ELK 之 LogstashLogstash 是一个接收,处理,转发日志的工具。支持系统日志,webserver 日志,错误日志,应用日志,总之包括所有可以抛出来的日志类型。在一个典型的使用场景下(ELK):用 Elasticsearch 作为后台数据的存储,kibana用来前端的报表展示。Logstash 在其过程中担任搬运工的角色,它为数据存储,报表查询和日志解析创建了一个功能强大的管道链。Logstash 提供了多种多样的 input,filters,codecs 和 output 组件,让使用者轻松实现强大的功能。

**参数**

-w # 指定线程,默认是 cpu 核数

-f # 指定配置文件

-t # 测试配置文件是否正常

-b # 执行 filter 模块之前最大能积累的日志,数值越大性能越好,同时越占内

存

**配置文件写法:**

# 日志导入

input {

}

# 日志筛选匹配处理

filter {

}

# 日志匹配输出

output {

}

**input 模块**

**列子1**

input {

# file为常用文件插件，插件内选项很多，可根据需求自行判断

file {

path => "/var/lib/mysql/slow.log"

# 要导入的文件的位置，可以使用\*，例如/var/log/nginx/\*.log

Excude =>”\*.gz”

# 要排除的文件

start\_position => "beginning"

# 从文件开始的位置开始读,end表示从结尾开始读

ignore\_older => 0

# 多久之内没修改过的文件不读取，0为无限制，单位为秒

sincedb\_path => "/dev/null"

# 记录文件上次读取位置，输出到null表示每次都从文件首行开始解析

type => "mysql-slow"

# type字段，可表明导入的日志类型

}

}

**例子2**

input {

# redis插件为常用插件，插件内选项很多，可根据需求自行判断

redis {

batch\_count => 1

# EVAL命令返回的事件数目，设置为5表示一次请求返回5条日志信息

data\_type => "list"

# logstash redis插件工作方式

key => "logstash-test-list"

# 监听的键值

host => "127.0.0.1"

# redis地址

port => 6379

# redis端口号

password => "123qwe"

# 如果有安全认证，此项为认证密码

db => 0

# 如果应用使用了不同的数据库，此为redis数据库的编号，默认为0。

threads => 1

# 启用线程数量

}

}

**filter 模块**

**例子**

filter { # 插件很多

if ([message] =~ "正则表达式") { drop {} }

# 正则匹配=~,!~,包含判断in,not in ，字符串匹配==,!=,等等，匹配之后可以做任何操作，这里过滤掉匹配行，除了做过滤操作，if后面可以作任意操作，甚至可以为匹配到的任意行做单独的正则分割操作

multiline {

pattern => "正则表达式"

negate => true

what => "previous"

# 多行合并，由于一些日志存在一条多行的情况，这个模块可以进行指定多行合并，通过正则匹配，匹配到的内容上面的多行合并为一条日志。

}

grok {

match => { "message" => "正则表达式"

# 正则匹配日志，可以筛选分割出需要记录的字段和值

}

remove\_field => ["message"]

# 删除不需要记录的字段

}

date {

match => ["timestamp","dd/MMM/yyyy:HH:mm:ss Z"]

# 记录@timestamp时间，可以设置日志中自定的时间字段，如果日志中没有时间字段，也可以自己生成

target=>“@timestamp”

# 将匹配的timestamp字段放在指定的字段 默认是@timestamp

}

ruby {

code => "event.timestamp.time.localtime"

# timestamp时区锁定

}

}

**output 模块**

**例子1**

output {

# stdout { codec => "rubydebug" }

# 筛选过滤后的内容输出到终端显示

elasticsearch { # 导出到es，最常用的插件

codec => "json"

# 导出格式为json

hosts => ["127.0.0.1:9200"]

# ES地址+端口

index => "logstash-slow-%{+YYYY.MM.dd}"

# 导出到index内，可以使用时间变量

user => "admin"

password => "xxxxxx"

# ES如果有安全认证就使用账号密码验证，无安全认证就不需要

flush\_size => 500

# 默认500，logstash一次性攒够500条的数据在向es发送

idle\_flush\_time => 1

# 默认1s，如果1s内没攒够500，还是会一次性把数据发给ES

}

}

**例子2**

output {

redis{ # 输出到redis的插件，下面选项根据需求使用

batch => true

# 设为false，一次rpush，发一条数据，true为发送一批

batch\_events => 50

# 一次rpush发送多少数据

batch\_timeout => 5

# 一次rpush消耗多少时间

codec => plain

# 对输出数据进行codec，避免使用logstash的separate filter

congestion\_interval => 1

# 多长时间进项一次拥塞检查

congestion\_threshold => 5

# 限制一个list中可以存在多少个item，当数量足够时，就会阻塞直到有其他消费者消费list中的数据

data\_type => list

# 使用list还是publish

db => 0

# 使用redis的那个数据库，默认为0号

host => ["127.0.0.1:6379"]

