最前沿開源監控 Prometheus專題講座

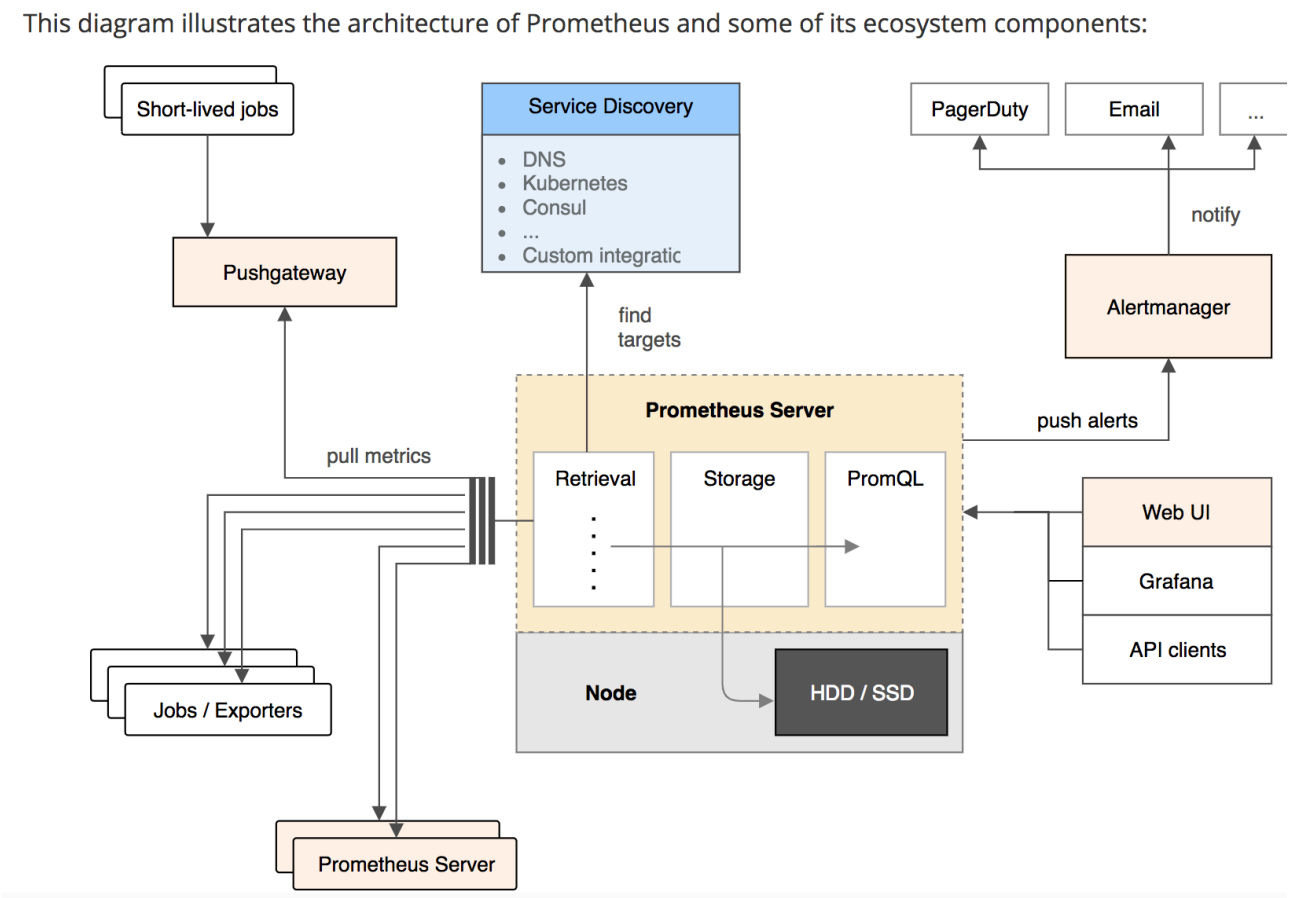
序章

一本专题讲座 章节介绍

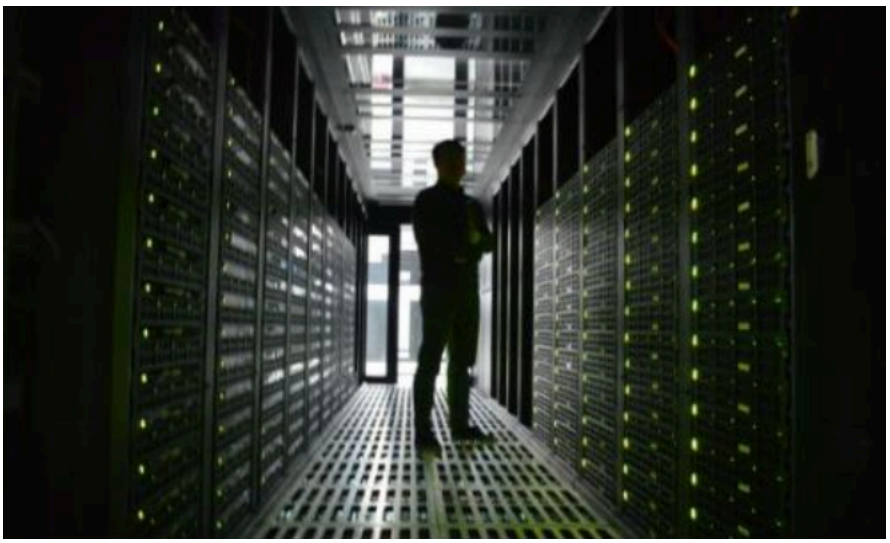
- 序章：做本专题讲座的内容简介
- 第一讲：企业级运维监控理论基础
- 第二讲：企业监控通用技术
- 第三讲：Prometheus 监控入门简介
- 第四讲：Prometheus 运行框架介绍
- 第五讲：Prometheus 数据格式介绍
- 第六讲：Prometheus 初探和配置（安装测试）
- 第七讲：Prometheus 数学理论基础学习 （prometheus 数学consule使用）
- 第八讲：Prometheus 命令行使用扩展
- 第九讲：企业级监控数据采集方法

- 第十讲：企业级监控数据采集脚本开发实践
- 第十一讲：Prometheus 之 exporter模块采集深入学习
- 第十二讲：Prometheus 之 Pushgateway模块深入学习
- 第十三讲：Grafana 超实用企业级监控绘图工具的结合
- 第十四讲：Prometheus 企业级实际使用（一）
- 第十五讲：Prometheus 企业级实际使用（二）
- 第十五讲：Prometheus 企业级实际使用（三）
- 第十六讲：Alertmanager连用（可选）
- 第十七讲：Pagerduty连用（重要）

