

从 ELK 到 EFK

(<https://www.tuicool.com/>)
时间 2017-10-18 07:02:44 公众账号

原文 http://mp.weixin.qq.com/s/UMzq0Mt2_nm5pWn1Spba3Q (http://mp.weixin.qq.com/s/UMzq0Mt2_nm5pWn1Spba3Q?utm_source=tuicool&utm_medium=referral)

主题 Logstash (/topics/11140060) ELK Stack (/topics/11400067) Elasticsearch (/topics/11020003)

作者：曹林华
本文为原创文章，转载请注明作者及出

背景

作为中国最大的在线教育站点，目前沪江日志服务的用户包含沪江网校，交易，金融，CCtalk（直播平台）等多个部门的多个产品的日志搜索分析业务，每日产生的各类日志有好十几种，每天处理约10亿条（1TB）日志，热数据保留最近7天数据，冷数据永久保存。

为什么做日志系统

首先，什么是日志？日志就是程序产生的，遵循一定格式（通常包含时间戳）的文本数据

通常日志由服务器生成，输出到不同的文件中，一般会有系统日志、应用日志、安全日志。这些日志分散地存储在不同的机器上。

通常当系统发生故障时，工程师需要登录到各个服务器上，使用 `grep / sed / awk` 等 Linux 脚本工具去日志里查找故障原因。在没有日志系统的情况下，首先需要定位处理请求的服务

器，如果这台服务器部署了多个实例，则需要去每个应用实例的日志目录下去找日志文件。每个应用实例还会设置日志滚动策略（如：每天生成一个文件），还有日志压缩归档策略等。

我认为，日志数据在以下几方面具有非常重要的作用：

- 数据查找：通过检索日志信息，定位相应的 bug，找出解决方案
- 服务诊断：通过对日志信息进行统计、分析，了解服务器的负荷和服务运行状态
- 数据分析：可以做进一步的数据分析，比如根据请求中的课程 id，找出 TOP10 用户感兴趣课程。

针对这些问题，为了提供分布式的实时日志搜集和分析的监控系统，我们采用了业界通用的日志数据管理解决方案 - 它主要包括 Elasticsearch、Logstash 和 Kibana 三个系统。通常，业界把这套方案简称为ELK，取三个系统的首字母，但是我们实践之后将其进一步优化为EFK，F代表Filebeat，用以解决Logstash导致的问题。下面，我们展开详细介绍。

文中涉及的 ELK stack 版本是：

Elasticsearch 5.2.2
Logstash 5.2.2
Kibana 5.2.2
Filebeat 5.2.2
Kafka 2.10



logstash kibana elasticsearch beats

Logstash：数据收集处理引擎。支持动态的从各种数据源搜集数据，并对数据进行过滤、分析、丰富、统一格式等操作，然后存储以供后续使用。

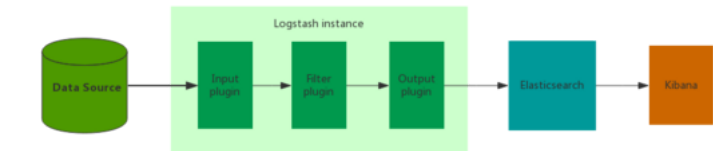
Kibana：可视化平台。它能够搜索、展示存储在 Elasticsearch 中索引数据。使用它可以很方便的用图表、表格、地图展示和分析数据。

Elasticsearch：分布式搜索引擎。具有高可伸缩、高可靠、易管理等特点。可以用于全文检索、结构化检索和分析，并能将这三者结合起来。Elasticsearch 基于 Lucene 开发，现在使用最广的开源搜索引擎之一，Wikipedia、StackOverflow、Github 等都基于它来构建自己的搜索引擎。

Filebeat：轻量级数据收集引擎。基于原先 Logstash-fowarder 的源码改造出来。换句话说：Filebeat就是新版的 Logstash-fowarder，也会是 ELK Stack 在 shipper 端的第一选择。

既然要谈 ELK 在沪江系统中的应用，那么 ELK 架构就不得不谈。本次分享主要列举我们曾经用过的 ELK 架构，并讨论各种架构所适合的场景和优劣供大家参考

简单版架构



这种架构下我们把 Logstash 实例与 Elasticsearch 实例直接相连。Logstash 实例直接通过 Input 插件读取数据源数据(比如 Java 日志， Nginx 日志等)，经过 Filter 插件进行过滤日志，最后通过 Output 插件将数据写入到 ElasticSearch 实例中。

