

# 饿了么监控体系的演进



# 目录

- 背景
- 遇到的问题
- 场景化
- 系统设计

# 背景



1. Statsd/Graphite/Grafana
2. ETrace
3. Zabbix
4. ELog

1. Statsd/Graphite/Grafana
2. ETrace/LinDB
3. ESM/InfluxDB/Grafana
4. ELK

1. EMonitor/LinDB
2. SLS



# 现状

1. 覆盖了饿了么所有的监控（业务监控，全链路监控，PaaS，IaaS等）
2. 覆盖所有应用及服务器
3. 每天采集原始数据 800T
4. 高峰计算事件 7000W/s



# 目录

- 背景
- 遇到的问题
- 场景化
- 系统设计

# 遇到的问题



1. 多套监控系统，包括收集，可视化及报警等
2. 各种上下文切换
3. 适合熟练工，不适合新同学

核心问题

1. 快速发现问题
2. 快速定位问题

核心用户

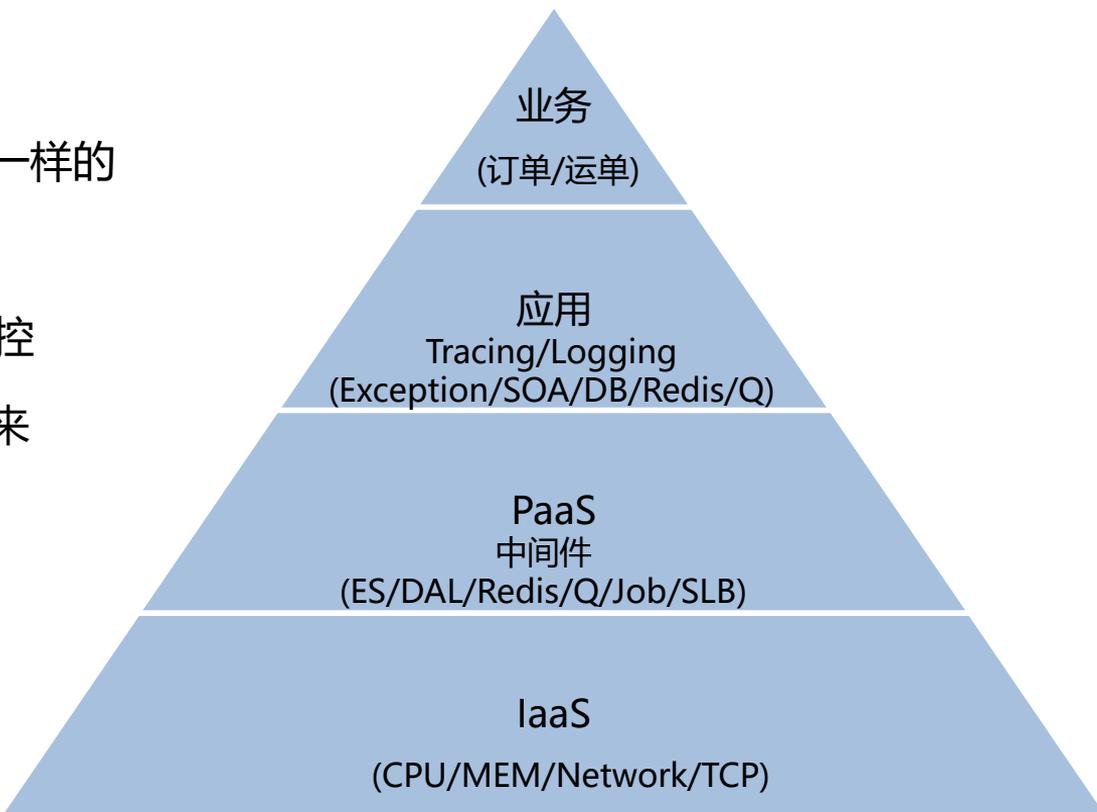
1. GOC
2. 开发人员

E-Monitor

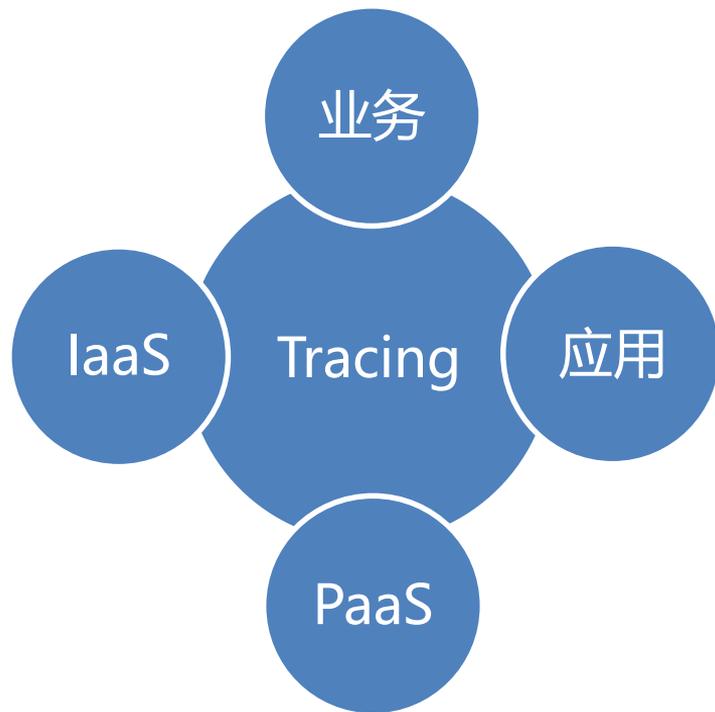
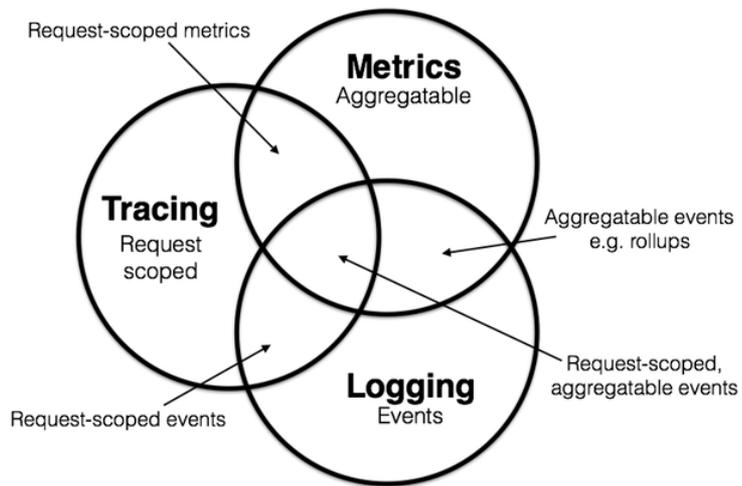


