监控系统

可观测性体系

- 指标：由监控系统时序性收集和记录的固定类型的可聚合数据，同样对单体应用较有效，但它们无法提供...
- 日志：日志是随时间发生的离散事件的不可变时间戳记录，对单体应用很有效，但分布式系统的故障通常...
- 跟踪：跟踪是跟随一个事务或一个请求从开始到结束的整体生命周期的过程，包括其所流经的组件
- 监控，处于整个生产环境稳定性建设需求金字塔的最底层

监控体系

对监控设置合理的预期

黑盒监控和白盒监控

著名的监控方法论

黄金指标

USE方法

RED方法

监控和告警设计哲学

可观测性体系

可观测性系统

■日志、指标和跟踪是应用程序可观测性的三大支柱

■前二者更多的是属于传统的“以主机为中心”的模型，而跟踪则“以流程为中心”

指标：由监控系统时序性收集和记录的固定类型的可聚合数据，同样对单体应用较有效，但它们无法提供足够的信息来理解分布式系统中调用（RPC）的生命周期

■Statsd、Prometheus...

日志：日志是随时间发生的离散事件的不可变时间戳记录，对单体应用很有效，但分布式系统的故障通常会由多个不同组件之间的互连事件触发

■ElasticStack、Splunk、Fluentd..

跟踪：跟踪是跟随一个事务或一个请求从开始到结束的整体生命周期的过程，包括其所流经的组件

■分布式跟踪是跟踪和分析分布式系统中所有服务中的请求或事务发生的情况的过程

◆“跟踪”意味着获取在每个服务中生成原始数据

◆“分析”意味着使用各种搜索、聚合、可视化和其它分析工具，帮助用户挖掘原始跟踪数据的价值

为什么需要监控系统

监控，处于整个生产环境稳定性建设需求金字塔的最底层

■它是运营一个可靠、稳定服务不可或缺的部分，它能避免SRE失去对生产系统的管控致使其蒙眼狂奔

■也是用来确保服务质量与产品目标保持一致的关键组件

监控的目的

■从规则的角度，监控信息是扩容、缩容和报警规则的数据来源

◆告警规则根据监控数据的分析结果生成异常信息并向用户报告故障

■从全局角度，基于监控信息，才能构建监控大盘

◆帮助SRE快速了解当前系统的情况，并回答当前系统表现的一些基本问题

■从长期角度，通过监控信息，可以分析系统的长期趋势

◆例如，数据库系统目前的数据量，以及增长速度和增长率

◆再例如，日活用户的数量、增长速度和增长率

■从实时角度，在系统出现变更的时候，可以通过监控系统，迅速了解新的变更是否异常

◆例如，缓存命中率是否下降，请求时延是否变长等

■从调度角度，系统出现告警信息时，监控系统能帮助SRE快速定位问题

监控体系

监控体系（自底向上）

■系统层监控

◆系统监控：CPU、Load、Memory、Swap、Disk IO、Processes、Kernel Parameters、……

◆网络监控：网络设备、工作负载、网络延迟、丢包率、……

■中间件及基础设施类系统监控

◆消息中间件：Kafka、RocketMQ和RabbitMQ等；

◆Web服务容器：Tomcat和Jetty等；

◆数据库及缓存系统：MySQL、PostgreSQL、MogoDB、ElasticSearch和Redis等；

◆数据库连接池：ShardingSphere等；

◆存储系统：Ceph等

■应用层监控

◆用于衡量应用程序代码的状态和性能

■业务层监控

◆用于衡量应用程序的价值，例如电子商务网站上的销售量

◆QPS、DAU日活、转化率；

◆业务接口：登录数、注册数、订单量、搜索量和支付量等；

对监控设置合理的预期

监控复杂应用程序本身也是一项复杂的工程项目，因此SRE团队一般有“监控专员”，但要避免“盯着屏幕找问题”

Google的实践经验

■尽量使用简单和快速的监控系统配合高效的工具进行事后分析

■最能反映真实故障的规则应该越简单越好，可预测性强，且非常可靠

■事关流量预测、容量规划的规则对错误和稳定性的要求较低，可以稍微复杂一些

■时间跨度较长的统计分析型指标采样率较低，也能容忍一定的错误率（偶发的错误率不会掩盖真正的长其趋）

■不常用的数据收集、汇总，以及告警配置应该定时删除

■若收集到的信息没有暴露给任何监控台，也没有被任何告警规则使用时，也应该定时删除

黑盒监控和白盒监控

黑盒监控与白盒监控的联系和区别

■黑盒监控

◆黑盒监控是指通过在外部对目标系统发起测试请求进行的监控行为

◆面向现象，代表了目前正在发生的（而非预测会发生的）问题，即“系统现在有故障”