这个阶段，日志的收集、过滤、输出等功能，主要由这三个核心组件组成 Input、Filter、Output

Input: 输入，输入数据可以是 File、Stdin（直接从控制台输入）、TCP、Syslog、Redis、Collectd 等

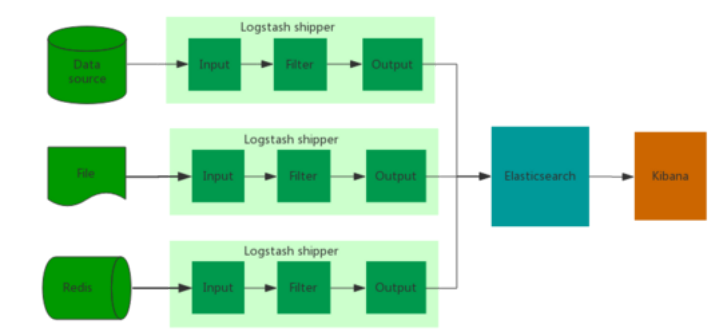
Filter: 过滤，将日志输出成我们想要的格式。Logstash 存在丰富的过滤插件: Grok 正则捕获、时间处理、JSON 编解码、数据修改 Mutate。Grok 是 Logstash 中最重要的插件，强烈建议每个人都要使用 Grok Debugger 来调试自己的 Grok 表达式

```
grok {
    match => ["message", "(?m)\[%{LOGLEVEL:level}\] \[%{TIMESTAMP_ISO8601:timestamp}\] \[%{DATA:logger}\] \[%{DATA:threadId}\] \[%{DATA:requestId}\] %GREEDYDATA:msgRawData"}]
}
```

Output: 输出，输出目标可以是 Stdout（直接从控制台输出）、Elasticsearch、Redis、TCP、File 等

这是最简单的一种ELK架构方式，Logstash 实例直接与 Elasticsearch 实例连接。优点是搭建简单，易于上手。建议供初学者学习与参考，不能用于线上的环境。

集群版架构



这种架构下我们采用多个 Elasticsearch 节点组成 Elasticsearch 集群，由于 Logstash 与 Elasticsearch 采用集群模式运行，集群模式可以避免单实例压力过重的问题，同时在线上各个服务器上部署 Logstash Agent，来满足数据量不大且可靠性不强的场景。

数据收集端：每台服务器上面部署 Logstash Shipper Agent 来收集当前服务器上日志，日志经过 Logstash Shipper 中 Input插件、Filter插件、Output 插件传输到 Elasticsearch 集群

数据存储与搜索：Elasticsearch 配置默认即可满足，同时我们看数据重要性来决定是否添加副本，如果需要的话，最多一个副本即可

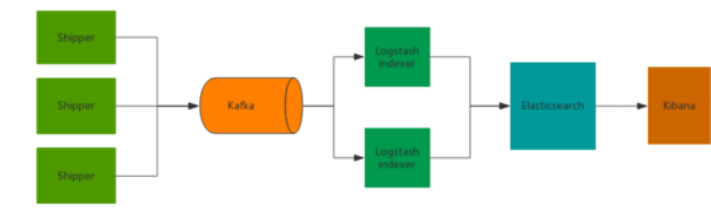
数据展示：Kibana 可以根据 Elasticsearch 的数据来做各种各样的图表来直观的展示业务实时状况

这种架构使用场景非常有限，主要存在以下两个问题

- 消耗服务器资源：Logstash 的收集、过滤都在服务器上完成，这就造成服务器上占用系统资源较高、性能方面不是很好，调试、跟踪困难，异常处理困难
- 数据丢失：大并发情况下，由于日志传输峰值比较大，没有消息队列来做缓冲，就会导致 Elasticsearch 集群丢失数据

这个架构相对上个版本略微复杂，不过维护起来同样比较方便，同时可以满足数据量不大且可靠性不强的业务使用。

引入消息队列



该场景下面，多个数据首先通过 Logstash Shipper Agent 来收集数据，然后经过 Output 插件将数据投递到 Kafka 集群中，这样当遇到 Logstash 接收数据的能力超过 Elasticsearch 集群处理能力的时候，就可以通过队列就能起到削峰填谷的作用，Elasticsearch 集群就不存在丢失数据的问题。