# 如何解决

1. 各层面向的用户及其视角是不一样的
2. 做好业务侧监控，并能联动
3. 标准化应用/PaaS/IaaS各层监控
4. 需要一个纽带来把各层串联起来
5. 端对端监控
6. 与其他系统集成

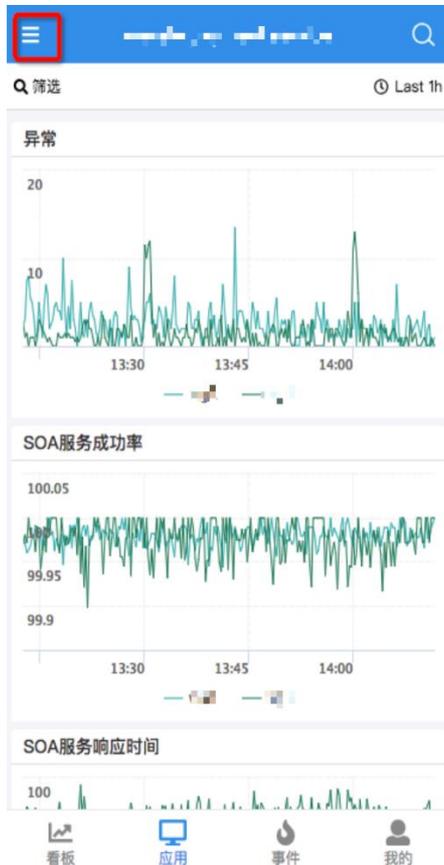


# 如何解决



# 如何解决

一套系统覆盖所有监控，支持多平台



# 目录

- 背景
- 遇到的问题
- 场景化
- 系统设计

# 业务大盘

## VS Grafana

1. 与业务更贴合
