ELK 实用技巧

分享篇

目錄

elk-in-action

> 工具，只是手段；做事，还得靠人。

关于**elk**

elk是什么

- elasticsearch、logstash、kibana的首字母
- 集日志收集、全文索引、日志分析于一身
- 崇尚'less is more'的哲学
- 开源、活跃

为什么要监控

> API服务可用性到底几个9，集群中的机器是否有浪费，下一波业务的高峰要不要添置机器？

在互联网行业，分布式环境天然的多样性，无中心化的设计

何谓 `elk`

在互联网行业有一种不成文的文化，`新名词`横行霸道。

elk本身也不是什么新技术，只是三种技术的简称。

为什么选择 ELK

- 足够简单（安装、部署等环节）
- 插件丰富（主流中间件，log范式均知此）
- 灵活，方便扩展
- 开源，社区活跃

> 用户群体知名度高
>
> - github
> - 维基百科
> - stackoverflow

那么，问题来了？还有比选择它更靠谱的吗？

怎么用

> 几经纠结，决心好好用下。那么，最好的方式就是调教。

规划

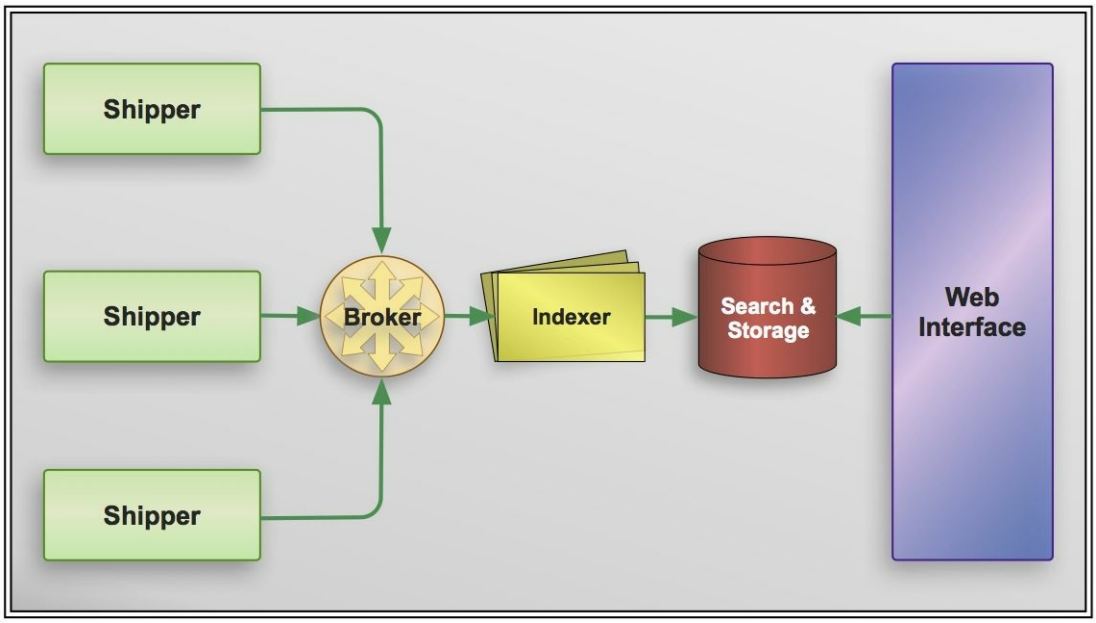
> 最终数据需要落到elasticsearch集群上，合理的规划机器将使你事半功倍。

1. logstash采集器，直接采用默认配置在目标机器上部署就好
2. elaticsearch需要花点功夫，好好规划下集群及切片设置
3. `redis`足以胜任broker角色

Logstash

安装

> 目标机器基本环境
>
> - ruby 1.8.7+
> - java 1.6+ （推荐1.7以上）



shipper等价于应用机器上的agent，通过监听事件统一规整到Broker(相当于一个buffer)，indexer是就是logstash的server部分。本身上来讲logstash不细分角色，其input-filter-output的机制，灵活度很高。对于Storage部分，Elasticsearch提供了全文索引，最后通过Kibana展现。

