



中山大學 软件工程学院  
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

# SSE316 : 云计算技术 Cloud Computing Technology

陈壮彬

软件工程学院

<https://zbchern.github.io/sse316.html>



# 云系统运维

- ❖ 站点可靠性工程
- ❖ 系统的问题/故障模式
- ❖ 分布式系统监控



# 云系统运维

- ❖ 站点可靠性工程
- ❖ 系统的问题/故障模式
- ❖ 分布式系统监控

# 2021年7月13日哔哩哔哩宕机事故



哔哩哔哩弹幕网

今天 02:20

昨晚，B站的部分服务器机房发生故障，造成无法访问。技术团队随即进行了问题排查和修复，现在服务已经陆续恢复正常。  
耽误大家看视频了，对不起！

☆ 收藏

📄 2338

💬 4559

👍 178635



哔哩哔哩弹幕网

7月14日 20:18

很抱歉昨晚#B站崩了#，耽误小伙伴们看视频了，我们将会赠送所有用户1天大会员。

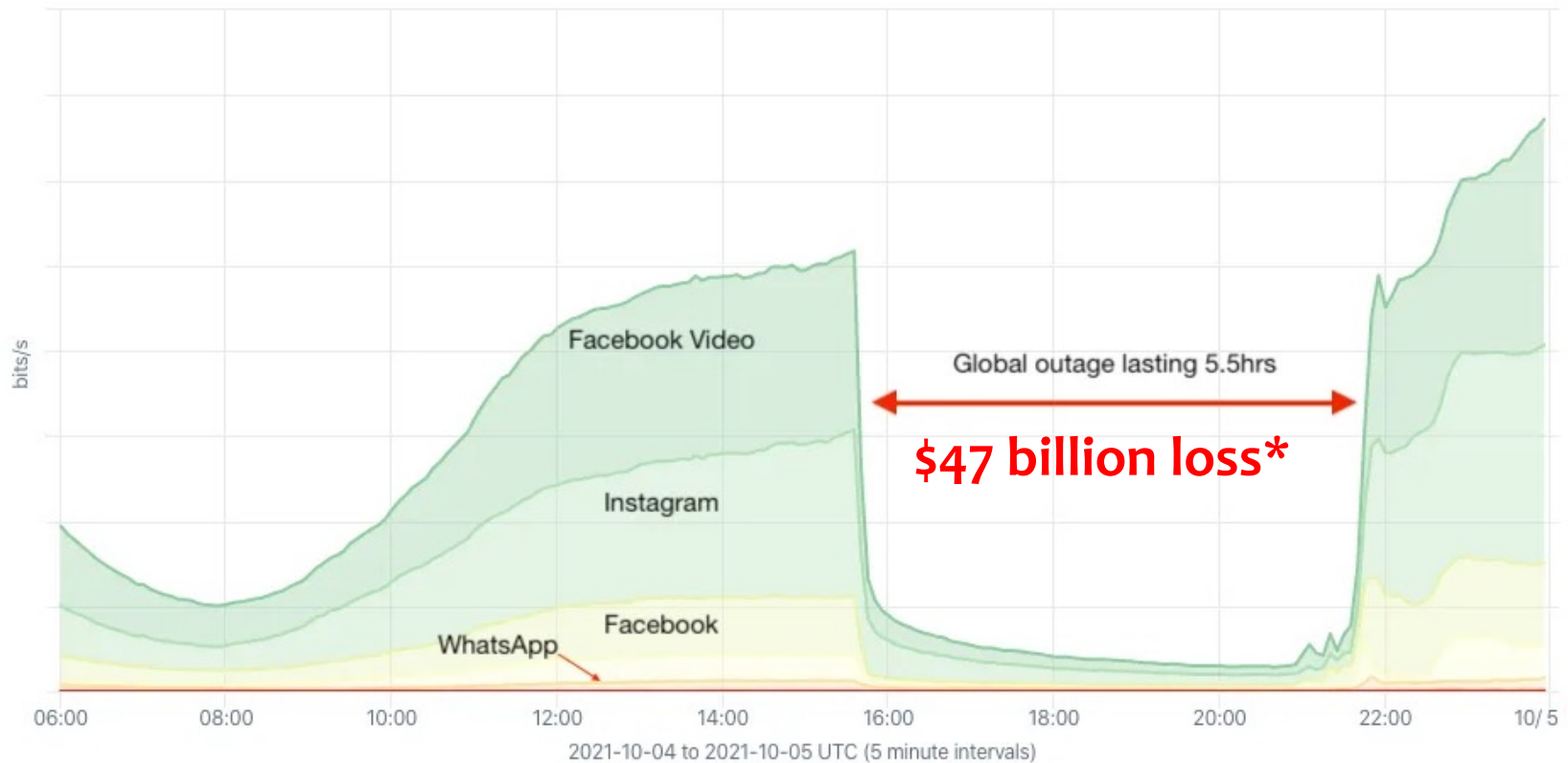
领取方式见评论👉



# 2021 Facebook 宕机事故



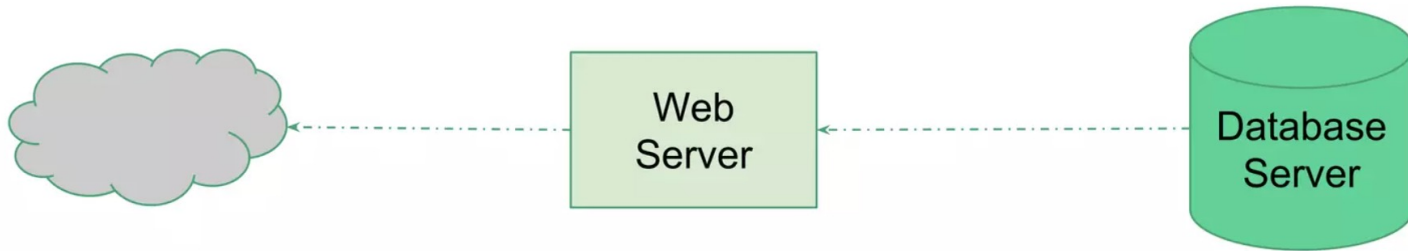
Top OTT Service by Average bits/s Internet Traffic served by Facebook  
Oct 04, 2021 06:00 to Oct 05, 2021 00:00 (18h) Global outage 4-Oct-2021



\*Data from: <https://www.datacenterdynamics.com/en/opinions/too-big-to-fail-facebooks-global-outage/>

\*\*Image from: [https://en.wikipedia.org/wiki/2021\\_Facebook\\_outage](https://en.wikipedia.org/wiki/2021_Facebook_outage)

It would have been easy if ...