2. Dashboard App
3. Chart Repo
4. Drill Down
5. 小工具



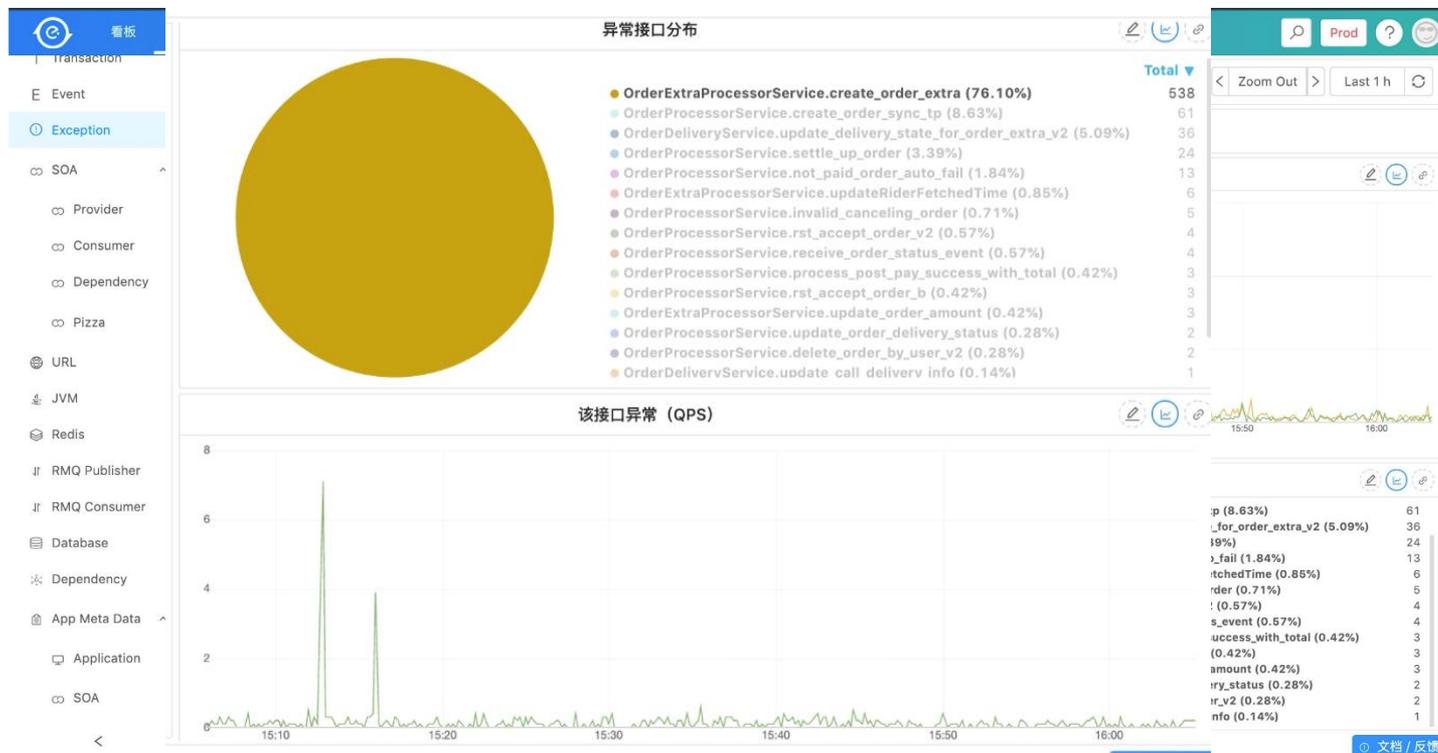
# 业务大盘



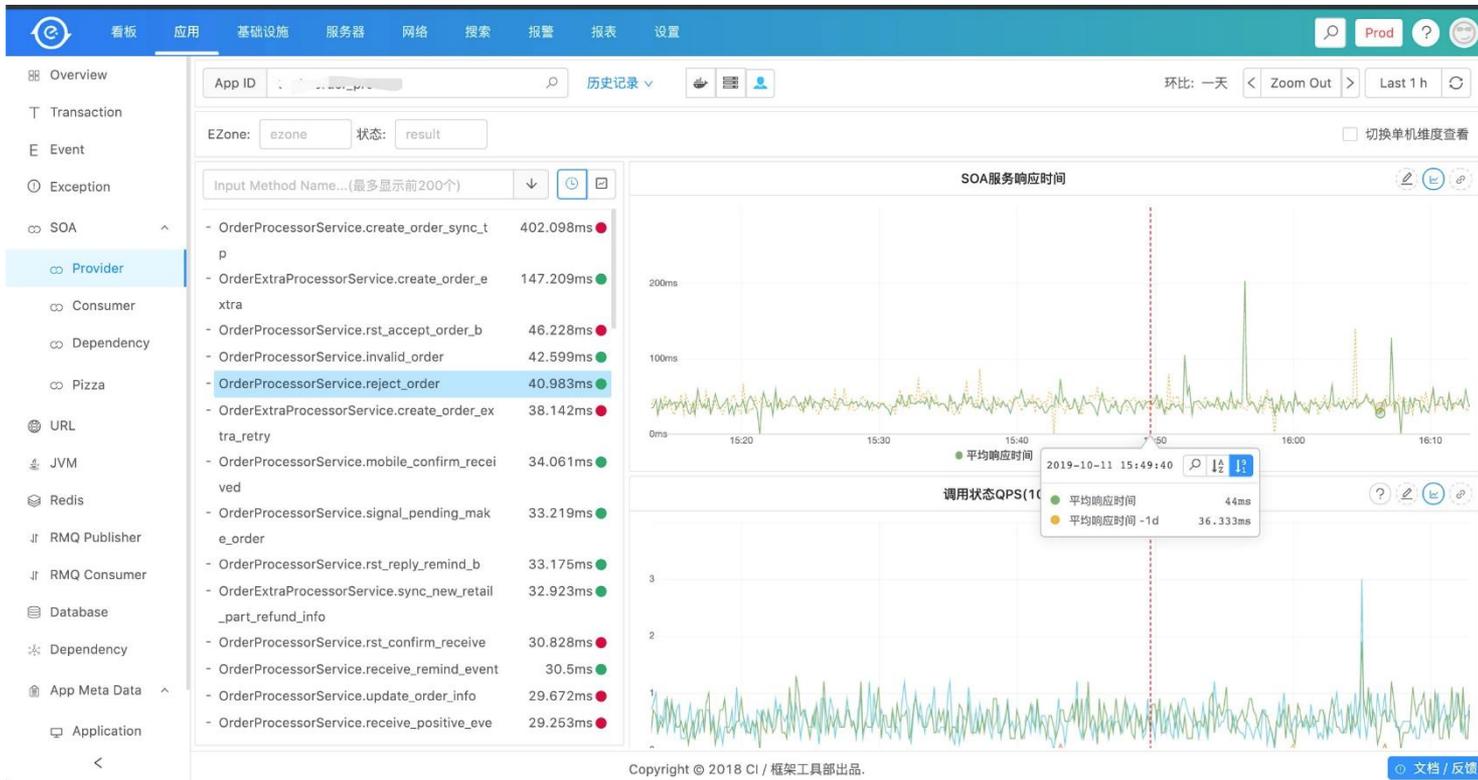
# 应用监控



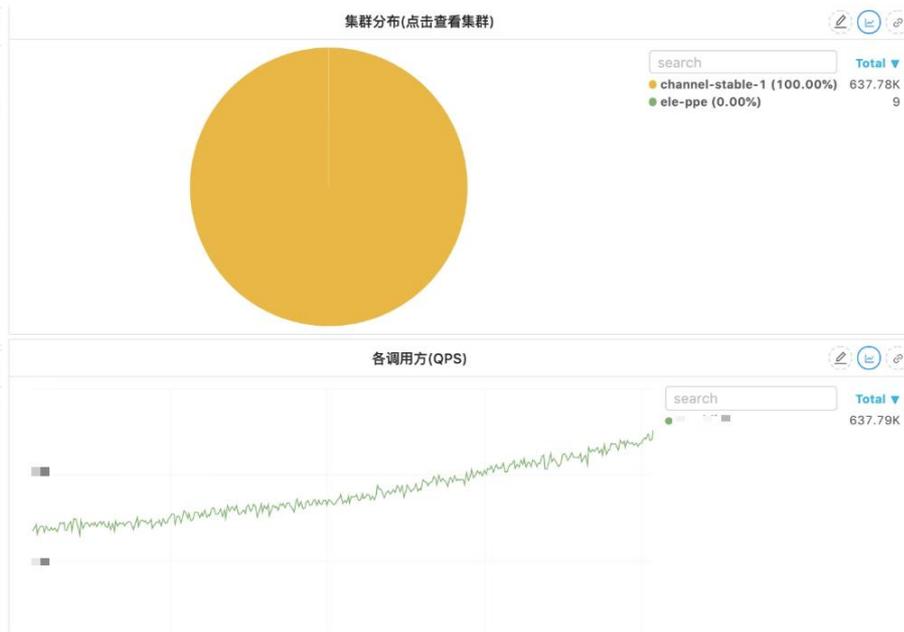
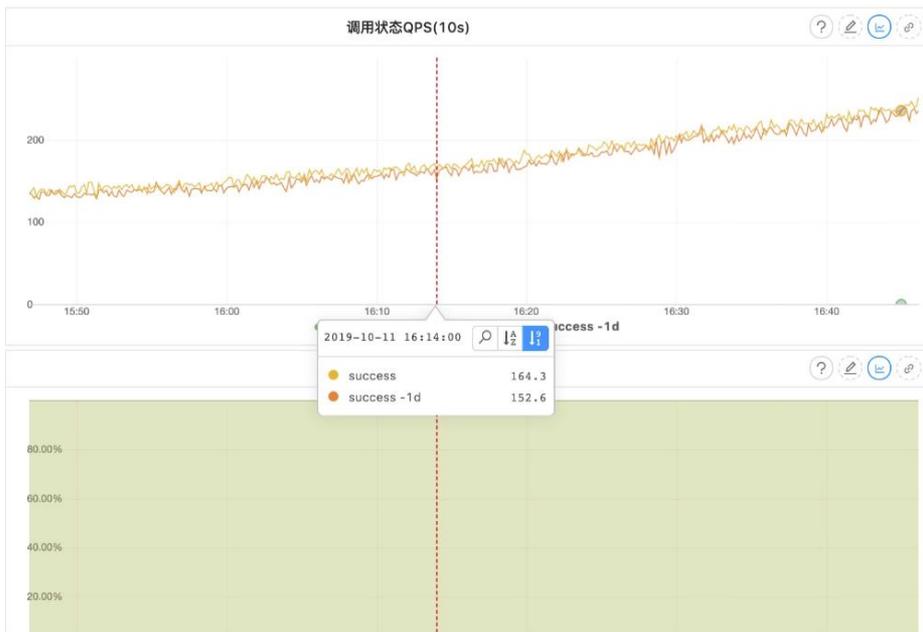
# 应用监控 – Exception



