目前该文档内容只是elk 6.3版本。

# logstash默认测试配置

input { stdin { } }

output { stdout { codec => rubydebug} }

hello

{

"@timestamp" => 2018-06-27T01:11:38.423Z,

"@version" => "1",

"message" => "hello\r",

"host" => "Admin-PC"

}

input {

stdin {

}

}

filter {

grok {

match => [ "message", "%{COMBINEDAPACHELOG}"]}

}

output {

stdout {

codec => rubydebug

}

}

107.187.90.29 - - [05/Sep/2015:01:14:02 +0000] "GET / HTTP/1.1" 200 453 "-" "curl/7.21.0"

{

"verb" => "GET",

"bytes" => "453",

"host" => "Admin-PC",

"agent" => "\"curl/7.21.0\"",

"httpversion" => "1.1",

"response" => "200",

"@timestamp" => 2018-06-27T01:08:12.984Z,

"request" => "/",

"clientip" => "107.187.90.29",

"referrer" => "\"-\"",

"ident" => "-",

"timestamp" => "05/Sep/2015:01:14:02 +0000",

"@version" => "1",

"auth" => "-",

"message" => "107.187.90.29 - - [05/Sep/2015:01:14:02 +0000] \"GET / HTTP/1.1\" 200 453 \"-\" \"curl/7.21.0\"\r"

}

input {

stdin {

}

}

filter{

json{

source => "message"

}

mutate{

add\_field => {

"test\_name" => "%{[loggerName]}"

}

}

}

output {

stdout {

codec => rubydebug

}

}

定义了一个从命令行接受输入，解析json，并输出到命令的解析器。

两层解析

input {

stdin {

}

}

filter{

json{

source => "message"

}

json{

source => "message"

target => "jsoncontent"

}

}

output {

stdout {

codec => rubydebug

}

}

过滤器对嵌套的json信息做第二次解析。

以上两个配置可能会在java程序的log分析中用到。

# Dsl语言

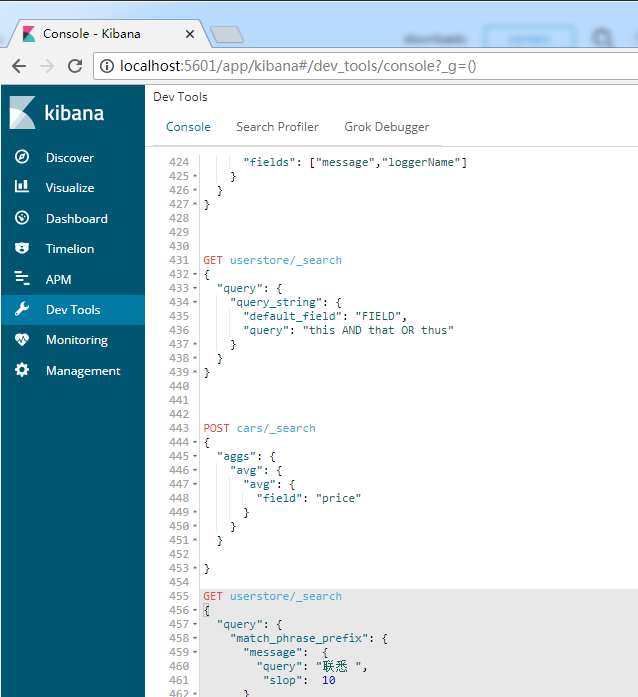
Elasticsearch也支持sql语句，但是dsl功能更强大。可以某种程度上把es理解成一个nosql数据库。默认分词英文以单词分词，中文以单个字符分词。

Dsl查询是以restapi进行的。查询的url

GET userstore/\_search

Userstore为index的名称

在kibana的devtool里有编辑器可以方便使用



详细内容参照官方文档，这里只列举最简单的用法，由于elasticsearch的查询功能实在是强大，深入的使用请参阅官方文档。特别涉及分词的时候，如何选择最优化的结果。

