分布式之elk日志架构的演进

孤独烟 架构师小秘圈 1周前

> 作者：孤独烟
> 来自：打杂的ZRJ

引言

好久没写分布式系列的文章了，最近刚好有个朋友给我留言，想看这方面的知识。其实这方面的知识，网上各种技术峰会的资料一抓一大把。博主也是凑合着写写。感觉自己也写不出什么新意，大家也凑合看看。

日志系统的必要性?

我15年实习的时候那会，给某国企做开发。不怕大家笑话，生产上就两台机器。那会定位生产问题，就是连上一台机器，然后用使用 `grep / sed / awk` 等 Linux 脚本工具去日志里查找故障原因。如果发现不在这台机器上，就去另一台机器上查日志。有经历过上述步骤的童鞋们，请握个抓！
然而，当你的生产上是一个有几千台机器的集群呢？你要如何定位生产问题呢？又或者，你哪天有这么一个需求，你需要收集某个时间段内的应用日志，你应该如何做?
为了解决上述问题，我们就需要将日志集中化管理。这样做，可以提高我们的诊断效率。同时也有利于我们全面理解系统。

正文

组件简介

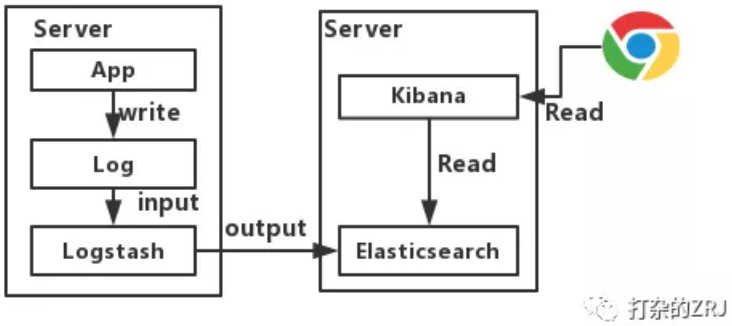
这里大概介绍一下ELK组件在搭建日志系统过程中所扮演的角色，这边了解一下即可，具体的会在后文进行说明。
大家应该都知道ELK指的是:（Elasticsearch＋Logstash＋Kibana）。其中

- `Logstash` :负责采集日志
- `Elasticsearch` :负责存储最终数据、建立索引、提供搜索功能
- `Kibana` ： 负责提供可视化界面

好了，知道上面的定义，可以开始讲演进过程了

实习版

OK，这版算是Demo版，各位开发可以在自己电脑上搭建练练手，如下图所示。



这种架构下我们把 Logstash实例与Elasticsearch实例直接相连，主要就是图一个简单。我们的程序App将日志写入Log，然后**Logstash将Log读出，进行过滤，写入Elasticsearch**。最后浏览器访问Kibana，提供一个可视化输出。

缺点

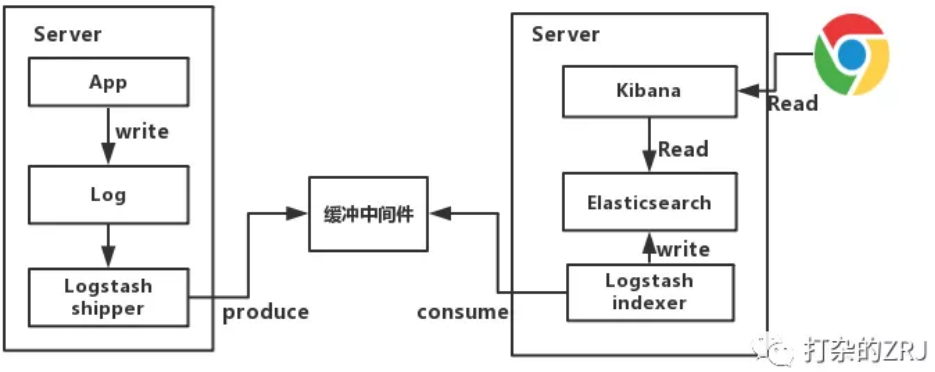
该版的缺点主要是两个

- 在大并发情况下，日志传输峰值比较大。如果直接写入ES,ES的HTTP API处理能力有限，在日志写入频繁的情况下可能会超时、丢失，所以需要一个缓冲中间件。
- 注意了，Logstash将Log读出、过滤、输出都是在应用服务器上进行的，这势必会造成服务器上占用系统资源较高，性能不佳，需要进行拆分。

