



分层次构建应用系统的可观测性

刘征

2019-9, 社区布道师, elastic, @martinliu

关于我

刘征, elastic 社区布道师



Ops背景

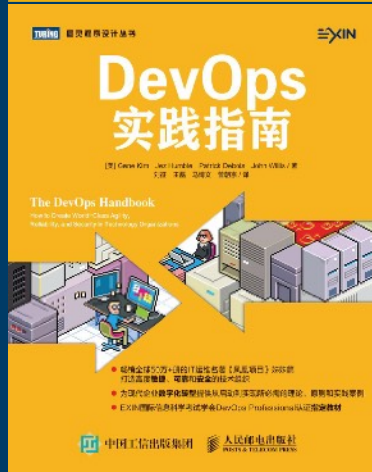
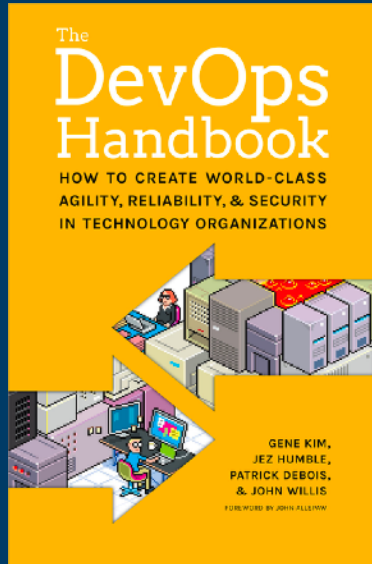
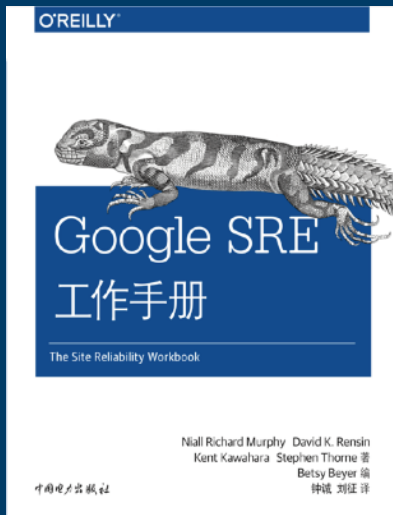
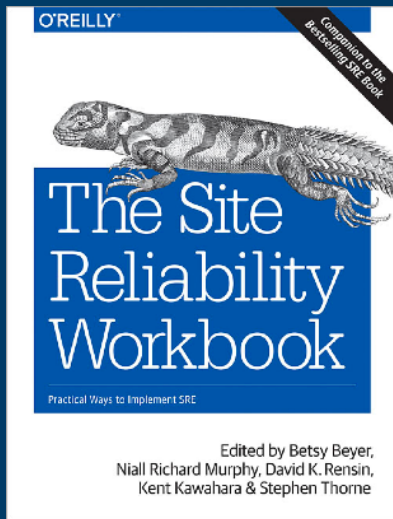
云计算

工具链

中国DevOps社区组织者

译者:

- DevOps Handbook
- The Reliability Workbook 【翻译中】



议程

破解云原生应用的可观测性

- 1 云原生应用的主要特征，它在系统可运维方面的挑战
- 2 可观测性的定义和本质，这是新生事物还是老生常谈
- 3 分层次的构建云原生应用系统的可观测性，用四步法破解这个难题
- 4 构建可观测性的难点和挑战，准备的越充分，就能走的越远
- 5 演示，Q&A



Baron SchSchwarz
CTO of VividCortex



监控告诉我们系统的那些部分是工作的。可观测性告诉我们那里为何不工作了。

云原生应用的三个特点

应用架构

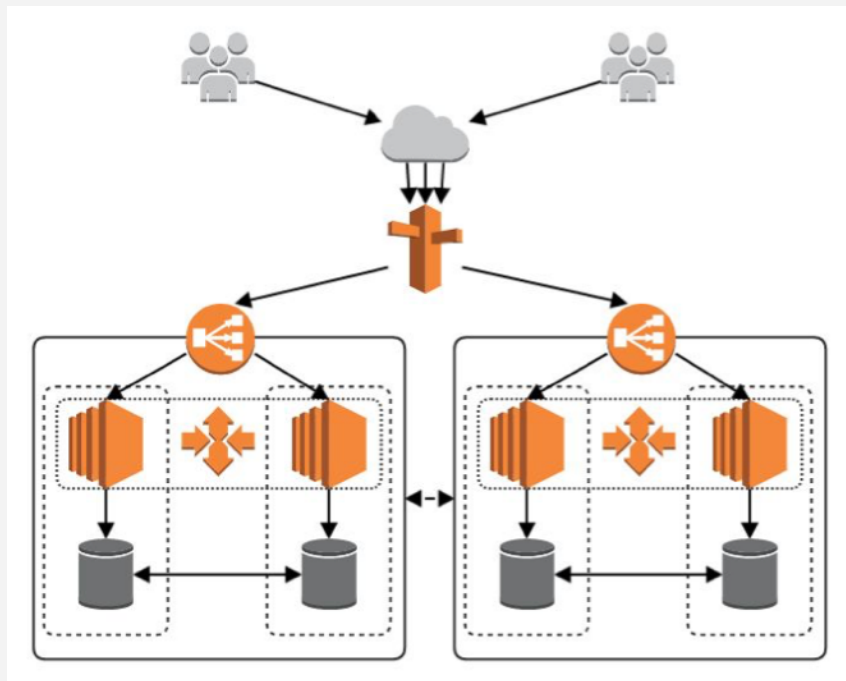
- 从单体应用向微服务过渡
- 应用架构过渡为松耦合系统
- 应用版本迭代更快、周期更短

基础设施

- 容器化、应用自身快、轻、微
- Kubernetes成为默认的平台
- IaaS、PaaS和CaaS平台底层

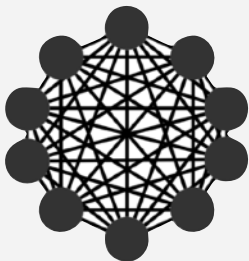
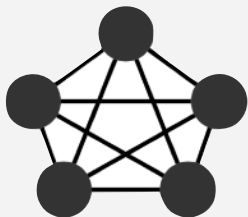
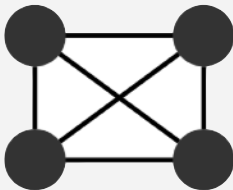
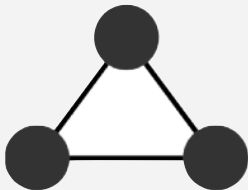
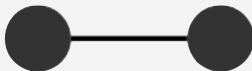
生命周期

- DevOps流水线持续部署
- 服务变更低成本和低风险
- 呈现高频率和全自动变更



Monolith at scale
互联网应用
基于延迟的路由
多区部署
负载均衡
多应用副本
自动扩容
数据库读写分离副本

服务间通讯



Micro service at Scale



Wheel of Doom

This diagram, taken from late 2014, illustrates the interactions between services running on Hailo's platform.

微服务应用和传统单体的对比

在重要运维事件方面的差异

运维部署的复杂度	扩展性	回滚/故障隔离	服务期限承诺
必须借助自动化配置管理工具和Docker	速度更快、按需扩展	更快、更方便	服务周期短
更高的复杂度	更高的资源利用率	实现更好的故障隔离	船小好调头
正常水平	复制整个虚拟机或技术栈 速度慢	速度慢、难度大 成本高、风险高	周期长、不靠谱 没有回头路、无后悔药