# redis 的地址和端口，会覆盖全局端口

key => xxx

# list或channel的名字

password => xxx

# redis的密码，默认不使用

port => 6379

# 全局端口，默认6379，如果host已指定，本条失效

reconnect\_interval => 1

# 失败重连的间隔，默认为1s

timeout => 5

# 连接超时的时间

workers => 1

# 工作进程

}

}

**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)

filter {

if [action] == "login" {

mutate { remove\_field => "secret" }

}

}

若是正则匹配，成功后添加自定义字段：

filter {

if [message] =~ /\w+\s+\/\w+(\/learner\/course\/)/ {

mutate {

add\_field => { "learner\_type" => "course" }

}

}

}

在一个条件里指定多个表达式：

output {

# Send production errors to pagerduty

if [loglevel] == "ERROR" and [deployment] == "production" {

pagerduty {

...

}

}

}

在in条件，可以比较字段值：

filter {

if [foo] in [foobar] {

mutate { add\_tag => "field in field" }

}

if [foo] in "foo" {

mutate { add\_tag => "field in string" }

}

if "hello" in [greeting] {

mutate { add\_tag => "string in field" }

}

if [foo] in ["hello", "world", "foo"] {

mutate { add\_tag => "field in list" }

}

if [missing] in [alsomissing] {

mutate { add\_tag => "shouldnotexist" }

}

if !("foo" in ["hello", "world"]) {

mutate { add\_tag => "shouldexist" }

}

}

not in示例：

output {

if "\_grokparsefailure" not in [tags] {

elasticsearch { ... }

}

}

@metadata field

在logstash1.5版本开始，有一个特殊的字段，叫做@metadata。@metadata包含的内容不会作为事件的一部分输出。

input { stdin { } }

filter {

mutate { add\_field => { "show" => "This data will be in the output" } }

mutate { add\_field => { "[@metadata][test]" => "Hello" } }

mutate { add\_field => { "[@metadata][no\_show]" => "This data will not be in the output" } }

}

output {

if [@metadata][test] == "Hello" {

stdout { codec => rubydebug }

}

}

输出结果

$ bin/logstash -f ../test.conf

Pipeline main started

asdf

{

"@timestamp" => 2016-06-30T02:42:51.496Z,

"@version" => "1",

"host" => "windcoder.com",

"show" => "This data will be in the output",

"message" => "asdf"

}

"asdf"变成message字段内容。条件与@metadata内嵌的test字段内容判断成功，但是输出并没有展示@metadata字段和其内容。

不过，如果指定了metadata => true，rubydebug codec允许显示@metadata字段的内容。

stdout { codec => rubydebug { metadata => true } }

输出结果

$ bin/logstash -f ../test.conf

Pipeline main started

asdf

{

"@timestamp" => 2016-06-30T02:46:48.565Z,

"@metadata字段及其子字段内容。" => {

"test" => "Hello",

"no\_show" => "This data will not be in the output"

},

"@version" => "1",

"host" => "windcoder.com",

"show" => "This data will be in the output",

"message" => "asdf"

}

现在就可以见到@metadata字段及其子字段内容。

只有rubydebug codec允许显示@metadata字段的内容。

只要您需要临时字段但不希望它在最终输出中，就可以使用@metadata字段。

最常见的情景是filter的时间字段，需要一临时的时间戳。如：

input { stdin { } }

filter {

grok { match => [ "message", "%{HTTPDATE:[@metadata][timestamp]}" ] }

date { match => [ "[@metadata][timestamp]", "dd/MMM/yyyy:HH:mm:ss Z" ] }

}

output {

stdout { codec => rubydebug }

}

输出结果

$ bin/logstash -f ../test.conf

Pipeline main started

02/Mar/2014:15:36:43 +0100

{

"@timestamp" => 2014-03-02T14:36:43.000Z,

"@version" => "1",

"host" => "windcoder.com",

"message" => "02/Mar/2014:15:36:43 +0100"

}

# scrapy日志处理，分割出项目名称（根据log路径分割）、日志的级别（message分割）、time、详细信息messages

input {

file{

path=>"/ELK/logfile/\*\*/\*"

type=>"20181218"

start\_position=>"beginning"

sincedb\_path=>"/dev/null"

}

}

filter{

grok {match =>{"path"=>"/ELK/logfile/(?<project\_name>[0-9a-zA-Z]+\_[0-9a-zA-Z]+)"}

}

grok {match =>{"message"=>"(?<time>%{YEAR}-%{MONTHNUM}-%{MONTHDAY})%{DATA:engine }%{LOGLEVEL:loglevel}:%{GREEDYDATA:messages }"}

remove\_field => ['engine']

}

}

output{

stdout{

codec=>rubydebug

}

}