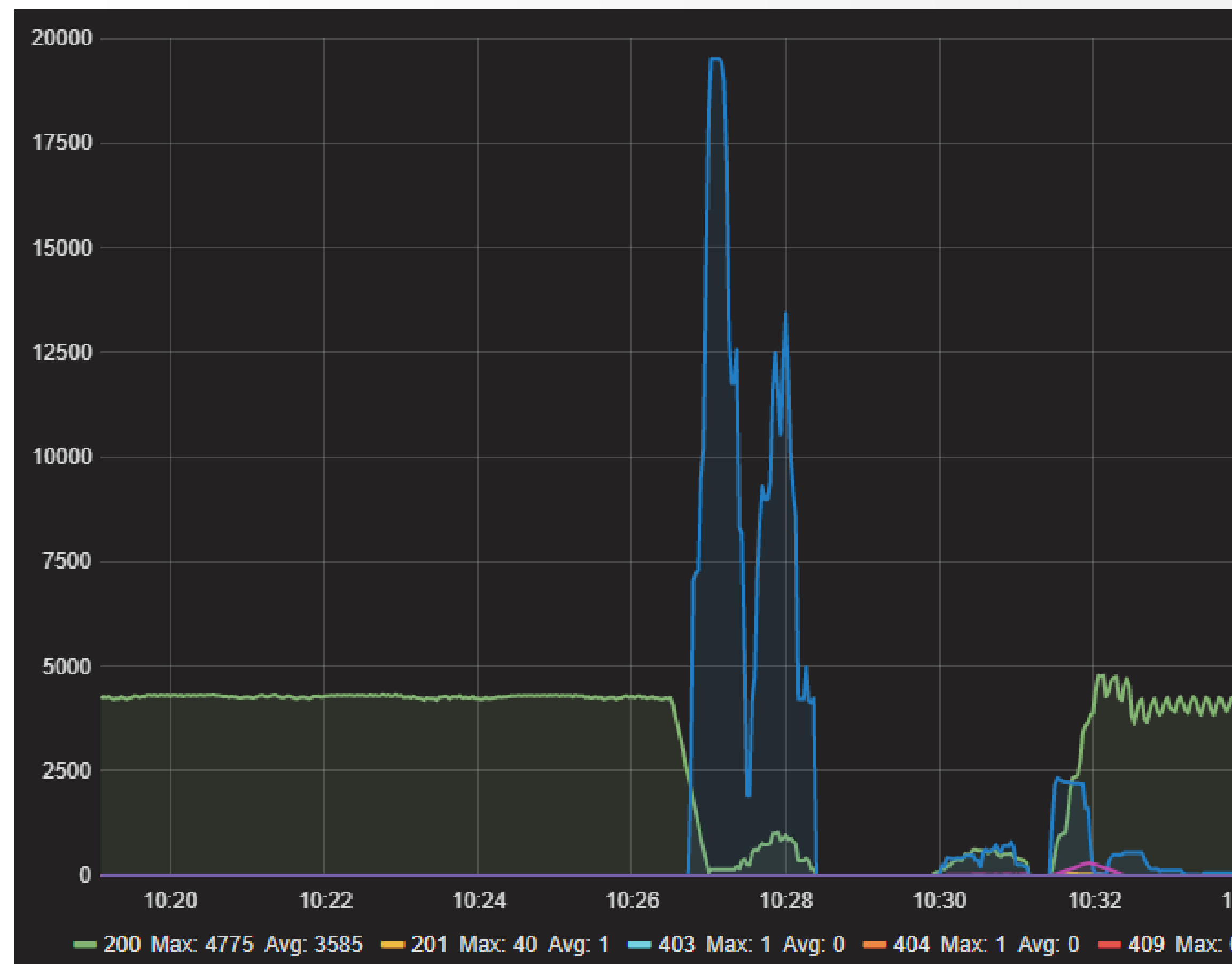
Prometheus+Grafana

- 增加多种 metrics
- apiserver 与 etcd 访问的延迟
(operation&resource)
- apiserver 流控比例
- Controller manager Sync 延迟统计 (resource)
- Controller manager 队列饱和度
- Scheduler 队列饱和度
-

如何优化 Kubernetes ?