目前业界在日志服务场景中，使用比较多的两种消息队列为：Kafka VS Redis。尽管 ELK Stack 官网建议使用 Redis 来做消息队列，但是我们建议采用 Kafka。主要从下面两个方面考虑：

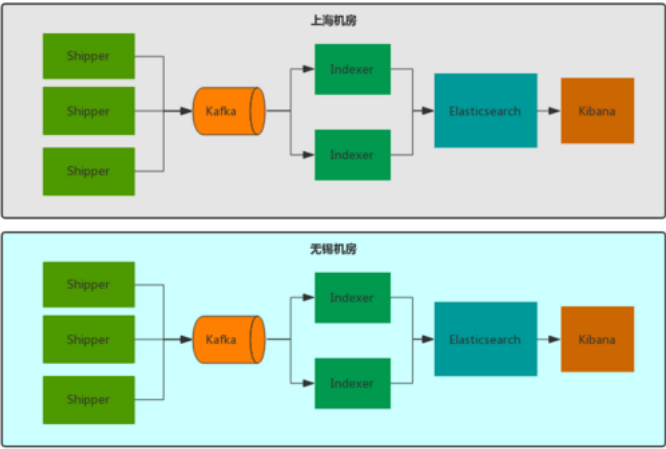
- 数据丢失：Redis 队列多用于实时性较高的消息推送，并不保证可靠。Kafka保证可靠但有点延时。
- 数据堆积：Redis 队列容量取决于机器内存大小，如果超过设置的Max memory，数据就会抛弃。Kafka 的堆积能力取决于机器硬盘大小。

综合上述的理由，我们决定采用 Kafka 来缓冲队列。不过在这种架构下仍然存在一系列问题

- Logstash shipper 收集数据同样会消耗 CPU 和内存资源
- 不支持多机房部署

这种架构适合较大集群的应用部署，通过消息队列解决了消息丢失、网络堵塞的问题。

多机房部署

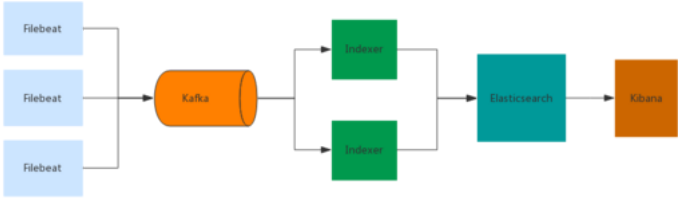


随着沪江业务的飞速增长，单机房的架构已经不能满足需求。不可避免的，沪江的业务需要分布到不同机房中，对于日志服务来说也是不小的挑战。当然业界也有不少成熟的方法，比如阿里的单元化、腾讯的 SET 方案等等。单元化在这边不详细展开，大家可以参考微博的【单元化架构】。

最终我们决定采用单元化部署的方式来解决 ELK 多机房中遇到的问题(延时、专线流量过大等)，从日志的产生、收集、传输、存储、展示都是在同机房里面闭环消化，不存在跨机房传输与调用的问题。因为交互紧密的应用尽量部署在同机房，所以这种方案并不会给业务查询造成困扰。

Logstash、Elasticsearch、Kafka、Kibana 四个集群都部署到同一机房中，每个机房都要每个机房自己的日志服务集群，比如A机房业务的日志只能传输给本机房 Kafka，而A机房 Indexer 集群消费并写入到A机房 Elasticsearch 集群中，并由A机房 Kibana 集群展示，中间任何一个步骤不依赖B机房任何服务。

引入Filebeat



Filebeat 是基于原先 logstash-forwarder 的源码改造出来的，无需依赖 Java 环境就能运行，安装包10M不到。

如果日志的量很大，Logstash 会遇到资源占用高的问题，为解决这个问题，我们引入了Filebeat。Filebeat 是基于 logstash-forwarder 的源码改造而成，用 Golang 编写，无需依赖 Java 环境，效率高，占用内存和 CPU 比较少，非常适合作为 Agent 跑在服务器上。

下面看看Filebeat的基本用法。编写配置文件，从 Nginx access.log 中解析日志数据：

```
# filebeat.yml
filebeat.prospectors:
- input_type: log
  paths: /var/log/nginx/access.log
  json.message_key:

output.elasticsearch:
  hosts: ["localhost"]
  index: "filebeat-nginx-%{+yyyy.MM.dd}"
```