■白盒监控

◆白盒监控是指依靠系统内部暴露的指标进行的监控行为

◆大量依赖对系统内部信息的检测，因此可以检测到即将发生的问题及那些被重试操作所掩盖掉的问题

◆可能面向现象（什么东西出了故障），也可能面向原因（发生故障的原因），这取决于白盒系统所提供的信息

在收集用于调试的遥测数据时，白盒监控必不可少

针对那些尚未发生但即将发生的问题，黑盒监控无能为力

著名的监控方法论

Google的四个黄金指标

■常用于在服务级别帮助衡量终端用户体验、服务中断、业务影响等层面的问题

■适用于应用及服务监控

Netflix的USE方法

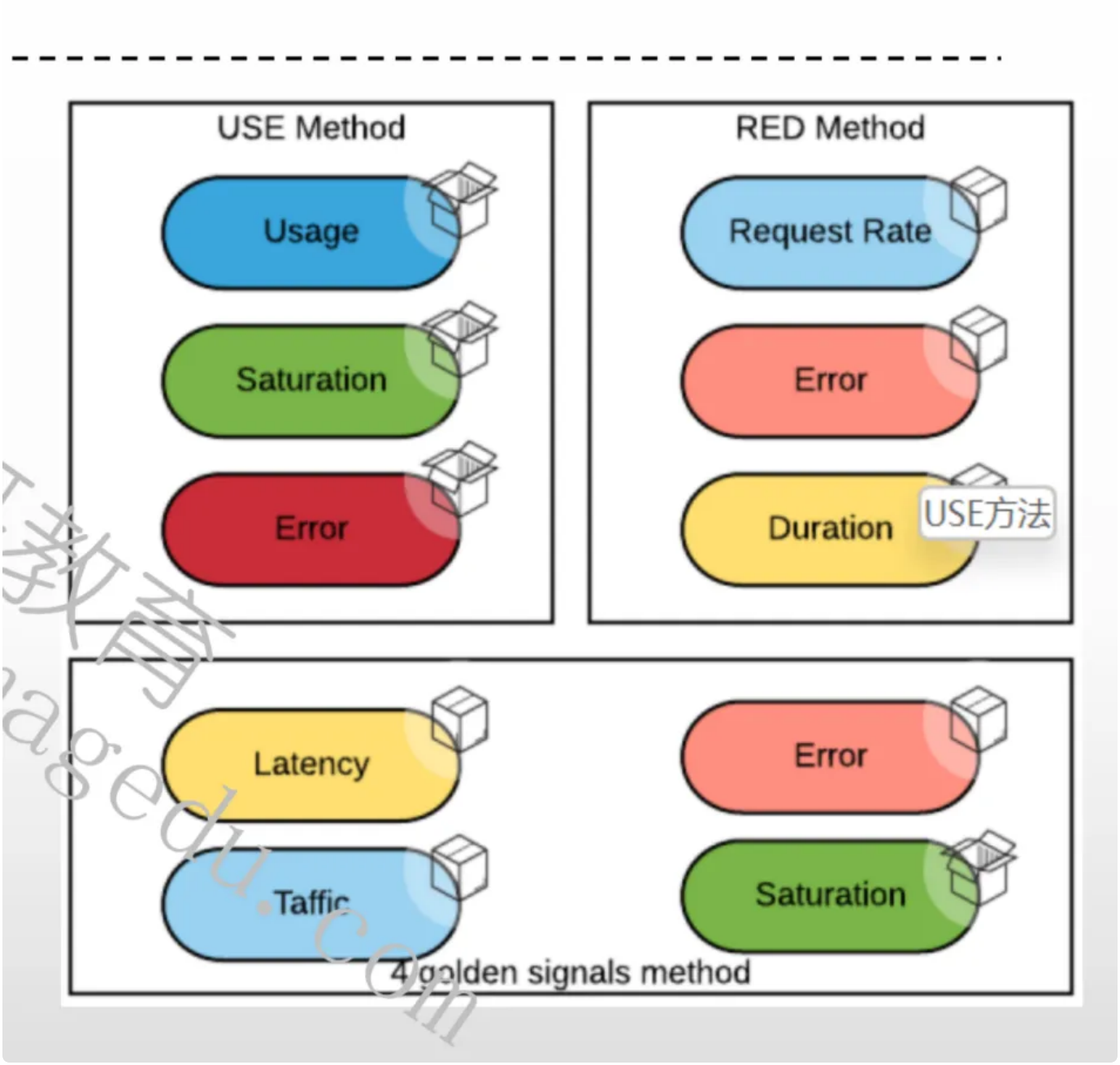
■全称为“Utilization Saturation and Errors Method”