应用部署到生产
环境只是刚刚开
始。



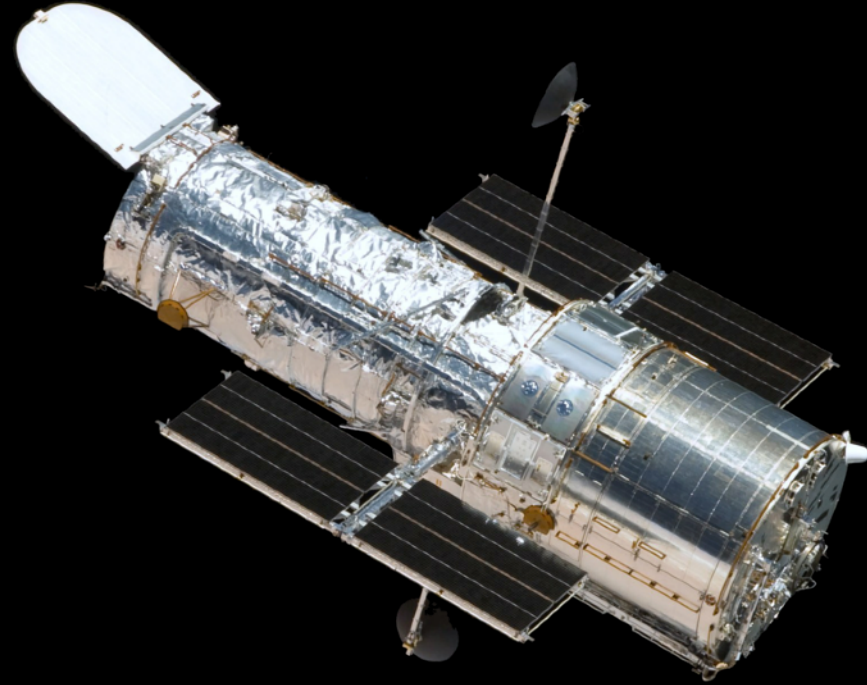
天下没有不宕机的系统，时刻做好恢复系统的准备。



如果系统是分布式的，那么它的故障也可能在哪里都存在。



Rethink Service Monitoring



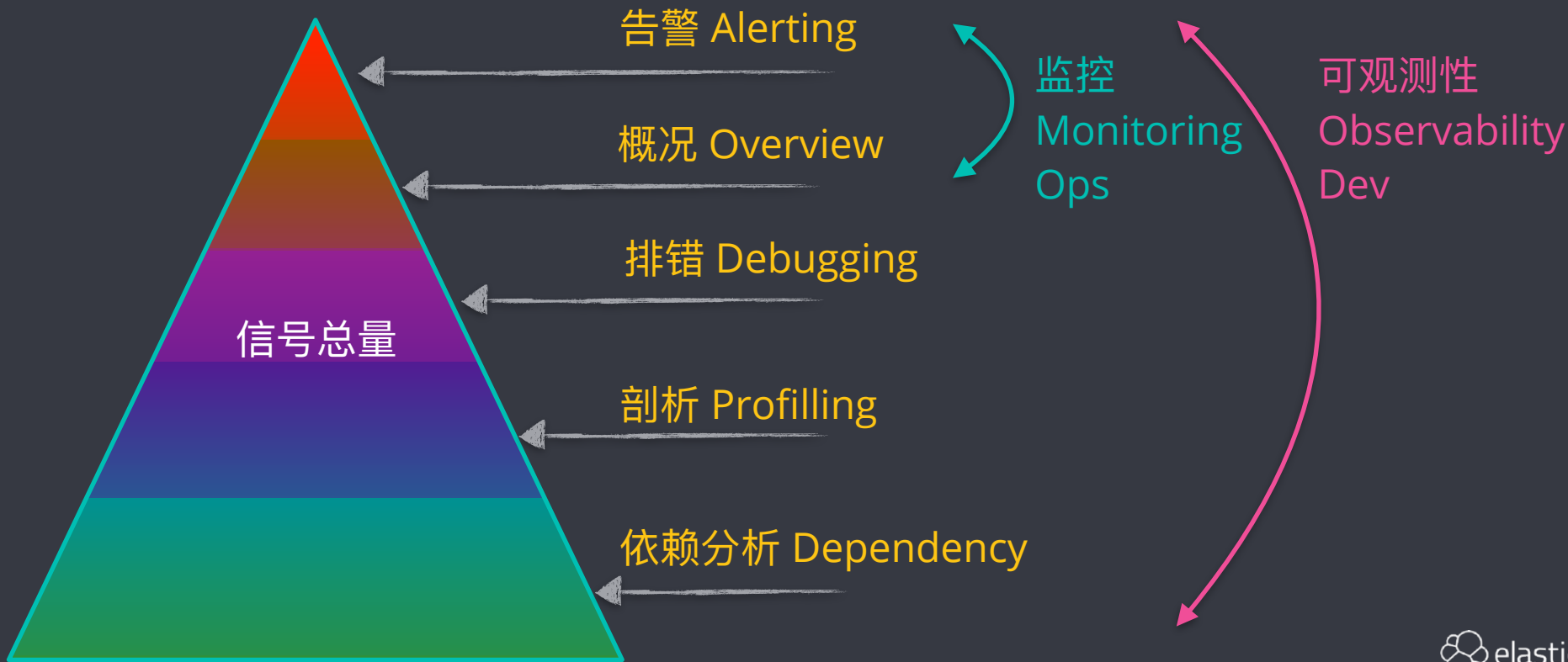
议程

破解云原生应用的可观测性

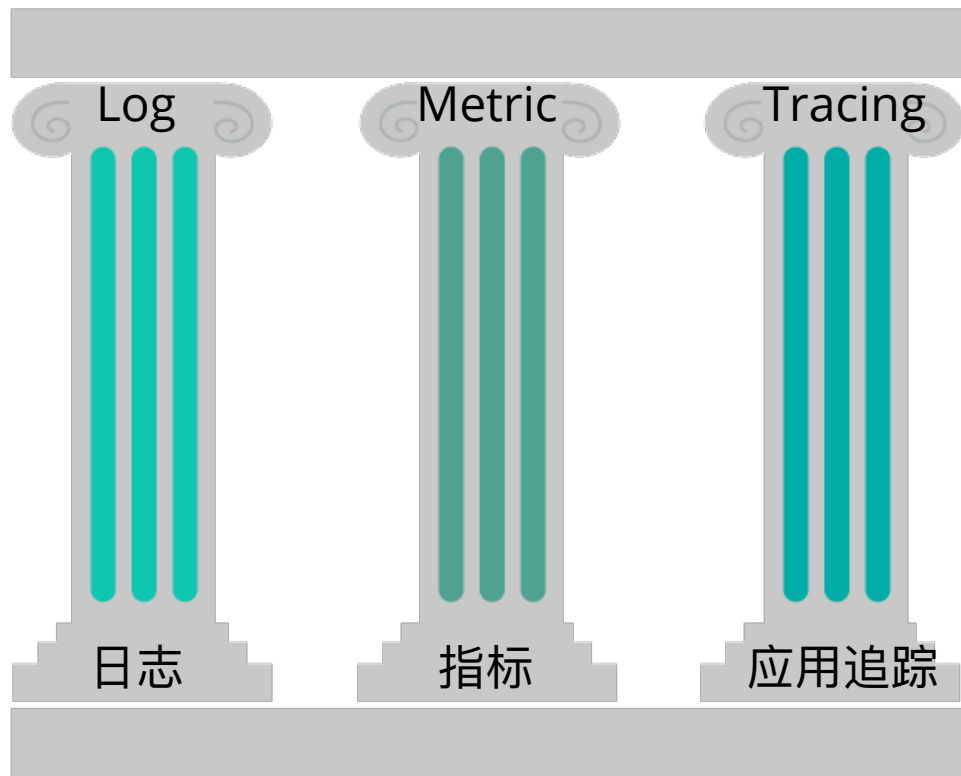
- 1 云原生应用的主要特征，它在服务监控方面的挑战
- 2 可观测性是新生事物还是老生常谈，它的真实本质
- 3 分层次的构建云原生应用系统的可观测性，用四步法破解这个难题
- 4 构建可观测性的难点和挑战，准备的越充分，就能走的越远
- 5 演示，Q&A