# 应用监控 - SOA

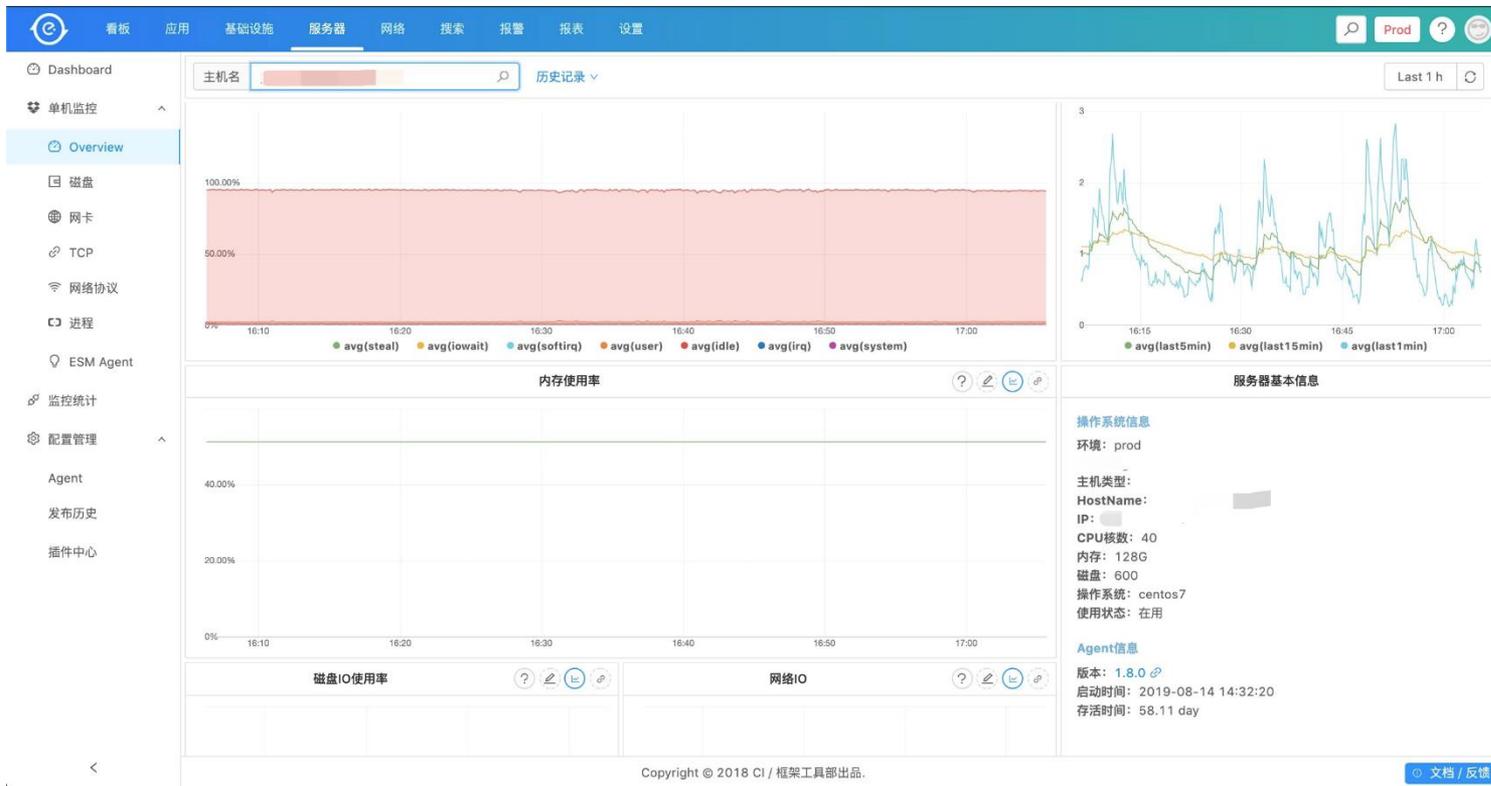


# 应用监控 - SOA





# 服务器监控



# Tracing

10-10 15:24:56.234 starlightServer :me.ele.trade.integration.order.process.api.OrderProcessorService 0

