**ELK Stack企业日志平台课程文档**

|  |  |
| --- | --- |
| 作者 | 李振良 （微信: olizhenliang） |
| 官方网站 | <http://www.ctnrs.com> |
| 课程地址 | https://ke.qq.com/course/305012 |
| 技术博客 | <http://lizhenliang.blog.51cto.com> |
| 技术公众号 |  |

目录

[一、ELK介绍 3](#_Toc11159388)

[二、ELK架构 3](#_Toc11159389)

[三、ElasticSearch 4](#_Toc11159390)

[3.1 基本概念 4](#_Toc11159391)

[3.2 集群部署 4](#_Toc11159392)

[3.3 数据操作 6](#_Toc11159393)

[3.4 常用查询 7](#_Toc11159394)

[示例数据 7](#_Toc11159395)

[match\_all 9](#_Toc11159396)

[from，size 9](#_Toc11159397)

[match 10](#_Toc11159398)

[bool 11](#_Toc11159399)

[range 12](#_Toc11159400)

[3.3 Head插件 13](#_Toc11159401)

[四、Logstash 14](#_Toc11159402)

[5.1 安装 14](#_Toc11159403)

[5.2 条件判断 15](#_Toc11159404)

[5.3 输入插件（Input） 16](#_Toc11159405)

[5.3.1 所有输入插件都支持的配置选项 16](#_Toc11159406)

[5.3.1 Stdin 16](#_Toc11159407)

[5.3.1 File 17](#_Toc11159408)

[5.3.2 TCP 18](#_Toc11159409)

[5.3.4 Beats 19](#_Toc11159410)

[5.4 编码插件（Codec） 20](#_Toc11159411)

[5.4.1 json/json\_lines 20](#_Toc11159412)

[5.4.3 multline 21](#_Toc11159413)

[5.4.4 rubydebug 24](#_Toc11159414)

[5.5 过滤器插件（Filter） 24](#_Toc11159415)

[5.5.1 json 24](#_Toc11159416)

[5.5.2 kv 25](#_Toc11159417)

[5.5.3 grok 26](#_Toc11159418)

[5.5.4 geoip 29](#_Toc11159419)

[5.5.5 date 31](#_Toc11159420)

[5.5.6 mutate 32](#_Toc11159421)

[5.6 输出插件（Output） 33](#_Toc11159422)

[5.6.1 ES 33](#_Toc11159423)

[五、Kibana 35](#_Toc11159424)

[六、引入Redis 35](#_Toc11159425)

[七、引入Filebeat 36](#_Toc11159426)

[6.1 安装 36](#_Toc11159427)

[6.1 配置文件 36](#_Toc11159428)

# 一、ELK介绍

ELKStacks是一个技术栈的组合，分别是Elasticsearch、Logstash、Kibana

ELK Stack：

1、扩展性：采用高扩展性分布式架构设计，可支持每日TB级数据

2、简单易用：通过图形页面可对日志数据各种统计，可视化

3、查询效率高：能做到秒级数据采集、处理和搜索

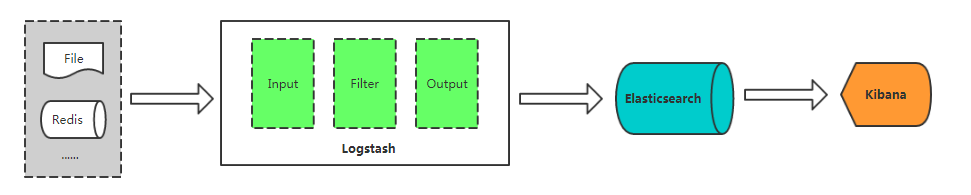
<https://www.elastic.co/cn/products/elasticsearch>

<https://www.elastic.co/cn/products/kibana>

<https://www.elastic.co/cn/products/beats/filebeat>

<https://www.elastic.co/cn/products/beats/metricbeat>

# 二、ELK架构



Logstash ：开源的服务器端数据处理管道，能够同时从多个来源采集数据、转换数据，然后将数据存储到数据库中。

Elasticsearch：搜索、分析和存储数据。

Kibana：数据可视化。

Beats ：轻量型采集器的平台，从边缘机器向 Logstash 和 Elasticsearch 发送数据。

Filebeat：轻量型日志采集器。

<https://www.elastic.co/cn/>

<https://www.elastic.co/subscriptions>

Input：输入，输出数据可以是Stdin、File、TCP、Redis、Syslog等。

Filter：过滤，将日志格式化。有丰富的过滤插件：Grok正则捕获、Date时间处理、Json编解码、Mutate数据修改等。

Output：输出，输出目标可以是Stdout、File、TCP、Redis、ES等。

# 三、ElasticSearch

## 3.1 基本概念

Node：运行单个ES实例的服务器

Cluster：一个或多个节点构成集群

Index：索引是多个文档的集合

Document：Index里每条记录称为Document，若干文档构建一个Index

Type：一个Index可以定义一种或多种类型，将Document逻辑分组

Field：ES存储的最小单元

Shards：ES将Index分为若干份，每一份就是一个分片

Replicas：Index的一份或多份副本

|  |  |
| --- | --- |
| ES | 关系型数据库（比如Mysql） |
| Index | Database |
| Type | Table |
| Document | Row |
| Field | Column |