1) 先简单看一眼 Prometheus的整体框架图 （很复杂看不懂？ 跟着大米老师放宽心）



2）我们来谈谈监控对运维的重要性和地位



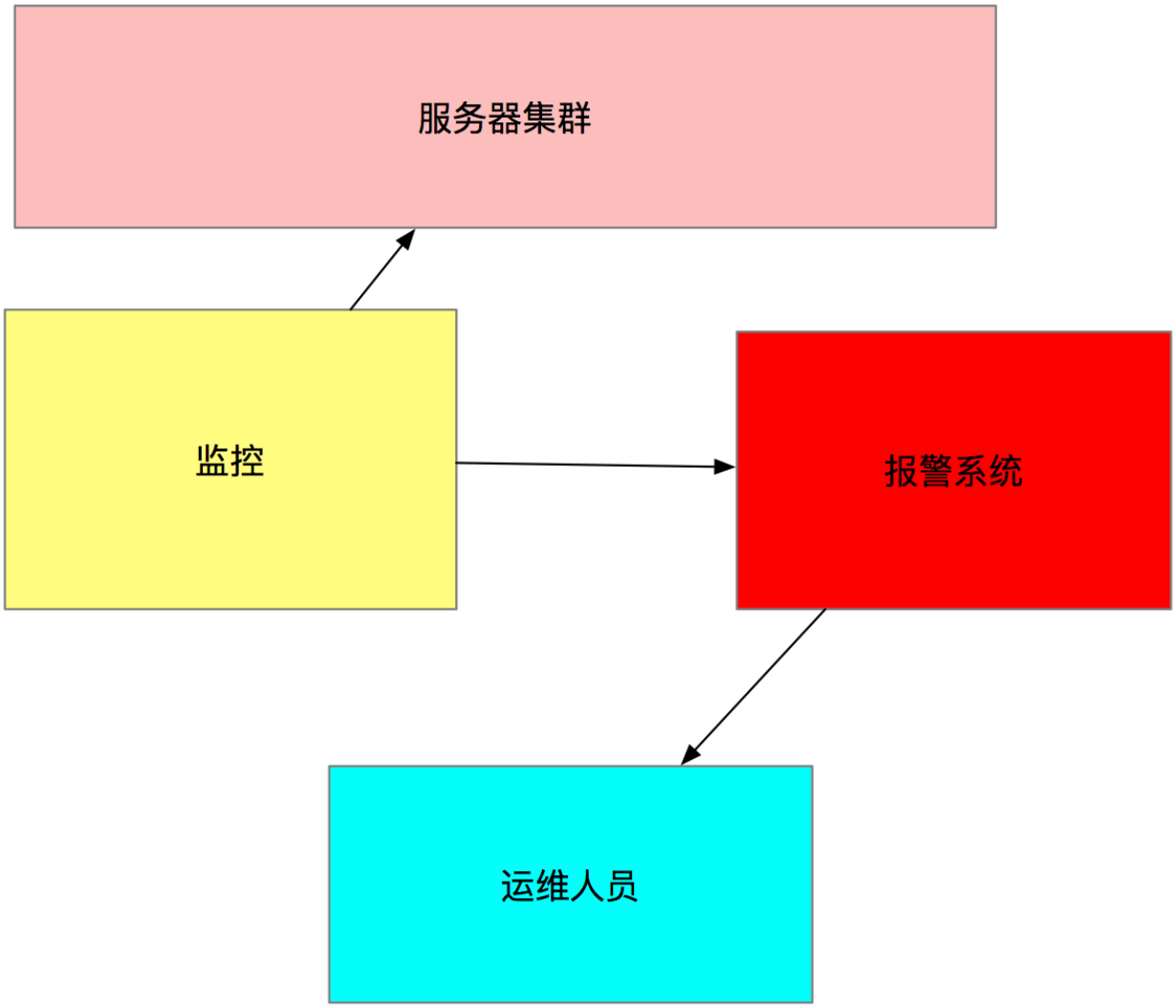
运维是什么？ 说白了就是管理服务器，保证服务器给线上产品提供稳定运行的服务环境 （大米运维 系列课程 第一阶段有详细阐述运维在企业中的定位 ）

监控是什么？ 说白了 就是用一种形式 去盯着 观察服务器 把服务器的各种行为表现都显示出来 用以发现问题 和 不足

报警是什么？ 监控和报警这两个词一定要分开说 分开理解！ 监控是监控 报警是报警。
监控是把行为表现展示出来，用来观察的
报警则是 当监控获取的数据 发生异常并且到达了某个临界点的时候，采用各种途径来通知用户 通知管理员 通知运维人员 甚至通知老板