apiserver 响应码统计

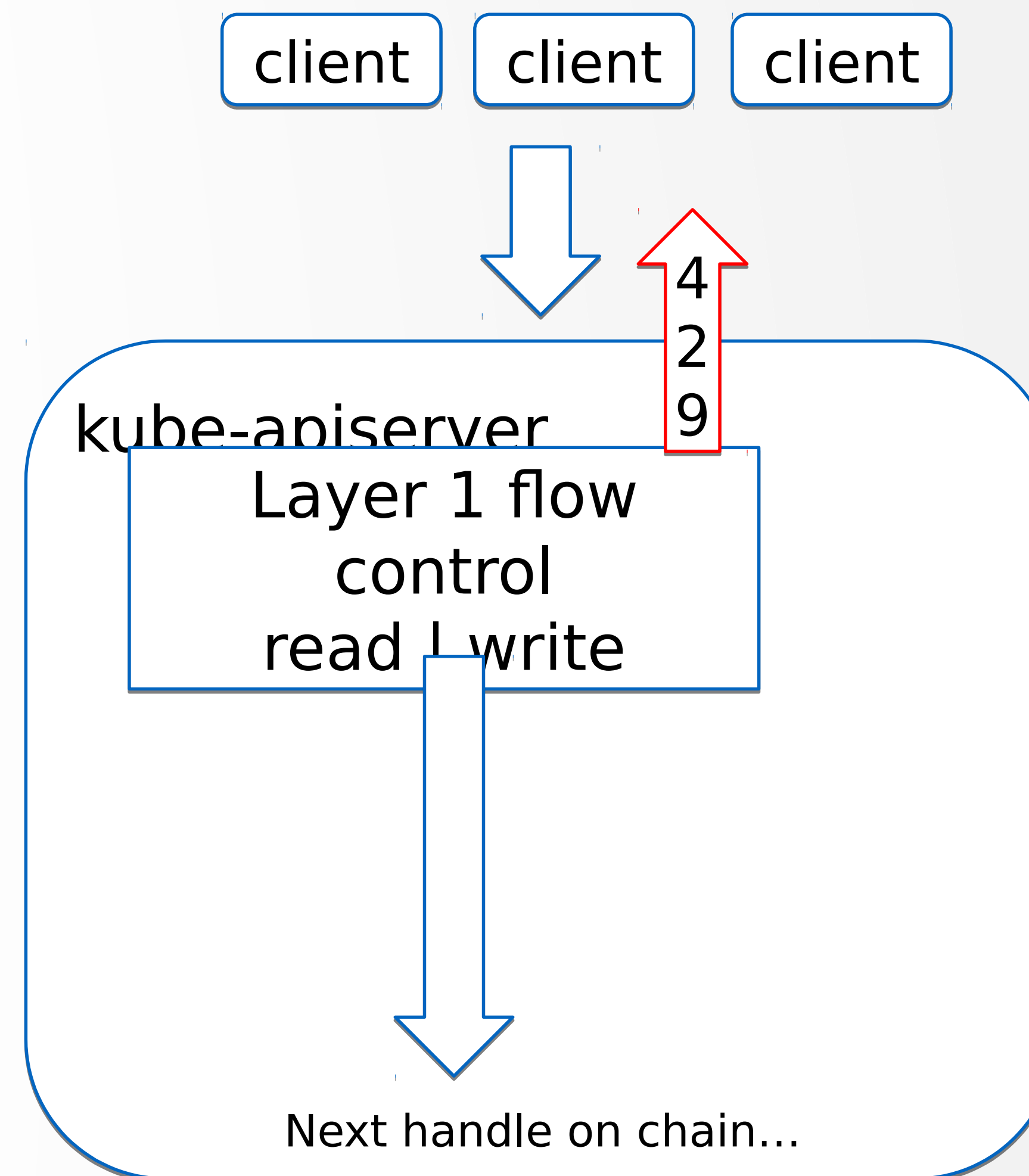
- 例如: APIServer 响应码统计
- 背景: Node - 2.5w / Pod - 15w
- 操作: 重启 APIServer
- 现象:
 - - 平时主要是 200(绿色), 4600 个每秒
 - - 重启 429(蓝色) 异常增高, 最多近 2w 个
 - - 有段时间没有 200 的响应
 - - 重启时间长达近 7 分钟
- 不符合我们的性能通过标准 (重启 / 升级 停机时间 < 3min)



如何优化 Kubernetes ?