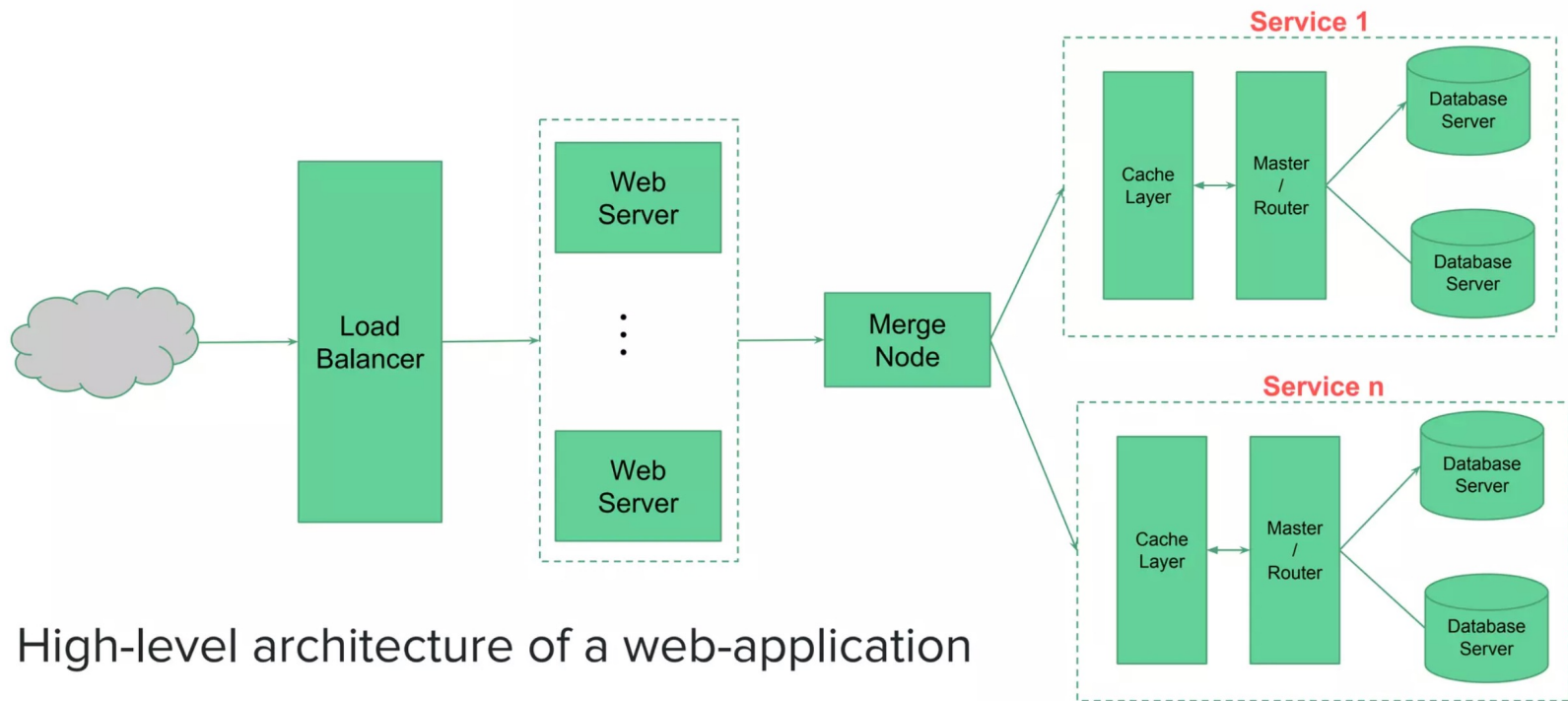


A simple web application

# 云系统运维



But most often in reality



每一个组件的故障率都不为零

# 软件系统维护



- 软件工程有时候和生养孩子类似：虽然生育的过程是痛苦和困难的，但是养育孩子成人才是真正需要花费大部分时间和精力  
的地方
- 有统计显示，一个软件系统的 **40%~90%** 的花销其实是在开发建设完成之后不断维护的过程中！



# 站点可靠性工程师



- 因此，如果软件工程职业主要专注于**设计和构建软件系统**，那么应该有另外一种职业**专注于整个系统的生命周期管理**
  - ✓ 软件系统的设计到部署
  - ✓ 软件系统服役的全过程，包括不断更新迭代
  - ✓ 软件系统顺利退役
- 这种职业被 Google 称为**站点可靠性工程师**（SRE, Site Reliability Engineering），负责
  - ✓ 确保大型软件系统运行得更可靠
  - ✓ 软件系统的架构设计
  - ✓ 运维流程的不断优化
  - ✓ 资源利用效率更高、扩展性更好

# SRE 的发展历程




- 软件系统发展的初期，雇佣系统管理员（sysadmin）运维复杂的系统是行业一直以来普遍的做法，主要负责
  - ✓ 将已开发好的软件组件部署到生产环境中，对外提供服务
  - ✓ 处理系统中各种需要人为干预的事件
  - ✓ 应对来自业务部门的需求变更
- 随着系统越来越复杂，组件越来越多，相关的事件和变更需求也越来越多，于是
  - ✓ 公司招聘更多的系统管理员
  - ✓ 与研发工程师形成两个部门：开发部（Dev）和运维部（Ops）

# 开发部和运维部之间的矛盾



- 70% 的生产事故来自软件系统的变更
  - **新功能上线**: 例如新代码中存在bug，或者新功能与现有功能之间存在未预料到的交互效应
  - **配置更新**: 例如数据库连接的配置被错误地更改，应用可能无法访问其数据源



这将引发开发部门和  
运维部门之间的矛盾！

# 开发部和运维部之间的矛盾



- 开发部门关注如何能够更**快速地构建和发布新功能**



# 开发部和运维部之间的矛盾



- 运维部门关心如何能**在值班期间避免故障**



# 开发部和运维部之间的矛盾



- 在现实生活中，公司内部这两股力量只能用最传统的政治斗争来保障各自的利益
  - ✓ 运维团队宣称：任何变更上线必须经过由运维团队制定的流程，有助于避免事故的发生
  - ✓ 研发团队反击：我们不是进行大规模更新，只是小修小补，不需要再走一遍流程

# Google 的解决之道：SRE



- SRE 模型是 Google 尝试从根本上避免这种矛盾的方法，通过雇佣专门的软件工程师创造软件系统来运维系统运行以替代传统模型中的人工操作
- SRE 团队有两种类型的工程师
  - ✓ 团队中 50% ~ 60% 是标准软件工程师，即应用开发人员
  - ✓ 剩下 40% ~ 50% 是基本具有标准软件工程师的技能，同时又具有一定程度其他技术能力的工程师（UNIX 系统内部细节和 1 ~ 3 层网络知识是 Google 最看重的）

SRE 是 DevOps 模型在 Google 的具体实践



- SRE 团队的成员有如下特点
  - ✓对重复性、手工性的工作天然排斥
  - ✓由足够的技能快速开发出系统以替代手工操作

本质上讲，SRE 就是用软件工程的思维和方法论完成以前由系统管理团队手动完成的任务。这些 SRE 倾向于通过设计、构建自动化工具来取代人工操作。



# 世界上第一个 SRE



- 玛格丽特·汉密尔顿是一名参加阿波罗登月计划的软件开发工程师，同时也是 MIT 教授
- 在阿波罗 7 研发期间，她的女儿劳拉偷偷按下控制台的某个键，使得整个模拟程序奔溃了，导致整个火箭发射程序意外终止
- 经过调试发现劳拉意外触发了 P01 这段子程序的执行。玛格丽特向项目组提交一个**软件改动，进行特殊状态检查**，用以检测飞行员是否意外触发 P01 程序的执行
- NASA 管理层觉得此事概率太小了，因此驳回。玛格丽特只能在飞行手册里加一句“请勿触发 P01 程序，”此事还被其他工程师嘲笑，认为是小题大做
- 几天后，阿波罗 8 执行飞行计划途中，飞行员意外触发 P01 程序！当时正值圣诞节，大部份工程师处于休假状态
- 所幸飞行手册提到了解决方法，飞行员因此幸免于难！
- 最终玛格丽特的软件改动提案通过

# 典型的 SRE 活动



- 软件工程：编写或修改代码，以及所有其他相关的设计和文档工作
- 系统运维工程：处理报警信息（定位、解决问题），必要时联系其他团队（拉起 warroom）；监控和优化系统的性能（处理速度、响应时间、资源使用率等）
- 琐事：与运维服务相关的重复性的、手工的劳动
- 流程负担：与运维服务不相关的行政工作，如招聘、会议等

# 琐事繁多是不是一定不好？



- 从心理学的角度看，已知和重复性的工作有一种让人平静的功效，完成这些事可以带来一种满足感和快速胜利感
- 少量的琐事不是什么大问题，但是一旦琐事的数量变多了，就会有害了

# On-call 工程师压力是不是越小越好？



- 虽然系统非常稳定对 on-call 工程师而言是非常幸福的事，但长时间不操作生产环境会导致自信心问题，包括过度自信及自信心不够
- 为避免此问题，最好保证每个工程师每个季度至少 on call 一两次的；同时，Google 每年举办一次维持数天的全公司灾难恢复演习
- 当然，压力太大，睡眠不足或不规律也绝不是什么好事

# SRE 做的怎么样？



如何衡量系统的可靠性？

# 传统的单体软件/硬件所采用的指标



- **故障率 ( Failure Rate )** : 在一定时间内, 系统或设备发生故障的平均频率。通常以 “每小时故障次数” 或 “每年故障次数” 等单位来表示
- **平均无故障时间 ( Mean Time to Failure, MTTF )** : 指系统或设备在发生首次故障前的平均运行时间, 通常用于描述那些不能或不需要被修复的系统或设备 ( 如云系统中的容器和 FaaS )
- **平均修复时间 ( Mean Time to Repair, MTTR )** : 指系统或设备从发生故障到被修复并恢复正常运行所需的时间
- **平均故障间隔时间 ( Mean Time Between Failures, MTBF )** : 指连续两次故障之间的平均时间, 包括故障发生时的停机时间