# 查询所有

GET userstore/\_search

{

"query": {

"match\_all": {}

}

}

# 全文本搜索

返回的结果有个max\_score表示与结果的匹配程度

## match

GET userstore/\_search

{

"query": {

"match": {

"level": "Error"

}

}

}

注意不分大小写 模糊匹配查询 注意中文和英文分词的不同，对于英文是单词匹配，比如a.b.c is 这种字符串a.b.c作为一个词进行匹配。如果是汉子则对每一个汉子匹配。

## match\_phrase

GET userstore/\_search

{

"query": {

"match\_phrase": {

"message": "Phone"

}

}

}

只要包含这个单词就可以 默认分词下中文用此方式搜索

## match\_phrase\_prefix

{

"query": {

"match\_phrase\_prefix": {

"message": {

"query": "悉 若",

"slop": 10

}

}

}

}

只要message包含 悉 若 两个字就算命中，无关顺序，无关两个字是否在一起。

## multi\_match

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html#type-best-fields>

GET userstore/\_search

{

"query": {

"multi\_match": {

"query": "眠麻寿",

"fields": ["message","detail\_message.data.userName"]

}

}

}

Fields包含query中的字符就可以了，只要有一个字符就算是命中，不写就是全文搜索。

GET userstore/\_search

{

"query": {

"multi\_match": {

"query": "麻旋牡",

"type": "phrase\_prefix",

"fields": ["message"]

}

}

}

使用"phrase\_prefix查询严格按照字符顺序进行查询。Message里包含麻旋牡字符才是命中查询。

## Common查询

GET /\_search

{

"query": {

"common": {

"body": {

"query": "this is bonsai cool",

"cutoff\_frequency": 0.001

}

}

}

}

大概意思就是 this is bonsai cool 是做分词查询，this is是英语差用词汇，所以结果里肯定大部分都是这种，cutoff\_frequency直接过滤掉这些常用词。具体我也不知道。

## Bool查询

{

"query": {

"bool": {

"must": { "match": { "title": "quick" }},

"must\_not": { "match": { "title": "lazy" }},

"should": [

{ "match": { "title": "brown" }},

{ "match": { "title": "dog" }}

]

}

}

}

以上的查询结果返回 title 字段包含词项 quick 但不包含 lazy 的任意文档。目前为止，这与 bool 过滤器的工作方式非常相似。

区别就在于两个 should 语句，也就是说：一个文档不必包含 brown 或 dog 这两个词项，但如果一旦包含，我们就认为它们 更相关 ：

{

"query": {

"bool": {

"must": [

{

"match\_phrase": {

"loggerName": "com.aorise.controller.TestController"

}

}

],

"filter": {

"range": {

"@timestamp": {

"gte": "2018-06-27T06:02:55.731Z",

"lte": "2018-06-27T06:03:00.149Z"

}

}

}

}

}

}

查询com.aorise.controller.TestController在某个时间段的log信息

## query\_string

GET userstore/\_search

{

"query": {

"query\_string": {

"default\_field": "threadId",

"query": "(43) OR (1)"

}

}

}

将query 两边做两次分词查询

# term-level queries

## Term Query

{

"query": {

"term": {

"level": {

"value": "error"

}

}

}

}

查找error的log，此查询使用了分词，如果error为err则匹配不到error，中文分词是以单个汉字匹配的。

## Terms

{

"query": {

"terms": {

"level": [

"info",

"error"

]

}

}

}

匹配info和error的log。

## terms\_set

PUT /my-index

{

"mappings": {

"\_doc": {

"properties": {

"required\_matches": {

"type": "long"

}

}

}

}

}

PUT /my-index/\_doc/1?refresh

{

"codes": ["ghi", "jkl"],

"required\_matches": 2

}

PUT /my-index/\_doc/2?refresh

{

"codes": ["def", "ghi"],

"required\_matches": 2

}

GET /my-index/\_search

{

"query": {

"terms\_set": {

"codes" : {

"terms" : ["abc", "def", "ghi"],

"minimum\_should\_match\_field": "required\_matches"

