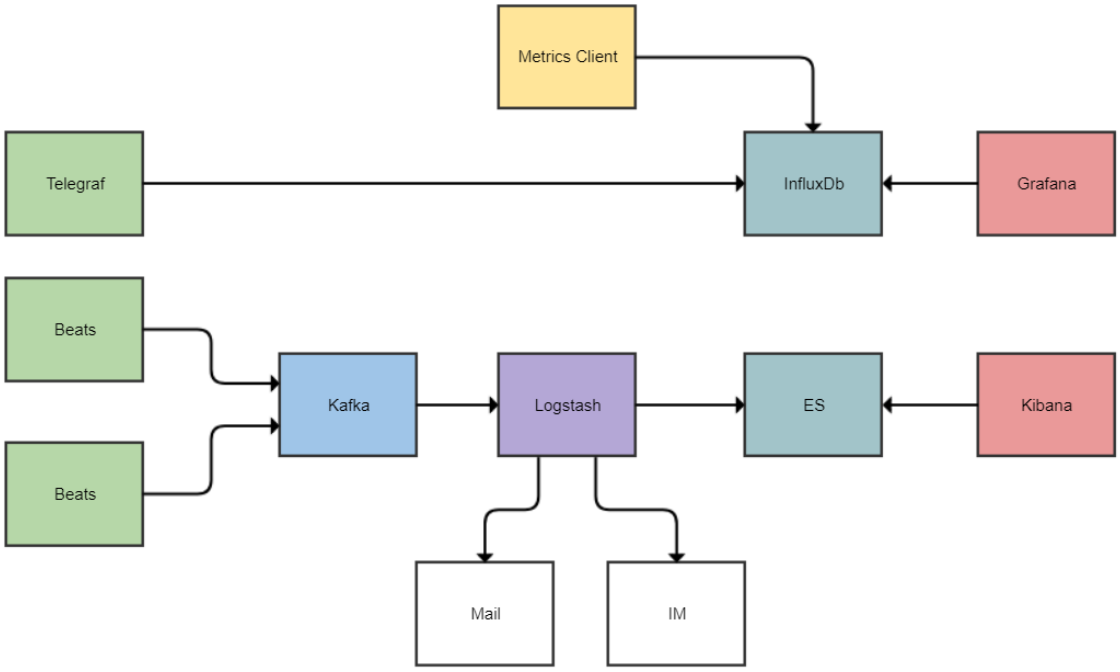
朱晔的互联网架构实践心得 S1E4：简单好用的监控六兄弟

【下载本文 PDF 进行阅读】

这里所说的六兄弟只指 **ELK 套件**（ElasticSearch+Logstash+Kibana）以及 **TIG 套件**（Telegraf+InfluxDb+Grafana）。