# 云系统更重要的指标



- 服务质量指标 ( Service level indicators ) : 对所提供的服务水平的某些方面的具体**定量衡量**
  - ✓ 错误率 ( Error rate ) : 请求处理失败的百分比, 比如 HTTP 5XX
  - ✓ 系统吞吐量 ( Throughput ) : 系统每秒处理的请求数
  - ✓ 延迟 ( Latency ) : 系统处理请求的时间
  - ✓ 可用性 ( Availability ) : 系统可用时间的百分比

# 云系统更重要的指标



- 服务质量目标 ( Service level objectives ) : 服务的某个 SLI 的目标值或者目标范围
  - ✓ 错误率 ( Error rate ) : 请求处理失败的百分比小于 0.1%
  - ✓ 系统吞吐量 ( Throughput ) : 99% 的请求在 200ms 内完成
  - ✓ 延迟 ( Latency ) : 在峰值负载期间, 系统将每秒处理 1000 个请求
  - ✓ 可用性 ( Availability ) : 服务在 99.99% 的时间内都可用

SLO 表明了系统的可靠性, 但选择一个合适的 SLO 是非常复杂的过程



# 选择合适的 SLO



- 是否可靠性越高越好？100% 的可靠性是不是一个正确的目标？

**事实上，可靠性只要到达一定的程度即可，并不是越高越好**

# 选择 SLO 考虑的因素



- 成本

可用性目标：99.9% -> 99.99%

增加的可用性：0.09%

服务收入：100 万美元

改进可用性后的价值： $100 \text{ 万美元} * 0.09\% = 900 \text{ 美元}$

在这种情况下，若达成此可用性目标的成本低于 900 美元，择是合理的投资，否则将亏本。

# 选择 SLO 考虑的因素



- 过高的 SLO 可能没有意义

用户体验主要受较不可靠的组件主导，用户在一个有着 99% 可靠性的智能手机上不能分辨出 99.99% 和 99.999% 的服务可靠性的区别。

# 选择 SLO 考虑的因素



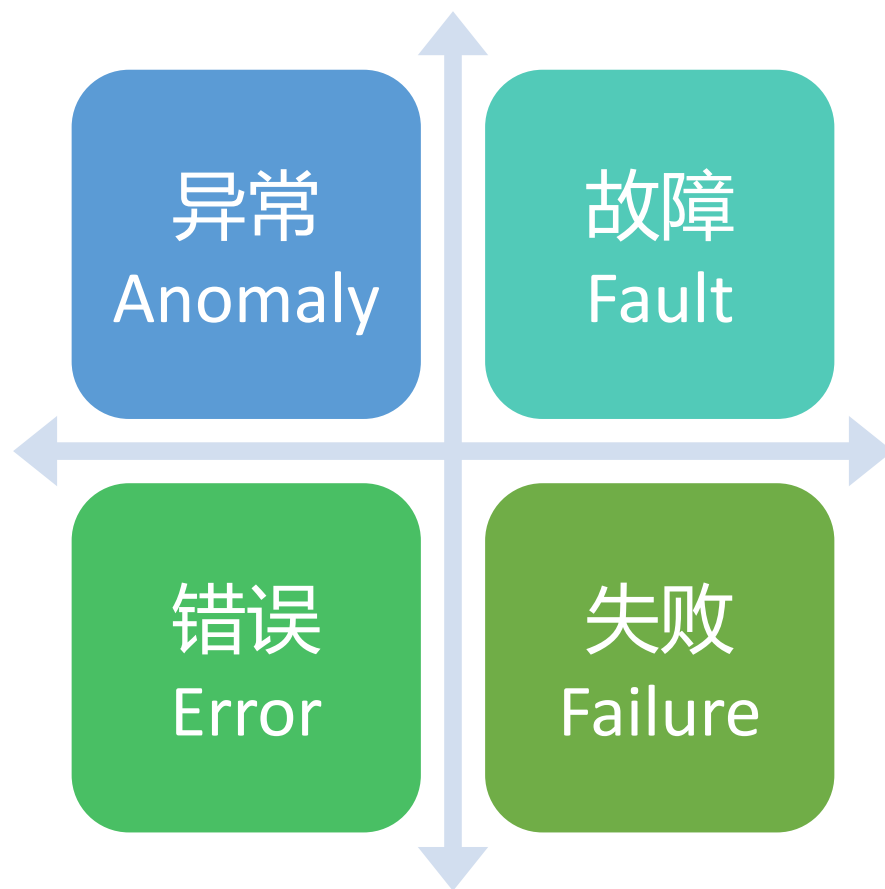
- 过高的 SLO 可能有害
  - 谷歌内部使用的锁服务为 Chubby。Chubby 的可靠性极高，以至于工程师们编写代码时直接假定 Chubby 不会奔溃
  - 当 Chubby 真的发生故障时，相关服务发生了灾难性的后果
  - 为了解决这个问题，谷歌管理层下令，如果 Chubby 在某个月内没有发生故障，则故意让其停机 5 分钟



# 云系统运维

- ❖ 站点可靠性工程
- ❖ 系统的问题/故障模式
- ❖ 分布式系统监控

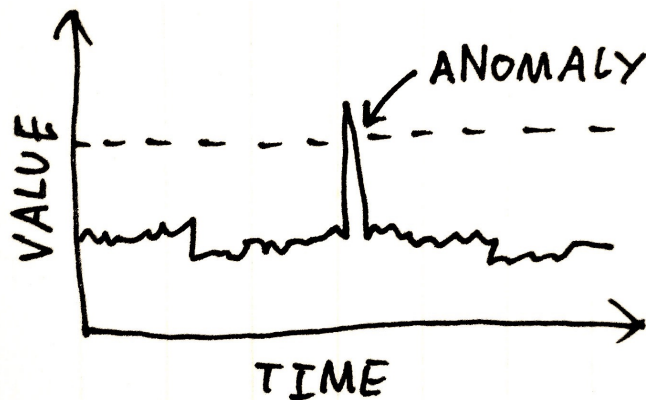
# 如何描述系统的问题/故障模式



# 异常 ( Anomaly )



- 异常通常是指系统行为偏离了预期的正常行为
- 在云计算中，异常可能指的是性能下降，系统的响应时间增加，或者出现了预期之外的系统行为
- 例如，假设云服务器CPU使用率的正常范围在 10% 到 50% 之间。如果 CPU 使用率突然飙升到 90%，那么这就是一个异常。需要注意的是，异常并不总是指示问题，但通常需要进一步的调查



CPU 利用率飙升

# 故障 ( Fault )



- 故障是指系统内部的一个**组件或者模块出现的问题**，包括硬件组件（如内存、硬盘、CPU等）和软件组件（如操作系统、应用程序等）
- 假设云服务器器的一个硬盘有问题，这可能会导致服务器停机





# 根因 ( root cause )



- 导致问题或故障的深层次、基本的原因。在完成了故障定位后，我们需要进一步调查和分析，找出问题的真正原因
- 比如，在上述服务器停机的例子中，我们可能通过故障定位发现**硬盘出了问题**。但进一步的根因定位可能会发现，是因为硬盘的**散热系统设计不合理**，导致硬盘过热并最终损坏