于是，我们的初级版诞生了！

初级版

在这版中，加入一个缓冲中间件。另外对Logstash拆分为Shipper和Indexer。先说一下，LogStash自身没有什么角色，只是根据不同的功能、不同的配置给出不同的称呼而已。Shipper来进行日志收集，Indexer从缓冲中间件接收日志，过滤输出到Elasticsearch。具体如下图所示



说一下，这个缓冲中间件的选择。

大家会发现，早期的博客，都是推荐使用redis。因为这是ELK Stack 官网建议使用 Redis 来做消息队列，但是很多大佬已经通过实践证明使用Kafka更加优秀。原因如下:

- `Redis` 无法保证消息的可靠性，这点 `Kafka` 可以做到
- `Kafka` 的吞吐量和集群模式都比 `Redis` 更优秀

- `Redis`受限于机器内存，当内存达到Max，数据就会抛弃。当然，你可以说我们可以加大内存啊？但是，在`Redis`中内存越大，触发持久化的操作阻塞主线程的时间越长。相比之下，`Kafka`的数据是堆积在硬盘中，不存在这个问题。

因此，综上所述，这个缓存中间件，我们选择使用`Kafka`。

缺点

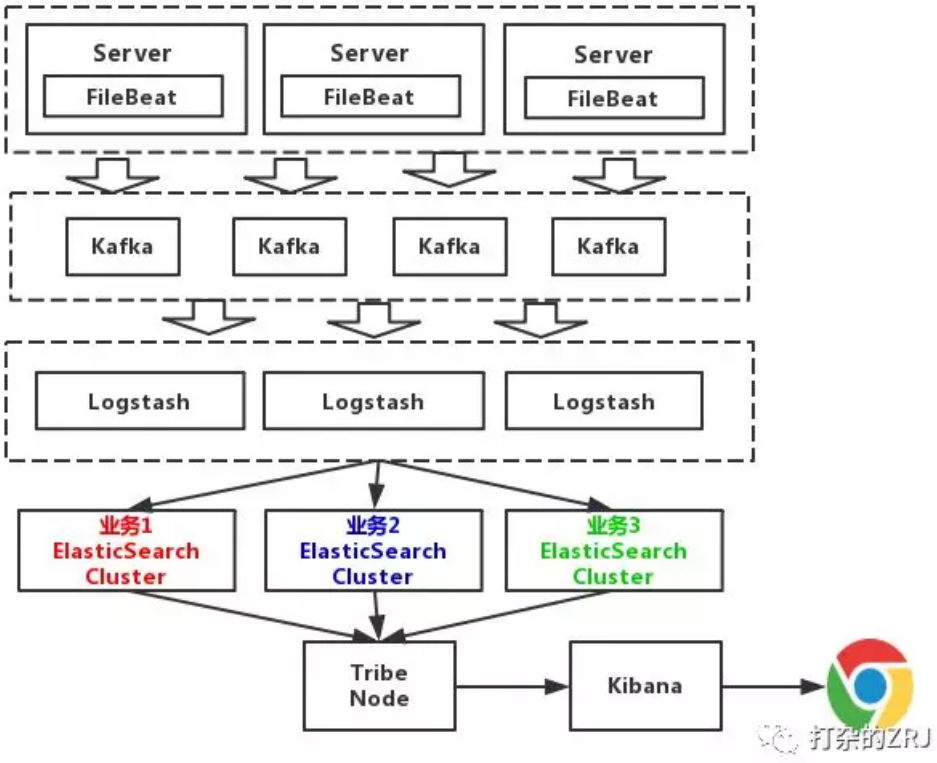
主要缺点还是两个

- `Logstash Shipper`是jvm跑的，非常占用`JAVA`内存！ 。据《ELK系统使用filebeat替代logstash进行日志采集》这篇文章说明，8线程8GB内存下，`Logstash`常驻内存660M（`JAVA`）。因此，这么一个巨无霸部署在应用服务器端就不大合适了，我们需要一个更加轻量级的日志采集组件。
- 上述架构如果部署成集群，所有业务放在一个大集群中相互影响。一个业务系统出问题了，就会拖垮整个日志系统。因此，需要进行业务隔离！

