# 筛选有用日志

1. 利用if删除无用日志

http://xiaorui.cc/2015/12/24/%E9%80%9A%E8%BF%87logstash-filter%E8%BF%87%E6%BB%A4%E5%B1%8F%E8%94%BD%E4%B8%8D%E9%9C%80%E8%A6%81%E7%9A%84%E6%97%A5%E5%BF%97/

input {

    file    {

                type => "producer"

                path => "/data/buzzMaster/extractor.log"

        }

}

filter {

          if ([message] =~ "^xiaorui.cc") {

             drop {}

          }

}

output  {

    stdout { codec => rubydebug}

}

1. 利用if筛选有用日志

http://blog.51cto.com/seekerwolf/2110174

input {

kafka {

topics\_pattern => "mysql.\*"

bootstrap\_servers => "x.x.x.x:9092"

*#auto\_offset\_rest => "latest"*

codec => json

group\_id => "logstash-g1"

}

}

*#终于到了关键的地方了，logstash的filter ，使用filter 过滤出来我们想要的日志，*

filter {

*#if 还可以使用or 或者and 作为条件语句，举个栗子： if “a” or “b” or “c” in [tags]，这样就可以过滤多个tags 的标签了，我们这个主要用在同型号的交换设备的日志正规化，比如说你有5台交换机，把日志指定到了同一个syslog-ng 上，收集日志的时候只能通过同一个filebeat，多个prospector加不同的tags。这个时候过滤就可以通过判断相应的tags来完成了。*

**if** "mysql\_slow\_logs" in [tags]{

grok {

*#grok 里边有定义好的现场的模板你可以用，但是更多的是自定义模板，规则是这样的，小括号里边包含所有一个key和value，例子：（?<key>value），比如以下的信息，第一个我定义的key是data，表示方法为：?<key> 前边一个问号，然后用<>把key包含在里边去。value就是纯正则了，这个我就不举例子了。这个有个在线的调试库，可以供大家参考，http://grokdebug.herokuapp.com/*

match => { "message" => "(?<date>\d{4}/\d{2}/\d{2}\s(?<datetime>%{TIME}))\s-\s(?<status>\w{2})\s-\s(?<respond\_time>\d+)\.\d+\w{2}\s-\s%{IP:client}:(?<client-port>\d+)\[\d+\]->%{IP:server}:(?<server-port>\d+).\*:(?<databases><\w+>):(?<SQL>.\*)"}

*#过滤完成之后把之前的message 移出掉，节约磁盘空间。*

remove\_field => ["message"]

}

}

**else** **if** "mysql\_sql\_logs" in [tags]{

grok {

match => { "message" => "(?<date>\d{4}/\d{2}/\d{2}\s(?<datetime>%{TIME}))\s-\s(?<status>\w{2})\s-\s(?<respond\_time>\d+\.\d+)\w{2}\s-\s%{IP:client}:(?<client-port>\d+)\[\d+\]->%{IP:server}:(?<server-port>\d+).\*:(?<databases><\w+>):(?<SQL>.\*)"}

remove\_field => ["message"]}

}

}

1. logstash 条件判断语句  
   使用条件来决定filter和output处理特定的事件。logstash条件类似于编程语言。条件支持if、else if、else语句，可以嵌套。   
   比较操作有：   
   相等: ==, !=, <, >, <=, >=   
   正则: =~(匹配正则), !~(不匹配正则)   
   包含: in(包含), not in(不包含)   
   布尔操作：   
   and(与), or(或), nand(非与), xor(非或)   
   一元运算符：   
   !(取反)   
   ()(复合表达式), !()(对复合表达式结果取反)   
     
     
   2、if[foo] in "String"在执行这样的语句是出现错误原因是没有找到叫做foo的field，无法把该字段值转化成String类型。所以最好要加field if exist判断。  
   判断字段是否存在，代码如下：  
     
     
   if ["foo"] {  
     mutate {  
       add\_field => { "bar" => "%{foo}"}  
     }  
   }

# Grok插件

### 常用表达式

* USERNAME 或 USER  
  用户名，由数字、大小写及特殊字符(.\_-)组成的字符串

比如：1234、Bob、Alex.Wong等

* EMAILLOCALPART  
  电子邮件用户名部分，首位由大小写字母组成，其他位由数字、大小写及特殊字符(\_.+-=:)组成的字符串。注意，国内的QQ纯数字邮箱账号是无法匹配的，需要修改正则

比如：stone、Gary\_Lu、abc-123等

* EMAILADDRESS  
  电子邮件

比如：stone@abc.com、Gary\_Lu@gmail.com、abc-123@163.com等

* HTTPDUSER  
  Apache服务器的用户，可以是EMAILADDRESS或USERNAME
* INT  
  整数，包括0和正负整数

比如：0、-123、43987等

* BASE10NUM 或 NUMBER  
  十进制数字，包括整数和小数

比如：0、18、5.23等

* BASE16NUM  
  十六进制数字，整数

比如：0x0045fa2d、-0x3F8709等

* BASE16FLOAT  
  十六进制数字，整数和小数
* WORD  
  字符串，包括数字和大小写字母

比如：String、3529345、ILoveYou等

* NOTSPACE  
  不带任何空格的字符串
* SPACE  
  空格字符串
* QUOTEDSTRING 或 QS  
  带引号的字符串

比如："This is an apple"、'What is your name?'等

* UUID  
  标准UUID

比如：550E8400-E29B-11D4-A716-446655440000

* MAC  
  MAC地址，可以是Cisco设备里的MAC地址，也可以是通用或者Windows系统的MAC地址
* IP  
  IP地址，IPv4或IPv6地址