# 错误 ( Error )

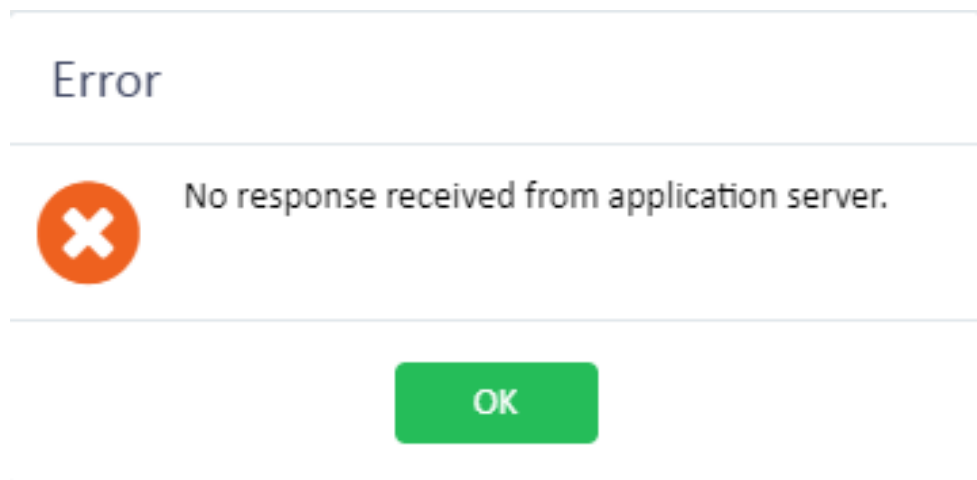


- 错误是指系统的部分或全部功能无法按预期的方式进行
- 比如上述的服务器因为硬盘故障导致停机便是错误

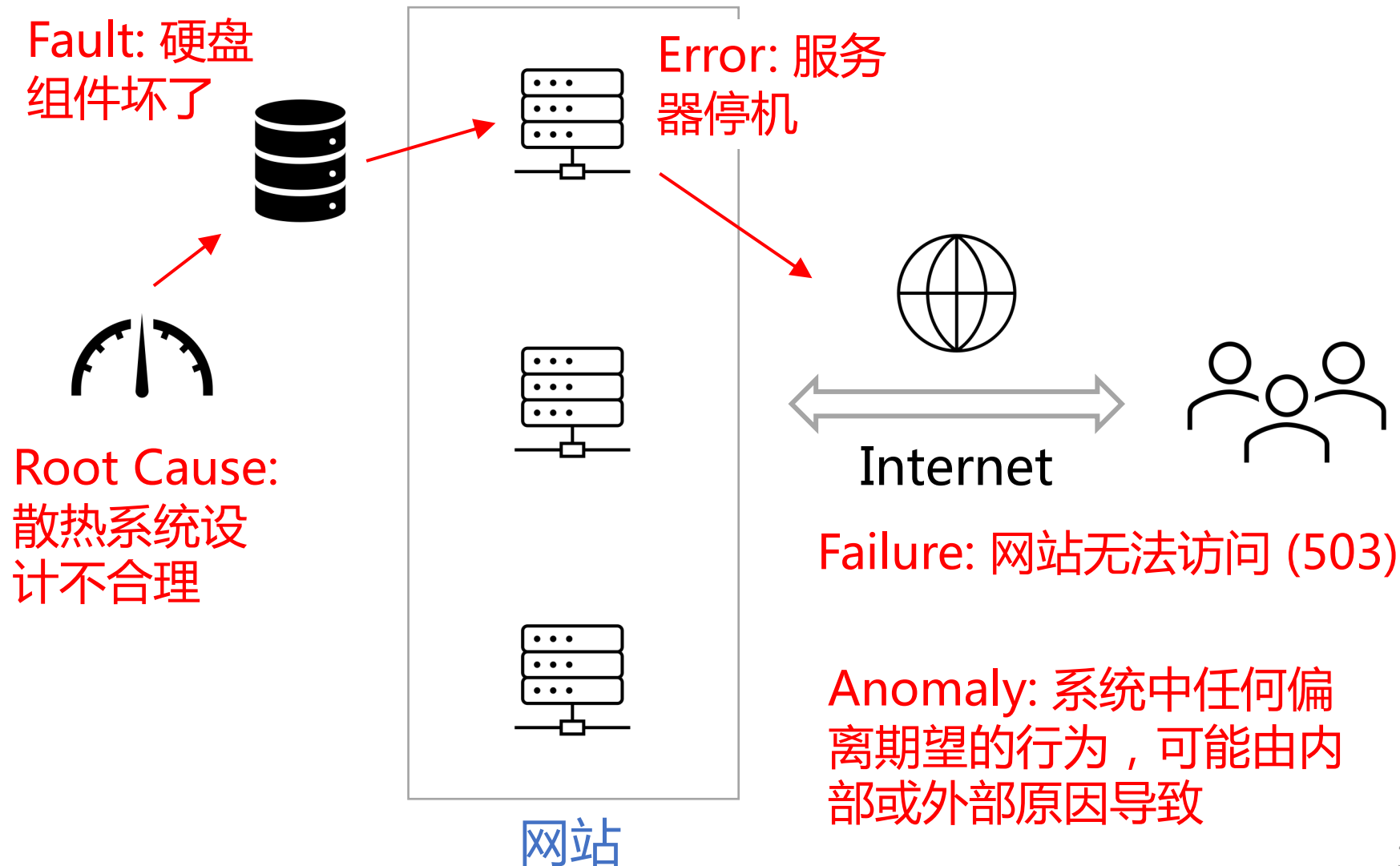
# 失败 ( Failure )



- 从用户角度可感知的系统无法提供预期服务的情况，从某种意义上讲是云提供商最关心的问题
- 例如，如果云服务由于硬件故障、软件错误或网络问题无法响应用户的请求，那么我们就可以说该服务已经失败



# 上述概念的关系



# 一个例子

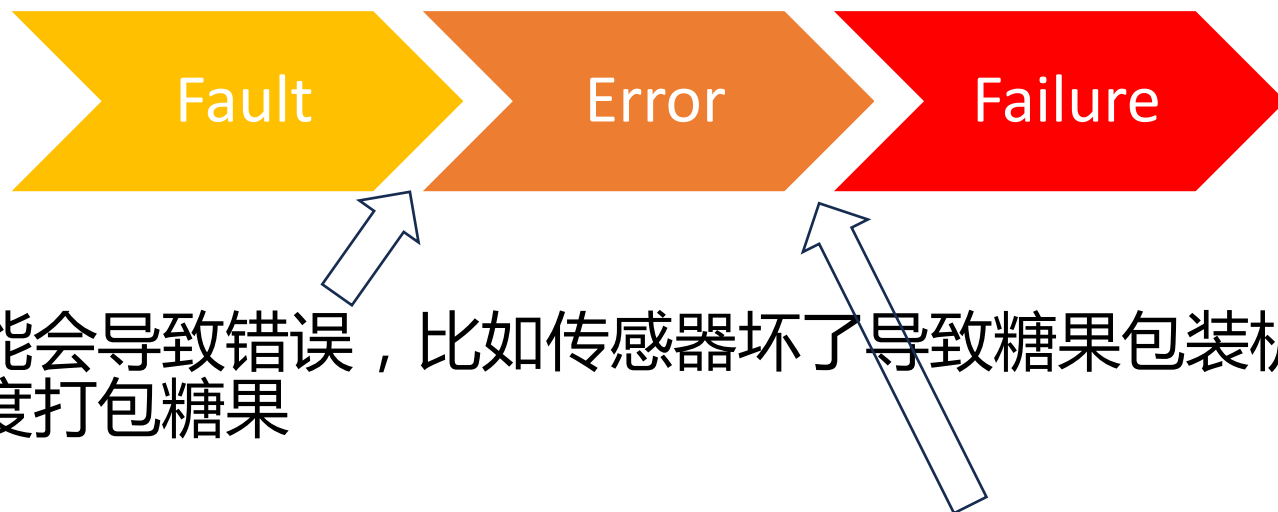


- 假设我们有一个自动化的糖果工厂。当工厂的糖果制作机器运行时，通常每小时可以生产 1000 个糖果。
- 有一天，你发现每小时只生产了 500 个糖果，这就是一个**异常**。
- 查找问题，你发现糖果包装机有一个**错误**，它没有按预期的速度打包糖果。
- 如果这个错误没有得到修复，那么可能会导致整个糖果生产线的**失败**，因为没有打包的糖果不能被送出去销售。
- 接着你发现原来是糖果包装机的一个传感器**故障**了，它没有正确检测到糖果的存在，所以包装机没有正确地打包糖果。
- 最后发现是传感器进水损坏了，这便是**根因**。