## 3.2 集群部署

确认时区：

date

如果时区是EST，要修改为CST（中国时区）

cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

配置YUM仓库，后面安装都用这个库：

# rpm --import https://packages.elastic.co/GPG-KEY-elasticsearch

# vi /etc/yum.repos.d/elastic.repo

[elastic-6.x]

name=Elastic repository for 6.x packages

baseurl=https://artifacts.elastic.co/packages/6.x/yum

gpgcheck=1

gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch

enabled=1

autorefresh=1

type=rpm-md

# yum install elasticsearch -y

# 节点1

cluster.name: elk-cluster

node.name: node-1

# node.master: true或fase #是否作为主节点

path.data: /home/es/es\_data

network.host: 192.168.1.195

http.port: 9200

discovery.zen.ping.unicast.hosts: ["192.168.1.195", "192.168.1.196", "192.168.1.197"]

discovery.zen.minimum\_master\_nodes: 2

# 节点2

cluster.name: elk-cluster

node.name: node-2

path.data: /home/es/es\_data

network.host: 192.168.1.196

http.port: 9200

discovery.zen.ping.unicast.hosts: ["192.168.1.195", "192.168.1.196", "192.168.1.197"]

discovery.zen.minimum\_master\_nodes: 2

# 节点3

cluster.name: elk-cluster

node.name: node-3

path.data: /home/es/es\_data

network.host: 192.168.1.197

http.port: 9200

discovery.zen.ping.unicast.hosts: ["192.168.1.195", "192.168.1.196", "192.168.1.197"]

discovery.zen.minimum\_master\_nodes: 2

cluster.name # 集群名称

node.name # 节点名称

path.data # 数据目录。可以设置多个路径，这种情况下，所有的路径都会存储数据。

集群主要关注两个参数：

discovery.zen.ping.unicast.hosts # 单播，集群节点IP列表。提供了自动组织集群，自动扫描端口9300-9305连接其他节点。无需额外配置。

discovery.zen.minimum\_master\_nodes # 最少主节点数

为防止数据丢失，这个参数很重要，如果没有设置，可能由于网络原因脑裂导致分为两个独立的集群。为避免脑裂，应设置符合节点的法定人数：(nodes / 2 ) + 1

换句话说，如果集群节点有三个，则最小主节点设置为(3/2) + 1 或2

查看集群节点：

curl -XGET 'http://127.0.0.1:9200/\_cat/nodes?pretty'

查看集群健康状态：

# curl -i -XGET <http://127.0.0.1:9200/_cluster/health?pretty>



green：所有的主分片和副本分片都已分配。你的集群是 100% 可用的。

yellow：所有的主分片已经分片了，但至少还有一个副本是缺失的。不会有数据丢失，所以搜索结果依然是完整的。不过，你的高可用性在某种程度上被弱化。如果 更多的 分片消失，你就会丢数据了。把 yellow 想象成一个需要及时调查的警告。

red：至少一个主分片（以及它的全部副本）都在缺失中。这意味着你在缺少数据：搜索只能返回部分数据，而分配到这个分片上的写入请求会返回一个异常。

green/yellow/red 状态是一个概览你的集群并了解眼下正在发生什么的好办法。

## 3.3 数据操作

**RestFul API格式**

curl -X<verb> ‘<protocol>://<host>:<port>/<path>?<query\_string>’-d ‘<body>’

|  |  |
| --- | --- |
| **参数** | **描述** |
| verb | HTTP方法，比如GET、POST、PUT、HEAD、DELETE |
| host | ES集群中的任意节点主机名 |
| port | ES HTTP服务端口，默认9200 |
| path | 索引路径 |
| query\_string | 可选的查询请求参数。例如?pretty参数将返回JSON格式数据 |
| -d | 里面放一个GET的JSON格式请求主体 |
| body | 自己写的 JSON格式的请求主体 |

查看索引：

curl http://127.0.0.1:9200/\_cat/indices?v

新建索引：

curl -X PUT 127.0.0.1:9200/logs-2018.05.22

删除索引：

curl -X DELETE 127.0.0.1:9200/logs-2018.05.22

## 3.4 常用查询

ES提供一种可用于执行查询JSON式的语言，被称为Query DSL。

### 示例数据

使用官方提供的示例数据：

https://www.elastic.co/guide/en/elasticsearch/reference/current/\_exploring\_your\_data.html

wget <https://raw.githubusercontent.com/elastic/elasticsearch/master/docs/src/test/resources/accounts.json>

导入数据：

curl -H "Content-Type: application/json" -XPOST "localhost:9200/bank/\_doc/\_bulk?pretty&refresh" --data-binary "@accounts.json"

curl "localhost:9200/\_cat/indices?v"

curl -X GET "localhost:9200/bank/\_search?q=\*&sort=account\_number:asc&pretty"

\_search

q=\* ES批量索引中的所有文档

sort=account\_number:asc 表示根据account\_number按升序对结果排序

以上方式是使用查询字符串替换请求主体。

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match\_all": {} },

"sort": [

{ "account\_number": "asc" }

]

}

'

这个区别在于不是传入的q=\* URI,而是向\_search API提供JSON格式的查询请求体。

### match\_all

match\_all：匹配所有文档。默认查询

示例：查询所有，默认只返回10个文档

curl -X GET "localhost:9200/bank/\_search?pretty" -H 'Content-Type: application/json' -d'

{

"query": { "match\_all": {} }

}

'

query告诉我们查询什么，match\_all使我们查询的类型。match\_all查询仅仅在指定的索引的所有文件进行搜索。

### from，size

除了query参数外，还可以传递其他参数影响查询结果，比如上面的sort，下嘛的size：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match\_all": {} },