很多时候 总是把监控和报警混在一起说 这是不正确的 需要纠正

如下图所示



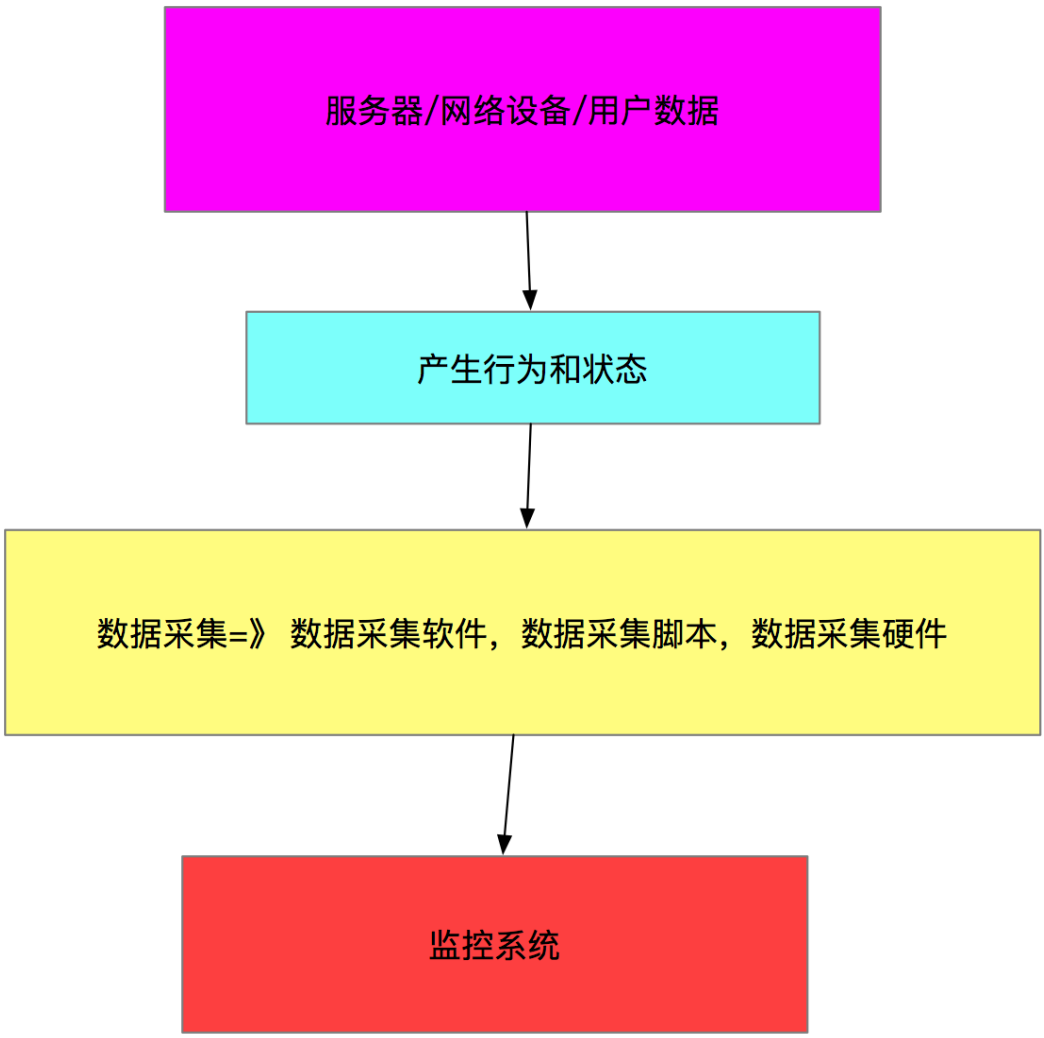
3） 我们先来谈谈监控都有哪些组成部分 和流程

监控本身看的是数据，数据从哪里来?

数据不是凭空从天上掉下来的，也不是研发人员主动给你的，只能是从运维数据采集来.

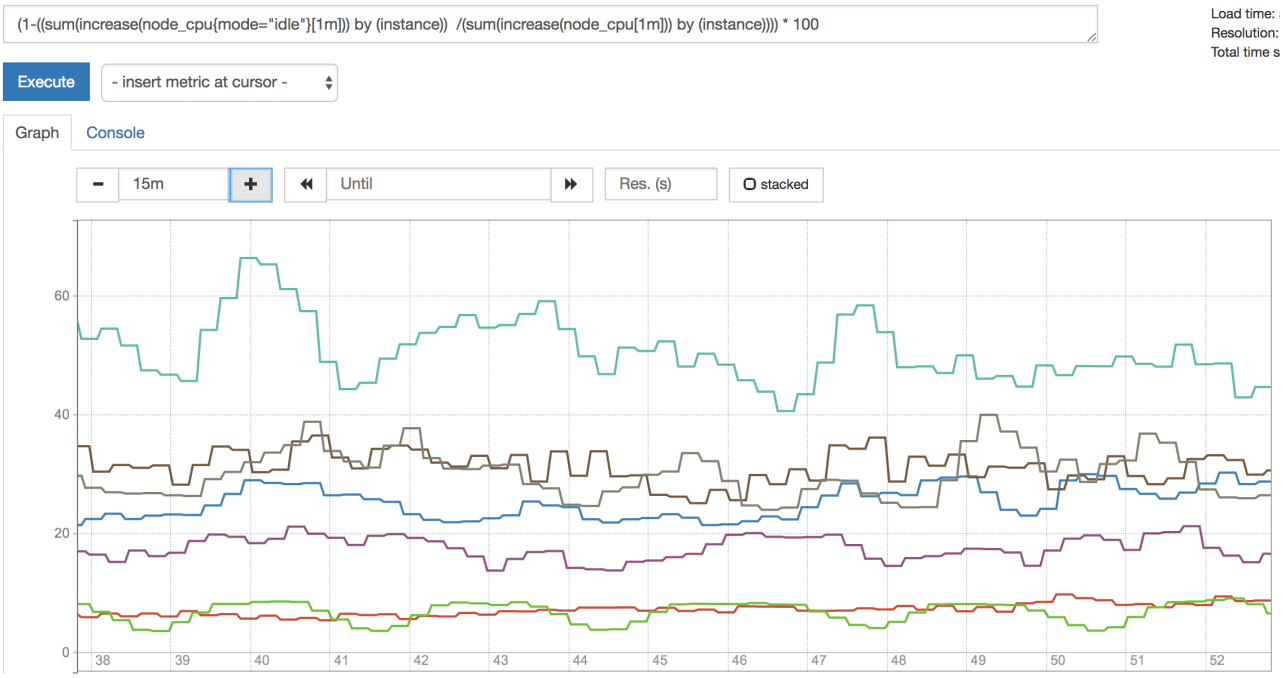
数据采集本身就是一门大学问，不仅仅是为监控提供服务/ 分析用户行为 / 安全策略