监控和可观测性之间的关系

从上往下是逐层递进的现象和原因



可观测性的三根支柱



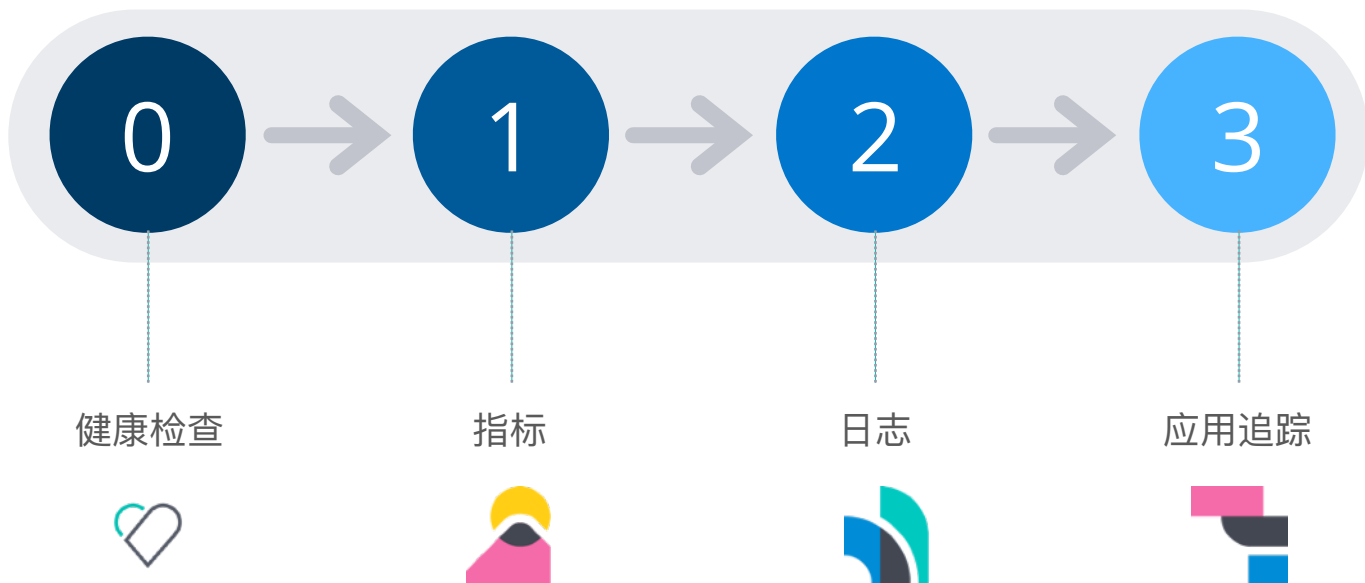
议程

破解云原生应用的可观测性

- 1 云原生应用的主要特征，它在系统可运维方面的挑战
- 2 可观测性的定义和本质，这是新生事物还是老生常谈
- 3 分层构建云原生应用系统的可观测性，四步法破解难题
- 4 构建可观测性的难点和挑战，准备的越充分，就能走的越远
- 5 演示，Q&A

分层级构建可观测性

适用于云原生应用的可观测性建设四步法



提供服务运行状态的红绿灯系统

♥ Level 0 : 健康检查

是否在运行中?



能处理更多工作量么?

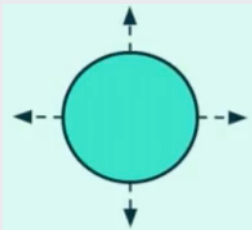
能处理它的工作么?

实施健康检查的三种模式

Level 0 : 健康检查

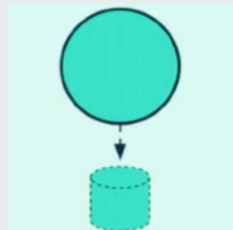
广播

- 将状态信息封包
- 发送到网络上
- 更新自己和邻居的状态



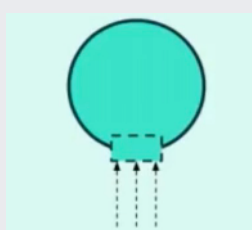
注册表

- 服务的ip/域名的端口信息
- 写入/更新到etcd或Zk
- 最新服务清单



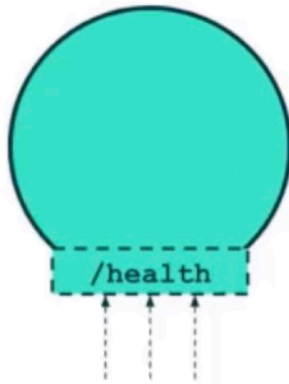
暴露

- 利用服务默认的服务端口
- 编写应用特有的健康服务
- 等待外部工具的轮询



实施向外暴露健康状态的接口

Level 0 : 健康检查



```
GET http://1.2.3.4:8080/health
```

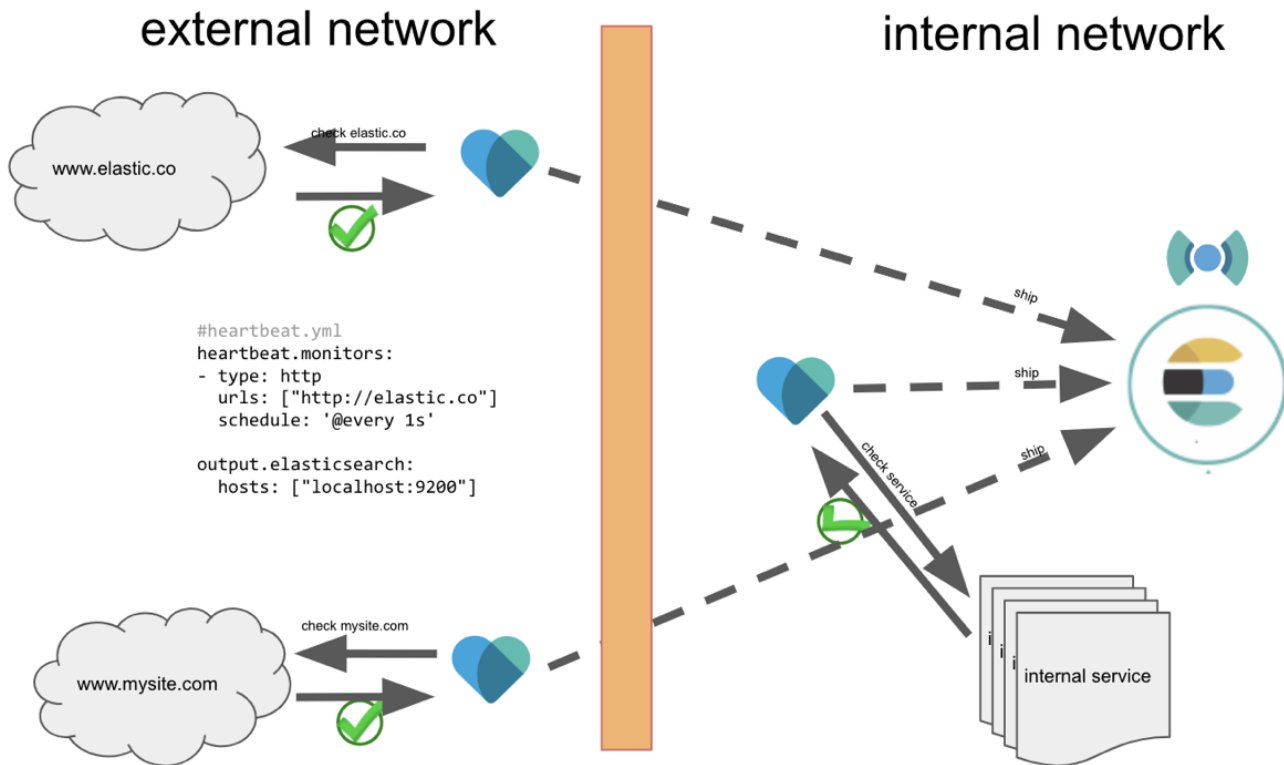
```
200 OK
```

```
{
  "service": "registration-service",
  "healthy": true,
  "workload": { "healthy": true },
  "dependencies": [
    { "name": "cassandra", "healthy": true },
    { "name": "billing-svc", "healthy": true }
  ]
}
```