比如：127.0.0.1、FE80:0000:0000:0000:AAAA:0000:00C2:0002等

* HOSTNAME  
  主机名称
* IPORHOST  
  IP或者主机名称
* HOSTPORT  
  主机名(IP)+端口

比如：127.0.0.1:3306、api.stozen.net:8000等

* PATH  
  路径，Unix系统或者Windows系统里的路径格式

比如：/usr/local/nginx/sbin/nginx、c:\windows\system32\clr.exe等

* URIPROTO  
  URI协议

比如：http、ftp等

* URIHOST  
  URI主机

比如：www.stozen.net、10.0.0.1:22等

* URIPATH  
  URI路径

比如：//www.stozen.net/abc/、/api.php等

* URIPARAM  
  URI里的GET参数

比如：?a=1&b=2&c=3

* URIPATHPARAM  
  URI路径+GET参数

比如：//www.stozen.net/abc/api.php?a=1&b=2&c=3

* URI  
  完整的URI

比如：http://www.stozen.net/abc/api.php?a=1&b=2&c=3

### 日期时间表达式

* MONTH  
  月份名称

比如：Jan、January等

* MONTHNUM  
  月份数字

比如：03、9、12等

* MONTHDAY  
  日期数字

比如：03、9、31等

* DAY  
  星期几名称

比如：Mon、Monday等

* YEAR  
  年份数字
* HOUR  
  小时数字
* MINUTE  
  分钟数字
* SECOND  
  秒数字
* TIME  
  时间

比如：00:01:23

* DATE\_US  
  美国日期格式

比如：10-15-1982、10/15/1982等

* DATE\_EU  
  欧洲日期格式

比如：15-10-1982、15/10/1982、15.10.1982等

* ISO8601\_TIMEZONE  
  ISO8601时间格式

比如：+10:23、-1023等

* TIMESTAMP\_ISO8601  
  ISO8601时间戳格式

比如：2016-07-03T00:34:06+08:00

* DATE  
  日期，美国日期%{DATE\_US}或者欧洲日期%{DATE\_EU}
* DATESTAMP  
  完整日期+时间

比如：07-03-2016 00:34:06

* HTTPDATE  
  http默认日期格式

比如：03/Jul/2016:00:36:53 +0800

### Log表达式

* LOGLEVEL  
  日志等级

比如：Alert、alert、ALERT、Error等

# Mutate插件

mutate插件 mutate插件是用来处理数据的格式的，你可以选择处理你的时间格式，或者你想把一个字符串变为数字类型（当然需要合法），同样的你也可以返回去做。可以设置的转换类型 包括： "integer"， "float" 和 "string"。

1. filter {
2. mutate {
3. *#接收一个数组，其形式为value，type*
4. *#需要注意的是，你的数据在转型的时候要合法，你总是不能把一个‘abc’的字符串转换为123的。*
5. convert => [
6. *#把request\_time的值装换为浮点型*
7. "request\_time", "float"，
8. *#costTime的值转换为整型*
9. "costTime", "integer"
10. ]
11. }
12. }

# Ruby插件

# Date插件

date插件 这里需要合前面的grok插件剥离出来的值logdate配合使用（当然也许你不是用grok去做）。

1. filter{
2. date{
3. *#还记得grok插件剥离出来的字段logdate吗？就是在这里使用的。你可以格式化为你需要的样子，至于是什么样子。就得你自己取看啦。*
4. *#为什什么要格式化？*
5. *#对于老数据来说这非常重要，应为你需要修改@timestamp字段的值，如果你不修改，你保存进ES的时间就是系统但前时间（+0时区）*
6. *#单你格式化以后，就可以通过target属性来指定到@timestamp，这样你的数据的时间就会是准确的，这对以你以后图表的建设来说万分重要。*
7. *#最后，logdate这个字段，已经没有任何价值了，所以我们顺手可以吧这个字段从event对象中移除。*
8. match=>["logdate","dd/MMM/yyyy:HH:mm:ss Z"]
9. target=>"@timestamp"
10. remove\_field => 'logdate'
11. *#还需要强调的是，@timestamp字段的值，你是不可以随便修改的，最好就按照你数据的某一个时间点来使用，*
12. *#如果是日志，就使用grok把时间抠出来，如果是数据库，就指定一个字段的值来格式化，比如说："timeat", "%{TIMESTAMP\_ISO8601:logdate}"*
13. *#timeat就是我的数据库的一个关于时间的字段。*
14. *#如果没有这个字段的话，千万不要试着去修改它。*
16. }
17. }

# Json插件