Tags

InterceptorTotalCost: 0  
me.ele.trade.integration.order.interceptor.ServerSOAInterceptor : 0  
Cost: 54

trade.order_process 1.1.... wg1	1ms	1.85(%)	SQL: ...select	Drill Down	10-10 15:24:56.235
trade.order_process 1.1.... wg1	0ms	0.00(%)	SQL: ...insert		10-10 15:24:56.237
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	DAL: ...record.insert		10-10 15:24:56.234
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	SQL: ...record_12.insert		10-10 15:24:56.234
trade.order_process 1.1.... wg1	1ms	1.85(%)	SQL: commit		10-10 15:24:56.237
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	DAL: unknowntable.commit		10-10 15:24:56.235
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	SQL: unknowntable.other		10-10 15:24:56.235
trade.order_process 1.1.... wg1	2ms	3.70(%)	SQL: ...update		10-10 15:24:56.238
me.ele.arch.das.capricorn... wg1	1ms	1.85(%)	DAL: ...update		10-10 15:24:56.237
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	SQL: ...update		10-10 15:24:56.237
me.ele.arch.das.capricorn... wg1	0ms	0.00(%)	SQL: ...update		10-10 15:24:56.238
trade.order_process 1.1.... wg1	1ms	1.85(%)	SQL: commit	Drill Down	10-10 15:24:56.240
trade.order_process 1.1.... wg1	0ms	0.00(%)	RMQ_PRODUCER: EChannel.basicPublish	Consume	10-10 15:24:56.242
trade.order_process 1.1.... wg1	0ms	0.00(%)	ETraceLink: AsyncCall	Drill Down	10-10 15:24:56.242
trade.order_process 1.1.... wg1	1ms	1.85(%)	SQL: order_process_record.select	Drill Down	10-10 15:24:56.242
trade.order_process 1.1.... wg1	0ms	0.00(%)	Redis: Stats		10-10 15:24:56.242
trade.order_process 1.1.... wg1	0ms	0.00(%)	RMQ_PRODUCER: EChannel.basicPublish	Consume	10-10 15:24:56.243
trade.order_process 1.1.... wg1	0ms	0.00(%)	RMQ_PRODUCER: EChannel.basicPublish	Consume	10-10 15:24:56.243
trade.order_process 1.1.... wg1	43ms	79.63(%)	SOACall: ISpoutService.pushOrder	Drill Down	10-10 15:24:56.243
trade.order_process 1.1.... wg1	2ms	3.70(%)	SOACall: ElemeOrderService.refresh_cache	Drill Down	10-10 15:24:56.286