用Heatbeat实施健康检查的系统架构

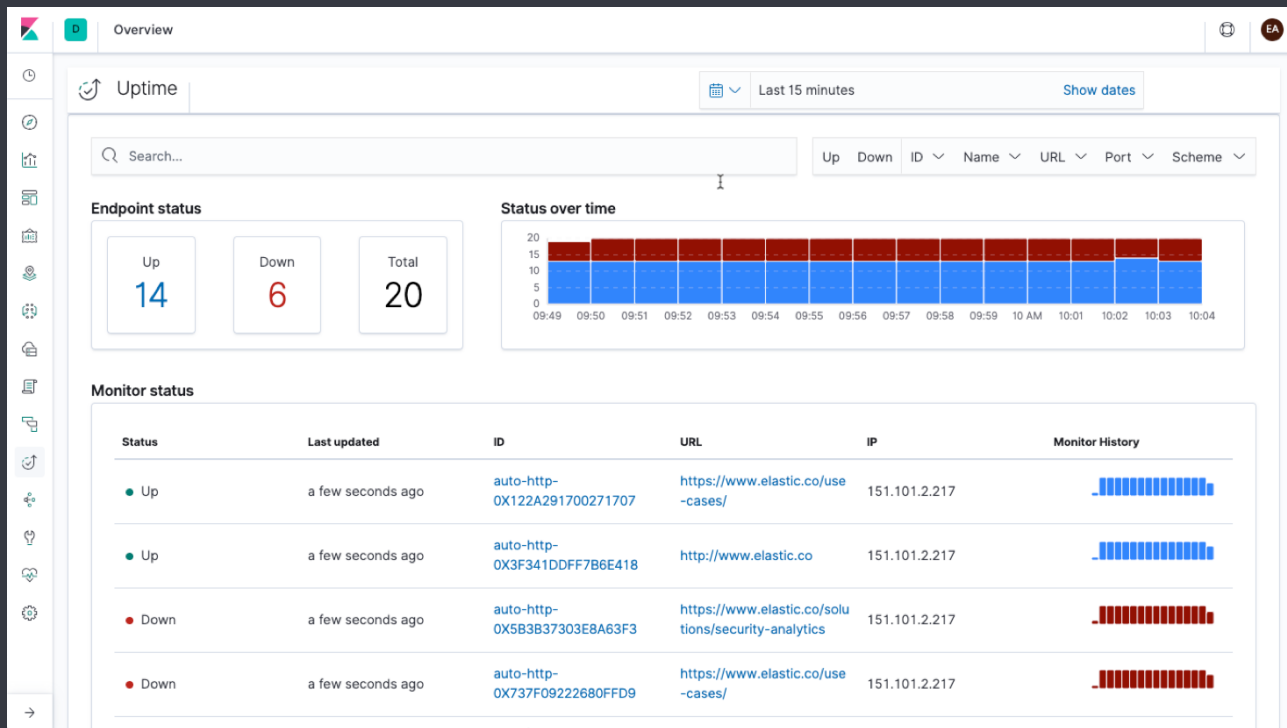
Level 0 : 健康检查

- 1 部署Heatbeat采集点
- 2 配置也运行每一个采集点
- 3 加载集中分析仪表盘
- 4 定义告警规则



Elastic Heatbeat监控服务健康检查的效果

Level 0 : 健康检查



可能的问题和挑战

- 定义何谓健康？
- 扫描互相依赖的复杂系统并不一定能识别出问题？
- 过度的检查导致资源浪费，甚至DDos自己！

指标是在许多个连续的时间周期里度量的KPI数值

Level 1: 指标

07/Jan/2019 16:10:00	all	2.58	0.00	0.70	1.12	0.05	95.55	server1	containerX	regionA
07/Jan/2019 16:20:00	all	2.56	0.00	0.69	1.05	0.04	95.66	server2	containerY	regionB
07/Jan/2019 16:30:00	all	2.64	0.00	0.65	1.15	0.05	95.50	server2	containerZ	regionC