上图显示了两套独立的体系，ELK 和 TIG（TIG 是我自己编出来的，网上没有类似于 ELK 这种约定俗成的说法）：

这两套体系都由收集器+存储+展示网站构成，青绿色的收集器，蓝绿色的存储，红色的展示网站。

这两套体系都有免费的组件可以使用，安装配置也相对简单（当然公司也要赚钱，他们肯定都主推 Cloud 版本，一般也不会用 Cloud 版本，肯定本地部署）。

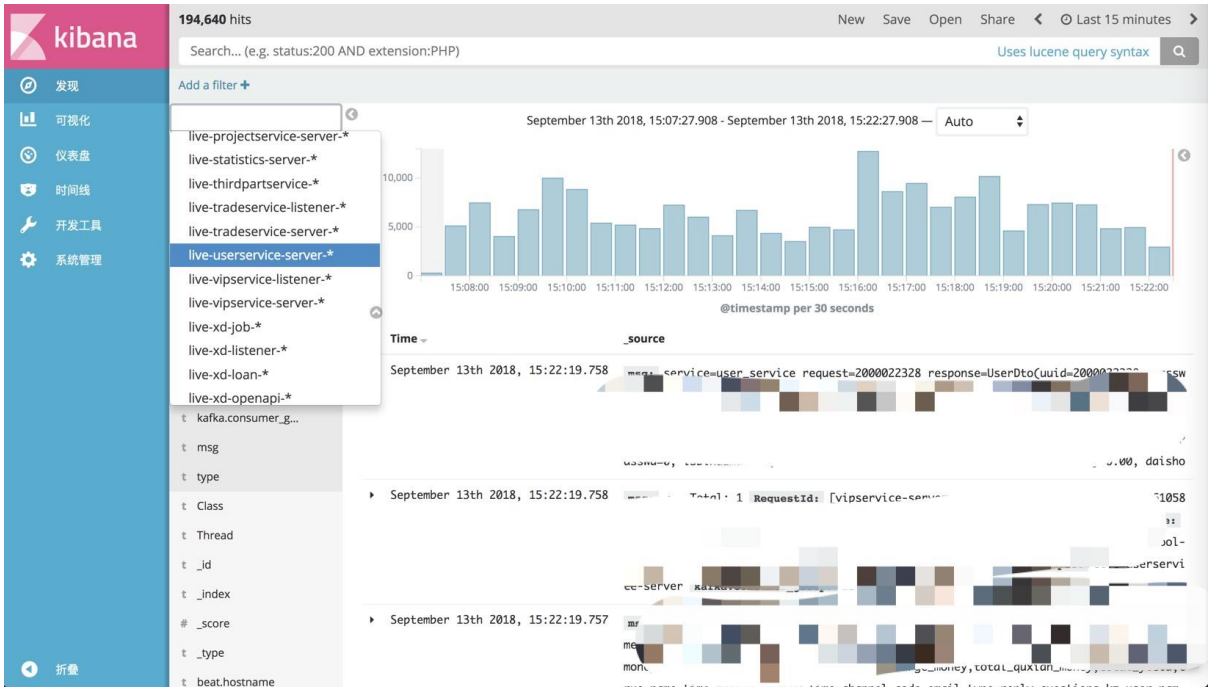
ELK 体系更多用于日志类数据的收集、保存、搜索、查看、报警。

TIG 体系更多用于各种 Metrics 指标类数据的收集、保存、查看、报警。

对于 ELK，由于日志数据量往往较大，并且突发日志激增的情况很普遍，写入索引没有这么快，所以一般会引入 Kafka 之类的消息队列在之前挡一挡。

对于 ELK，在进入 ES 之前数据会有一些过滤解析和额外的报警之类的需求，所以可以使用 logstash 在之前作为一个汇聚处理层，利用丰富的插件做各种处理。但是 logstash 的性能不是那么高，对资源的消耗很厉害，使用的时候需要注意。

有关 ELK



上图是 Kibana 的界面，这里可以看到我们把微服务各个组件的日志都收集到了 ES 中，在 Kibana 上可以使用表达式进行各种搜索，最常用的就是按照串联微服务整个流程的 RequestID 或用户的 UserID 搜索相关日志了。很多公司的开发习惯到服务器上去一台一台搜索日志，好一点会用 ansible 批量搜索，这样其实是非常不方便的：

- 文本的搜索会比 ES 索引数据库的搜索慢的多。
- 文本的搜索遇到文件大的话，占用服务器相当多的内存和 CPU 资源，影响到业务的进行。
- 文件日志一般会进行归档和压缩，想要搜索非当日的日志不那么方便。
- 权限不太好控制，而且原始的文件日志对外开放查询的话可能会有安全问题有信息泄露风险。
- 在把数据统一收集到 ES 的过程中，我们可以做很多额外的工作，包括脱敏，存储到其它数据源，发邮件和 IM 通知（比如可以和 Slack 或钉钉机器人整合）等等。

有关异常

我一直有一个观点，我认为再怎么强调异常都不过分，特别是一直上抛到业务表面的未处理异常以及服务中的系统异常。我们可以把异常区分为业务逻辑主动产生的可以预先知道是咋回事的业务异常以及无法预先知道的系统异常。对于系统异常往往意味着底层基础设施（如网络、数据库、中间件）等有抖动或故障或是代码中有 Bug（即使不是 Bug 也是逻辑不完善的情况），每一个异常，我们都需要逐一进行排查调查出根本原因，如果暂时没有时间调查的话，需要记录在案有时间再去调查。对于有些业务量特别大的系统，每天会有几十万的异常，大概有 100+以上的情况。最差最差那就做到这几点吧：