# 上述概念的关系



它们是从系统的内部问题（故障），到系统状态的不正确（错误），再到系统无法正常提供服务（失败）的过程。



- 故障可能会导致错误，比如传感器坏了导致糖果包装机没有按预期速度打包糖果
- 错误如果不处理可能会导致失败，比如糖果包装机无法工作，糖果生产线将无法生存糖果
- 当出现错误时，系统通常会尝试采取冗余措施或故障恢复策略以避免失败。如果这些措施成功，则不会有系统失败，否则将产生系统失败



# 云系统运维

- ❖ 站点可靠性工程
- ❖ 系统的问题/故障模式
- ❖ 分布式系统监控

# 分布式系统监控



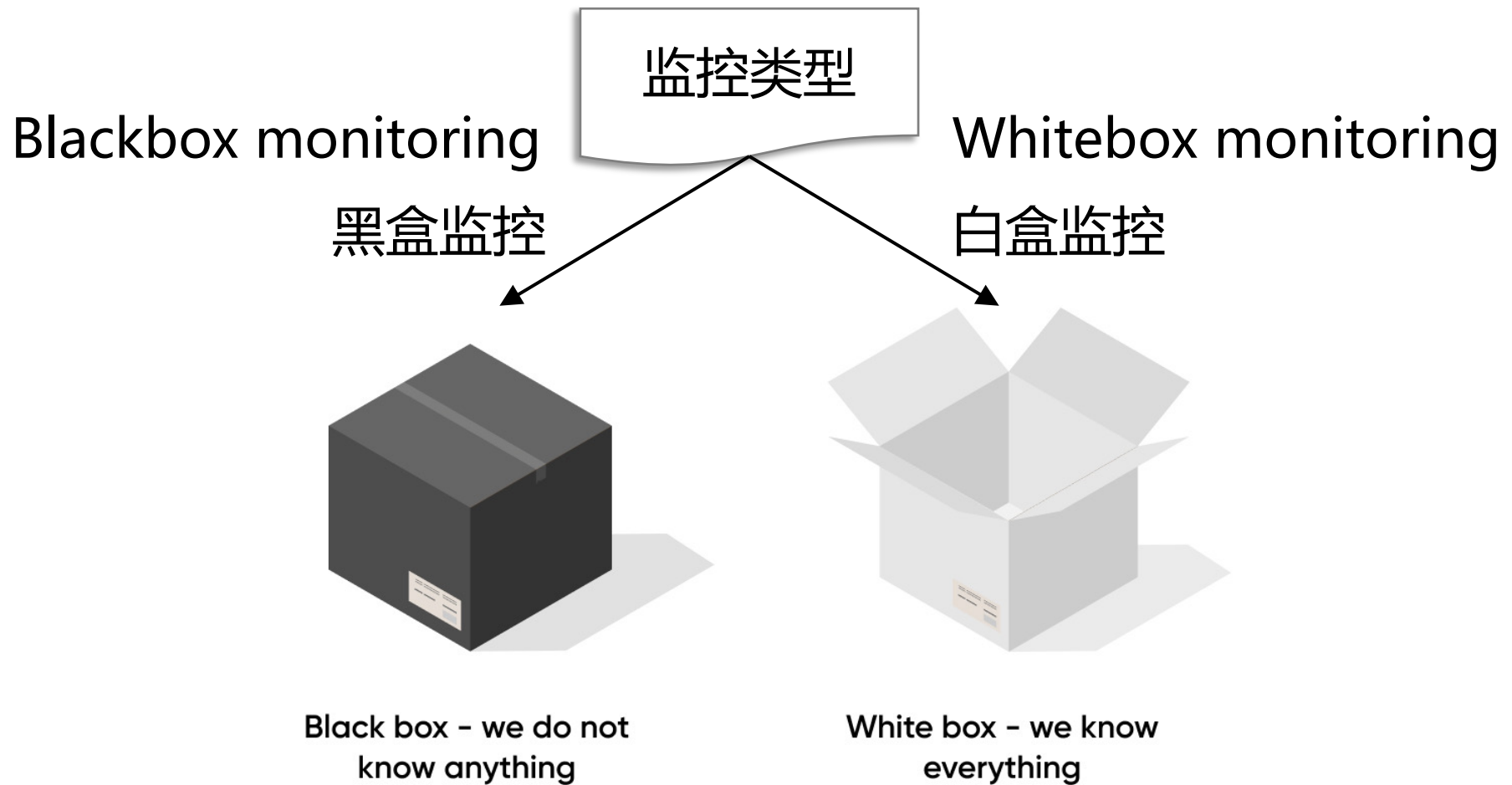
- 监控，处于整个生产环境需求金字塔模型的最底层，是运营一个可靠的稳定服务不可缺少的部分
- 服务运维人员依靠**实时收集、处理和分析分布式系统的信息和状态数据**，以维持系统的正常运行并保障系统性能

监控类型

监控数据



# 系统监控类型



- 白盒监控指的是直接监控系统内部的状态，需要对系统的内部结构和工作原理有深入的了解
- 白盒监控的优点在于，它可以提供非常详细和精确的信息，帮助我们深入理解系统的工作状态
- 但同时，白盒监控也需要花费更多的时间和资源来实施，因为需要对系统的内部结构和工作原理有深入的了解

- 白盒监控虽然能对系统内部有深入了解，但无法知道内部状态对外部用户的影响
- 黑盒监控则是从系统的外部，只关注系统的输入和输出，而不关心系统内部是如何工作的。黑盒监控常常通过模拟用户行为，检查系统是否能正常提供服务
- 但同时，当系统出现问题时，黑盒监控可能无法提供足够的信息来定位问题

- 系统监控数据的典型类型

● 指标 ( Metric )

● 日志 ( Log )

● 追踪 ( Trace )

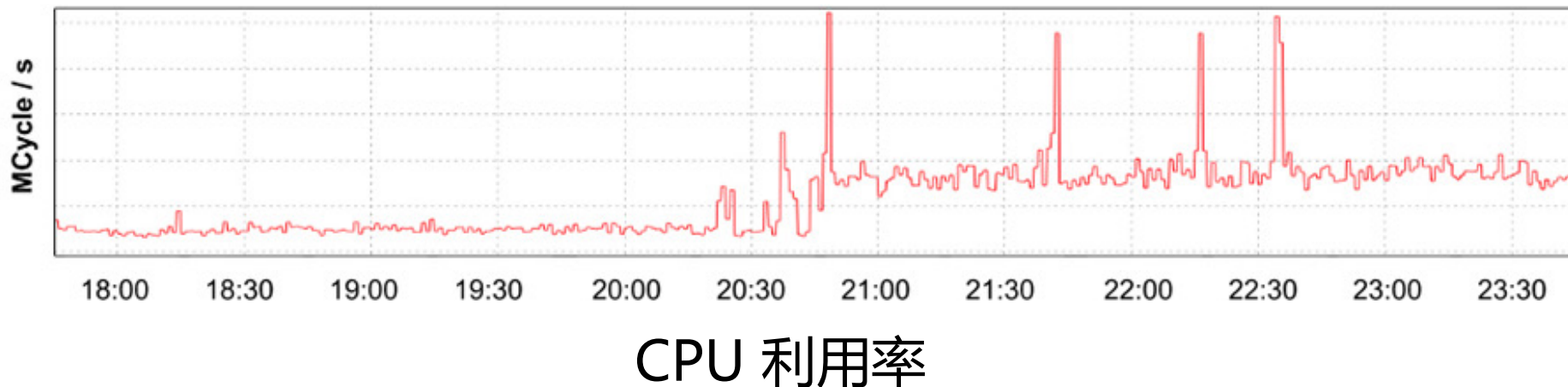
● 告警 ( Alert )

● 工单 ( Ticket )

# 指标 ( Metric )



- 对系统性能的定量衡量，也称系统关键性能指标（KPI, Key Performance Indicator）
- 以时间序列的形式收集和存储的，采样频率固定（比如 1s、1m），便于观察系统性能随时间的变化



# 四个黄金指标



- **延迟 ( Delay )** : 服务处理请求所需要的时间
  - ✓ HTTP 页面从请求到响应的时间差
  - ✓ 数据库查询从发起到数据返回到时间差
- **流量 ( Traffic )** : 通过系统的数据量
  - ✓ 对 Web 服务器来说, 是每秒 HTTP 请求数量
  - ✓ 对视频媒体系统来说, 是网络 I/O 速率
- **错误 ( Errors )** : 请求失败的速率
  - 显示失败 ( 例如 HTTP 5XX )
  - 隐式失败 ( 例如 HTTP 200 回复中包含了错误内容 )
  - 策略原因导致的失败 ( 例如要求回复在 1s 内, 任何超过 1s 的请求均为失败 )