在每10分钟里，度量CPU load指标，显示这个数值，并附加相关的元数

监控指标的三种来源/类型

Level 1: 指标



系统指标

- CPU使用率
- 磁盘使用率
- 网络带宽

应用指标

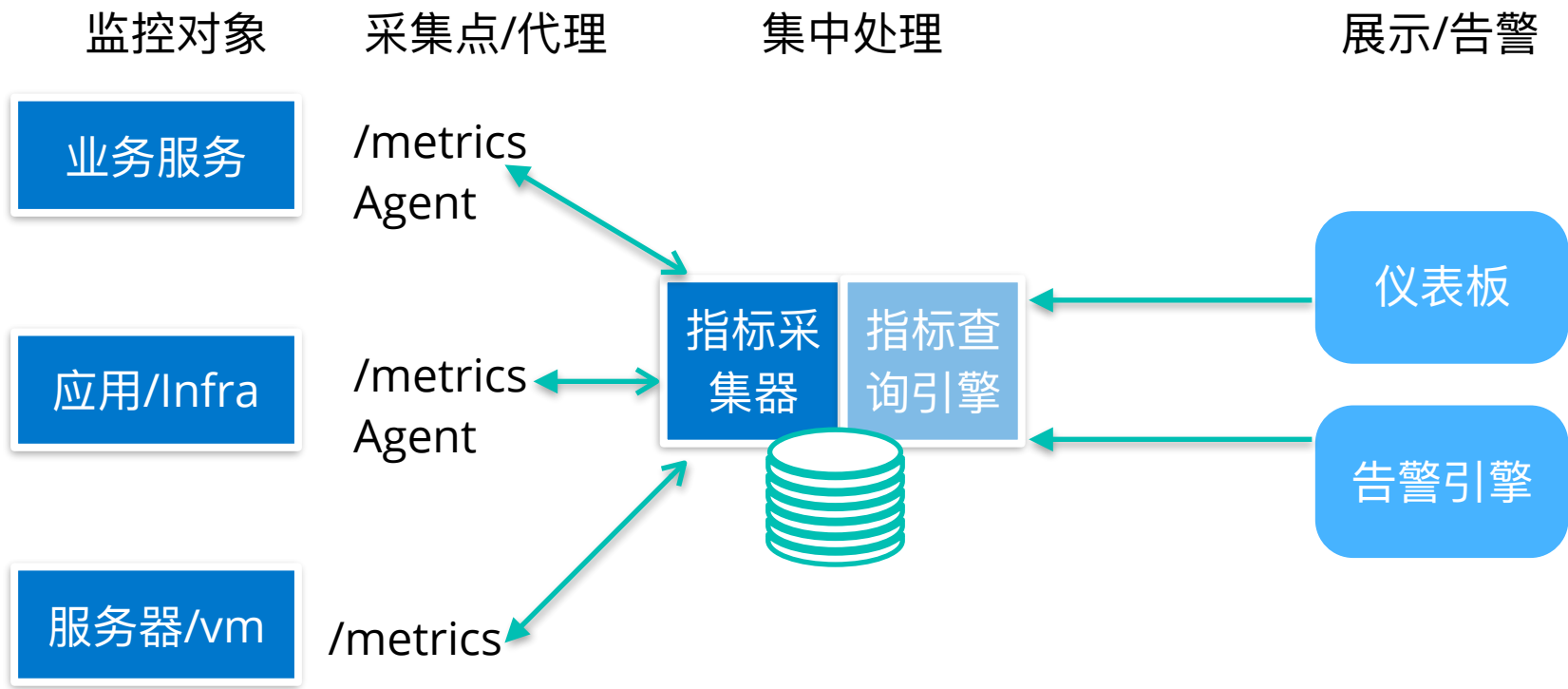
- 出错率
- 延迟
- 饱和度

业务指标

- 用户会话
- 订单数量
- 营业额

实施指标监控的部署模式

Level 1: 指标



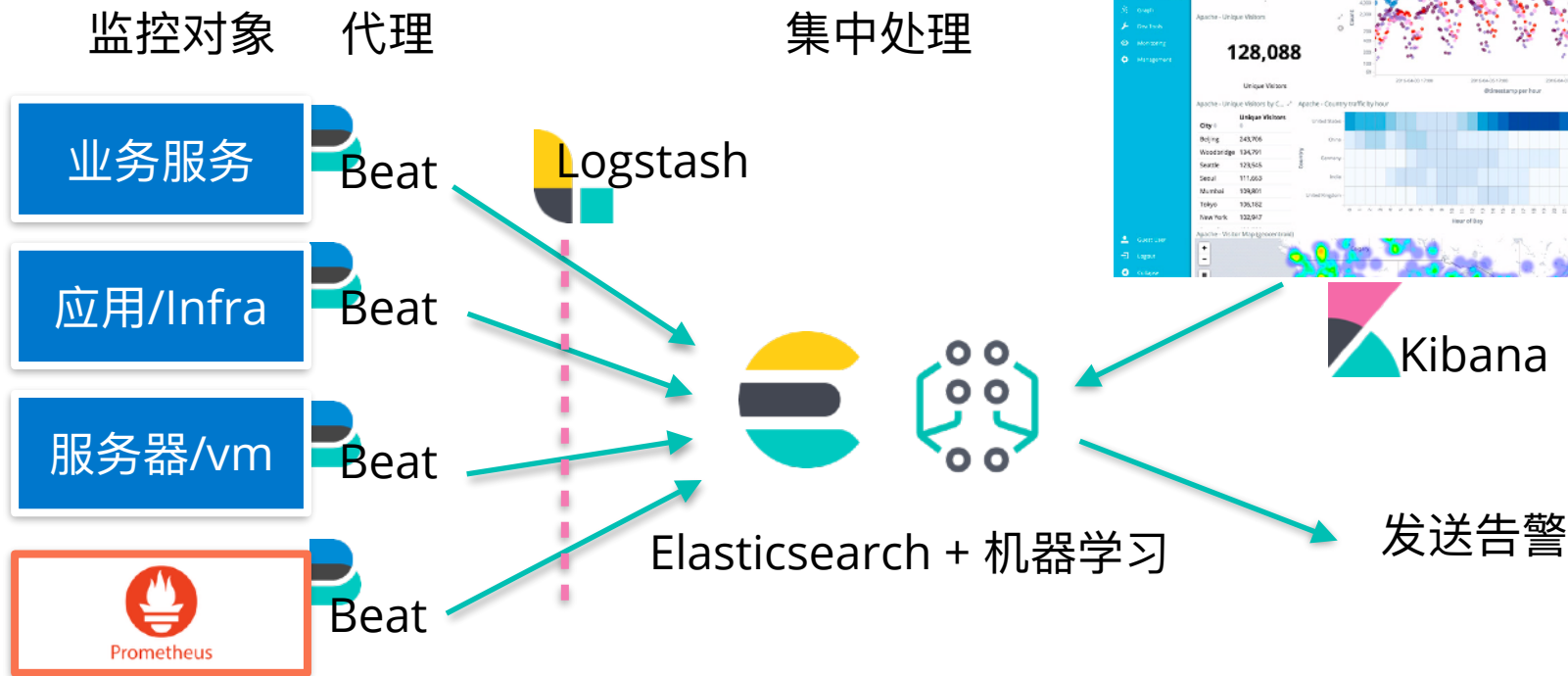
谨防对指标的过度依赖，它的挑战和不足

Level 1: 指标

现实的情况	局限性和挑战
不是每一个指标都值得告警	缺乏针对影响用户端征兆的告警
每个信息点的维度有限	对细粒度排错帮助有限，例如：缺少CustomerID
实时查询意味着数据无法长期保留	实时的短期数据查询和长期的趋势分析之间存在矛盾
传统的静态阈值管理正在失效	业务高峰期的最低阈值大于平时最高值， 动态基线和可能性预测功能匮乏
数据的聚合、统计、分析和展示花费大量成本	需要耗费一定开发人力成本， 运算消耗大量时间和存储空间成本

一种优化的指标监控部署模式

Level 1: 指标



破局指标监控窘境的可能性

Level 1 : 指标

局限性和挑战	需要的能力和工具
缺乏针对影响用户端征兆的告警	参考SRE的方法，定义基于SLO的告警 梳理有价值的指标与SLI的对应关系
对细粒度排错帮助有限，例如：缺少CustomerID	在指标数据生成的地方/监控代理上，或者指标发送到集中存储的中转站里丰富元数据，beats和logstash可方便的增加必要的键值型指标元数据
实时的短期数据查询和长期的趋势分析之间存在矛盾	将冷热和密度不同的指标数据放置在不同存储集群上进行统一查询和数据管理，elasticsearch可以设计实施冷热数据架构，支持多集群查询
业务高峰期的最低阈值大于平时最高值，动态基线和可能性预测功能匮乏	对关联的监控指标进行动态基线管理，Elastic Stack的机器学习功能可以在无督导的模式下进行指标的动态阈值运算，并做异常行为检测和告警
分析能力需要耗费一定开发人力成本，运算消耗大量时间和存储空间成本	中央数据存储内置数据的聚合、分析和统计等功能，elasticsearch在索引存储了指标数据后就能提供以上分析能力，数据上卷功能可节省大量空间

日志是那些长时间累积的事件记录

Level 2 : 日志

```
64.242.88.10 - - [07/Jan/2019:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1" 200 6291
```

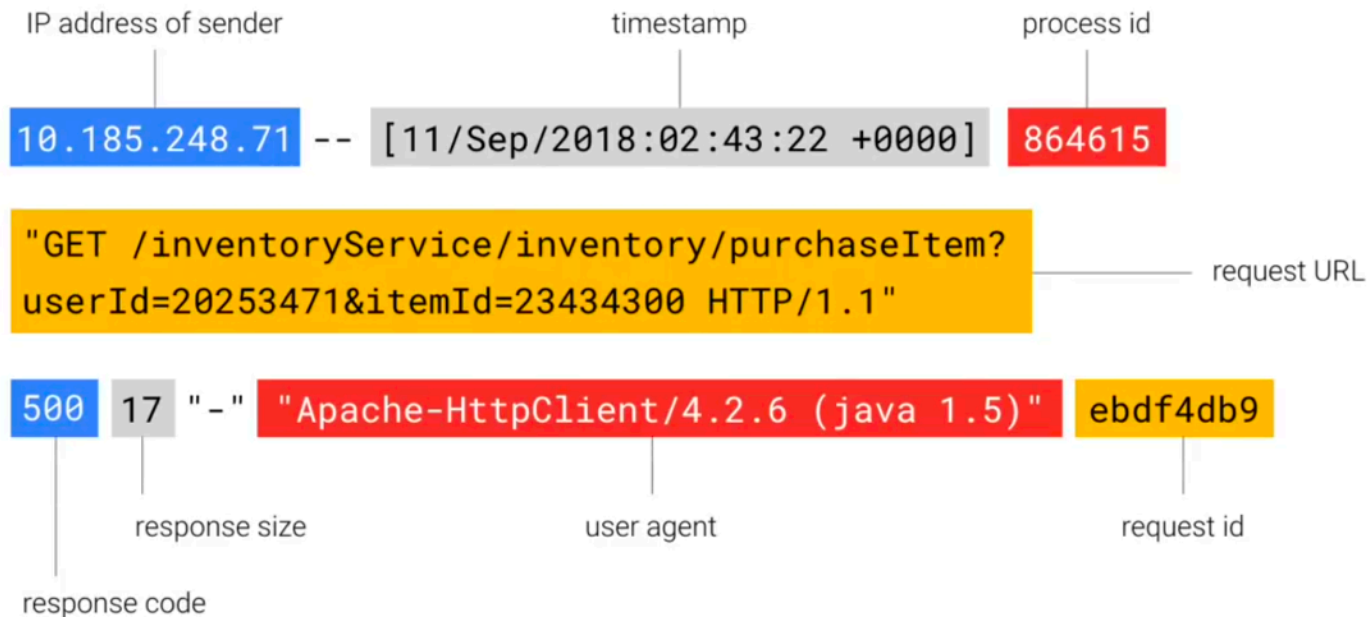
```
64.242.88.10 - - [07/Jan/2019:16:11:58 -0800] "POST /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1" 404 7352
```

```
64.242.88.10 - - [07/Jan/2019:16:20:55 -0800] "GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1" 200 5253
```

