<http://ilhee.tistory.com/category/2016/ELK>

http://geniedev.tistory.com/6

### [[elasticsearch] 엘라스틱서치 시작하기](http://geniedev.tistory.com/6)

Geniedev 2016.05.12 12:18

서비스에 검색 & 추천기능 추가를 위해  
**Elasticsearch** 와 **Solr** 그리고 은전한닢에 관한 내용을 정리한다.

엘라스틱서치와 솔라의 공통점

1. 루씬을 기반으로 한 텍스트 검색 엔진 라이브러리이다.

2. Restful API를 제공한다.

3. Cache를 지원한다.

4. 분산처리에 용이하다.

5. Spring Component를 지원한다.

**spring-data**[spring-data-solr](http://projects.spring.io/spring-data-solr/) 와 [spring-data-elasticsearch](http://projects.spring.io/spring-data-elasticsearch/) 를 참고한다.

엘라스틱서치와 솔라 비교

|  |  |  |
| --- | --- | --- |
|  | Solr | ElasticSearch |
| 도큐먼트 형식 | XML, CSV, JSON | JSON |
| 다중 스키마 문서 | 지원하지 않음 | 지원 |
| 조인 | 지원하지 않음 | Parent\_type/Children\_type |
| 분산 시스템 노드 연결 | 주키퍼 | 젠 디스커버리, 주키퍼 |
| MySQL 불러오기 | dataimportHandler | logstrash |

엘라스틱서치?

엘라스틱서치는 앞서 말했듯, 루씬을 기반으로 한 텍스트 검색 엔진 라이브러리이다. 사전 매핑 없이 JSON 문서 형식으로 입력하면 별도의 이벤트가 없어도 바로 색인을 시작한다. 이렇게 저장된 데이터는 별도의 재시작/갱신 없이도 바로 검색에 사용될 수 있다. 이는 곧 색인 작업이 완료됨과 동시에 검색이 가능하다는걸 의미한다. 이러한 특징들 덕분에 솔라와 비교하여 실시간 검색 엔진 구현에 좀 더 적합하다.

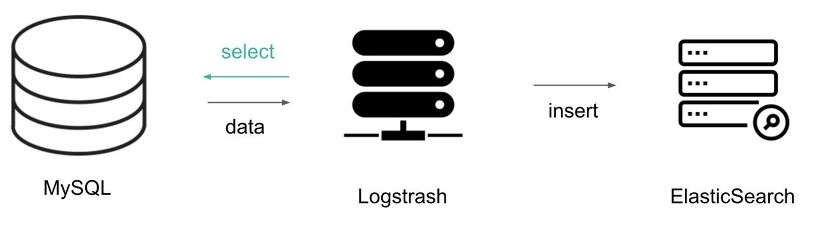
엘라스틱서치와 관계형 데이터베이스

엘라스틱서치를 쉽게 이해하기 위해 사용되는 용어들을 관계형 데이터베이스와 비교하면 쉽게 받아드릴수 있다.

|  |  |
| --- | --- |
| RDB | Elasticsearch |
| DATABASE | INDEX |
| TABLE | TYPE |
| ROW | DOCUMENT |
| COLUMN | FEILD |
| SCHEMA | MAPPING |

엘라스틱서치에서 MySQL 불러오기

1.9버전까지는 reiver라는 친구를 이용하여 MySQL에 데이터를 불러왔었는데, 2.0버전부터 공식적인 지원을 중단하였다. 엘라스틱은 이후 logstash를 이용하여 MySQL의 데이터를 불러오기를 권장하고 있다.



엘라스틱서치 설치

대충알아봤으니 설치해보자.

|  |  |  |
| --- | --- | --- |
| 1 | $ wget https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/tar/elasticsearch/2.3.2/elasticsearch-2.3.2.tar.gz | [cs](http://colorscripter.com/info#e) |

압축을 풀고 클러스터를 실행한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ ./bin/elasticsearch --cluster.name my\_cluster\_name --node.name my\_node\_name | [cs](http://colorscripter.com/info#e) |