中级版

这版呢，引入组件Filebeat。当年，Logstash的作者用golang写了一个功能较少但是资源消耗也小的轻量级的Logstash-forwarder。后来加入Elasticsearch后，以logstash-forwarder为基础，研发了一个新项目就叫Filebeat。

相比于Logstash，Filebeat更轻量，占用资源更少，所占系统的 CPU 和内存几乎可以忽略不计。毕竟人家只是一个二进制文件。那么，这一版的架构图如下，我直接画集群版



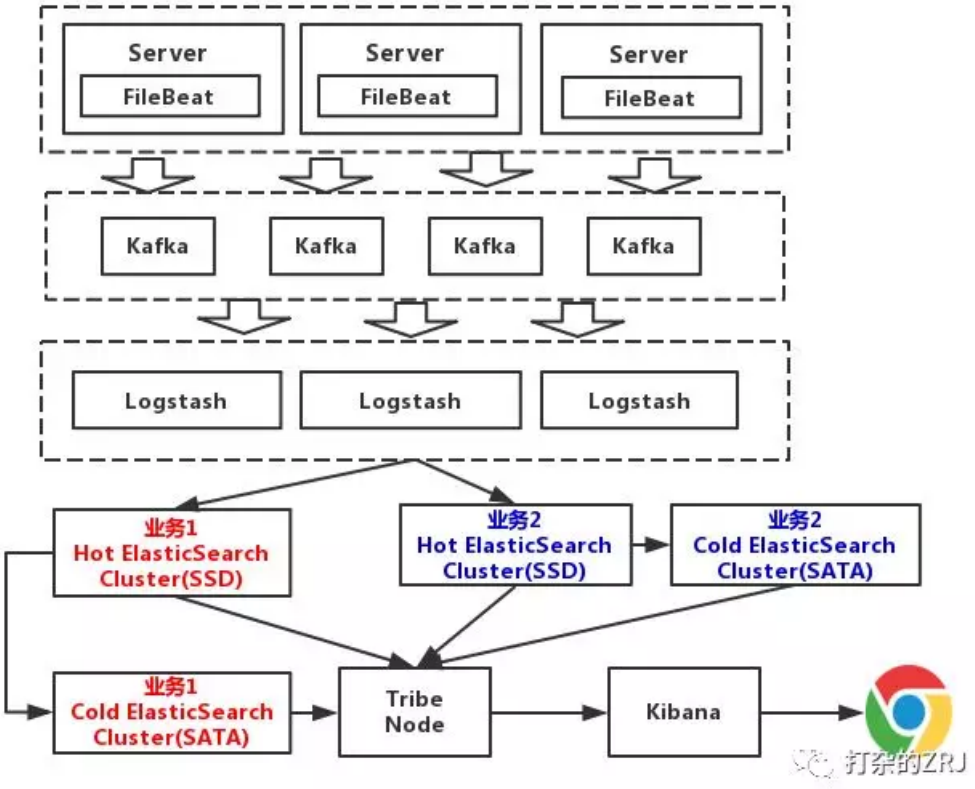
至于这个**Tribe Node**，中文翻译为部落结点，它是一个特殊的客户端，它可以连接多个集群，在所有连接的集群上执行搜索和其他操作。在这里呢，负责将请求路由到正确的后端ES集群上。

缺点

这套架构的缺点在于对日志没有进行冷热分离。因为我们一般来说，对一个礼拜内的日志，查询的最多。以7天作为界限，区分冷热数据，可以大大的优化查询速度。

高级版

这一版，我们对数据进行冷热分离。每个业务准备两个Elasticsearch集群，可以理解为冷热集群。7天以内的数据，存入热集群，以SSD存储索引。超过7天，就进入冷集群，以SATA存储索引。这么一改动，性能又得到提升，这一版架构图如下(为了方便画图，我只画了两个业务Elasticsearch集群)



隐患

这个高级版，非要说有什么隐患，就是敏感数据没有进行处理，就直接写入日志了。关于这点，其实现在JAVA这边，现成的日志组件，比如 `log4j` 都有提供这种日志过滤功能，可以将敏感信息进行脱敏后，再记录日志。