"size": 1

}

'

**注意：size未指定，默认为10**

**返回10-19的文档：**

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match\_all": {} },

"from": 10,

"size": 10

}

'

此功能实现分页功能非常有用。如果from未指定，默认为0

返回\_source字段中的几个字段：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match\_all": {} },

"\_source": ["account\_number", "balance"]

}

'

### match

**基本搜索查询，针对特定字段或字段集合进行搜索**

查询编号为20的账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match": { "account\_number": 20 } }

}

'

返回地址中包含mill的账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match": { "address": "mill" } }

}

'

返回地址有包含mill或lane的所有账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": { "match": { "address": "mill lane" } }

}

'

### bool

**查询包含mill和lane的所有账户：**

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": {

"bool": {

"must": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}

'

该bool must指定了所所有必须为真才匹配。

**查询包含mill或lane的所有账户：**

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": {

"bool": {

"should": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}

'

### range

* **range**

指定区间内的数字或者时间。

操作符：gt大于，gte大于等于，lt小于，lte小于等于

查询余额大于或等于20000且小于等于30000的账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'

{

"query": {

"bool": {

"must": { "match\_all": {} },

"filter": {

"range": {

"balance": {

"gte": 20000,

"lte": 30000

}

}

}

}

}

}

'

## 3.5 Head插件

wget <https://npm.taobao.org/mirrors/node/latest-v4.x/node-v4.4.7-linux-x64.tar.gz>

tar -zxvf node-v4.4.7-linux-x64.tar.gz

# vi /etc/profile

NODE\_HOME=/usr/local/node4.4

PATH=$NODE\_HOME/bin:$PATH

export NODE\_HOME PATH

或者

# yum install nodejs npm -y

**安装grunt：**

# npm install -g grunt-cli

#忽略脚本继续进行安装

# npm install phantomjs-prebuilt@2.1.14 --ignore-scripts

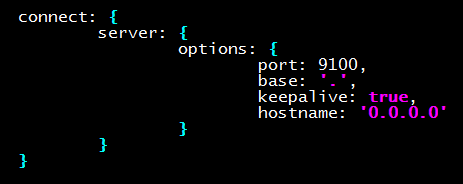
# npm install

**修改监听和ES地址：**

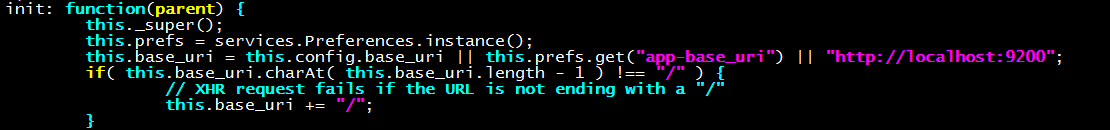
# git clone git://github.com/mobz/elasticsearch-head.git

# cd elasticsearch-head

# vi Gruntfile.js



# vi \_site/app.js



**运行head：**

# grunt server

# npm run start

**访问：**<http://ip:9100>

出现问题：“集群健康值: 未连接”

**解决方法：**

# vi $ES\_HOME/config/elasticsearch.yml

# 增加如下字段

http.cors.enabled: true

http.cors.allow-origin: "\*"

说明：CORS（跨域访问），这个字段默认为false，5.x版本后增加的。

# 四、Logstash

## 4.1 安装

<https://www.elastic.co/guide/en/logstash/current/configuration.html>

# yum install logstash -y

# /usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/test.conf

|  |  |
| --- | --- |
|  |  |
| -f, --path.config CONFIG\_PATH | 指定配置文件。使用文件，目录，或者通配符加载配置信息，如果指定目录或者通配符，按字符顺序加载。 |
| -e, --config.string CONFIG\_STRING | 指定字符串输入 |
| -w, --pipeline.workers COUNT | 指定管道数量，默认3 |
| --log.level LEVEL | 指定Logstash日志级别,fatal/error/warn/info/debug/trace |
| -r，--config.reload.automatic | 配置文件自动重新加载。默认每3s检查一次配置文件更改。  --config.reload.interval <interval> 修改时间间隔。  如果没有启用自动加载，也可以向Logstash进程发送SIGHUP（信号挂起）信号重启管道，例如：kill -1 14175 |
|  |  |
| -t, --config.test\_and\_exit | 检查配置文件是否正确 |

## 4.2 条件判断

使用条件来决定filter和output处理特定的事件

比较操作：

* 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)

可以像其他编程语言那样，条件if判断、多分支，嵌套。

if EXPRESSION {

...

} else if EXPRESSION {

...

} else {

...

}

## 4.3 输入插件（Input）

Input：输入，输出数据可以是Stdin、File、TCP、Syslog、Redis、Kafka等。

<https://www.elastic.co/guide/en/logstash/current/input-plugins.html>

### 4.3.1 所有输入插件都支持的配置选项

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| add\_field | hash | No | {} | 添加一个字段到一个事件 |
| codec | codec | No | plain | 用于输入数据的编解码器 |
| enable\_metric | boolean | No | true |  |
| id | string | No |  | 添加一个ID插件配置，如果没有指定ID，则Logstash将生成一个ID。强烈建议配置此ID，当两个或多个相同类型的插件时，这个非常有用的。例如，有两个文件输入，添加命名标识有助于监视 |
| tags | array | No |  | 添加任意数量的标签，有助于后期处理 |
| type | string | No |  | 为输入处理的所有事件添加一个字段，自已随便定义，比如linux系统日志，定义为syslog |