我们来看看压测数据：

压测环境

- 虚拟机 8 cores 64G内存 540G SATA盘
- Logstash 版本 2.3.1
- Filebeat 版本 5.5.0

压测方案

Logstash / Filebeat 读取 350W 条日志 到 console，单行数据 580B，8个进程写入采集文件

压测结果

项目	workers	cpu usr	总共耗时	收集速度
Logstash	8	53.7%	210s	1.6w line/s
Filebeat	8	38.0%	30s	11w line/s

Filebeat 所消耗的CPU只有 Logstash 的70%，但收集速度为 Logstash 的7倍。从我们的应用实践来看，Filebeat 确实用较低的成本和稳定的服务质量，解决了 Logstash 的资源消耗问题。

最后，分享给大家一些血泪教训，希望大家以我为鉴。

1. Indexer 运行一段时间后自动挂掉

突然有一天监控发现日志不消费了，排查下来发现消费 Kafka 数据的 indexer 挂掉了。所以，Indexer 进程也是需要 supervisor 来监控的，保证它时刻都在运行。

2. Java异常日志输出

开始我们在通过 grok 切割日志的时候，发现 Java 的 Exception 日志输出之后，会出现换行的问题。后来使用 Logstash **codec/multiline** 插件来解决。

```
input {
  stdin {
    codec => multiline {
      pattern => "^\[["
      negate => true
      what => "previous"
    }
  }
}
```

3. 由于时区导致日志8小时时差

Logstash 2.3版本 date插件配置如下，查看解析结果发现@timestamp比中国时间早了8小时。

解决方案 Kibana 读取浏览器的当前时区，然后在页面上转换时间内容的显示。

```
date {
  match => [ "log_timestamp", "YYYY-MM-dd HH:mm:ss.SSS" ]
  target => "@timestamp"
}
```

4.Grok parse failure

我们遇到线上 node 日志突然有几天日志查看不出来。后来拉出原始日志对比才发现生成出来的日志格式不正确，同时包含 JSON 格式和非 JSON 格式的日志。但是我们用grok解析的时候采用是 json 格式。建议大家输出日志保证格式一致同时不要出现空格等异常字符，可以使用在线 grok debug (<http://grokdebug.herokuapp.com/>) 来调试正则。

总结

基于 ELK stack 的日志解决方案的优势主要体现于：

- 可扩展性：采用高可扩展性的分布式系统架构设计，可以支持每日 TB 级别的新增数据。
- 使用简单：通过用户图形界面实现各种统计分析功能，简单易用，上手快
- 快速响应：从日志产生到查询可见，能达到秒级完成数据的采集、处理和搜索统计。
- 界面炫丽：Kibana 界面上，只需要点击鼠标，就可以完成搜索、聚合功能，生成炫丽的仪表板。

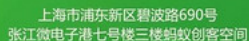


参考资料

- <https://www.elastic.co/guide/en/beats/filebeat/1.3/filebeat-overview.html>
- <https://zhuanlan.zhihu.com/p/26399963>

技术沙龙推荐

点击下方图片即可阅读



● ● ● ● ● ● ●



- 1. FACT：一款固件类分析测试平台 (/articles/ljylVbY)
- 2. FACT：一款固件类分析测试平台 (/articles/3yYneuA)
- 3. Damon Edwards：IT运营是最可预测的DevOps差异化因素 (/articles/Ur6vYvR)
- 4. 18种适合于各种层次开发人员的PHP工具 (/articles/73En2qU)
- 5. DevOps采用现状情况报告 (/articles/qFnIrlz)
- 6. 开源SSH双因素登陆认证系统JXOTP了解一下 (/articles/RNrqgeN)

JS

Object.assign()

+

<

>

JS

Object.assign()

Object.assign()