流控优化

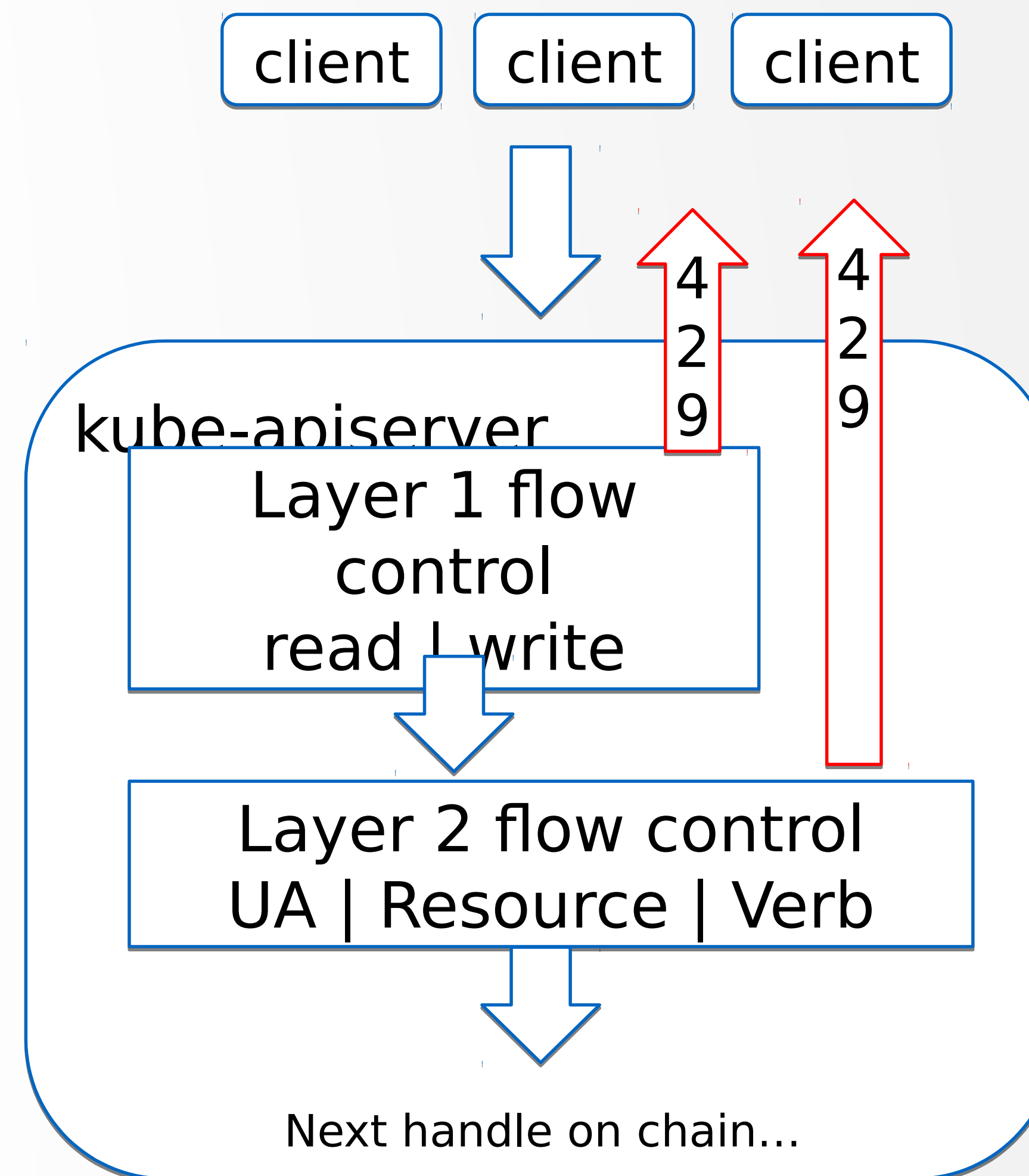
- 响应码 429 - 流控
- 默认流控重试时间 1s
- 机理：重启时所有 client 需要重新 list&watch 导致并发请求量过高（包含 bootstrap controller），超过流控。超过流控之后，返回**响应码 429**，同时在响应头里添加 **Retry-After** 告诉 client 1s 之后再试，造成 APIServer 长时间过载。



如何优化 Kubernetes ?