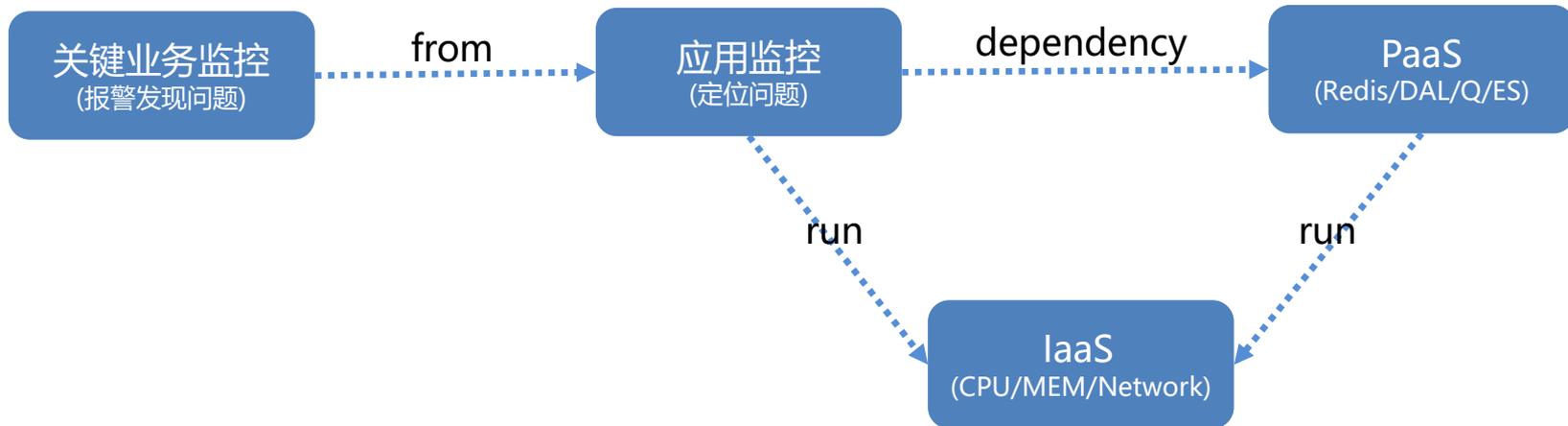
题目的请求

Copyright © 2016 C17 视觉工具部出品  
Copyright © 2016 C17 视觉工具部出品

文档 / 反馈



# E-Monitor



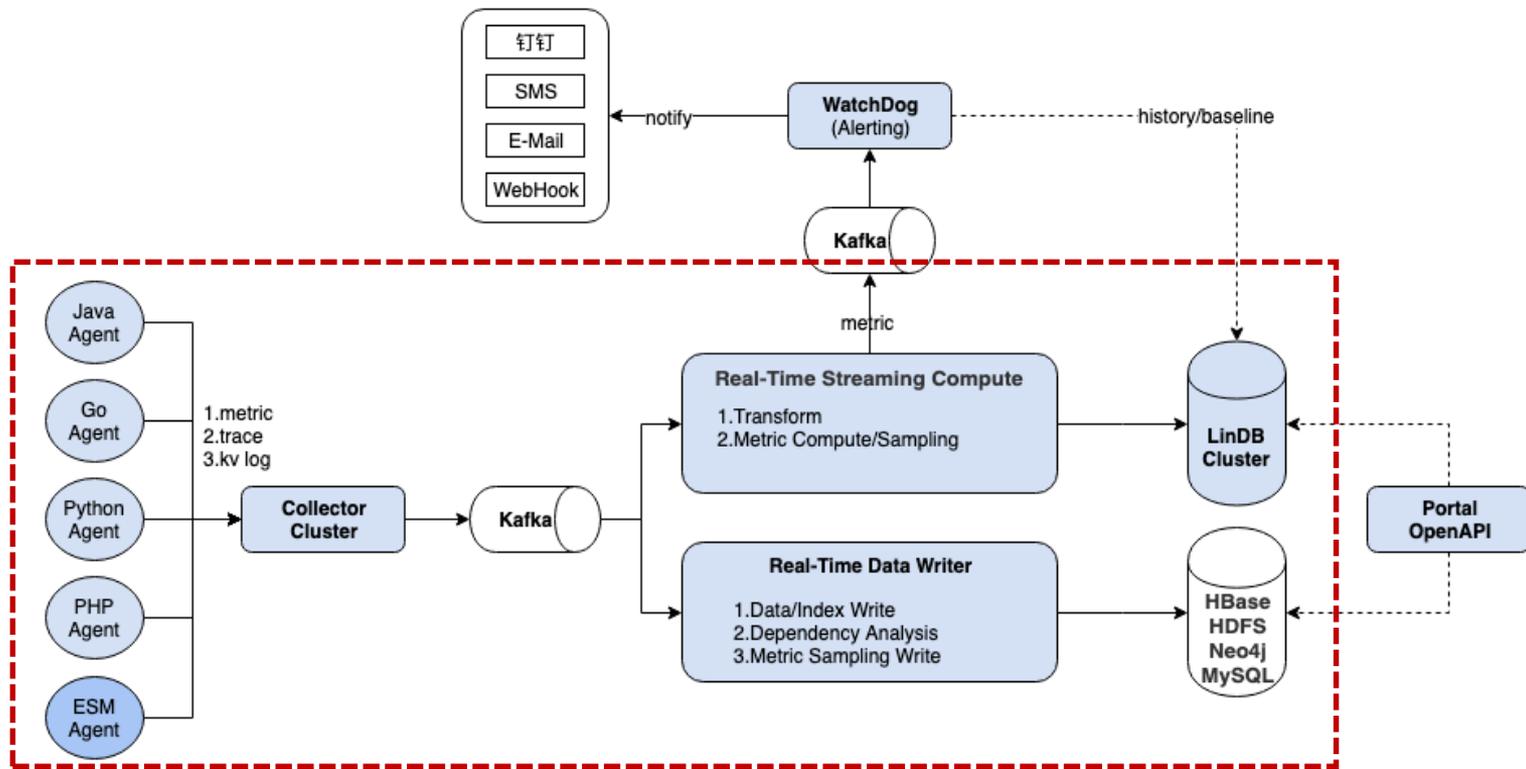
# 目录

- 背景
- 遇到的问题
- 场景化
- 系统设计

# 整体架构

1. Pipeline -> Lambda
2. 支持多IDC
3. 全量日志, 通过指标+采样的方式
4. 支持 Java/Golang/Python/PHP/C++/Node.js
5. 所有监控数据计算窗口为 10S
5. 自研 + 开源组件构建了整套系统

# 整体架构



# 踩过的坑



1. Kafka broker 节点 IO Hang 住，导致所有 Producer 线程全部 Hang 住，流量掉底
2. HBase 上构建了索引，导致 HBase 热点严重
3. 系统稳定性
4. 生产效率



1. 基于 Kafka Client 封装了一个 Broker 与 Thread 绑定的版本，即一个线程负责某一 Broker 的写入，当某一节点写入有问题，数据自动 Balance 到别的节点
2. 不支持全文检索，有时看起很用的功能，其实不一定是用户真正需要的
3. 从 Pipeline 处理所有数据流，到计算和写存储分离类似 Lambda，计算采用类 SQL
4. 所有的数据都转换成 Metrics，外加自定义的可视化组件，阶段性的前进，每个阶段只做 1-2 件重要的事情