### 4.3.1 Stdin

标准输入。

示例：

# cat config/logstash.conf

input {

stdin {

}

}

filter {

}

output {

stdout{codec => rubydebug }

}

### 4.3.1 File

<https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| close\_older | number | No | 3600 | 单位秒，打开文件多长时间关闭 |
| delimiter | string | No | \n | 每行分隔符 |
| discover\_interval | number | No | 15 | 单位秒，多长时间检查一次path选项是否有新文件 |
| exclude | array | No |  | 排除监听的文件，跟path一样，支持通配符 |
| max\_open\_files | number | No |  | 打开文件最大数量 |
| path | array | YES |  | 输入文件的路径，可以使用通配符  例如/var/log/\*\*/\*.log，则会递归搜索 |
| sincedb\_path | string | No |  | sincedb数据库文件的路径，用于记录被监控的日志文件当前位置 |
| [sincedb\_write\_interval](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html" \l "plugins-inputs-file-sincedb_write_interval" \o "sincedb_write_interval) | number | No | 15 | 单位秒，被监控日志文件当前位置写入数据库的频率 |
| [start\_position](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html" \l "plugins-inputs-file-start_position" \o "start_position) | [string](https://www.elastic.co/guide/en/logstash/current/configuration-file-structure.html" \l "string" \o "Stringedit), one of ["beginning", "end"] | No | end | 指定从什么位置开始读取文件：开头或结尾。默认从结尾开始，如果要想导入旧数据，将其设置为begin。如果sincedb记录了此文件位置，那么此选项不起作用 |
| |  |  | | --- | --- | | [stat\_interval](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html" \l "plugins-inputs-file-stat_interval" \o "stat_interval) |  | | number | No | 1 | 单位秒，统计文件的频率，判断是否被修改。增加此值会减少系统调用次数。 |

input {

file {

path =>"/var/log/messages"

tags =>"nginx"

tags =>"access"

type =>"syslog"

}

}

filter {

}

output {

stdout{codec => rubydebug }

}

### 4.3.2 TCP

通过TCP套接字读取事件。与标准输入和文件输入一样，每个事件都被定位一行文本。

input {

tcp {

port =>12345

type =>"nc"

}

}

filter {

}

output {

stdout{codec => rubydebug }

}

# nc 192.168.1.196 12345

### 4.3.4 Beats

从Elastic Beats框架接收事件。

# cat config/logstash.conf

input {

beats {

port => 5044

}

}

filter {

}

output {

stdout { codec => rubydebug }

}

filebeat配置文件：

filebeat.prospectors:

- type: log

paths:

- /var/log/messages

tags: ["system-log","123"]

fields:

level: debug

output.logstash:

hosts: ['127.0.0.1:5044']

## 4.4 编码插件（Codec）

Logstash处理流程：input->decode->filter->encode->output

### 4.4.1 json/json\_lines

该解码器可用于解码（Input）和编码（Output）JSON消息。如果发送的数据是JSON数组，则会创建多个事件（每个元素一个）

如果传输JSON消息以\n分割，就需要使用json\_lines。

input {

stdin {

codec =>json {

charset => ["UTF-8"]

}

}

}

filter {

}

output {

stdout{codec => rubydebug }

}

测试数据：{"a":123,"b":456,"c":[1,2,3]}

### 4.4.2 multline

匹配多行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| [auto\_flush\_interval](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-auto_flush_interval" \o "auto_flush_interval) | number | No |  |  |
| [charset](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-charset" \o "charset) | string | No | UTF-8 | 输入使用的字符编码 |
| [max\_bytes](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-max_bytes" \o "max_bytes) | bytes | No | 10M | 如果事件边界未明确定义，则事件的的积累可能会导致logstash退出，并出现内存不足。与max\_lines组合使用 |
| |  |  | | --- | --- | | [max\_lines](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-max_lines" \o "max_lines) |  | | number | No | 500 | 如果事件边界未明确定义，则事件的的积累可能会导致logstash退出，并出现内存不足。与max\_bytes组合使用 |
| [multiline\_tag](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-multiline_tag" \o "multiline_tag) | string | No | multiline | 给定标签标记多行事件 |
| [negate](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-negate" \o "negate) | boolean | No | false | 正则表达式模式，设置正向匹配还是反向匹配。默认正向 |
| [pattern](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-pattern" \o "pattern) | string | Yes |  | 正则表达式匹配 |
| [patterns\_dir](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-patterns_dir" \o "patterns_dir) | array | No | [] | 默认带的一堆模式 |
| [what](https://www.elastic.co/guide/en/logstash/current/plugins-codecs-multiline.html" \l "plugins-codecs-multiline-what" \o "what) | [string](https://www.elastic.co/guide/en/logstash/current/configuration-file-structure.html" \l "string" \o "Stringedit), one of ["previous", "next"] | Yes | 无 | 设置未匹配的内容是向前合并还是向后合并。 |

input {

stdin {

codec => multiline {

pattern => "pattern, a regexp"

negate => "true" or "false"

what => "previous" or "next"

}

}

}

例如，将JAVA堆栈跟踪是多行的，通常从最左边开始，每个后续行都缩进。

input {

stdin {

codec => multiline {

pattern => "^\s"