流控优化

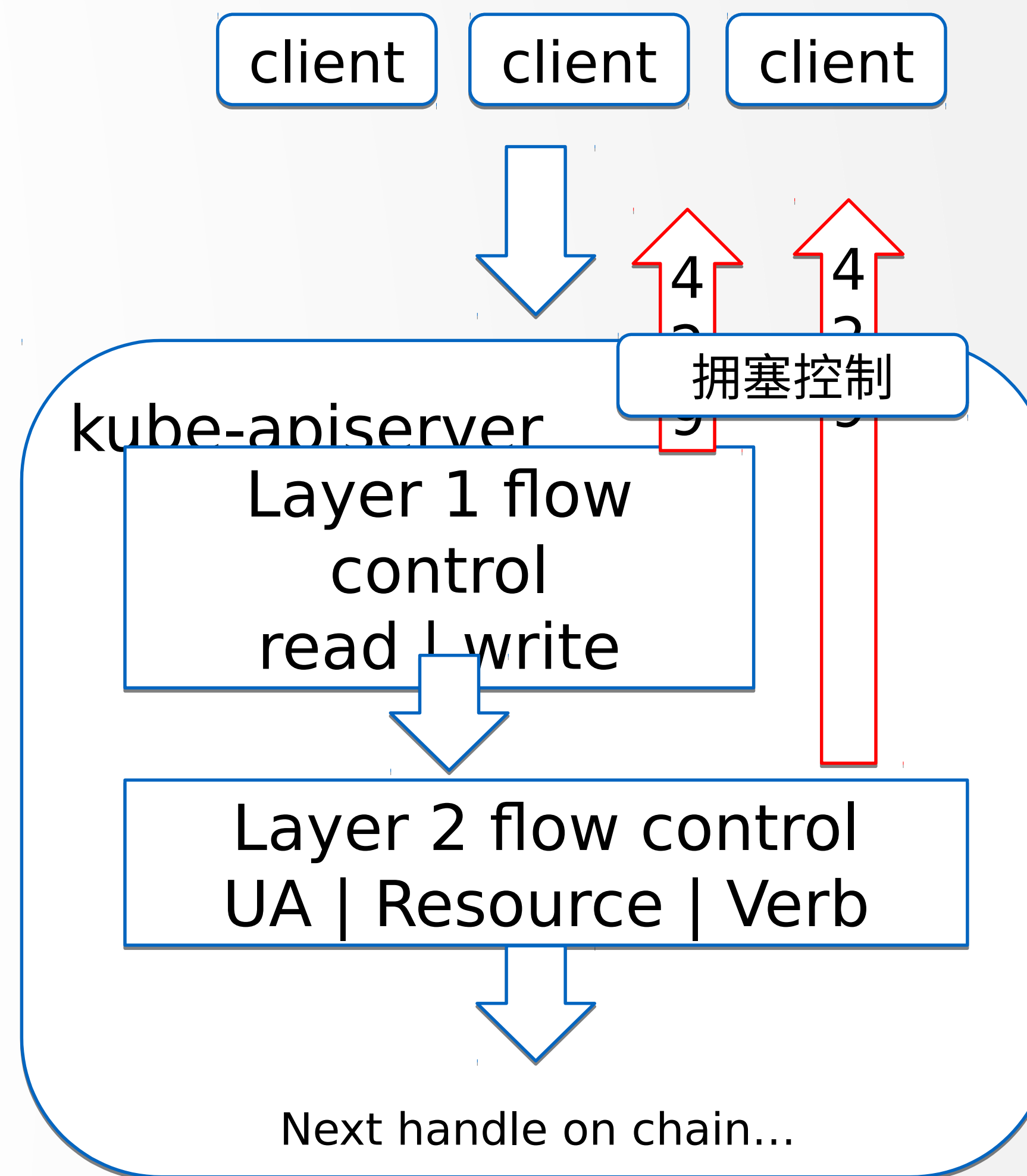
- 多级流控：在读写流控的基础上，根据 UserAgent、Resource、Verb 进行细粒度流控
- 例如：通过 UserAgent，限制 kubelet，优先允许 Master 组件初始化 (API Server Bootstrap Controller、其他 Controller 和 Scheduler 等)



如何优化 Kubernetes ?