# 计算 - Shaka

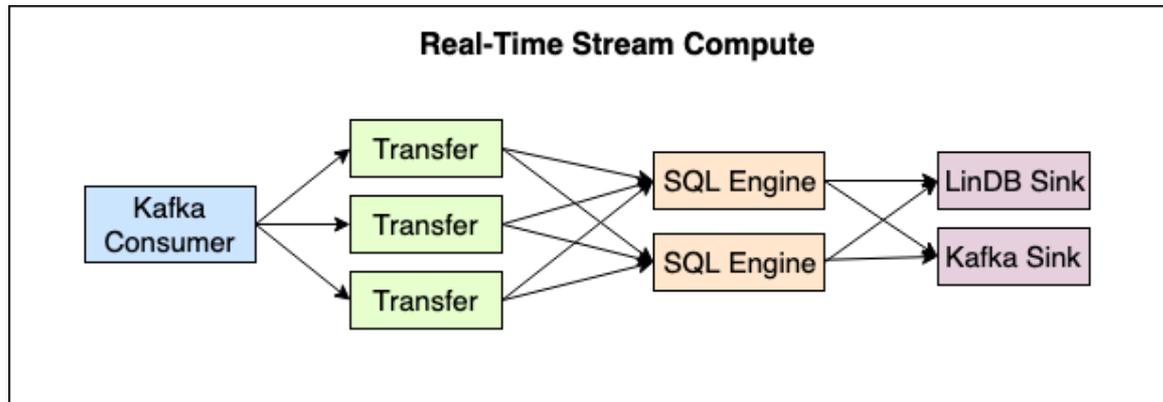


1. 随着数据量不断增加，系统开始出现不稳定的情况，有时出问题之后需要较长时间来恢复
2. 计算是整系统的资源大户，也是整个系统最核心的组件之一



做好数据的 Sharding 对一个计算类组件非常重要，越早做 Sharding 效果越好

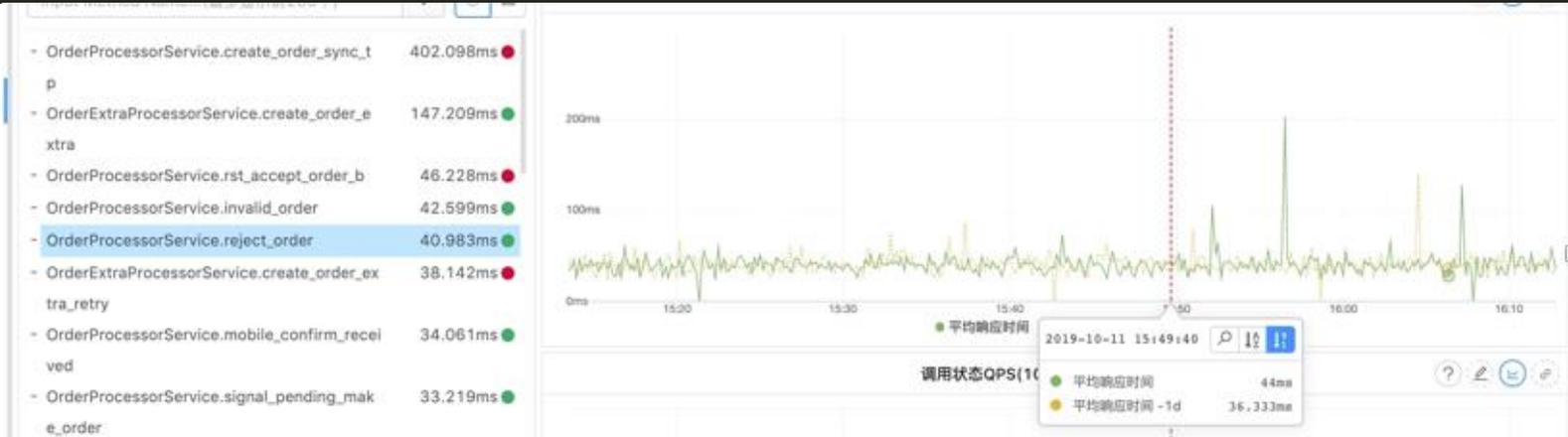
1. 写 Kafka 之前就按一定的数据特性来 Sharding，不同类型的数据写不同的 Topic/Partition
2. Shaka 内部又按不同 Event 类型 Sharding 到不同的 SQL Engine



1. 基于 CEP(Esper) 实现类 SQL 的计算
2. 非结构化的数据转换成结构化数据
3. UDF 处理异常数据分析及采样

# 计算 - Shaka

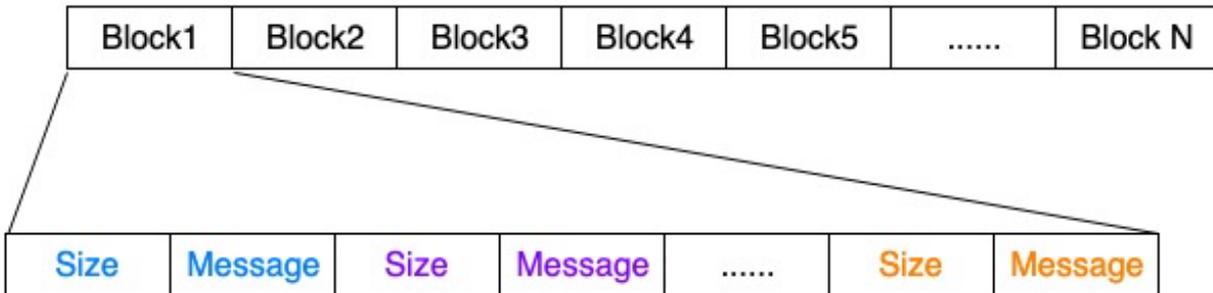
```
@ Name ('soa_provider')
@Metric(name = '{appId}.soa_provider', tags = {'result', 'method', 'ezone'}, fields = {'timerCount', 'timerSum', 'timerMin', 'timerMax', 'hist'}, fieldMap = {'histogramCount', 'timerCount'}, sampling = 'sampling')
@Metric(name = 'etrace.dashboard.soa_provider', tags = {'result', 'appId', 'ezone'}, fields = {'timerCount', 'timerSum', 'timerMin', 'timerMax', 'hist'}, fieldMap = {'histogramCount', 'timerCount'})
select header.ezone as ezone,
header.appId as appId,
result as result,
method as method,
trunc_sec(timestamp, 10) as timestamp,
f_max(max(duration)) as timerMax,
f_min(min(duration)) as timerMin,
f_sum(sum(duration)) as timerSum,
f_sum(count(1)) as timerCount,
hist(duration) as hist,
sampling('Timer', duration, header.msg) as sampling
from soa_service
group by header.appId, method, result, header.ezone, trunc_sec(timestamp, 10);
```



# 存储 - Data

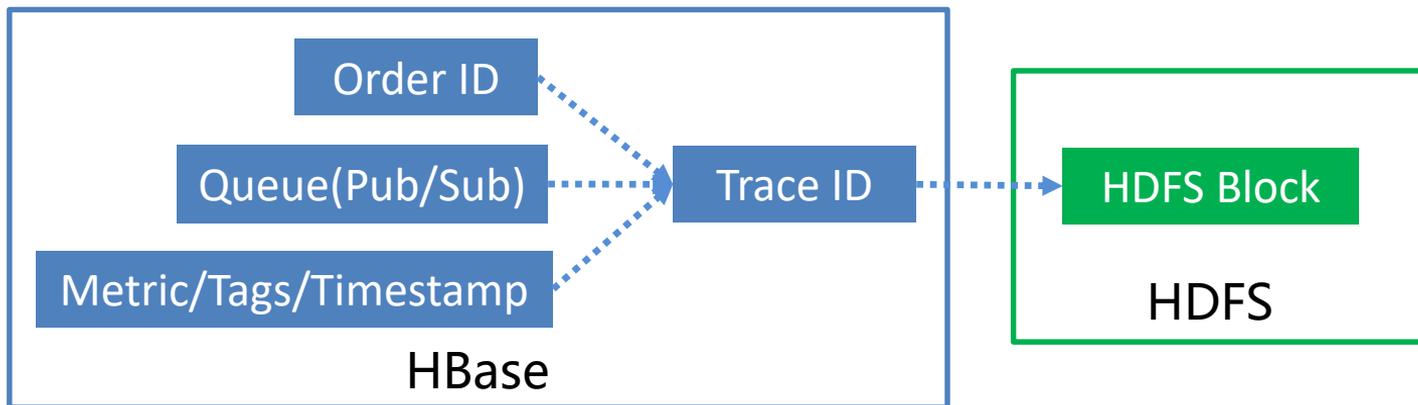
1. HDFS + HBase 存储所有的 Raw Data, HDFS 存储所有的 Raw Data, HBase 存储简单的索引 (Request ID/Trace ID + RPC ID => file + block offset + message Offset)
2. 64 KB Block + Snappy
3. Block 压缩前置到 Collector 完成