■主要用于分析系统性能问题，可以指导用户快速识别资源瓶颈以及错误的方法

■应用于主机指标监控

Weave Cloud的RED方法

Weave Cloud基于Google的四个黄金指标的原则下结合Prometheus以及Kubernetes容器实践，细化和总结的方法论，特别适合于云原生应用以及微服务架构应用的监控和度量



黄金指标

延迟（Latency）

■服务请求所需要的时长，例如HTTP请求平均延迟

■需要将失败请求和成功请求分开来对待，以免产生误导性的结果

流量（Traffic）

■使用系统中的某个高层次的指标针对系统负载需求所进行的度量

■通常使用某业务程序的关键指标，例如Web服务器的请求量、流媒体系统的网络I/O速率或者并发会话数量、存储系统的TPS

错误（Errors）

■请求失败的速率，用于衡量错误发生的情况

■例如，HTTP 500错误数等显式失败，返回错误内容或无效内容等隐式失败，以及由策略原因导致的失败　（例如强制要求响应时间超过30毫秒的请求视为错误）；

饱和度（Saturation）

■衡量资源的使用情况，用于表达应用程序有多“满”

■通常是系统中目前最为受限的某种资源的某个具体指标的度量

◆例如内存受限的系统中，即为内存；在I/O受限的系统中，即为I/O

■很多系统在达到100%的利用率之前性能会严重下降，因此，还应该使用“利用率”指标进行度量

USE方法

USE方法由Netflix的内核和性能工程师RendanGregg提出，主要用于分析系统性能问题

主要包括如下三个指标

■利用率(Utilization)

◆关注系统资源的使用情况。这里的资源主要包括但不限于：CPU，内存，网络，磁盘等等

◆接入于100%的利用率通常是系统性能瓶颈的标志

■饱和度(Saturation)

◆例如CPU的平均运行排队长度，这里主要是针对资源的饱和度(注意，不同于4大黄金信号)

◆任何资源在某种程度上的饱和都可能导致系统性能的下降

■错误数(Errors)

◆错误计数

◆例如：“网卡在数据包传输过程中检测到的以太网网络冲突了14次”

内存指标示例

■利用率：sum by(pod,namespace) (container_memory_working_set_bytes{container="",pod!=""})

■饱和度：

◆max by (pod,namespace) (container_memory_working_set_bytes{container!=""} / on (pod,namespace,container)

■错误数：sum by (pod,namespace) (rate(container_memory_failures_total{container!="POD"}[5m]))

RED方法

RED方法是Weave Cloud在基于Google的4个黄金指标的原则下结合Prometheus以及Kubernetes容器实践，细化和总结的方法论，特别适合于云原生应用以及微服务架构应用的监控和度量

在四大黄金指标的原则下，RED方法可以有效地帮助用户衡量云原生以及微服务应用下的用户体验问题；

RED方法主要关注以下3种关键指标

■(Request)Rate：每秒钟接收的请求数；

■(Request)Errors：每秒失败的请求数；

■(Request)Duration：每个请求所花费的时长；

示例

■请求速率：sum(rate(istio_requests_total{reporter="source"}[5m]))

■请求失败数：sum(rate(istio_requests_total{reporter="source",response_code~”5.*”}[5m]))

■请求时长：

◆sum by
(le,destination_service_name,destination_service_namespace)
(rate(istio_request_duration_seconds_bucket[5m]))

监控和告警设计哲学

为监控系统和告警系统添加新规则时，回答如下问题有助于减少误报

■该规则是否能够检测到一个目前检测不到的、紧急的、有操作性的，并且即将发生或者已经发生的用户可见故障?

■是否可以忽略这条告警？什么情况可能会导致用户忽略这条告警，如何避免?

■这条告警是否确实显示了用户正在受到影响？是否存在用户没有受到影响也可以触发这条规则的情况?

◆例如测试环境和系统维护状态下发出的告警是否应该被过滤掉?

■收到告警后，是否要进行某个操作？是否需要立即进行该操作，还是可以等到第二天早上再进行？该操作是否可以被安全地自动化？该操作效果是长期的，还是短期的?

■是否也会有其他人收到相关的紧急告警，这些紧急告警是否是不必要的?

这些问题反映了如下理念

■每天的紧急告警次数不应该太多

■每个紧急告警都应该可以具体操作

■每个紧急告警的响应都应该依赖于某种智能分析过程

◆这意味着，若某个紧急告警只是需一个固定的动作，它就不应该成为紧急告警

■每个紧急告警都应该事关某个新的问题，而不应该彼此重叠