# 四个黄金指标



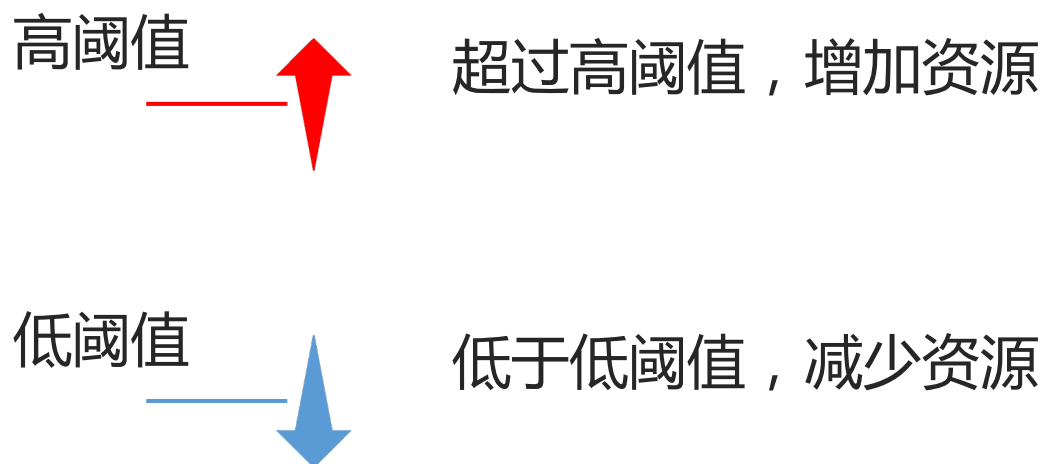
- **饱和度 ( Saturation )** : 描述服务容量有多 “满 , ” 通常是系统中目前最为受限的某种资源的某个具体指标的度量
  - ✓ 在内存受限的系统中 , 为内存使用率
  - ✓ 在 I/O 受限的系统中 , 为 I/O 使用率

对这四个黄金指标进行检测 , 同时在某个指标发生故障时 ( 或即将要发生故障时 ) 发出告警 , 能做到这些服务的监控就基本差不多了。

# 指标如何帮助系统进行问题检测



- 监控系统各类状态是否处于正常范围内，通常用于检测系统异常（Anomaly）和失败（Failure）



- 关键难题是如何设置指标及响应的阈值



# 指标监控工具 - Prometheus



- 普罗米修斯 ( Prometheus ) 是一款开源的**系统监控和警报工具包**，它的核心组件是一个时间序列数据库，用于存储所有收集到的监控数据。
- Prometheus 通过 HTTP 协议从被监控的服务中拉取指标，然后对这些数据进行存储和查询。



<https://prometheus.io/>

# 指标监控工具 - Grafana



- Grafana 则是一个**开源的指标分析和可视化套件**，可以连接到多种不同的数据源（包括 Prometheus）。
- Grafana 提供了丰富的数据可视化选项，包括图表、表格、热图等，并提供了一套警报机制，允许用户定义基于特定指标的警报规则。



## Grafana

<https://go2.grafana.com/grafana-cloud.html>



<https://play.grafana.org/d/000000012/grafana-play-home?orgId=1>

# 日志 ( Log )



- 日志是**系统活动的详细记录**。它通常包括时间戳，事件的发生地点（例如，具体的服务器或服务），以及事件的详细描述
- 比如用户在执行某个操作时发生错误，系统生成一条日志记录错误的信息，包括错误的类型，发生的时间，以及可能的原因

```
1 public void setTemperature(Integer temperature) {  
2     // ...  
3     logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);  
4     if (temperature.intValue() > 50) {  
5         logger.info("Temperature has risen above 50 degrees.");  
6     }  
7 }
```