每一条事件记录，都证明那里曾经发生过什么事

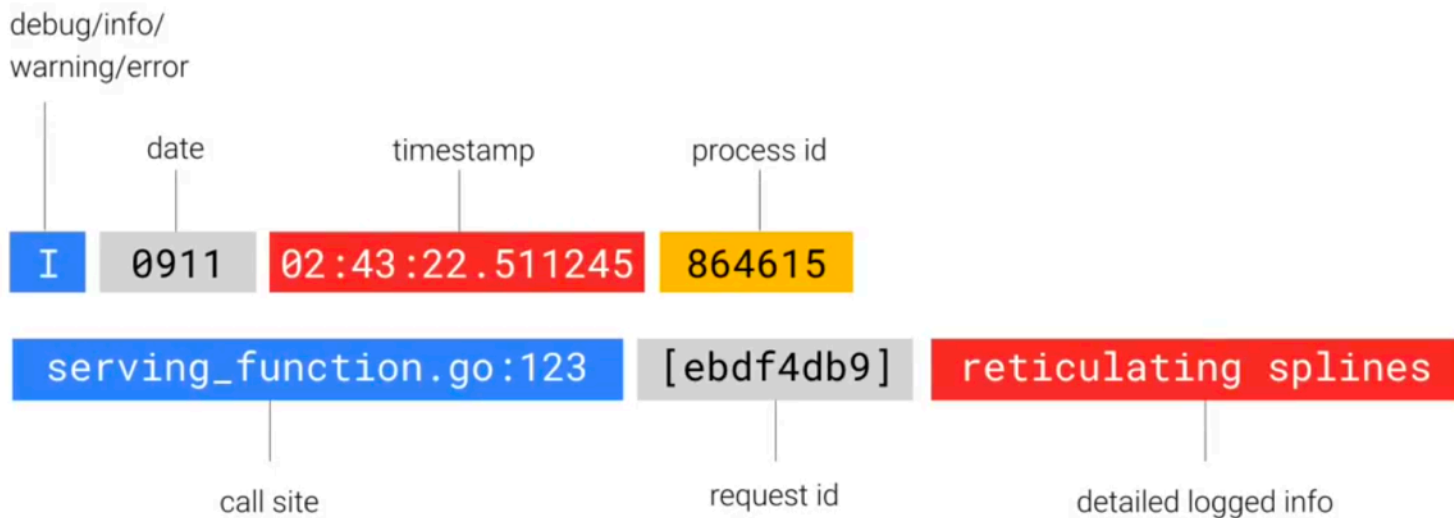
日志类型-请求日志

Level 2: 日志



日志类型-排错日志

Level 2: 日志



打下管理大数据量日志的三个前提条件

Level 2 : 日志



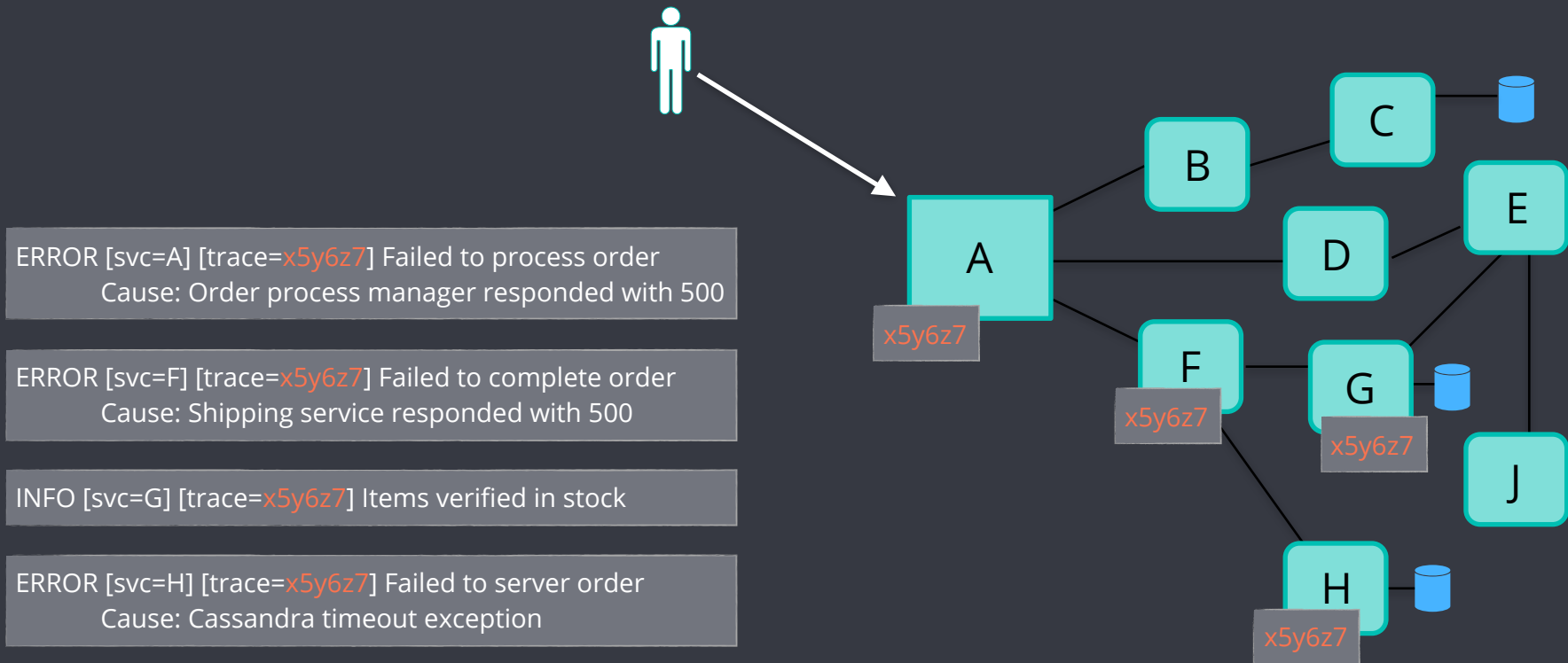
可集中化
Centralised

可全文搜索
Searchable

可关联
Correlated

构建可进行关联分析的日志信息流

Level 2 : 日志



构造高维度结构化日志，释放排错日志的无限潜能

Level 2 : 日志

2018-02-20T16:38:23+00:00 ERROR Read time out

谢谢啊~~ 真的是比没有强👍

什么时候发生？

Timestamp 2018-02-20T16:38:23+00:00

日志信息是什么？

Level Error Log Read time out

是什么服务？

Service registration-service team Beijing-T1

运行的是什么程序？

commit 5938deumw4 Build 5938deumw4 Runtime java-1.8.0_160

程序运行在哪里？

Region ap-northeast-1 Node Reg-host012

谁受到了影响？

customerID 87762 userID 87762

追踪标识是什么？

traceID x5y6z7

JSON

应用的响应速度缓慢或者加载速度异常?

Level 3: 追踪

03:43:45 Request "GET cyclops.ESProductDetailView"

03:43:57 Response "cyclops.ESProductDetailView 200 OK"

12秒 zzzzZZZZzzzz

应用服务调用报错?

Level 3: 追踪

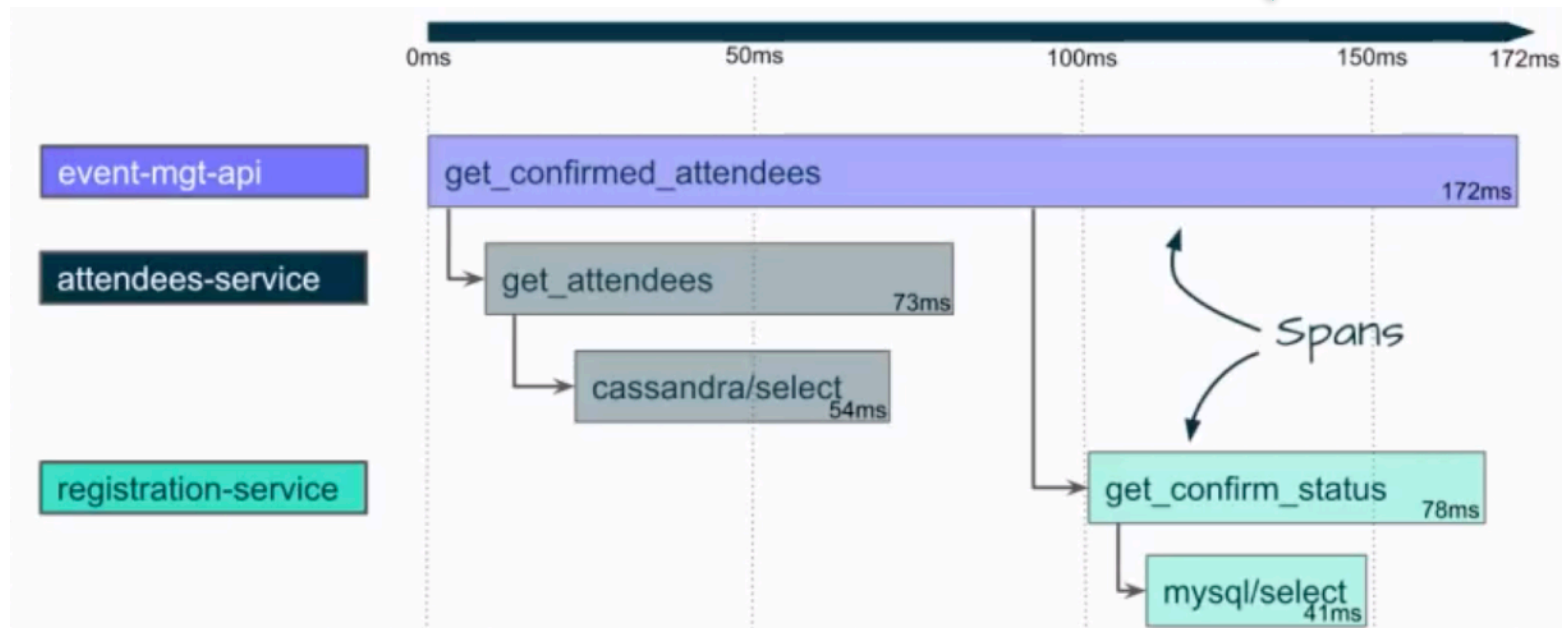
03:43:59 Request "POST /api/checkout"