数据采集是什么？ 数据采集的方式



4） 我们来看一个用**Prometheus + Grafana** 的一个数据监控采集成图

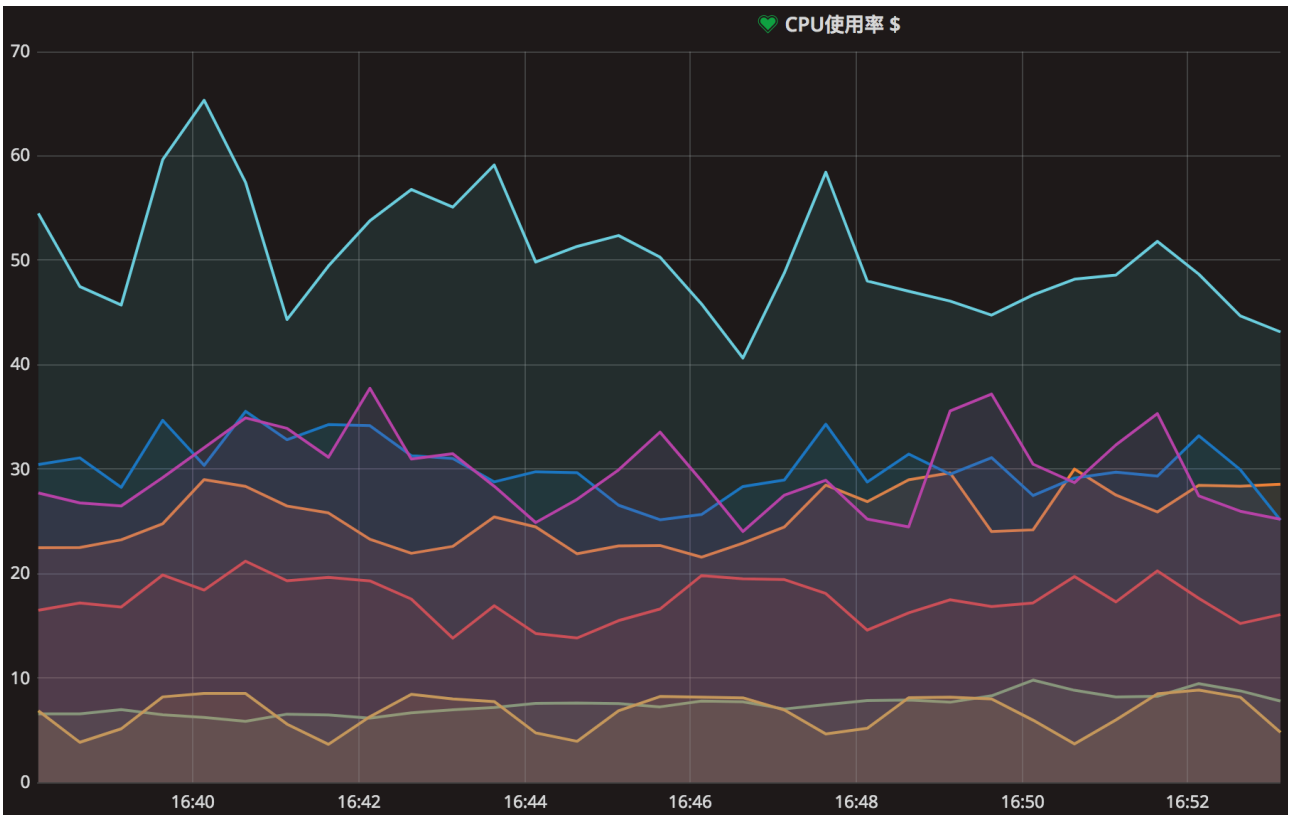
prometheus 数据采集后的成图

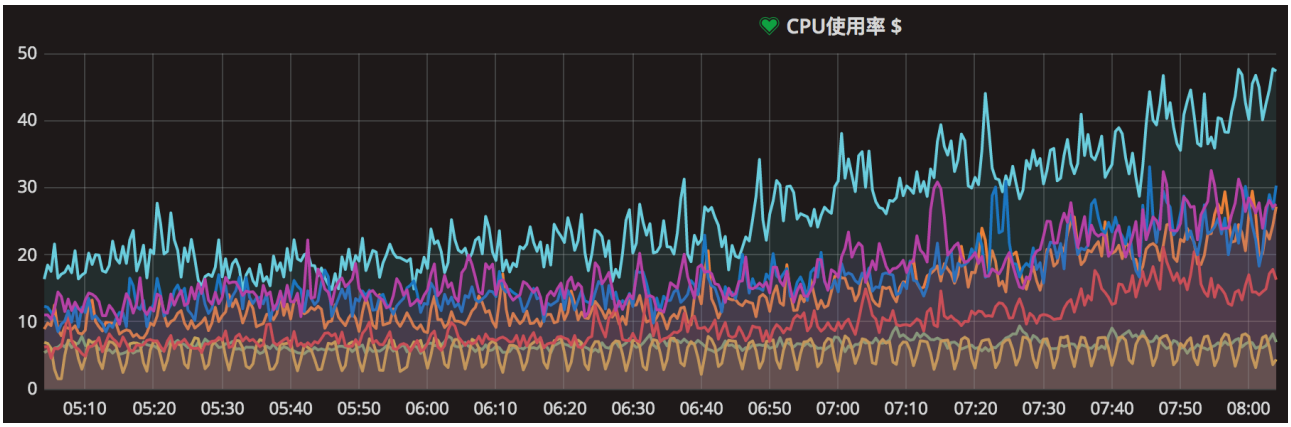


集联Grafana这个强力的绘图工具后，更加美观和直观

—Prometheus + Grafana成图

其实 上下两个图 都来自于同一份采集数据，既不冲突 又大大提供利用率





5） 我们来谈谈报警

报警跟监控 严格来说 是需要分开对待的

因为报警 也有专门的报警系统

报警系统 包括几种主要的展现形式： 短信报警，邮件报警，电话报警（语音播报）,通讯软件

不像监控系统 比较成型的报警系统 目前大多数都是收费的 商业化

报警系统中 最重要的一个概念之一 就是对报警阈值的理解

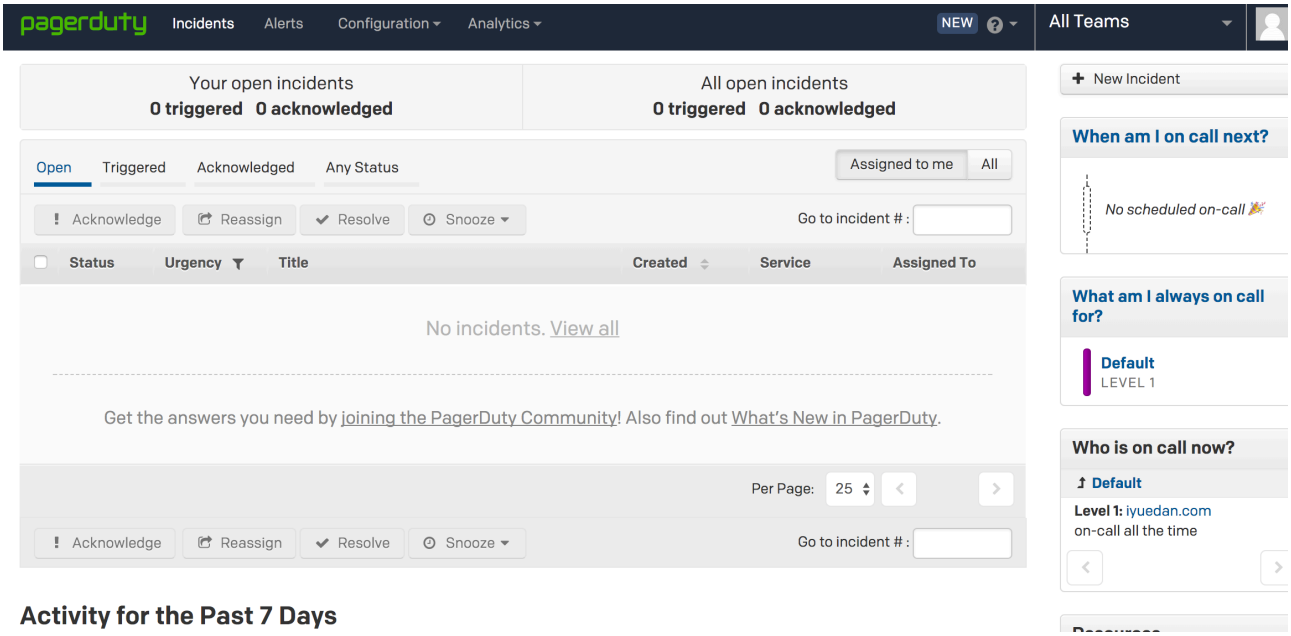
阈值(Trigger Value)，是监控系统中 对数据到达某一个临界值的定义

例如： 通过监控发现，当前某一台机器的CPU突然升高，到达了 99%的使用率，99 就是作为一次报警的触发阈值

6） 我们来看一眼 本章节给大家即将讲到的一款 优秀的美国商业报警系统

Pagerduty 的年头不短了，在企业中 尤其是外企 出镜率非常高

虽然是收费监控，但是费用对于一个企业来说很便宜 甚至对于 个人用户来说其实也不算高



Pagerduty 拥有 短信，电话，邮件 所有的报警机制

Pagerduty 还有非常实用的必要的 运维值班管理制度 和 报警升级等等 扩展功能 往后我们会陆续介绍到

Pagerduty的优点非常多，使用率非常高（外企几乎清一色的使用，国内企业很多也在使用）

但是有优点就肯定也有不足

Pagerduty有几个小问题 需要提高

- 对中文的支持不好 或者说几乎不支持 （指的是 语音播报方面 ）=> 那咱们为啥不多提高下自己的英文能力？^_^
- 站点主要在美国和海外 网速有时候不太给力 => 可以走代理的方式 加快速度