- 全面梳理代码，千万不要吃掉异常了，往往很多时候 Bug 无法找到原因就是不知道这里吃掉的到底是什么异常。使用 ELK 我们可以很方便搜索过滤日志，多记一点异常或非正常流程的 Error 非常有助于我们修 Bug。
- 我们需要对异常出现的频次进行监控和报警，比如 XXException 最近 1 分钟有 200 条异常，时间久了我们会对这些异常有感觉，看到这样的量我们知道这必然是抖动，如果出现 XXException 最近 1 分钟有 10000 条异常，那么我们知道这不一定是网络抖动了，这是依赖服务挂的节奏，马上需要启动应急响应的排查流程。
- 确保 100%关注和处理好空指针、数组越界、并发错误之类的异常，这每一个异常基本就是一个 Bug 了，会导致业务无法继续的，有的时候这些异常因为绝对数量小会在众多异常中埋没，需要每天单独看这些异常进行逐一解决。这一个异常如果影响到了一个用户正常的流程，那么这个用户可能就流失了，虽然这一个用户只是千万用户中的一员，但是给这一个用户带来的感受是很差的。我一直觉得我们要先于用户发现问题解决问题，最好是等到客服反馈过来的时候（大多数非付费类互联网产品的用户不会因为遇到一个阻碍流程的问题去打客服电话，而是选择放弃这个产品）已经是一个带有修复时间点的已知问题。

做的更好一点甚至我们可以为每一个错误分配一个 ID，如果这个错误有机会透传到用户这端，在 500 页面上不那么明显的地方显示一下这个 ID，如果用户截屏反馈问题的话，可以轻易通过这个错误 ID 在 ELK 中找到相应错误，一键定位问题。

有关 TIG

上图是 Grafana 的截图，Grafana 支持挺多数据源，InfluxDb 也是其中的一个数据源，类似于 InfluxDb 的产品还有 Graphite，也是不错的选择。Telegraf 是 InfluxDb 公司的收集数据的 Agent 套件，会有相当多的插件，这些插件并不复杂，自己也可以通过 Python 简单编写，就是有点费时间，有现成的么就用，说白了就是从各个中间件暴露出来的 Stats 接口收集格式化数据然后写入 InfluxDb 中去。我们来看看 Telegraf 支持的插件（图片截取自 https://github.com/influxdata/telegraf）：

Input Plugins

- activemq
- aerospike
- amqp_consumer (rabbitmq)
- apache
- aurora
- aws cloudwatch
- bcache
- beanstalkd
- bond
- burrow
- cassandra (deprecated, use jolokia2)
- ceph
- cgroup
- chrony
- conntrack
- consul
- couchbase
- couchdb
- cpu
- DC/OS
- diskio
- disk
- disque
- dmcache
- dns query time
- docker
- dovecot
- elasticsearch
- exec (generic executable plugin, support JSON, influx, graphite and nagios)
- fail2ban
- fibaro