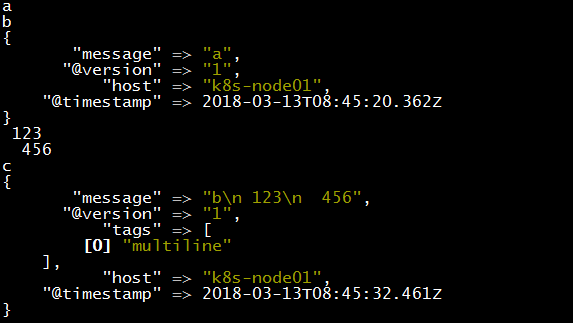
what => "previous"

}

}

}

表示以任意空白开头的行都属于上一行。



input {

stdin {

codec => multiline {

pattern => "^\["

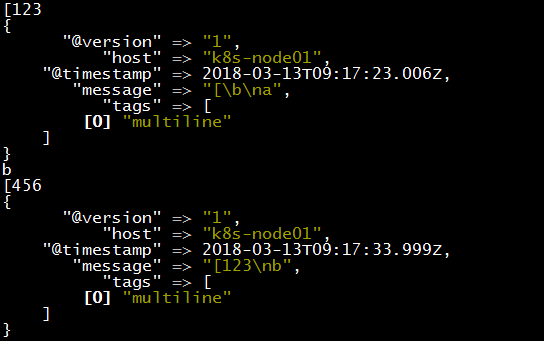
negate => true

what => "previous"

}

}

}



input {

stdin {

codec => multiline {

# Grok pattern names are valid! :)

pattern => "^%{TIMESTAMP\_ISO8601} "

negate => true

what => "previous"

}

}

}

不以时间戳开始的行应与前一行合并。

### 4.4.3 rubydebug

采用ruby awsone print库来解析日志。

## 4.5 过滤器插件（Filter）

Filter：过滤，将日志格式化。有丰富的过滤插件：Grok正则捕获、date时间处理、JSON编解码、数据修改Mutate等。

所有的过滤器插件都支持以下配置选项：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| add\_field | hash | No | {} | 如果过滤成功，添加任何field到这个事件。例如：add\_field => [ "foo\_%{somefield}", "Hello world, from %{host}" ]，如果这个事件有一个字段somefiled，它的值是hello，那么我们会增加一个字段foo\_hello，字段值则用%{host}代替。 |
| add\_tag | array | No | [] | 过滤成功会增加一个任意的标签到事件例如：add\_tag => [ "foo\_%{somefield}" ] |
| enable\_metric | boolean | No | true |  |
| id | string | No |  |  |
| periodic\_flush | boolean | No | false | 定期调用过滤器刷新方法 |
| remove\_field | array | No | [] | 过滤成功从该事件中移除任意filed。例：remove\_field => [ "foo\_%{somefield}" ] |
| remove\_tag | array | No | [] | 过滤成功从该事件中移除任意标签，例如：remove\_tag => [ "foo\_%{somefield}" ] |

### 4.5.1 json

JSON解析过滤器，接收一个JSON的字段，将其展开为Logstash事件中的实际数据结构。

当事件解析失败时，这个插件有一个后备方案，那么事件将不会触发，而是标记为\_jsonparsefailure，可以使用条件来清楚数据。也可以使用[tag\_on\_failure](https://www.elastic.co/guide/en/logstash/current/plugins-filters-json.html" \l "plugins-filters-json-tag_on_failure" \o "tag_on_failure)

input {

stdin {

}

}

filter {

json {

source => "message"

target => "content"

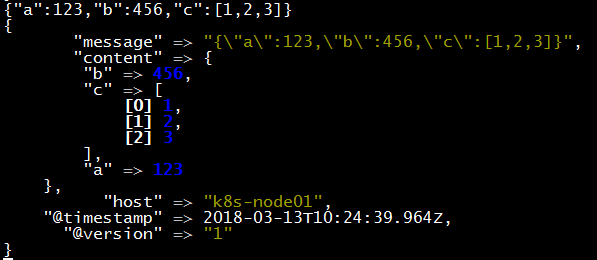
}

}

output {

stdout{codec => rubydebug }

}



### 4.5.2 kv

自动解析key=value。也可以任意字符串分割数据。

field\_split 一串字符，指定分隔符分析键值对

例如URL查询字符串拆分参数：

filter {

kv {

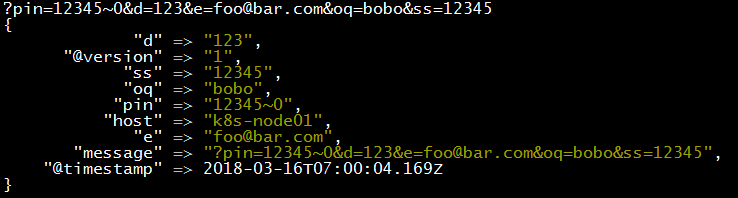
field\_split => "&?"

}

}

根据&和?分隔。

www.baidu.com?pin=12345~0&d=123&e=foo@bar.com&oq=bobo&ss=12345



### 4.5.3 grok

grok是将非结构化数据解析为结构化。

这个工具非常适于系统日志，mysql日志，其他Web服务器日志以及通常人类无法编写任何日志的格式。

默认情况下，Logstash附带约120个模式。也可以添加自己的模式（patterns\_dir）

<https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns>

<http://grok.ctnrs.com> Grok调试网站

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| break\_on\_match | boolean | No | true |  |
| keep\_empty\_captures |  | No | false | 如果true将空保留为事件字段 |
| match | hash | No | {} | 一个hash匹配字段=>值 |
| named\_captures\_only | boolean | No | true | 如果true，只存储 |
| overwrite | array | No | [] | 覆盖已存在的字段的值 |
| pattern\_definitions |  | No | {} |  |
| patterns\_dir | array | No | [] | 自定义模式 |
| patterns\_files\_glob | string | No | \* | Glob模式，用于匹配patterns\_dir指定目录中的模式文件 |
| tag\_on\_failure | array | No | \_grokparsefailure | tags没有匹配成功时，将值附加到字段 |
| tag\_on\_timeout | string | No | \_groktimeout | 如果Grok正则表达式超时，则应用标记 |
| timeout\_millis | number |  | 30000 | 正则表达式超时时间 |