其他的问题 基本上没有了

7） 最后 我们来谈谈 Prometheus相比其他老款监控的 不可被替代的巨大优势，以及一些不足有待提高的地方

- 监控数据的精细程度 绝对的第一 可以精确到 1 ~ 5秒的采集精度 4 5分钟 理想的状态 我们来算算采集精度 （存储 性能）
- 集群部署的速度 监控脚本的制作 （指的是熟练之后） 非常快速 大大缩短监控的搭建时间成本
- 周边插件很丰富 exporter pushgateway 大多数都不需要自己开发了
- 本身基于数学计算模型，大量的实用函数 可以实现很复杂规则的业务逻辑监控（例如QPS的曲线弯曲 凸起 下跌的 比例等等模糊概念）
- 可以嵌入很多开源工具的内部 进行监控 数据更准时 更可信（其他监控很难做到这一点）
- 本身是开源的，更新速度快，bug修复快。支持N多种语言做本身和插件的二次开发
- 图形很高大上 很美观 老板特别喜欢看这种业务图 （主要是指跟Grafana的结合）

一些不足的地方

- 因其数据采集的精度 如果集群数量太大，那么单点的监控有性能瓶颈 目前尚不支持集群 只能workaround
- 学习成本太大，尤其是其独有的数学命令行（非常强大的同时 又极其难学《=自学的情况下），中文资料极少，本身的各种数学模型的概念很复杂（如果没人教 自己一点点学英文官网 得 1-3 个月入门）
- 对磁盘资源也是耗费的较大，这个具体要看 监控的集群量 和 监控项的多少 和保存时间的长短

- 本身的使用 需要使用者的数学不能太差 要有一定的数学头脑（这个说真的 我觉得才是最难克服的）
- 再难也不用担心 有大米 无难事！

8） 序章结束， 之后跟着大米老师 一起来学习这一款高大上的prometheus监控 一起做监控的明日之星吧

最前沿開源監控 Prometheus專題講座

第一讲 企业级运维监控理论基础

— 第一讲内容

- 监控在企业中的重要性
- 监控在运维工作中的比重
- 数据采集的重要性
- 运维监控种类的分类
- 监控做不好的各种后果
- 监控做好了的各种好梦

1） 监控在企业中的重要性

追溯到监控不发达的时代，那时候 基本属于一个 出行基本靠走，安全基本靠狗🐶

记得在2006年 我刚刚开始做运维的时候，哪里有什么自动化监控的概念

至少都得是人工盯着（不至于靠狗🐶。。。。。）

服务器大概是2，3台的样子 记不清楚了

放在一个比较简陋的机房里（说机房 不如说就是个 不用的仓库）

所有服务器 都接着单独的显示器运行着

每天上班 第一件事 就是走进去巡视一下，看看各种软件打印出来的输出信息 是否有报错 如果有 就拿个Excel 记录一下，再发邮件给老板。。。。。

想起来都觉得 臊得慌

不过其实在那个年代 也是没有办法 运维本身就很原始，各种运维技术都处于拓荒的阶段

如今的企业中 动不动运维就要负责 成百上千 甚至上万台机器（之前在一家公司 维护过近30000+服务器 虚拟机+物理机 横跨美国7个州）

没有高大上的方法 是绝对支撑不起来这种规模的 监控的

服务器随时随地 都有可能出故障，出问题，需要被监控住

不管是虚拟机还是物理机 还是各种软件，各种线上产品，操作系统，网络设备，办公环境，业务数据，用户体验

说句夸张点的话： 现在连员工上个班 打卡机都得弄个监控，打卡机自己本身可能会坏，打卡记录 还得被HR监控

各种监控啊~ ~ ~ ~ 都是运维的事啊~ ~ ~

监控在企业中 扮演着重要的监督者的角色，任何一个地方出现问题 我们都需要及时的知道

很多情况下 企业对某种类型的监控 需要非常的敏感(采集的精度)，例如用户正常访问 这种业务级别的监控

一旦出现了问题 需要在妙极的时间 立刻知道，（时间=钱）不然就是毁灭性的灾难和损失 由其是针对哪些大规模的企业

所以说 监控对于企业的重要性 不言而喻了

2） 监控在运维工作中的比重

基础运维（系列第一阶段）一线 主要扮演着 一个处理日常任务，及时救火 这样的角色

而监控的搭建 和 数据采集的工作 很多时候 需要依赖于 运维开发的协助（开发 创新）

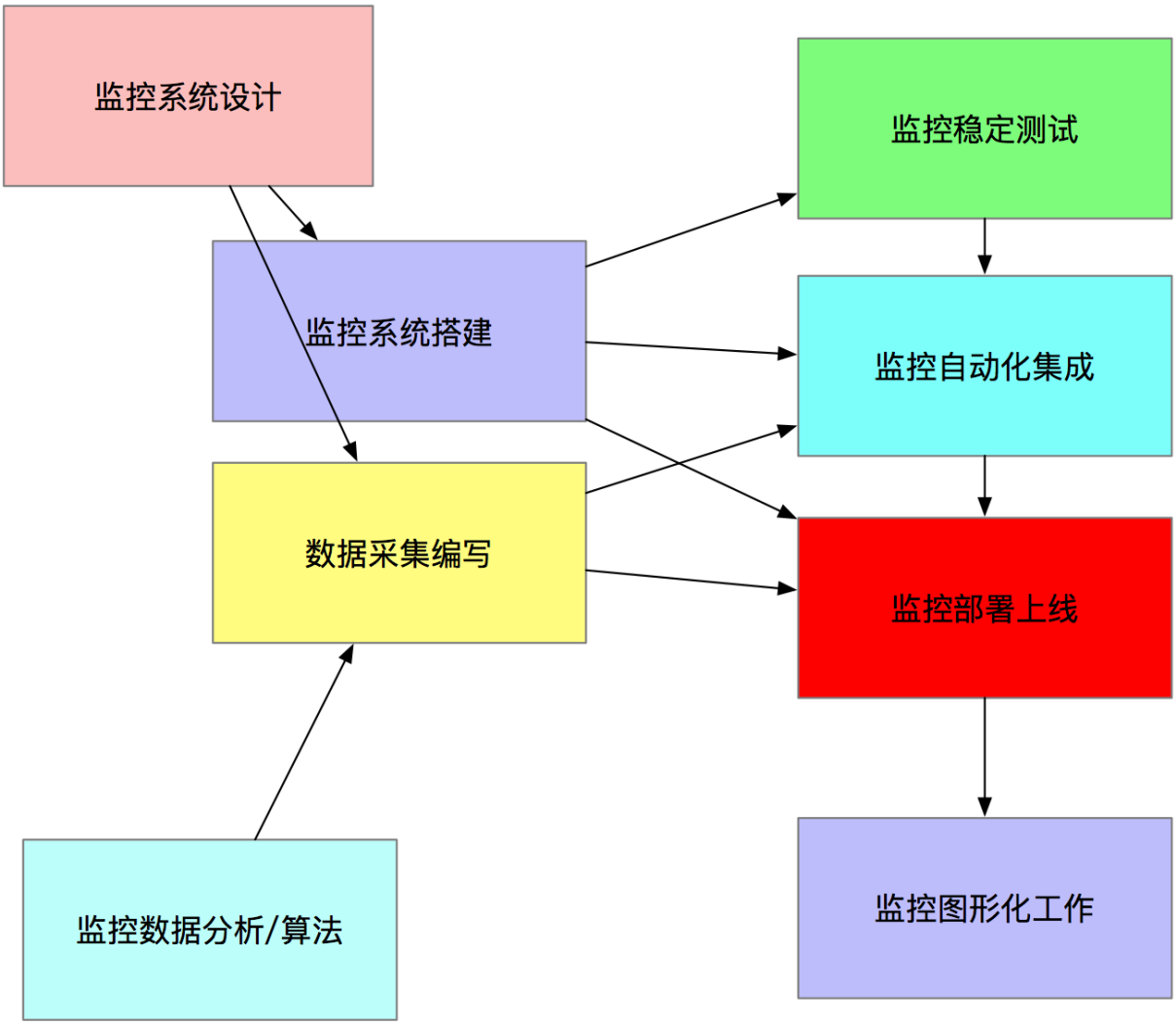
在大米运维课堂 第一阶段前三节课中，有详细介绍各种运维职位的职责（没有看的同学 感兴趣的话 可以去回顾一下课程）