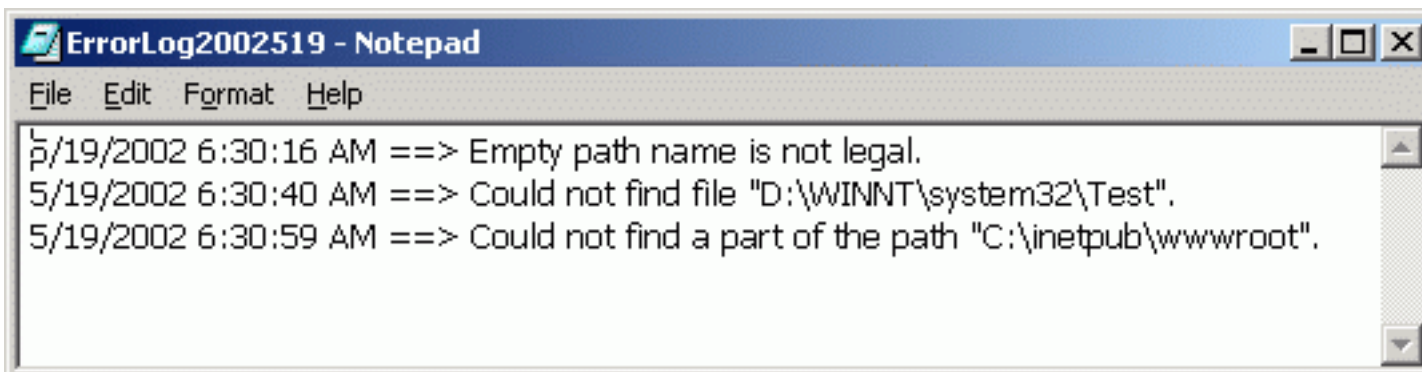


```
1 0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.  
2 0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

# 日志如何帮助系统进行问题检测



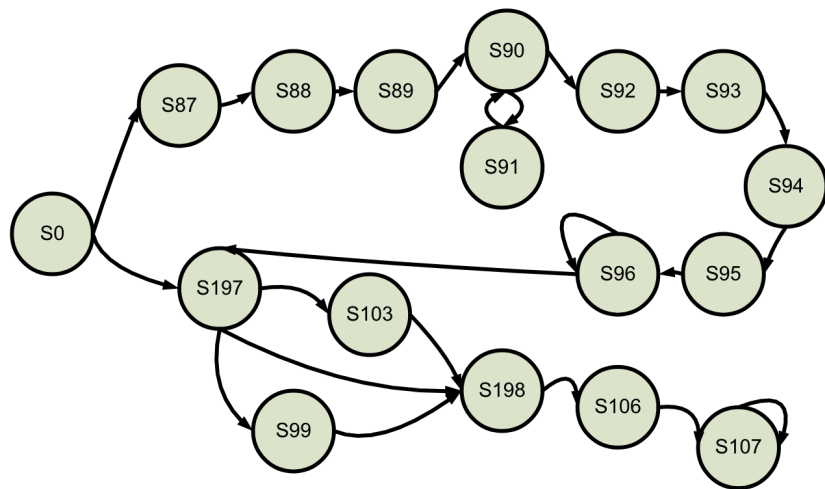
- 当出现错误日志即表明系统存在相应的问题



# 日志如何帮助系统进行问题检测



- 基于日志图结构<sup>[1, 2]</sup>的系统行为分析
  - ✓ 代表系统正常运行时产生的 log，每一个节点代表一种类型的 log
  - ✓ 若系统行为偏离此图，则表明可能出现问题
    - 出现新类型的 log 或缺失某条 log
    - log 间的转换异于此图
    - 节点的转换性能太差（如延迟，循环次数等）



Hadoop Mapreduce Task

State	Interpretation
S87~ S96	Initialization when a new job submitted
S197	Add a new map/reduce task
S103	Select remote data source
S99	Select local data source
S198	Task complete
S106	Job complete
S107	Clear task resource

The interpretations of states

[1] Fu et al. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. ICDM 2009.

[2] Nandi et al. Anomaly Detection Using Program Control Flow Graph Mining from Execution Logs. KDD 2016.

# 日志记录的最佳实践



- **结构化日志**：尽可能以结构化的格式（如JSON）记录日志，结构化的日志可以更容易地进行查询和分析。
- **平衡日志数量与信息量**：太少的日志无法记录系统详细信息，太多的日志对存储和计算带来压力，需要识别系统关键日志。
- **统一日志级别**：使用标准的日志级别（如debug, info, warn, error, fatal）可以帮助你更好地过滤和查找日志。
- **日志内容要明确**：确保每条日志都有具体、明确的含义，可以帮助其他开发人员或运维人员更快地理解和解决问题。避免使用含糊不清或过于技术性的词汇。
- **日志轮换和清理**：为了避免硬盘空间被日志文件占满，应定期进行日志轮换和清理。同时，根据业务需求和法规要求，设定适合的日志保留策略。
- ...

# ELK 堆栈



- Elasticsearch、Logstash 和 Kibana 通常被统称为 ELK 堆栈，是一套**开源的日志管理和分析解决方案**：
  - ✓Elasticsearch：Elasticsearch 提供了一个分布式、多租户的**全文搜索引擎**可以在几分钟内处理 PB 级的数据
  - ✓Logstash：Logstash 是一个开源的**数据收集引擎和传输工具**，可以灵活地获取并标准化来自系统、Web或其他来源的日志
  - ✓Kibana：Kibana则是 ELK 堆栈的**可视化组件**，无论是进行深度的数据分析，还是快速搜索你的应用日志，或者是监控你的系统健康状况，Kibana都能帮助你更好地理解你的数据



# Kibana 日志分析界面



elastic

Find apps, content, and more. Ex: Discover

Discover

Options

New

Open

Alerts

Share

Inspect

Save

logs\*

cluster\_block\_exception

log.level: error

Search field names

Filter by type 0

Selected fields

@timestamp

log.level

message

Available fields

\_id

\_index

\_score

agent.ephemeral\_id

agent.id

agent.name

agent.type

agent.version

data\_stream.dataset

data\_stream.namespace

data\_stream.type

ecs.version

elastic\_agent.id

elastic\_agent.snapshot

elastic\_agent.version

3,391 hits

Documents

Field statistics

BETA

4,000

2,000

0

17 Jun 15, 2022

18

19

20

21

22

23

24 Jun 16, 2022

01

02

03

04

05

06

07

08

09

10

11

12

13

14

15

16

17

Jun 15, 2022 @ 17:00:00.000 - Jun 16, 2022 @ 17:24:57.284 (interval: Auto - 30 minutes)

Columns

1 field sorted

	@timestamp	log.level	message
<input type="checkbox"/>	Jun 15, 2022 @ 22:43:07.584	error	failed to publish events: 503 Service Unavailable: {"error":{"root_cause":[{"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"}],"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"},"status":503}
<input type="checkbox"/>	Jun 15, 2022 @ 22:43:06.480	error	failed to perform any bulk index operations: 503 Service Unavailable: {"error":{"root_cause":[{"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"}],"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"},"status":503}
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:26.861	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:26.090	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:25.963	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:25.568	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:25.513	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:25.452	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:25.094	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:24.830	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:24.523	error	failed to publish events: temporary bulk send failure
<input type="checkbox"/>	Jun 15, 2022 @ 22:42:24.259	error	failed to publish events: temporary bulk send failure

Rows per page: 100

1

2

3

4

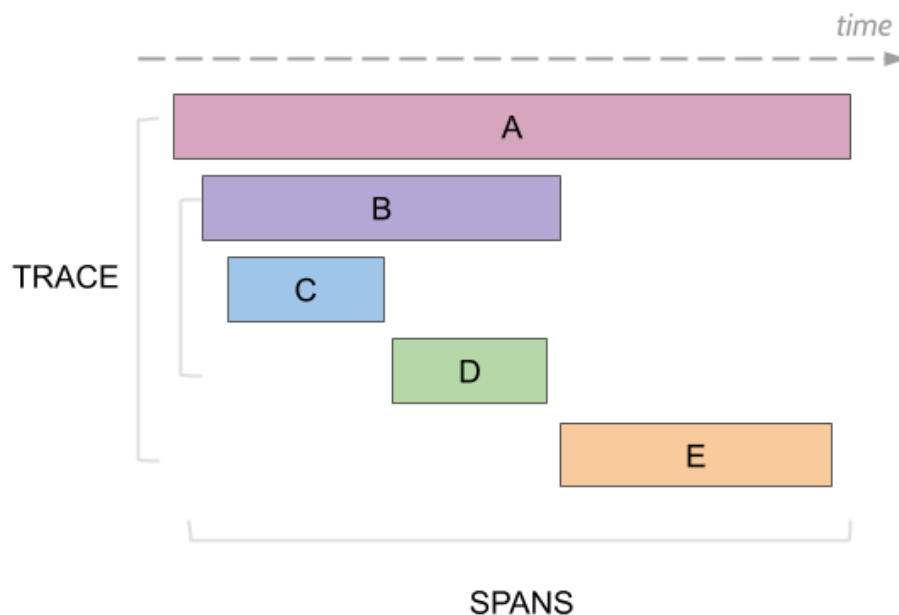
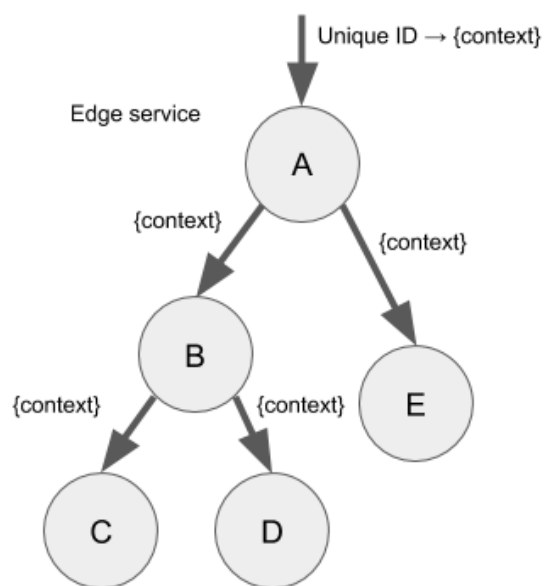
5



# 追踪 ( Trace )



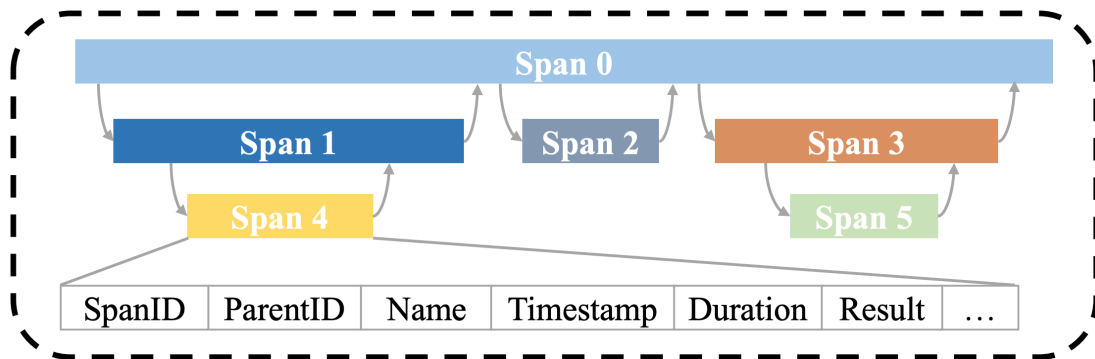
- 跟踪记录**单个请求在系统中的传播路径**。它可以帮助我们理解系统的内部行为，找出性能瓶颈，以及确定故障的来源。
- 例如，一个发送到云服务的请求可能会经过多个服务。追踪可以展现这个请求**在哪些服务之间传递**，每个服务处理**请求花费了多长时间**，以及**是否在某个服务中发生了错误或延迟**。



# Span



- Span 代表了一个**单独的工作单元**，比如一个函数调用或者一个操作。在追踪系统中，一次请求会形成一个 Trace，而 Trace 内部由多个 Span 组成



- 一个 Span 通常包含以下信息：
  - ✓ 一个 Span ID，唯一标识这个 Span
  - ✓ 一个 Trace ID，唯一标识这个请求或事务
  - ✓ 一个操作名，描述了这个 Span 做了什么
  - ✓ 一个开始时间和一个结束时间
  - ✓ 零个或多个引用到其他 Span，形成了一种父子或因果关系
  - ✓ 零个或多个关联的键值对，提供了更多关于这个操作的信息

# 例子



- 假设我们有一个在线购物网站，用户进行了一次购物操作，这将形成一个 Trace，其中包含了多个 Span：
  - 第一个 Span 可能是**用户点击购物车按钮**，操作名可能是“点击购物车”，键值对可能包括用户 ID、购物车内的商品 ID 等信息
  - 第二个 Span 可能是**后端系统检查库存**，操作名可能是“检查库存”，键值对可能包括商品 ID、库存数量等信息
  - 第三个 Span 可能是**付款操作**，操作名可能是“付款”，键值对可能包括付款金额、付款方式等信息
  - 第四个 Span 可能是**更新数据库**，操作名可能是“更新数据库”，键值对可能包括更新的记录 ID、更新的字段等信息

# 追踪如何帮助系统进行问题检测



- **监控**：追踪可以帮助我们监控系统的运行情况，找出性能瓶颈，优化系统的性能
- **故障诊断**：追踪可以帮助我们找出系统的故障源头，找出问题发生的具体位置和原因