03:43:59 Response "/api/checkout 500 ERROR"

多服务的应用全调用追踪

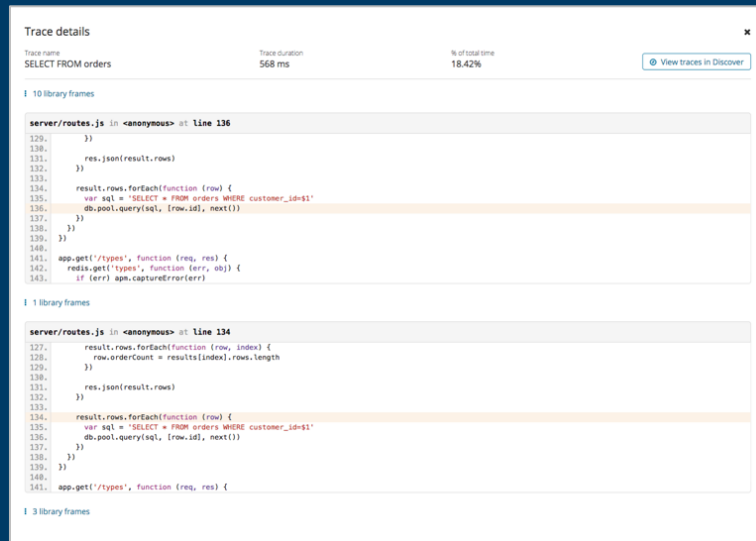
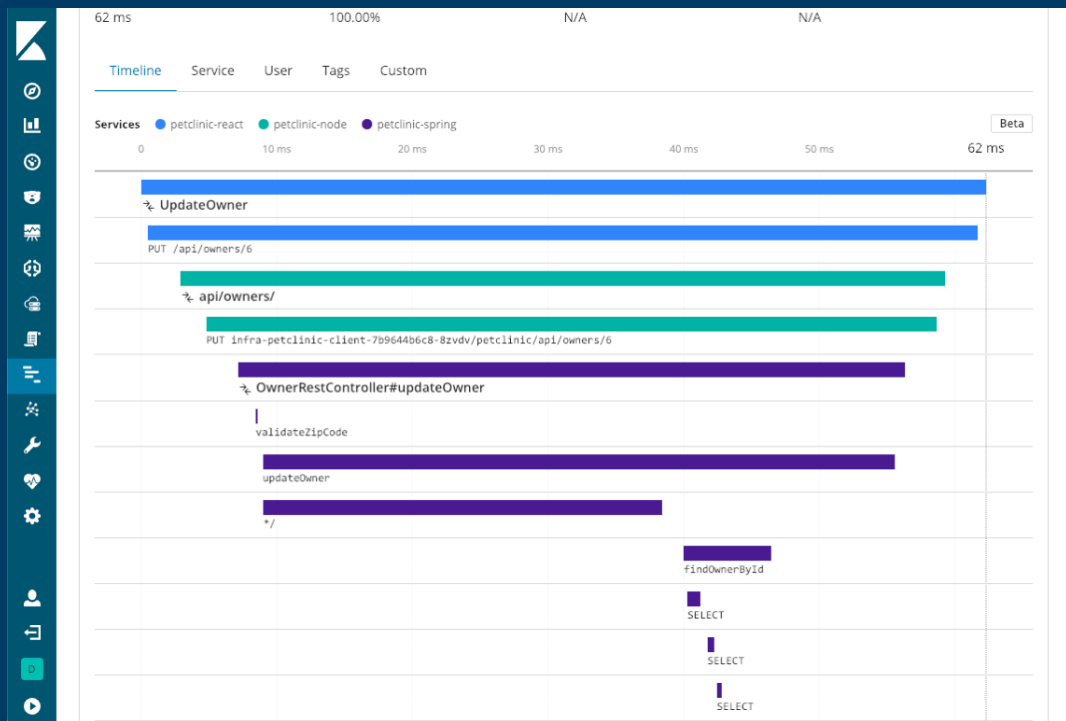
Level 3: 追踪

Trace-追踪



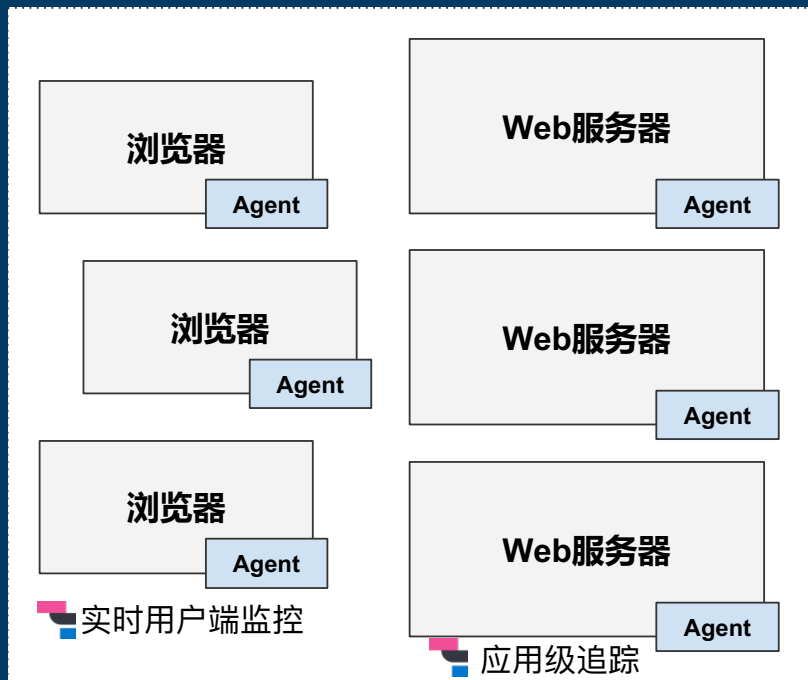
Elastic APM低成本的在框架和代码级别深入洞察调用链

Level 3: 追踪



实施应用全链路追踪系统的架构

Level 3: 追踪



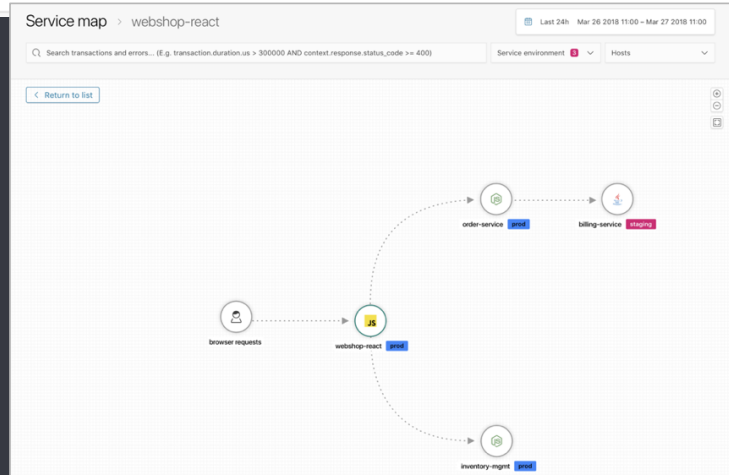
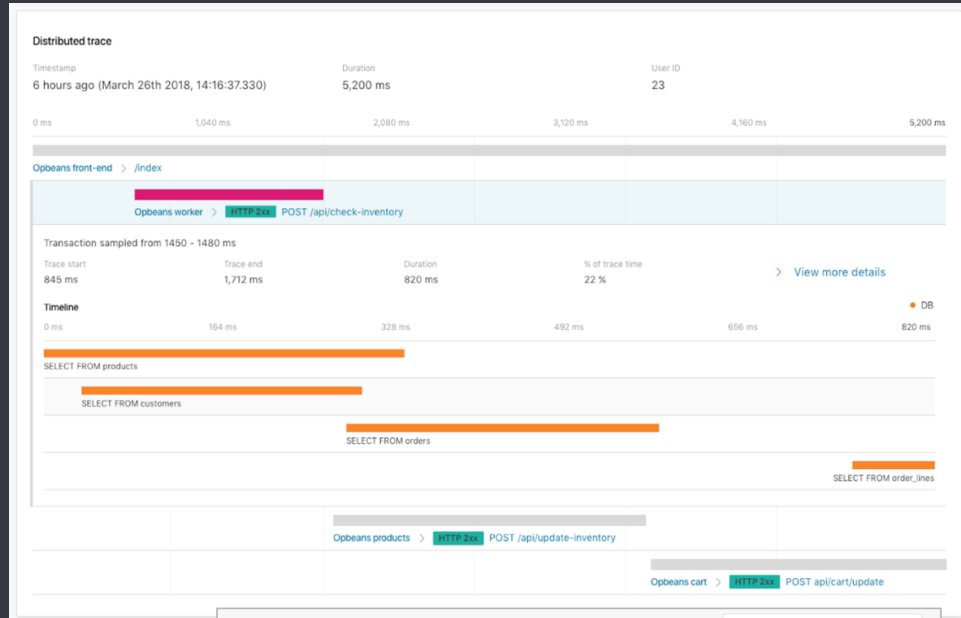
跨多服务的分布式追踪和关联映射