流控优化

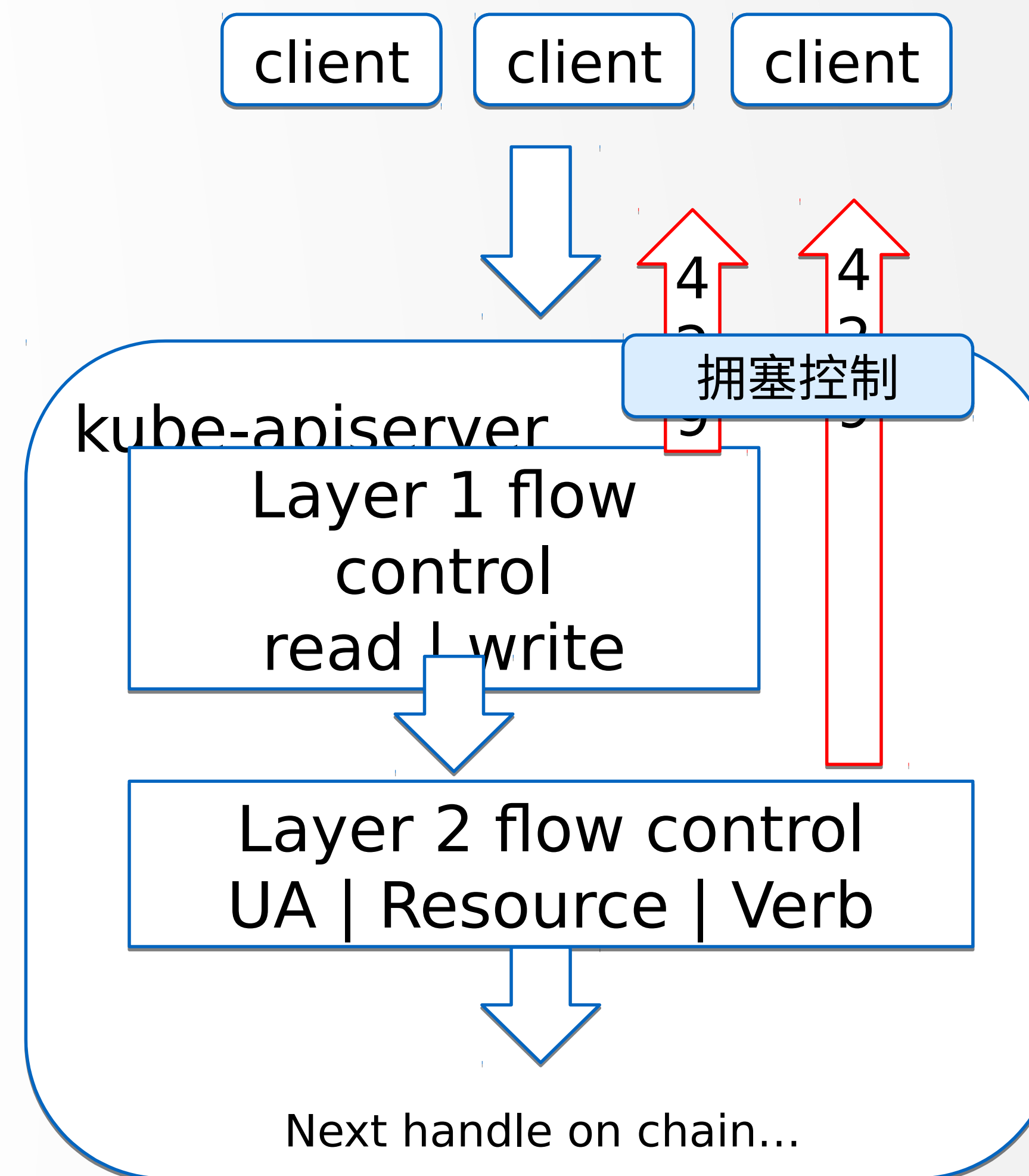
- 拥塞控制: 静态 Retry-After 改为动态值, 根据系统当前繁忙程度调节 1 到 8s (token bucket)



如何优化 Kubernetes ?