- **容量规划**：追踪可以帮助我们了解系统的负载情况，从而进行容量规划
- **服务依赖分析**：追踪可以显示出服务之间的依赖关系，有助于我们理解系统的运行流程。

# 追踪的限制和挑战



- **性能开销**：追踪系统需要在运行时收集大量的数据，这可能会影响到应用的性能
- **数据的管理和存储**：大量的数据需要有效的方法来存储和管理这些数据，同时需要考虑数据的安全和隐私问题
- **数据分析和可视化**：大量的追踪数据需要强大的工具才能进行有效的分析和可视化。而且，即使有了这些工具，理解和解释追踪数据仍然需要一定的专业知识
- **分布式追踪的复杂性**：在分布式系统中，追踪可能会变得非常复杂。跟踪一次请求可能涉及到多个服务，甚至多个数据中心。此外，服务之间可能存在复杂的调用关系，使得追踪更加困难
- ...

# 告警 ( alert ) 和工单 ( ticket )



- 日志 ( log )、指标 ( metric ) 和追踪 ( trace ) 的确可以被看作是系统监控的元数据，也就是描述系统运行状态的数据
- 告警 ( alert ) 和工单 ( ticket ) 则是基于这些元数据产生的系统运维数据，也就是描述系统运行问题和解决过程的数据，提供更多语义信息

# 告警 (Alert)



- 监控系统的健康状态，当系统出现值得关注的问题，能够及时发送警报，使我们能够快速响应和解决问题
- 假设假设我们有一个在线购物网站，我们可以为定义一些警报规则监控系统重要指标，例如，如果 CPU 使用率超过 80%，或者响应时间超过 2 秒，就触发警报

Search by ID, title, or affected resource		Status == Active	Severity == Low, Medium, High	Time == Last month	Add filter
<input type="checkbox"/> Severity ↑↓	Alert title ↑↓		Affected resource ↑↓	Activity start time (UTC+2) ↑↓	MITRE ATT&CK® tactics
<input type="checkbox"/> High	Detected Petya ransomware indicators	Sample alert	Sample-VM	12/15/20, 3:54 PM	Execution
<input type="checkbox"/> High	Detected suspicious file cleanup commands	Sample alert	Sample-VM	12/15/20, 3:54 PM	Defense Evasion
<input type="checkbox"/> High	Digital currency mining container detected	Sample alert	Sample-Kubern...	12/15/20, 3:54 PM	Execution
<input type="checkbox"/> High	Potential SQL Injection	Sample alert	Sample-DB	12/15/20, 3:54 PM	
<input type="checkbox"/> High	Phishing content hosted on Azure Webapps	Sample alert	Sample-App	12/15/20, 3:54 PM	Collection
<input type="checkbox"/> Medium	Suspicious PHP execution detected	Sample alert	Sample-VM	12/15/20, 3:54 PM	Execution
<input type="checkbox"/> Medium	User accessed high volume of Key Vaults	Sample alert	Sample-KV	12/15/20, 3:54 PM	

# 微软 Azure 设置告警规则



## Alert format

Alert ID, Alert type, Alert title, Alert time, Severity, Component, etc.

[Dashboard](#) > [Event Grid Topics](#) > [mytopic0130 | Alerts](#) >

## Create alert rule

Rules management

✓ Whenever the total dead lettered events is greater than 10 count

\$ 0.00

Select condition

Total \$ 0.00

**i** In an alert rule with multiple conditions, you can only select one value per dimension within each condition.

### Action group

Send notifications or invoke actions when the alert rule triggers, by selecting or creating a new action group. [Learn more](#)

Action group name

Contains actions

Email when deadletter count is greater than 10

1 Email Azure Resource Manager Role ⓘ

[Select action group](#)

### Alert rule details

Provide details on your alert rule so that you can identify and manage it later.

Alert rule name \* ⓘ

Alert when deadletter counter goes above 10 ✓

Description

Specify the alert rule description

Severity \* ⓘ

Sev 3 ▼

Enable alert rule upon creation



Create alert rule

- 案例来源：<https://learn.microsoft.com/en-us/azure/event-grid/set-alerts>



# 告警机制的步骤



- **定义警报规则**：定义告警规则检测需要关注的问题，例如指标过高、日志出现特殊关键字、追踪路径错误
- **收集数据**：不断地收集和监控日志（log）、指标（metric）和追踪（trace）等系统运维数据
- **触发警报**：当收集到的数据满足我们定义的规则时，我们的系统会自动触发警报。这个警报可以通过各种方式发送，例如电子邮件、短信、手机应用推送等
- **处理警报**：收到警报后，我们需要立即采取行动解决问题。这可能涉及到查看详细的日志、指标和追踪信息，找出问题的源头，然后进行修复

# 工单 ( Ticket )



- 用来追踪用户的问题请求或者服务请求的记录，能够追踪问题解决的全过程

**Incident ID**  
**Resolved**  
**Critical**

## *Disk firmware update disabled disk cache*

Service: Storage	# of impacted requests: ~100,000
Datacenter: DC #4	# of impacted accounts: ~10,000

### Summary

Writing to a big data storage platform experienced high failure counts.

### Diagnosis

Firmware upgrade to a game drive service inadvertently disabled write cache. At the beginning, there was no direct impact on the service because the number of machines getting into bad state was small and the system was built to tolerate such instances. However, as more and more machines were getting upgraded, the overall latency of the service stack was slowly accumulating and at some point got tipped. It took quite some time to detect the incident which unfortunately deteriorated into a critical issue.

# 工单系统的主要流程



- **创建工单**：当用户或者系统遇到问题时，会创建一个工单。例如，如果一个用户发现网站登录功能出现问题，他可以提交一个工单来报告这个问题（[微软提交工单案例](#)）
- **分配工单**：系统会把工单分配给合适的团队或个人进行处理。例如，如果问题是关于数据库的，工单可能会被分配给数据库管理团队
- **处理工单**：团队或个人开始处理工单，寻找问题的原因，解决问题，然后更新工单状态。例如，数据库管理团队可能需要检查数据库日志，找出问题的原因
- **关闭工单**：当问题被解决，工单将被关闭，同时会有详细的解决报告，用来记录问题的解决过程和结果

# 工单系统的作用



- **问题追踪和管理**：工单系统能够帮助追踪和管理系统问题，使得运维团队可以快速理解和定位问题，防止问题被忽视或遗忘
- **提高效率**：工单系统可以自动分配工单，这不仅提高了处理问题的效率，还能确保问题被正确地解决
- **故障库**：每一个工单都是一个问题的记录，它记录了问题的发生、处理和解决过程。这些工单可以作为故障库，帮助我们积累故障处理的经验和知识
- **提供学习资源**：工单系统中的问题解决方案可以作为学习资源，帮助运维团队学习和提高
- **改进运维策略**：通过分析工单系统中的数据，我们可以发现系统的弱点和问题，改进运维策略



中山大學

SUN YAT-SEN UNIVERSITY

软件工程学院

SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬

软件工程学院

<https://zbchern.github.io/sse316.html>