grok模式的语法是 %{SYNTAX:SEMANTIC}

SYNTAX模式名称

SEMANTIC匹配文本的标识符

例如：%{NUMBER:duration} %{IP:client}

例如：虚构http请求日志抽出有用的字段

55.3.244.1 GET /index.html 15824 0.043

filter {

grok {

match => {

"message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}"

}

}

}

#### 自定义模式

如果默认模式中没有匹配的，可以自己写正则表达式。

# cat /opt/patterns

ID [0-9A-Z]{10,11}

filter {

grok {

patterns\_dir =>"/opt/patterns"

match => {

"message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration} %{ABC:id}"

}

}

}

55.3.244.1 GET /index.html 15824 0.043 15BF7F3ABB

#### 多模式匹配

一个日志可能有多种格式，一个匹配可以有多条规则匹配多种格式。

一条匹配模式，如果匹配不到，只会到message字段。

例如：新版本项目日志需要添加日志字段，需要兼容旧日志匹配

input {

stdin {

}

}

filter {

grok {

patterns\_dir =>"/opt/patterns"

match => [

"message", "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration} %{ID:id}",

"message", "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration} %{TAG:tag}"

]

}

}

output {

stdout{codec => rubydebug }

}

使用[]，=>换成逗号

### 4.5.4 geoip

开源IP地址库：

https://dev.maxmind.com/geoip/geoip2/geolite2/

IP地址库使用示例：

filter {

grok {

match => {

"message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}"

}

}

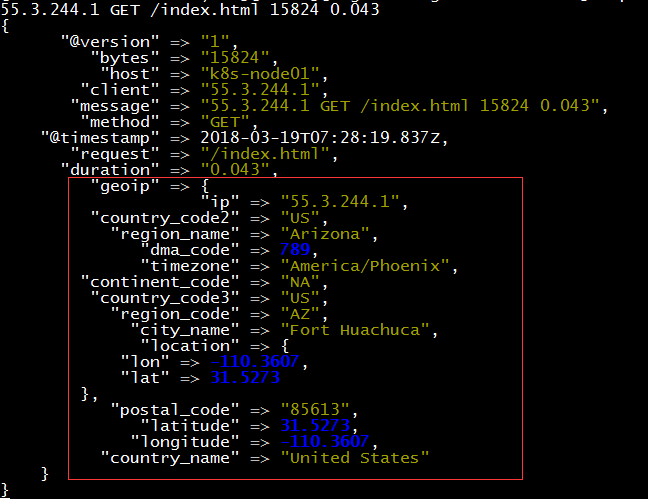
geoip {

source => "client"

database => "/opt/GeoLite2-City.mmdb"

}

}



filter {

grok {

match => {

"message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}"

}

}

geoip {

source => "client"

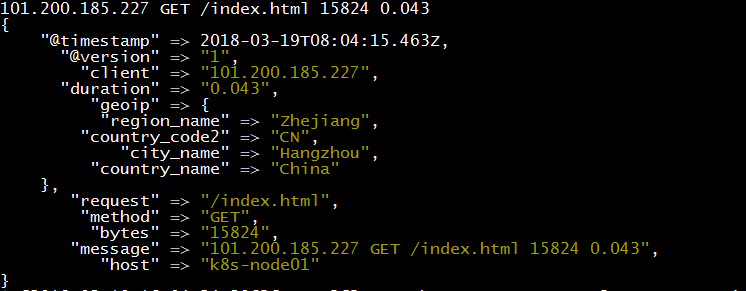
database => "/opt/GeoLite2-City.mmdb"

target => "geoip"

fields => ["city\_name", "country\_code2", "country\_name","region\_name"]

}

}



只保留关键字段。

### 4.5.5 date

日志时间过滤器用于从字段中解析日期，然后使用日期或时间戳作为事件的logstash时间戳。

如果不使用date插件，那么Logstash将处理事件作为时间戳。时间戳字段是Logstash自己添加到内置字段@timestamp，默认是UTC时间，比北京时间少8个小时。

插入到ES中保存的也是UTC时间，创建索引也是根据这个时间创建的。但Kibana是根据你当前浏览器的时区显示的（对timestamp加减）。

filter {

if "nginx-access" in [tags] {

grok {

match => {

"message" => "%{IPV4:remote\_addr} - (%{USERNAME:remote\_user}|-) \[%{HTTPDATE:time\_local}\] \"%{WORD:request\_method} %{URIPATHPARAM:request\_uri} HTTP/%{NUMBER:http\_protocol}\" %{NUMBER:http\_status} %{NUMBER:body\_bytes\_sent} \"%{GREEDYDATA:http\_referer}\" \"%{GREEDYDATA:http\_user\_agent}\""