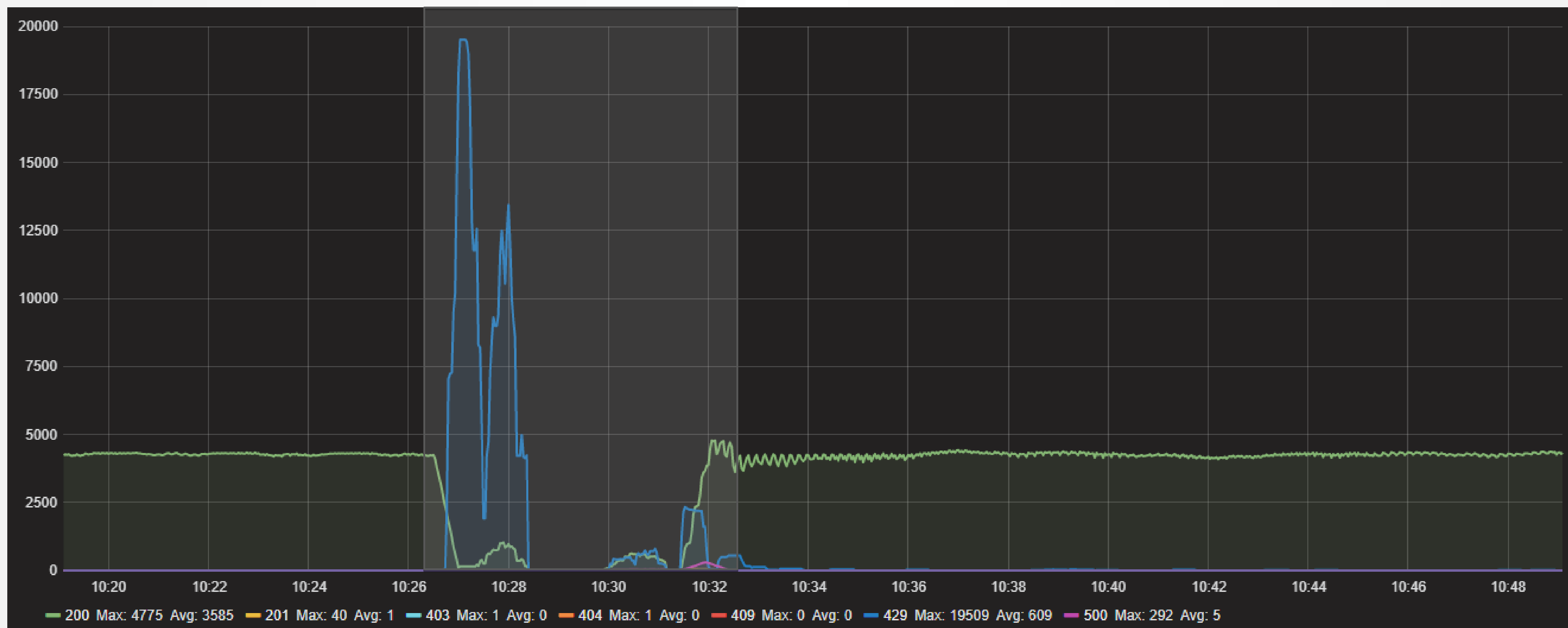
流控优化

- 拥塞控制：静态 Retry-After 改为动态值，根据系统当前繁忙程度调节 1 到 8s (token bucket)
- 算法：初始化时间窗口为 1s、容量 2^{10} 的 bucket，每次被流控尝试拿 Token
 - - 可用量 $> 1/2$ ，返回 1s
 - - 可用量 $> 1/4$ ，返回 2s
 - - 可用量 $> 1/8$ ，返回 4s
 - - 可用量不足或者不可用，返回 8s



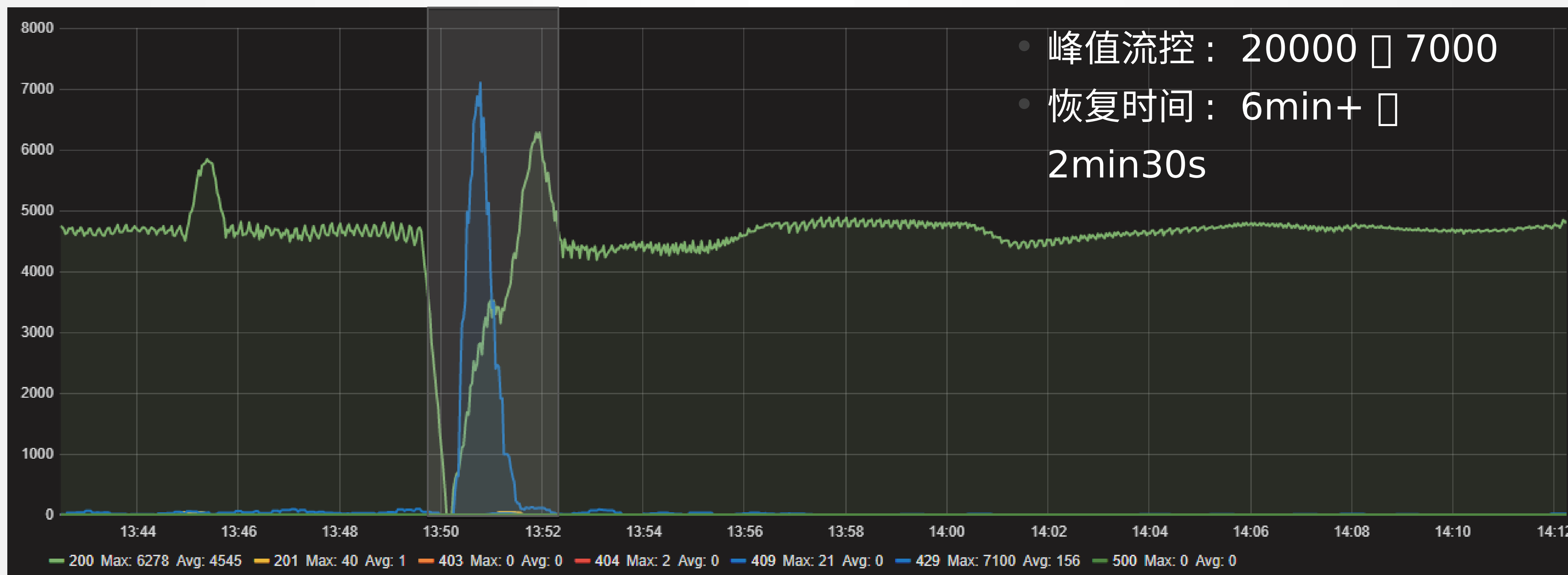
如何优化 Kubernetes ?

apiserver 响应码统计



如何优化 Kubernetes ?