不管是哪一种运维（哪怕你是运维架构师 运维专家）在紧急的时候 人人都要扮演起 救火英雄的角色

而救火 指的是 及时的发现和解决线上出现的各种故障 问题

那么为了要做到 及时的发现问题，那么一个好的完善的监控系统 就很自然的作为 运维工作中的第一优先级任务

一个最理想的 完整的 监控体系的搭建 有赖于如下这么几个方面的工作 以及流程



值得一说的是，在大部分的企业中，都无法做到 如上图的如此完善的监控创作体制

很多时候 有些其实很重要的部分 确被当成可有可无 一带而过

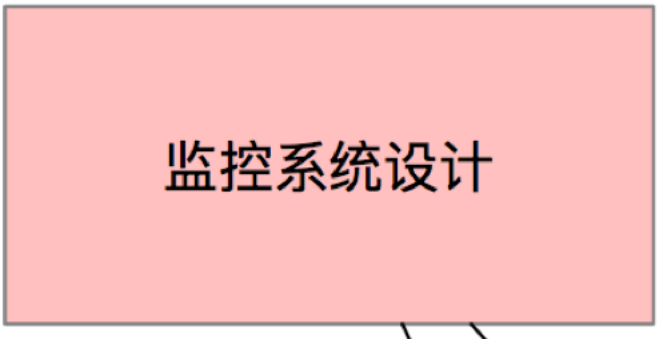
我本人觉得 非常不可取的同时 也感到无奈

因为很多时候 客观因素的力量太强大了。。。。。事事完美无缺 难啊！

（但是我们作为一个运维架构师 依然要有追求完美的信念 ）

接下来我们分别解释一下 监控流程中的 这些部分的内容

1） 监控系统设计（架构师）：



首当其冲 就是压在运维架构师的肩膀上的任务

设计如果都设计不好，那么后面的事儿也是白干

设计部分包括如下的内容：

- 评估系统的业务流程 业务种类 架构体系

各个企业的产品不同，业务方向不同，程序代码不同，系统架构更不同

对于各个地方的细节 都需要有一定程度的认知 才可以开起设计的源头

- 分类出所需的监控项种类

一般可分为： 业务级别监控 / 系统级别监控 / 网络监控 / 程序代码监控/ 日志监控 / 用户行为分析监控/ 其他种类监控

大的分类 还有更多的细小分类

这里给出几个例子

例如：

- 业务监控 可以包含 用户访问QPS，DAU日活，访问状态（http code）， 业务接口(登陆，注册，聊天，上传，留言，短信，搜索)，产品转化率，充值额度，用户投诉 等等这些很宏观的概念（上层）
- 系统监控 主要是跟操作系统相关的 基本监控项 CPU/ 内存 / 硬盘 / IO / TCP链接 / 流量 等等 (Nagios - plugins, prometheus)
- 网络监控 （IDC）对网络状态的监控（交换机，路由器，防火墙，VPN） 互联网公司必不可少 但是很多时候又被忽略 例如：内网之间（物理内网，逻辑内网 可用区 创建虚拟机 内网IP ） 外网 丢包率 延迟 等等
- 日志监控 监控中的重头戏（Splunk,ELK），往往单独设计和搭建， 全部种类的日志都有需要采集 (syslog, soft, 网络设备，用户行为)
- 程序监控 一般需要和开发人员配合，程序中嵌入各种接口 直接获取数据 或者特质的日志格式

- 监控技术的方案/软件选取（主观因素）

各种监控软件层出不穷，开源的 商业的 自行开发的 几百种的可选方案

架构师凭借一些因素 开始选材.

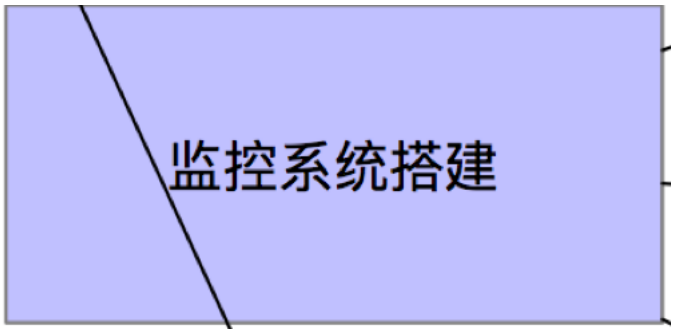
针对企业的架构特点，大小，种类，人员多少 等等 选取合适的技术方案

- 监控体系的人员安排

运维团队的任务划分，责任到人，分块进行

开发团队的配合人员选取，很多监控涉及的工作 都需要跟开发人员配合 才可以进行

2） 监控系统的搭建



- 单点服务端的搭建(prometheus)
- 单点客户端的部署
- 单点客户端服务器测试
- 采集程序单点部署
- 采集程序批量部署
- 监控服务端HA / cloud (自己定制)
- 监控数据图形化搭建（Grafana）
- 报警系统测试(Pagerduty)
- 报警规则测试
- 监控+报警联合测试
- 正式上线监控