使用这些插件运维或开发自己不需要费什么力气就可以把我们所有的基础组件都监控起来了。

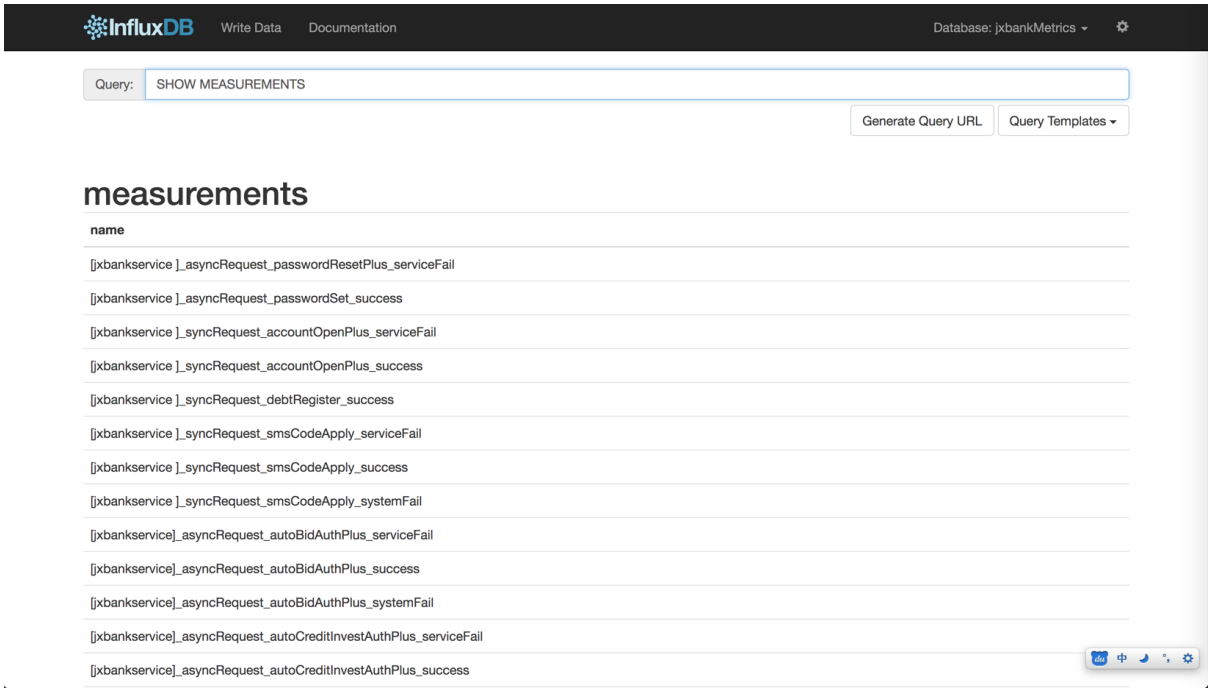
有关打点

如文本一开始的架构图所示，除了我们可以使用 Telegraf 的各种插件来收集各种存储、中间件、系统层面的指标之外，我们还做了一个 MetricsClient 小类库，让程序可以把各种打点的数据保存到 InfluxDb。其实每一条进入 InfluxDb 的 Measurement 记录只是一个事件，有下面这些信息：

- 时间戳

- 各种用于搜索的 Tag
- 值（耗时、执行次数）

如下图我们可以看到在这个 bankservice 中，我们记录了各种异步同步操作的成功、业务异常、系统异常事件，然后在 Grafana 进行简单的配置，就可以呈现出需要的图了。



对于 MetricsClient，可以在代码中手工调用也可以使用 AOP 的方式进行调用，我们甚至可以为所有方法加上这个关注点，自动收集方法的执行次数、时间、结果（正常、业务异常、系统异常）打点记录到 InfluxDb 中，然后在 Grafana 配置自己需要的 Dashboard 用于监控。

对于 RPC 框架也是建议框架内部自动整合打点的，保存 RPC 方法每次执行的情况，细化到方法的粒度配置出一些图表来，在出现事故的时候一键定位到疑似出问题的方法。通过 AOP 方+RPC 框架自动打点其实已经可以覆盖大部分需求了，当然如果我们在代码中再加一些业务层面的打点就更好了。