}

}

}

}

## Range

{

"query": {

"range": {

"@timestamp": {

"gte": "2018-06-27T06:02:55.731Z",

"lte": "2018-06-27T06:03:00.149Z"

}

}

}

}

查询指定时间之间的log

GET \_search

{

"query": {

"range" : {

"date" : {

"gte" : "now-1d/d",

"lt" : "now/d"

}

}

}

}

## Exists Query

GET /\_search

{

"query": {

"exists" : { "field" : "user" }

}

}

## prefix

{

"query": {

"prefix": {

"level": {

"value": "inf"

}

}

}

}

和term不同在于这个可以部分搜索。

## Wildcard Query

GET userstore/\_search

{

"query": {

"wildcard": {

"level": {

"value": "i\*o"

}

}

}

}

匹配与通配符表达式具有匹配字段的文档（**not analyzed**）。支持的通配符是 “\*”，它匹配任何字符序列（包括空字符）；还有 “？”，它匹配任何单个字符。请注意，此查询可能很慢，因为它需要迭代多个项。为了防止极慢的通配符查询，通配符项不应以通配符 “\*” 或 “？” 开头。通配符查询对应 **Lucene** 的 **WildcardQuery**。

## Regexp Query

GET userstore/\_search

{

"query": {

"regexp":{

"level":{

"value":"i.\*o",

"boost":1.2

}

}

}

}

正则表达式查询

## fuzzy\_query

<https://scsundefined.gitbooks.io/elasticsearch-the-definitive-guide-cn/content/s05/07_02_fuzzy_query.html>

## Type Query

GET /\_search

{

"query": {

"type" : {

"value" : "\_doc"

}

}

}

查询所有的type是\_doc的

# 聚合查询

POST /cars/transactions/\_bulk

{ "index": {}}

{ "price" : 10000, "color" : "red", "make" : "honda", "sold" : "2014-10-28" }

{ "index": {}}

{ "price" : 20000, "color" : "red", "make" : "honda", "sold" : "2014-11-05" }

{ "index": {}}

{ "price" : 30000, "color" : "green", "make" : "ford", "sold" : "2014-05-18" }

{ "index": {}}

{ "price" : 15000, "color" : "blue", "make" : "toyota", "sold" : "2014-07-02" }

{ "index": {}}

{ "price" : 12000, "color" : "green", "make" : "toyota", "sold" : "2014-08-19" }

{ "index": {}}

{ "price" : 20000, "color" : "red", "make" : "honda", "sold" : "2014-11-05" }

{ "index": {}}

{ "price" : 80000, "color" : "red", "make" : "bmw", "sold" : "2014-01-01" }

{ "index": {}}

{ "price" : 25000, "color" : "blue", "make" : "ford", "sold" : "2014-02-12" }

## Terms聚合查询

按照terms做聚合查询

GET /cars/transactions/\_search

{

"size": 0,

"aggs": {

"best\_colur": {

"terms": {

"field": "color.keyword",

"size": 10

}

}

}

}

查询出所有的颜色的数量

可以先查询，对查询结果做聚合

GET nursery-operation-manage/\_search

{

"query": {

"match": {

"thread": "pool-2-thread-1"

}

},

"size": 0,

"aggs": {

"logger\_level": {

"terms": {

"field": "level.keyword",

"size": 10

}

}

}

}

## Avg均值查询

{

"size": 0

, "aggs": {

"E": {

"avg": {

"field": "price"

}

}

}

}

## 多级聚合查询

{

"size": 0,

"aggs": {

"best\_colur": {

"terms": {

"field": "color.keyword",

"size": 10

},

"aggs": {

"max\_price": {

"max": {

"field": "price"

}

}

}

}

}

}