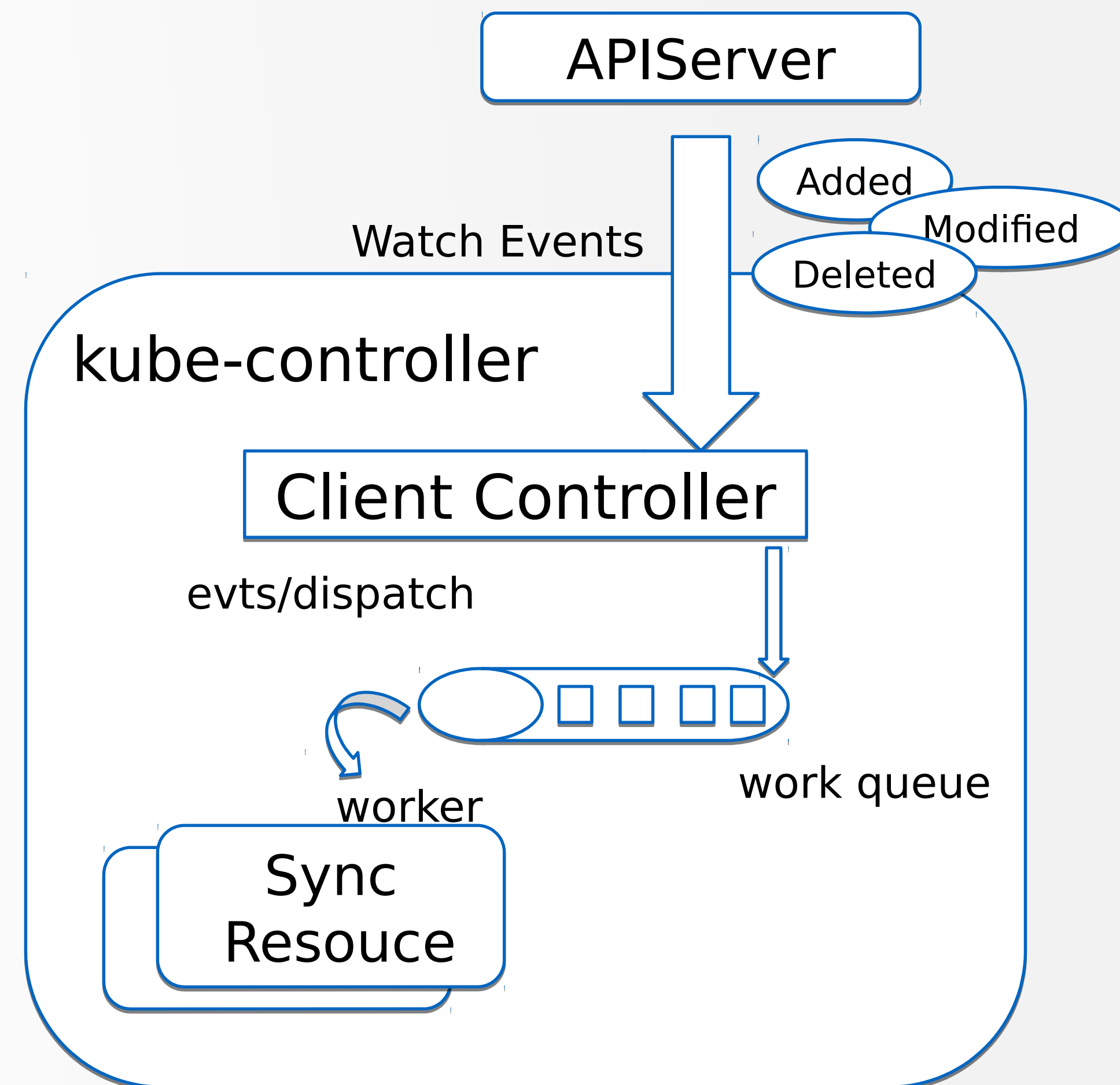
apiserver 响应码统计



如何优化 Kubernetes ?