如果我们为每一个业务行为，配置两个图，一个是调用量，一个是调用性能，如下图：

那么：

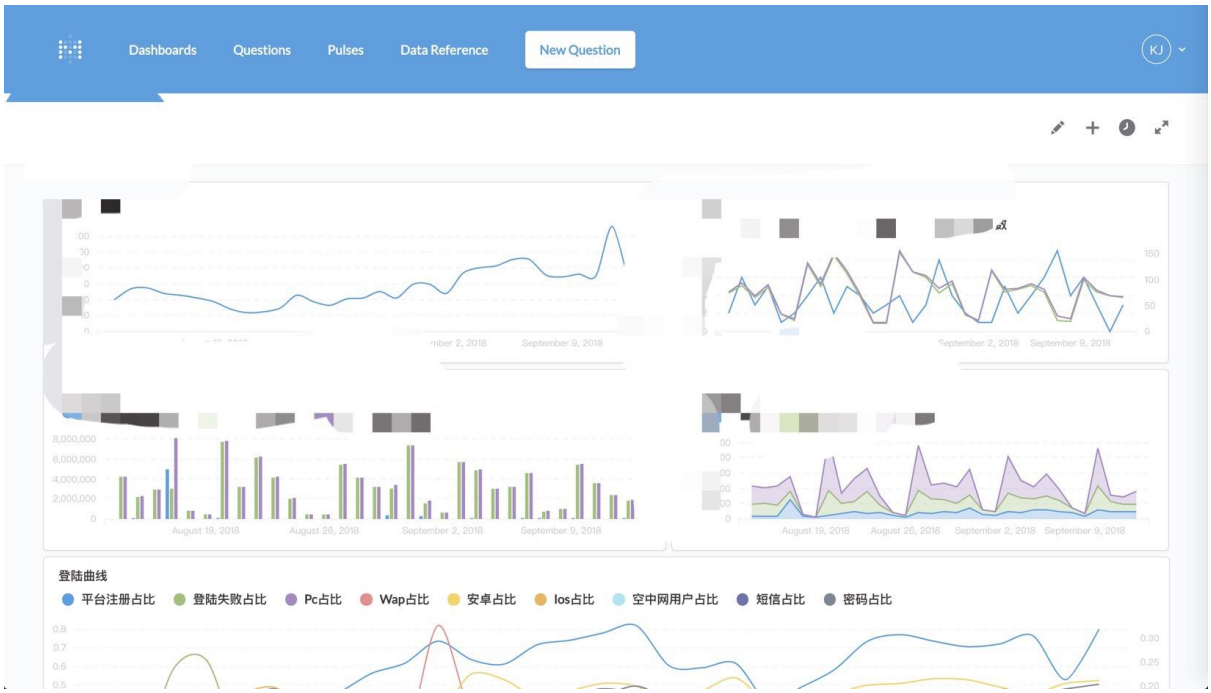
- 出现问题的时候，我们可以在很短的时间内判断出哪块有问题。
- 还可以初步判断出问题的原因是异常导致还是突增的压力所致。

这里推荐的配置方式是根据数据流，从前到后，每一个环节配置一下数据处理的数量和性能：

- 上游进来的数据
- 发送到 MQ 的数据
- MQ 接收到的数据
- MQ 处理完成的数据
- 和外部交互的请求
- 得到外部响应的请求
- 落库的请求
- 查缓存的请求

出了问题可以及时定位到出问题的模块，或至少是业务线，会比无头苍蝇好很多（当然，如果我们没有事先配置自己需要的 Dashboard 那也是白搭）。Dashboard 一定是需要随着业务的迭代不断去维护的，别经过几轮迭代之前的打点早已废弃，到出了问题的时候再看 Dashboard 全是 0 调用。

其它



Grafana 对接 InfluxDb 数据源挺好的，但是对接 MySQL 做一些查询总感觉不是特别方便，这里推荐一个开源的系统 Metabase，我们可以方便得保存一些 SQL 进行做一些业务或监控之类的统计。你可能会说了，这些业务统计是运营关注的，而且我们由 BI，我们需要自己做这些图表干啥，我想说我们即使搞技术也最好有一个自己的小业务面板，不是说关注业务量而是能有一个地方让我们知道业务跑的情况，在关键的时候看一眼判断一下影响范围。

好了，说到这里，你是否已看到了通过这六兄弟，其实我们打造的是一个立体化的监控体系，分享一个排查问题的几步走方式吧，毕竟在出大问题的时候我们的时间往往就只有那么几分钟：

- 关注异常或系统层面的压力报警，关注业务量掉 0（指的是突然下落 30%以上）报警。
- 通过 Grafana 面板配置的业务 Dashboard 判断系统哪个模块有压力问题、性能问题。
- 通过 Grafana 面板配置的服务调用量和业务进出量，排除上下游问题，定位出问题的模块。
- 通过 Kibana 查看相应模块是否出现错误或异常。

- 根据客户反馈的错误截屏，找到错误 ID，在 Kibana 中搜索全链路日志找问题。
- 对于细节问题，还有一招就是查请求日志了。我们可以在 Web 端的系统做一个开关，根据一定的条件可以开启记录详细的 Request 和 Response HTTP Log 的开关，有了每一个请求详细的数据，我们可以根据用户信息“看到”用户访问网站的整个过程，这非常有助于我们排查问题。当然，这个数据量可能会非常大，所以需要慎重开启这么重的 Trace 功能。

有打点、有错误日志、有详细请求日志，还怕定位不到问题？