}

}

date {

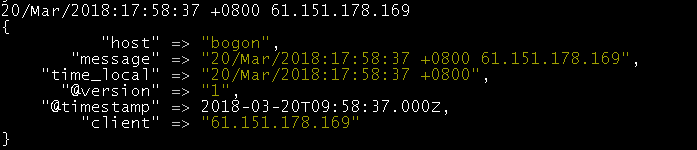
locale => "en"

match => ["time\_local", "dd/MMM/yyyy:HH:mm:ss Z"] # 默认target是@timestamp，所以time\_local会更新@timestamp时间。

}

}

}



还是以UTC时间存储，但@timestamp不是采集时当前系统时间了，而是根据匹配模式里的time\_local时间，可以写个别的时间测试。

### 4.5.6 mutate

修改指定字段的内容。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| convert | hash | No |  | 将字段的值转换为其他类型 |
| gsub | array | No |  | 用字符串替换正则匹配的内容，只支持字符串类型。 |
| replace | hash | No |  | 用新值替换一个字段 |

比如字段类型转换：

filter {

mutate {

convert => { "fieldname" => "integer" }

}

}

## 4.6 输出插件（Output）

Output：输出，输出目标可以是Stdout、ES、Redis、File、TCP等。

### 4.6.1 ES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting** | **Input type** | **Required** | **Default** | **Description** |
| hosts | URL | No |  |  |
| index | string | No | logstash-%{+YYYY.MM.dd} | 将事件写入索引。默认按日期划分。 |
| user | string | No |  | ES集群用户 |
| password | password | No |  | ES集群密码 |

output {

elasticsearch {

hosts => "localhost:9200"

index => "ytjh-admin-%{+YYYY.MM.dd}"

}

}

**Lostash->ES**

input {

file {

path => ["/var/log/messages"]

type => "system"

tags => ["syslog","test"]

start\_position => "beginning"

}

file {

path => ["/var/log/audit/audit.log"]

type => "system"

tags => ["auth","test"]

start\_position => "beginning"

}

}

filter {

}

output {

if [type] == "system" {

if [tags][0] == "syslog" {

elasticsearch {

hosts => ["http://192.168.0.217:9200","http://192.168.0.218:9200","http://192.168.0.219:9200"]

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

}

stdout { codec=> rubydebug }

}

else if [tags][0] == "auth" {

elasticsearch {

hosts => ["http://192.168.0.217:9200","http://192.168.0.218:9200","http://192.168.0.219:9200"]

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

}

stdout { codec=> rubydebug }

}

}

}

# 五、Kibana

# yum install kibana –y

# vi /etc/kibana/kibana.yml

server.port: 5601

server.host: "0.0.0.0"

elasticsearch.url: <http://localhost:9200>

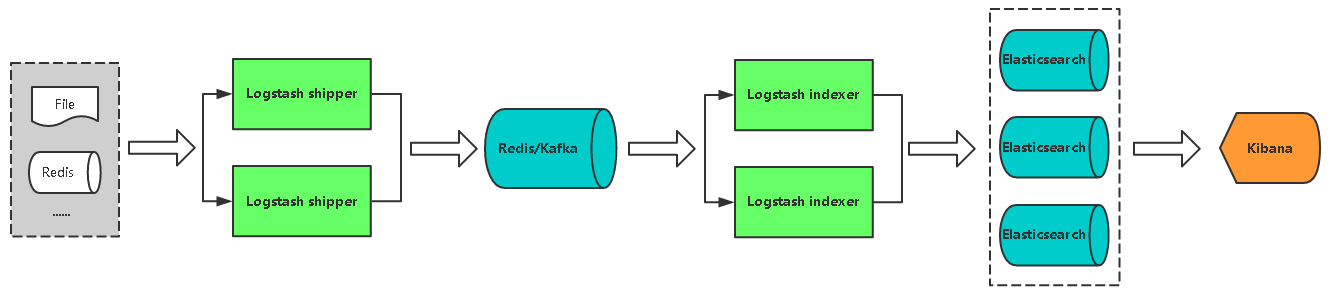
# systemctl start kibana

# systemctl enable kibana

访问：

<http://192.168.1.248:5601>

# 六、引入Redis



优势：

1、相比上图，在多台服务器，大量日志情况下可减少对ES压力，队列起到缓冲作用，也可以一定程度保护数据不丢失。（当Logstash接收数据能力超过ES处理能力时，可增加队列均衡网络传输）