hdfs://data/fx/bucket/20190802/10/0-192.168.0.1.data



# 存储 - Data

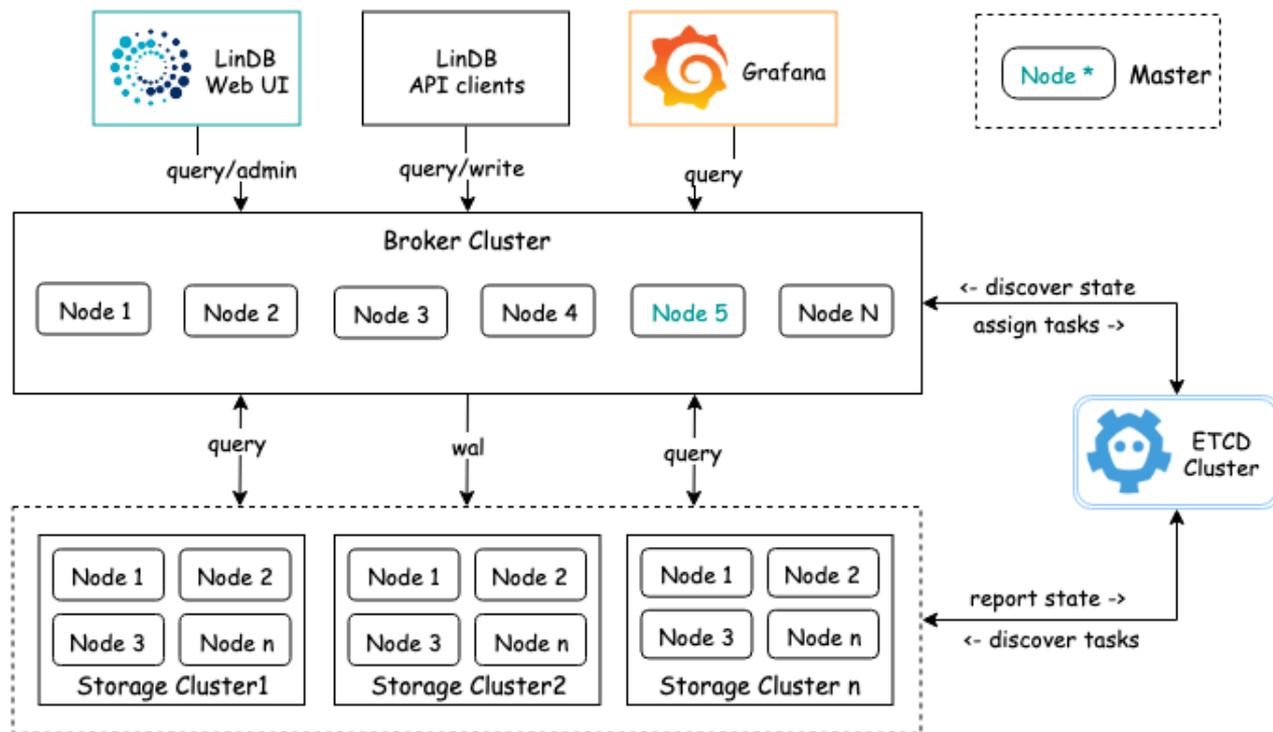
1. HBase 按天建表
2. Pre-Split
3. 多个二级索引



# TSDB - LinDB

1. 采用 Metric + Tags + Fields 方式
2. 基于 Series Sharding, 支持水平扩展
3. 自动的 Rollup, s->m->h->d
4. 高可靠性, 支持多副本, 支持跨机房
5. 自监控, 数据治理
6. 列式存储, 具有时序特性的 LSM 存储结构
7. 倒排索引

# TSDB - LinDB



# TSDB - LinDB

时序数据特性（根据其时间特性可以分为不随时间变化和随时间变化的数据）

1. Time Series => Metric + Tags: 这部分数据基本都是字符串，而且该数据占数据包的大头，但是不会随时间变化而变化，

尽量把字符串转换成数值来存储，以降低存储成本

2. Fields: 这部分数据基本都是数值，并且随着时间变化而变化，但是数值类型容易做压缩

Metrics	Tags	Timestamp + Fields
cpu.load	host=1.1.1.1,zone=sh	2019-01-10 21:00:00.000 => 1.0
cpu.load	host=1.1.1.1,zone=sh	2019-01-10 21:00:10.000 => 1.0
cpu.load	host=1.1.1.1,zone=sh	2019-01-10 21:00:20.000 => 1.0
cpu.load	host=1.1.1.1,zone=sh	2019-01-10 21:00:30.000 => 1.0
cpu.load	host=1.1.1.1,zone=sh	2019-01-10 21:00:40.000 => 1.0

1. 36台服务器，分不同集群
2. 每天增量写入 140T
3. 高峰TPS: 750W DPS/s
4. 10S 存 30天，历史可查2年以上
5. 磁盘占用 50T (压缩率在60倍左右)
6. 查询P99: 500ms ~ 1s

**THANK YOU !**