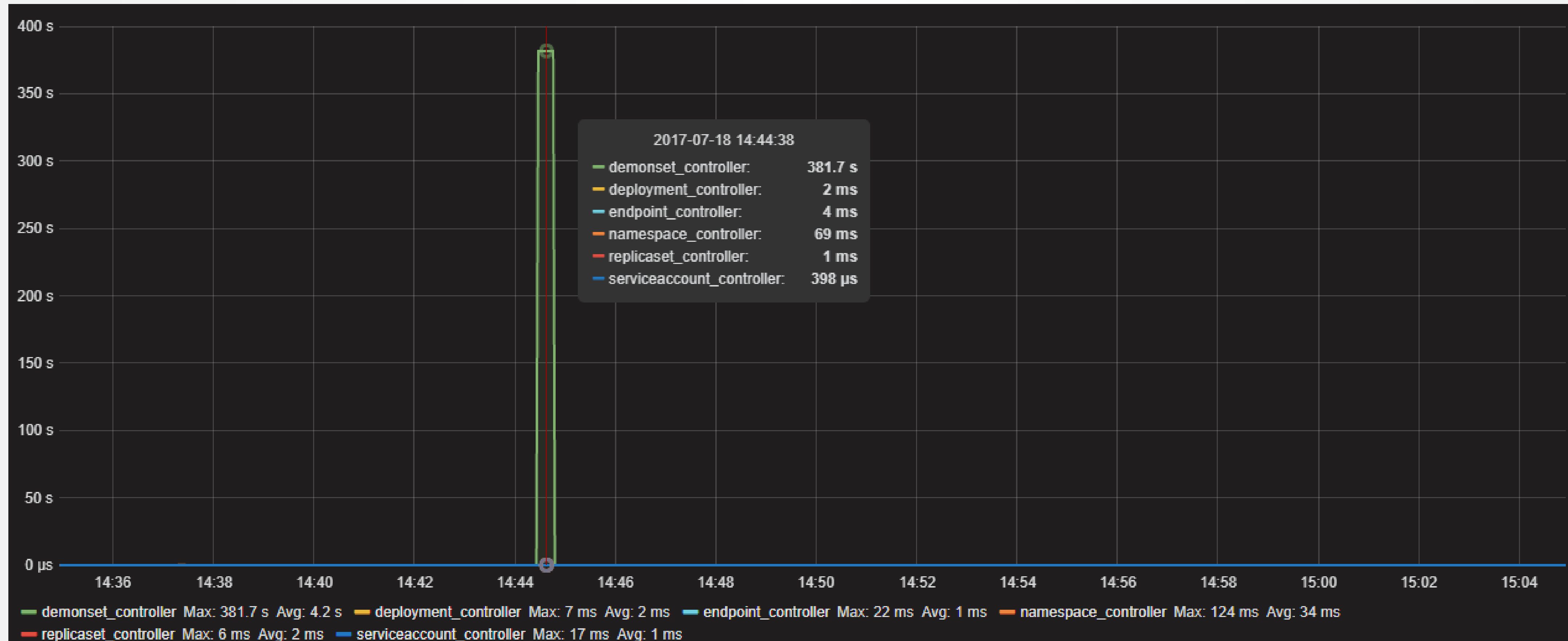
另一个例子

- Controller manager Sync 延迟统计 (resource)
- 什么是 Sync ?
 - 控制器在处理一种资源的过程称为一个 Sync (增删改)
 - 每隔一段时间 (普通资源 12 ~ 24 小时、Namespace 5 min, Endpoint 30s) **全量同步**一次 (防止漏处理, 来自 etcd2 的毛病)
- 泡一杯咖啡坐等结果……



如何优化 Kubernetes ?

Controller Sync Latency



如何优化 Kubernetes ?

Controller Sync Latency

- 发现 DaemonSet Controller 同步时间异常
- 原因：每次 DaemonSet 同步时，循环 node 时，在 node 内部重复 List Pod ，时间复杂度 $O(\text{PodNum} * \text{NodeNum} * \text{DaemonSetNum})$
- 能否优化过程：过程无法去除，因为 DaemonSet Controller 需要检查 node 可调度状态，那么就必须要取得 Node 上所有 Pod 的状态
- 优化：空间换时间，对 pod list 一次，进行分组，node 只根据检查调度到自身的 pod ，空间复杂度 $O(N)$ ，时间复杂度 $O(\max(P, N) * D)$
- 优化：加速 Pod List 的过程
- 优化：增加 Field 索引
- 结果：DaemonSet 峰值去除

如何优化 Kubernetes ?

加速 Pod List 过程

- 原理： Lister 接口接收 LabelSelector ，但此时 Pod 传入 LabelSelector 为空，则无需经过 Label 的比较
- 实现： API Expansion 机制， k8s 允许我们自定义资源操作接口，而不对原生逻辑造成侵入，不会影响 client 代码生成
- 方式： `$SRC/pkg/client/listers/$VERSION/${RESOURCE}_expansion.go`