2、将收集的日志统一在Indexer中处理。

如果日志量大可采用Kafka做缓冲队列，相比Redis更适合大吞吐量。

# yum install redis –y

# vi /etc/redis/redis.conf

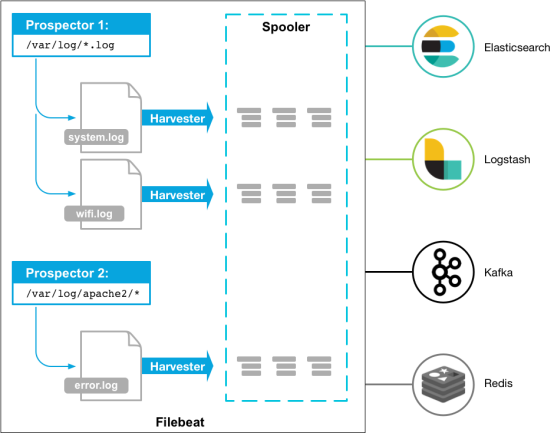
bind 0.0.0.0

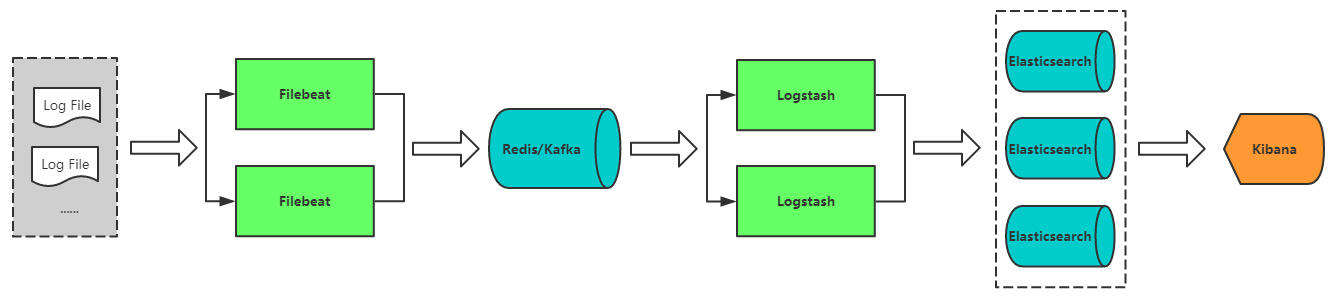
requirepass 123456

# systemctl start redis

# systemctl enable redis

# 七、引入Filebeat





如果只采集日志，Filebeat比Logstash更适合，filebeat占用资源少，配置简单。

采集系统日志示例：

# yum install filebeat –y

# vi /etc/filebeat/filebeat.yml

filebeat.prospectors:

- type: log

paths:

- /var/log/messages

tags: ["system-log"]

fields:

level: debug

output.logstash:

hosts: ['127.0.0.1:5044']

# systemctl start filebeat

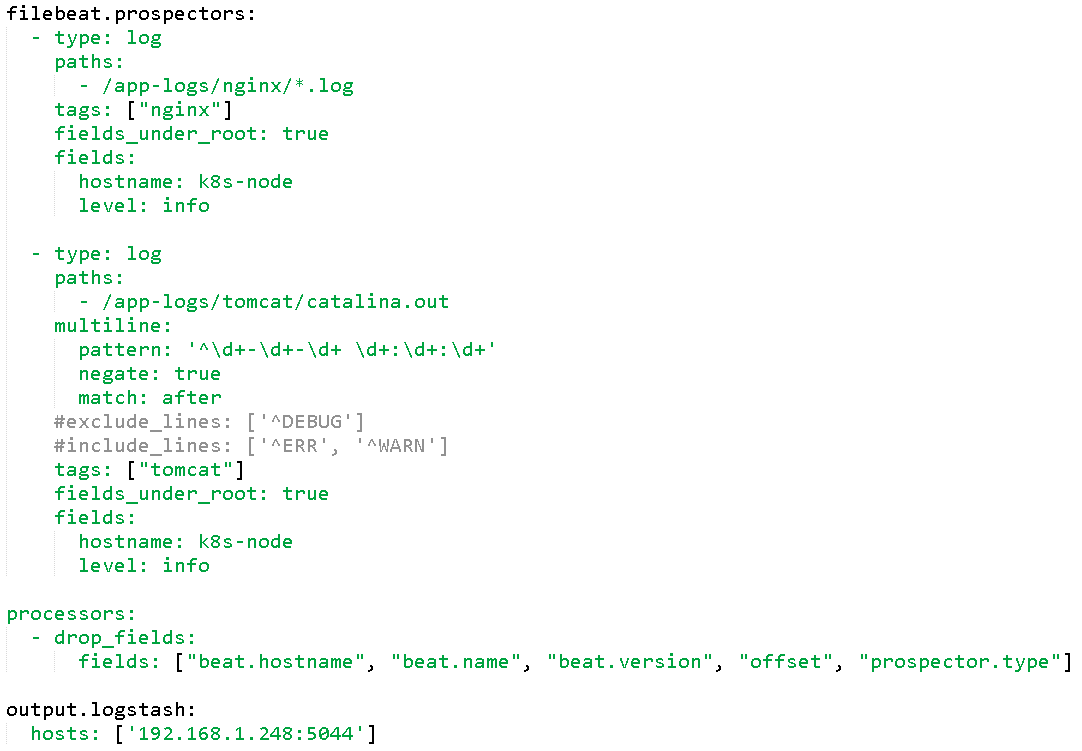
# systemctl enable filebeat

或者输出到logstash：

output.logstash:

hosts: ['192.168.1.248:5044']

filebeat配置文件示例：



paths # 指定监控的文件，可通配符匹配。如果要对目录递归处理，例如/var/log/\*/\*.log

encoding：指定监控的文件编码类型，使用plain和utf-8都可以处理中文日志

exclude\_lines: ['^DEBUG'] # 排除 DEBUG开头的行

include\_lines: ['^ERR', '^WARN'] # 只读取ERR，WARN开头的行

fields\_under\_root #设置true，则新增字段为根目录。否则是fields.level

fields # 新增字段，向输出的日志添加额外的信息，方面后面分组统计

processors # 定义处理器，在发送数据之前处理事件。

drop\_fields # 指定删除的字段

output.console: # 以JSON格式控制台输出

pretty: true

多行匹配参数：

**multiline.pattern: ^\[**

**multiline.negate: true**

**multiline.match: after**

**上面配置的意思是：不以[开头的行都合并到上一行的末尾**

pattern：正则表达式

negate：true 或 false；默认是false，匹配pattern的行合并到上一行；true，不匹配pattern的行合并到上一行

match：after 或 before，合并到上一行的末尾或开头