3） **数据采集的编写**

数据采集编写

- 可选用脚本作为数据采集途径

例如： shell / python / awk / lua （Nginx 安全控制，功能分类）/ php / perl/ go 等等

shell： 运维的入门脚本，任何和性能/后台/界面无关的逻辑 都可以实现最快速的开发

（shell 是在运维领域里 开发速度最快 难度最低的）

python: 各种扩展功能 扩展库 功能丰富 ，伴随各种程序的展示+开发框架（如django）等 可以实现快速的中高档次的平台逻辑开发. 目前在运维届 除去shell这个所有人必须会的脚本之外，火爆程度就属python了

awk: 本身是一个实用命令 也是一门庞大的编程语言. 结合shell脚本 或者独立 都可以使用。

在文本和标准输出处理上 有很大的优势

lua: 多用于nginx的模块结合 是比较新型的一个语言

php：老牌子的开发语言，在大型互联网开发中，目前有退潮的趋势 （PHP语言，php-fpm）不过在运维中 工具开发还是很依赖PHP

perl: 传说中 对文本处理最快的脚本语言 （但是代码可读性不强）

go: 新型的语言 目前在开发和运维中 炒的很热 工资也高 C语言 在各种后端服务逻辑的编写上 开发速度快 成行早

作为监控数据采集， 我们首推 shell + python ， 如果说 数据采集选取的模式 对性能/后台/界面 不依赖，那么shell速度最快 成本最低（公司往往喜欢快的）

- 数据采集的形式分类

） 一次性采集： 例如我们使用比较简单的 shell ./monitor.sh (ps -ef | grep, netstats -an | wc)+ crontab的形式按10秒 / 30秒 / 一分钟 这样的频率去 单词采集

这种形式的优点=» 一次性采集的模式 稳定性较好 不容易出现各种错误 和性能瓶颈，且开发逻辑简单 实现快速

这种形式的缺点=⟩ 一次性采集 对于有些采集项目 实现起来不够智能 也不够到位 例如 日志的实时采集（使用一次性采集 日志文件 200/5xx diff grep 也可以实现 但是很low 不够准确 不够直观）

）后台式采集： 采集程序以守护进程运行在Linux后台，持续不断的采集数据： pormetheus exporter 例如python/go开发的daemon程序 后台持续不断的采集

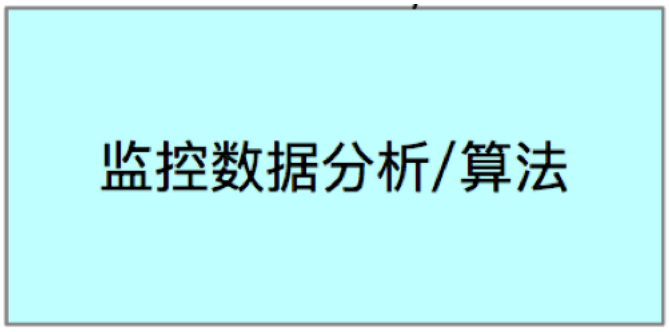
优点： 后台采集程序 数据准确性高 采集密度精细 管理方便

缺点：后台采集程序 如果开发过程不够仔细 可能会出现各种 内存泄漏 僵尸进程 性能瓶颈的问题， 且开发周期较长

）桥接式采集： 本身以后台进程运行 但是采集不能独立 依然跟服务器关联 以桥接方式收集采集数据

例如：NRPE for nagios

4） 监控数据分析和算法



例如： 采集CPU 的七种 等待状态参数 ， 采集用户每秒访问请求量 QPS

对于这些"基本单位"的数据采集 本身是必须的 也是没有疑问的

但是这里有一个问题

采集回来的单位数据 如果没有懂行的人 将它们形成 监控公式 和 报警阈值

那采集数据就没有任何的意义了

按上面两个例子来举例：

CPU来举例

```
top - 15:04:10 up 14 days, 23:47,  1 user,  load average: 1.23, 1.04, 1.08
Tasks: 378 total,   1 running, 377 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.9 us,  0.6 sy,  0.0 ni, 98.3 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
```

CPU采集回来的 平均负载数值，以及CPU的时间片分布百分比 nagios top

如果作为一个运维架构师 不懂得 Linux中CPU各种参数的深入原理
例如 平均负载是如何计算的，CPU的时间片分布是如何分类的，什么叫作 用户态/内核态 CPU等待/处理时间 什么是 Interuptable/uniteruptable CPU等待
等等这些概念。 那么即便数据被采集回来的再精细准确 你也利用不好

所以 这一点我给大家强调的是： 监控的数据分析和算法 其实非常依赖 运维架构师对Linux操作系统的各种底层知识的掌握， （这也是其中一个 我推荐prometheus的原因）
如果 我们使用那种老式的傻瓜式的监控 如nagios， 里面的监控脚本很全面(shell sh return, bin)，生成报警规则 和 阈值也很简单，缺点却显而易见： 监控的太粗糙 实用性不强， 另外 也不利于我们希望提高的同学
（例如：naigios 监控中 对CPU高不高的监控判断 就依据一个 当前的load值 >5 就警告 >10就报警， 我请问这种方式的CPU报警 有什么意义么？ 有利于我们通过监控找到真正的问题么？）

注：本阶段课程 主题是prometheus 我们可能没有过多的时间 给大家细讲Linux系统模型和内核参数 等等这些底层的优化知识，这些高级内容的具体学习 请参考 大米运维课堂（系列课程） 第三第四阶段 的高级课程

另外还有一个问题

例如业务级别监控的算法 运维自身无法做到十分专业
因为本身跟操作系统无关，是跟数据算法相关

举个例子：如果我想通过Prometheus实现对用户访问QPS的 精确监控

那么对于 监控图形 曲线 QPS上涨 QPS下跌，QPS凸起，QPS和历史数据的比较方法 等等这些 都属于业务级别的监控阈值类型
需要有专业的数据分析人员的协助 才可以算出优良的算法
例如： 如果我现在想针对 当前QPS下跌率进行报警计算， 那么用什么养的公式 针对我们的业务类型更贴切?
我是选择 计算当前5分钟内的平均值 当 < 一个固定数值的时候 报警合适?

我是选择 计算当前10分钟的总量 然后和 前一个小时同一时段比较合适?

我是选择 计算当前一小时的平均值 和 过去一周内 每一天的同一时段的时间比较合适?