何为 `log`

@timestamp + data

log的生命周期

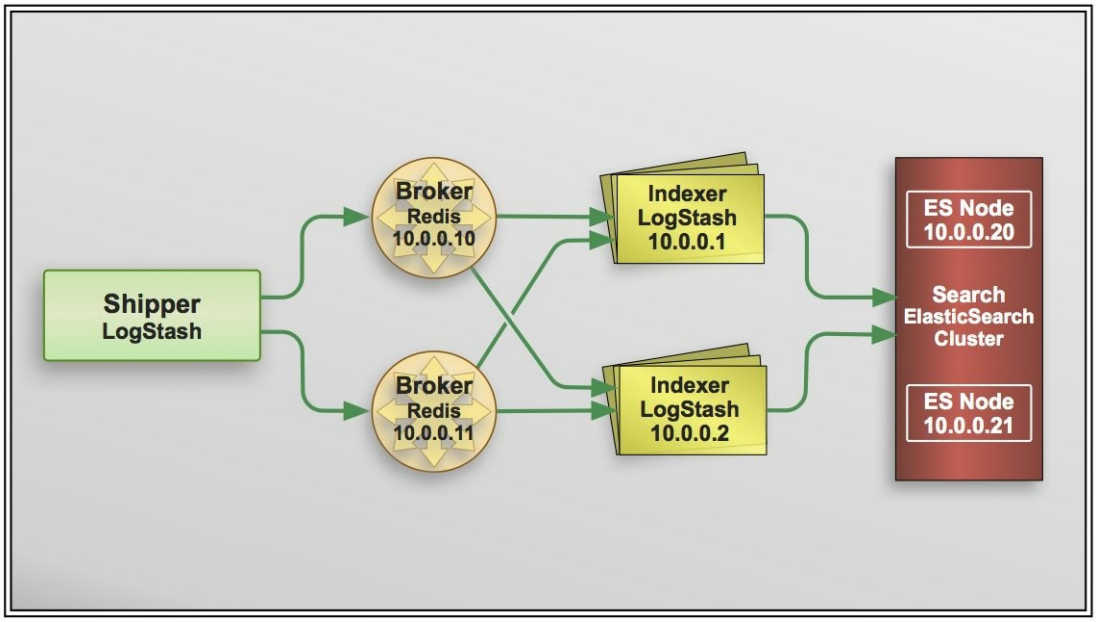
- 记录 --> 删除
- 记录 --> 收集 --> 分析 --> 删除

> 持续改善，才能释放生产力。

几种角色

input -> filter -> output

一个符合SLA协议的部署结构大致可以是这样的，官方参考。



收集器 shipper

部署在`目标机器`上的logstash，实时监控log文件的变化，并将其传递给中间人（建议的做法）。

代理人 broker

`broker`在这里，更多的用途是`解耦`。

索引器 indexer

indexer将broker中的数据通过`elasticsearch`的output插件持久化到`elasticsearch`集群。

Tips

- 了解你所要收集的数据，合理规划（多思考）
- 删除信息冗余的字段(eg: `grok` 插件中的 `messages`)
- 推荐基于 `broker` 模式进行中心化的日志收集

配置文件

> 可以指定一个配置文件，也可以指定一个文件夹

```
./bin/logstash -f xxx.conf
./bin/logstash -f /xxx/xxx/
```

java异常信息

```
2015-05-07 19:40:48,103|ß|uplus.error|ß|main|ß|ERROR|ß|websock|ß|cn.youja.uplus.logging.U
java.lang.ArithmeticException: / by zero
        at cn.youja.uplus.logging.UplusLoggerTest.ah(UplusLoggerTest.java:29)
        at cn.youja.uplus.logging.UplusLoggerTest.test(UplusLoggerTest.java:20)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.j
        at java.lang.reflect.Method.invoke(Method.java:597)
        at junit.framework.TestCase.runTest(TestCase.java:164)
        at junit.framework.TestCase.runBare(TestCase.java:130)
        at junit.framework.TestResult$1.protect(TestResult.java:106)
        at junit.framework.TestResult.runProtected(TestResult.java:124)
        at junit.framework.TestResult.run(TestResult.java:109)
        at junit.framework.TestCase.run(TestCase.java:120)
        at junit.framework.TestSuite.runTest(TestSuite.java:230)
        at junit.framework.TestSuite.run(TestSuite.java:225)
```

基于filter插件中的 `multiline` 和 `grok` 进行数据提取

```
input {stdin {}}
filter {
 multiline {
   pattern => "^%{TIMESTAMP_ISO8601}"
   negate => true
   what => "previous"
 }
 grok{
  match => {"message" =>["%{TIMESTAMP_ISO8601:rec_time}|ß|%{NOTSPACE:logger}|ß|%{NOTSPACE
   remove_field => [ "message" ]
 }
}
output {
```

```
  stdout { codec => rubydebug }
}
`
```

> 日志收集起来的目的是为了可视化的统计程序报错，用于持续改善。真正的定位问题和解决问题，属于另一个范畴。`message`字段是`multiline`插件的产物，成功`grok`之后，建议移除message字段。

```
{
        "@version" => "1",
      "@timestamp" => "2015-05-08T15:39:29.830Z",
            "host" => "vm_50_76.youja.cn",
            "tags" => [
        [0] "multiline"
    ],
        "rec_time" => "2015-05-07 19:40:48,103",
          "logger" => "uplus.error",
     "thread_name" => "main",
        "loglevel" => "ERROR",
          "module" => "websock",
    "class_method" => "cn.youja.uplus.logging.UplusLoggerTest$TalkShow.run(UplusLoggerTes
        "msg_info" => "除数不能为0.",
           "stack" => "java.lang.ArithmeticException: / by zero"
}
```

ElasticSearch

> 很简单，在你的需求上多做点功课。