Level 3: 追踪

- 查看端到端调用链路视图和单独的某个交易
- 基于特定的端到端的多服务追踪标识
- 参考OpenTracing API, 并与之兼容, 而且与W3C 追踪上下文规范保持一致
- 关注追踪数据量

application transaction rate
x all microservices
x cost of network and storage
x weeks of data retention

way, way too much \$\$\$\$



议程

破解云原生应用的可观测性

- 1 云原生应用的主要特征，它在系统可运维方面的挑战
- 2 可观测性的定义和本质，这是新生事物还是老生常谈
- 3 分层构建云原生应用系统的可观测性，四步法破解难题
- 4 构建可观测性的难点和挑战，准备的越充分，就能走的越远
- 5 演示，Q&A

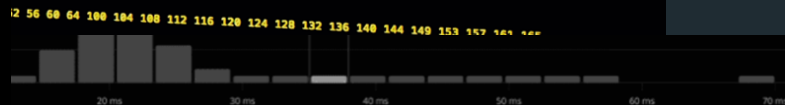
L2: 日志

491 total, 3 running, 488 sleeping, 3188 threads
2.87, 2.54, 2.28 CPU usage: 8.85% user, 6.89% sys, 84.25% idle
: 284M resident, 47M data, 17M linkedit.
: 280767 total, 4854M resident, 118M private, 1997M shared.
5G used (3187M wired), 1288M unused.
vsize, 1111M framework vsize, 341912008(16284) swapins, 348809712(0) swapouts.
packets: 36759414/22G in, 19823346/12G out. Disks: 87127310/2211G read, 32948289/2

```
02:15:09 Morgans-MacBook-Pro kernel[0]: AppleThunderboltNHIType2::waitForOK2G5x - retries = 9
02:15:09 Morgans-MacBook-Pro kernel[0]: RTC: Maintenance 2016/5/6 07:15:09, sleep 2016/5/6 05:15:14
02:15:09 Morgans-MacBook-Pro kernel[0]: AppleCmIn::systemWakeCall - messageType = 0xE0000340
02:15:09 Morgans-MacBook-Pro kernel[0]: Previous sleep cause: -113
02:15:09 Morgans-MacBook-Pro kernel[0]: in6_unlink_ifa: IPv6 address 0xcfa69419aec2bd09 has no prefix
02:15:09 Morgans-MacBook-Pro syslogd[45]: ASL Sender Statistics
02:15:09 Morgans-MacBook-Pro com.apple.CDScheduler[40]: SysAdmChk: Throttling activity execution. memFlags:0x0 thermalLevel:0x1
02:15:09 Morgans-MacBook-Pro kernel[0]: IOTThunderboltSwitch->(0x0)::...
02:15:09 Morgans-MacBook-Pro kernel[0]: TBT W (2): 0x...
02:15:09 Morgans-MacBook-Pro kernel[0]: ...
jack - Thunderbolt HPD packet for route = 0x0 port = 11 unplug = 0
mplete - took 137 milliseconds
```

```
tive
Proxy-
sived Status Packet, Payload 2: device was reinitialized
using process name locationd as bundle ID (this is expected for daemons without bundle ID
using process name apsd as bundle ID (this is expected for daemons without bundle ID
DomainNSURLErrorDomain Code=-1009 "The Internet connection appears to be offline." UserInfo={NSUnderlyingError=
using process name locationd as bundle ID (this is expected for daemons without bundle ID
using process name apsd as bundle ID (this is expected for daemons without bundle ID
oad device bootloaded
using process name apsd as bundle ID (this is expected for daemons without bundle ID
```

```
DomainNSURLErrorDomain Code=-1009 "The Internet connection appears to be offline." UserInfo={NSUnderlyingError=
Min Code=-1009 "The Internet connection appears to be offline." UserInfo={NSUnderlyingError=
L operation mode from AUTO to SUSPENDED
L, enterQuietMode(true)
tries, 0
```



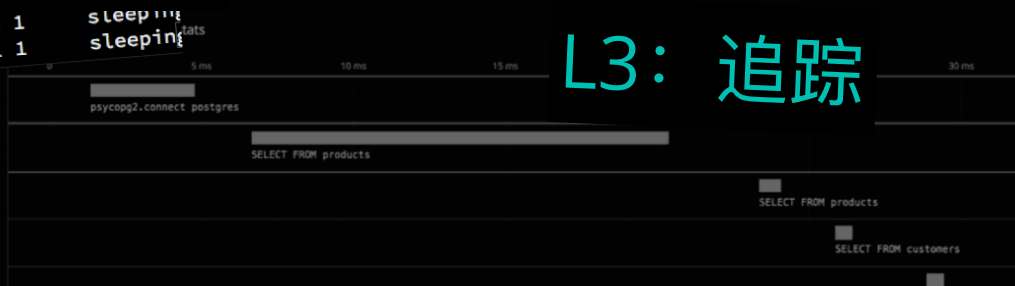
L1: 指标

	%CPU	TIME	#TH	#WQ	#PORTS	MEM	PURG	CMPRS	OCRP	PPID	STATE
MAND	7.7	00:05.46	1/1	0	23	8488K	0B	0B			
h	0.0	00:00.32	1	0	19	6764K	0B	0B			
in	0.0	00:00.04	2	1	29	2152K	0B	0B			
orker					54	3564K	0B	0B			
orker					54	3548K	0B	0B			
cklook					85	4760K	32K	0B			
ebEngi					95	14M	0B	0B			
orker					65	5576K	0B	0B			
orker	0.0	00:00.16	3	1	57	5500K	0B	0B			
eam Deck	0.0	00:02.15	35	1	649-	46M-	1476K	52921			
pd	0.0	00:00.02	2	1	40	128K	0B	1104K			
orker	0.0	00:00.06	3	1	54	64K	0B	3536K			
orker	0.0	00:00.07	3	1	51	60K	0B	3264K			
orker	0.0	00:00.07	3	2	59+	2916K+	0B	1460K			
.apple.ic	0.0	00:00.23	3	1	51	60K	0B	3256K			
orker	0.0	00:00.06	3	1	51	60K	0B	3188K	73302 1		sleeping
orker	0.0	00:00.07	3	1	51	60K	0B	3304K	73301 1		sleeping
orker	0.0	00:00.08	3	1	51	60K	0B				



可观测性

L3: 追踪





知道
知道的

不知道
不知道的

健康检查

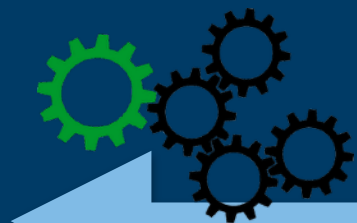
指标
{告警}

指标
{查询}

追踪

日志

事件



监控 & 可靠性

排错 & 探索



工具的可用性是实施可观测性的关键

成本低的 追踪埋点

- 并不知道那些流行的APM工具
- 先从框架上追踪监听的门槛低
- 追踪结果往往意外而有意义

容易浏览 查询分析

- 直观的内置统计分析仪表盘
- 在集中的日志、指标和APM之间关联和跳转
- 支持较复杂运维分析场景的查询的定制，无代码开发

可靠性 值得信任

- 集中数据存储集群具备高可用性，高速进行大数据量查询
- 能应对数据规模的线性增长
- 满足所有团队的集中式访问



THANK YOU