。。。。

艺体类推，这些数据算法 本身跟Linux无关，只有非常专业的 数据计算团队 才可以给出一个最合理的算法 协助我们的报警规则

5） 监控稳定测试

不管是一次性采集，还是后台采集，只要是在Linux上运行的东西 都会多多少少对系统产生一定的影响

稳定性测试 就是通过一段时间的单点部署观察 对线上有没有任何影响

6）监控自动化

监控客户端的批量部署，监控服务端的HA再安装，监控项目的修改，监控项目的监控集群变化

种种这些地方 都需要大量的人工

自动化的引进 会很大程度上 缩短我们对监控系统的维护成本

这里给出几个实例： Puppet（配置文件部署），Jenkins(CI 持续集成部署)， CMDB（运维自动化的最高资源管理平台 和 理念 ）。。。。等等

利用好如上这几个聚的例子，就可以实现 对监控自动化的掌控

（自动化工具和编程 相关的学习 请参考 大米运维课堂 第二 / 三阶段的课程）

7 监控图形化工作

采集的数据 和 准备好的监控算法，最终需要一个好的图形展示 才能发挥最好的作用

监控的设计搭建需要大量的技术知识，但是对于一个观察者来说（老板），往往不需要多少技术，只要能看懂图就好（例如 老板想看看 当前用户访问量状况，想看看 整体CPU高不高 等等）

所以， 监控的成图工作 也是很重要的一个内容

本阶段课程中，会详细介绍一个 很实用的高大上的图形集成工具=》 grafana的实用和搭建

第一讲内容结束，更多内容敬请关注

最前沿開源監控 Prometheus專題講座

第二讲 企业监控通用技术

— 第二讲内容

- 介绍企业目前在监控上的发展阶段
- 介绍企业当前实用的通用技术和工具
- 企业监控目前面临的一些问题
- 介绍监控的最终理想化

1） 介绍企业目前在监控上的各个发展阶段

早期企业无监控 （路远靠走 安全靠狗 ^_^）

全部都是人工盯着 服务器 操作系统 软件 网络 等等

中前期企业 半自动脚本监控

利用shell脚本这种类似的形式，做最简单的监控脚本

循环登陆机器 查看一些状态 之后人工记录

无报警 无自动化 无监控图形

中期企业 自动化程序/脚本/软件/监控

脚本更新换代 开始使用各种开源非开源软件 程序 进行监控的搭建和开发

监控形成图形化，加入报警系统 ，有一定的监控本身自动化实现

这个阶段监控开始逐步成型 但是仍然缺乏精确度和稳定程度 报警的精细度

中后期企业 集群式监控 各种外援监控方案

监控开始自成体系 加入各种自动化

除去自身开发和搭建监控系统外，还会大量使用各种外围监控 （各种商品监控 例如云计算监控 监控宝友盟等等）

监控发展出 内监控+ 外监控（内监控是企业自己搭建的自用监控， 外监控是 使用外援的商业监控产品 往往对产品的最外层接口和用户行为进行更宏观的监控）

当前和未来监控

根据目前的发展状况

未来的监控 主要会在几个方面 不断的提高

- 监控准确性 真实性

- 监控高度集成自动化 无人值守
- 监控成本的日益降低
- 监控和CMDB的集成化以及自愈系统的发展

2） 介绍企业当前实用的通用技术和工具

- 脚本监控（当前依然使用最原始的 脚本运行的形式 采集和监控的公司 依然不在少数 很多时候是为了节约成本）
- 开源/非开源工具监控 如：Nagios / Cacti / icinga / Zabbix / Ntop / prometheus / 等等。。
- 报警系统： Pagerduty / 自建语音报警系统 / 自建邮件系统/ 自建短信通知 / 各种商业报警产品

3） 企业监控目前面临的一些问题

- 监控自动化依然不够
- 很少能和CMDB完善的结合起来
- 监控依然需要大量的人工
- 监控的准确性和真实性 提高的缓慢
- 监控工具和方案的制定 较为潦草
- 对监控本身的重视程度 依然有待提高 （其实是 最糟糕）

4） 介绍监控的未来最终理想化

未来理想中最完美的监控 我这里给出两个定义词汇

- 完整自愈式监控体系

监控和报警 总归还是只能发现问题。 出现问题之后的处理 依然需要人工的大量干预
未来当自愈系统完善之后，各种层级的问题 都会被各种自动化 持续集成 人工智能 灾备 系统缓冲 等等技术自行修复

- 真实链路式监控

监控和报警的准确性 真实性 发展到最终级的一个理想化模型

举个例子： 当系统发出报警信息后，往往是各个层级的报警 一大堆一起发出 把真实引起问题的地方掩盖住

非常不利于我们即时的发现和处理问题

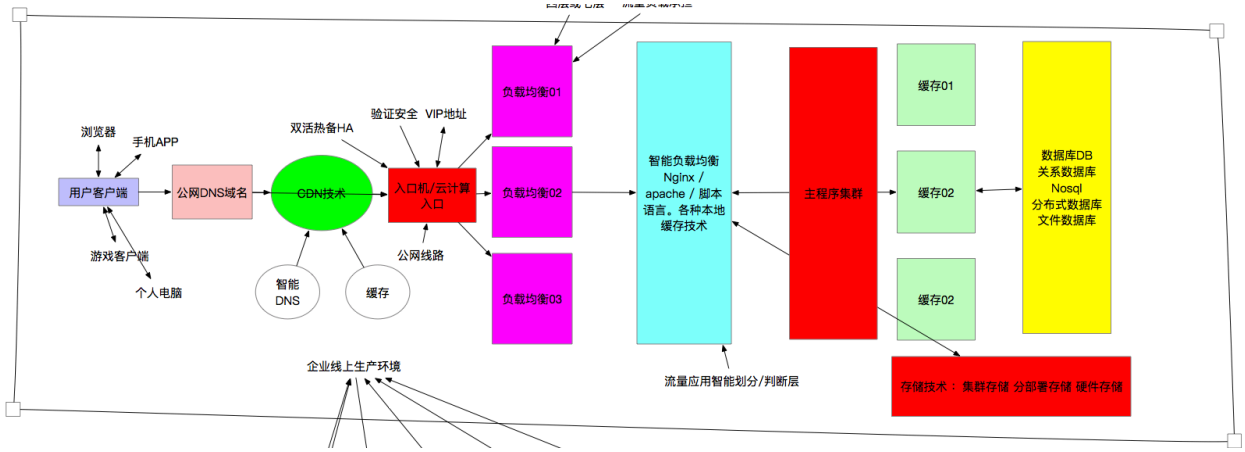
例如：真实发生的问题 是在于 数据库的一个新的联合查询 对系统资源消耗太大

造成各个方面的资源被大量消耗 间接的就引起各种链路的问题

于是乎 各个层面的报警 接踵而至， 日志在报警，慢查询在报警，数据库CPU 内存报警， 程序层TCP链接堆积报警，HTTP返回码5xx 499报警

所有系统CPU 缓存报警， 企业级监控用户流量下降报警

种种一堆一堆被连带出来的报警 全都暴露出来了，结果真正的背后原因 这一个祸根的DB查询反而没有被监控发现 或者说发现了 但是被彻底淹没了



最终理想的未来报警系统 可以把所有无关的报警全部忽略掉，以链路的方式 对问题一查到底 把最终引起问题的地方 报警出来 让运维和开发 即时做出响应和处理

这就是未来的终极化监控报警系统的构想。 （这个理念 说起来容易 实现起来非常难 需要大量的 彻底的跟业务和代码结合起来，并且配合精密的前沿新算法 以及所有运维开发的功能努力 才有可能在未来的5-10年最终实现）

第二讲结束， 第三讲开始 我们正式开始prometheus的学习 敬请期待.