Web technology for developers >

JavaScript >


JavaScript reference >

Standard built-in objects >

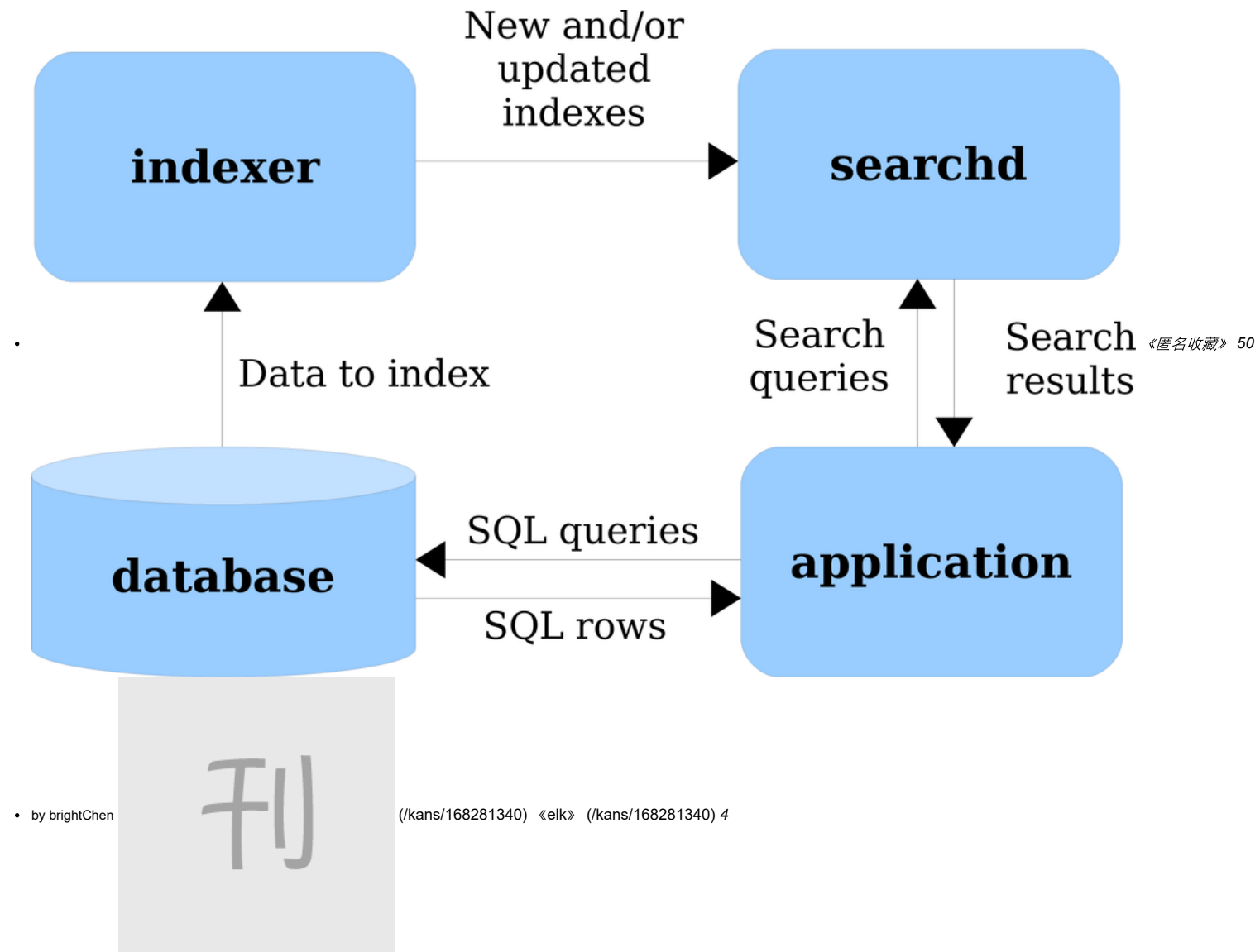
Object >

Object.assign()

The **Object.assign()** method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.



```
1 const object1 = {
2   a: 1,
3   b: 2,
4   c: 3
5 };
6
7 const object2 = Object.assign({c: 4, d: 5}, object1);
8
9 console.log(object2.c, object2.d);
10 // expected output: 3 5
```



我来评几句

登录后评论

已发表评论数(0)

相关站点



公众账号 (/sites/ilzq6rE)

+ 订阅

热门文章

- 1. 高效客户端持续集成实践之路 (/articles/vuENbu3)
- 2. FACT：一款固件类比分析测试平台 (/articles/ljyIVbY)
- 3. FACT：一款固件类比分析测试平台 (/articles/3yYneuA)
- 4. Damon Edwards：IT运营是最可预测的DevOps差异化因素 (/articles/Ur6vYvR)
- 5. 18种适合于各种层次开发人员的PHP工具 (/articles/73En2qU)

关于我们 (<https://www.tuicool.com/about>) 移动应用 (<https://www.tuicool.com/mobile>) 意见反馈 (<https://www.tuicool.com/bbs/go/issues>) 官方微博 (<https://weibo.com/tuicool2012>)