다음과 같이 실행되는걸 볼 수 있다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | [2016-05-12 02:16:39,444][INFO ][node                     ] [Wicked] version[2.3.2], pid[3898], build[b9e4a6a/2016-04-21T16:03:47Z]  [2016-05-12 02:16:39,445][INFO ][node                     ] [Wicked] initializing ...  [2016-05-12 02:16:39,904][INFO ][plugins                  ] [Wicked] modules [reindex, lang-expression, lang-groovy], plugins [analysis-seunjeon], sites []  [2016-05-12 02:16:39,921][INFO ][env                      ] [Wicked] using [1] data paths, mounts [[/ (/dev/xvda1)]], net usable\_space [4.4gb], net total\_space [7.7gb], spins? [no], types [ext4]  [2016-05-12 02:16:39,921][INFO ][env                      ] [Wicked] heap size [1007.3mb], compressed ordinary object pointers [true]  [2016-05-12 02:16:39,921][WARN ][env                      ] [Wicked] max file descriptors [4096] for elasticsearch process likely too low, consider increasing to at least [65536]  [2016-05-12 02:16:41,595][INFO ][node                     ] [Wicked] initialized  [2016-05-12 02:16:41,599][INFO ][node                     ] [Wicked] starting ...  [2016-05-12 02:16:41,654][INFO ][transport                ] [Wicked] publish\_address {172.000.000.000:9300}, bound\_addresses {[::]:9300}  [2016-05-12 02:16:41,662][INFO ][discovery                ] [Wicked] elasticsearch/jdsJR3LSTkKcCRc8YE4rlQ  [2016-05-12 02:16:44,769][INFO ][cluster.service          ] [Wicked] new\_master {Wicked}{jdsJR3LSTkKcCRc8YE4rlQ}{172.31.6.226}{172.31.6.226:9300}, reason: zen-disco-join(elected\_as\_master, [0] joins received)  [2016-05-12 02:16:44,793][INFO ][http                     ] [Wicked] publish\_address {172.000.000.000:9200}, bound\_addresses {[::]:9200}  [2016-05-12 02:16:44,793][INFO ][node                     ] [Wicked] started    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

기본적으로 ::1:9300 으로 바운딩 되어 있는데 config/elasticsearch.yml 파일에서

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # ---------------------------------- Network -----------------------------------  #  # Set the bind address to a specific IP (IPv4 or IPv6):  #   network.host: 0.0.0.0  #  # Set a custom port for HTTP:  #  # http.port: 9200    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

이부분을 수정해주면 된다.

로그스태시 설치

MySQL을 import 하기 위해서 logstash를 설치하자.

|  |  |  |
| --- | --- | --- |
| 1 | $ wget https://download.elastic.co/logstash/logstash/logstash-2.1.1.tar.gz | [cs](http://colorscripter.com/info#e) |

기본설정파일을 작성하고

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\* logstash.conf \*/  input {      stdin {}  }    output {      elasticsearch {          hosts => ["localhost:9200"]      }      stdout {          codec => rubydebug      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

적용한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ bin/logstash -f logstash.conf --configtest | [cs](http://colorscripter.com/info#e) |

Jdbc plugin을 설치한다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4 | $ bin/plugin list jdbc$ bin/plugin install logstash-input-jdbc  $ wget http://xbib.org/repository/org/xbib/elasticsearch/importer/elasticsearch-jdbc/2.1.1.2/elasticsearch-jdbc-2.1.1.2-dist.zip  $ sudo apt-get install unzip  $ cp elasticsearch-jdbc-2.1.1.2/lib/mysql-connector-java-5.1.33.jar lib/  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

Jdbc plugin을 위한 설정파일을 작성하고

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /\* logstash.conf \*/  input {      jdbc {          jdbc\_driver\_library => "lib/mysql-connector-java-5.1.33.jar"          jdbc\_driver\_class => "com.mysql.jdbc.Driver"          jdbc\_connection\_string => "jdbc:mysql://localhost:3306/[db-name]"          jdbc\_user => "[db-user]"          jdbc\_password => "[password]"          statement => "SELECT \* FROM [tbl-name]"          schedule => "\* \* \* \* \*"      }  }  output {      elasticsearch {          hosts => ["localhost:9200"]      }    stdout {          codec => rubydebug      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

적용하면 끝

|  |  |  |
| --- | --- | --- |
| 1  2 | $ bin/logstash -f logstash-jdbc.conf --configtest  $ bin/logstash -f logstash-jdbc.conf | [cs](http://colorscripter.com/info#e) |

은전한닢 설치

한국에서 서비스할꺼고, 검색어는 한국어일테니 형태소로 나누어 검색이 가능하게 해야한다.  
훌륭한 오픈소스 한국어 형태소 분석기인 은전한닢을 적용하자.

|  |  |  |
| --- | --- | --- |
| 1 | $ ./bin/plugin install org.bitbucket.eunjeon/elasticsearch-analysis-seunjeon/2.3.2.0 | [cs](http://colorscripter.com/info#e) |

**은전한닢**<https://bitbucket.org/eunjeon/seunjeon/src/6e8a067fb9a12bcdcdd7f858fd84714c94835f04/elasticsearch/>

인덱스 생성

설치가 모두 끝났으니 간단히 사용해 보자. 우선 데이터베이스에 해당하는 인덱스를 생성한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X POST http://ip:9200/test | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | {      "mappings" : {          "users" : {              "properties" : {                  "index" : { "type" : "integer" },                  "email" : { "type" : "string" },                  "nickName" : { "type" : "string"},                  "tag" : { "type" : "string"}              }          }      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

이번엔 은전한닢이 분석기로 적용된 인덱스를 생성한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X POST http://ip:9200/enjun | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | {      "settings" : {          "analysis" : {                                                   // 은전한닢 형태소 분석기 적용              "analyzer" : {                  "korean\_analyzer" : {                      "type":"custom",                      "tokenizer":"seunjeon\_default\_tokenizer"                  }              },              "tokenizer": {                  "seunjeon\_default\_tokenizer": {                      "type": "seunjeon\_tokenizer"                  }              }          }      },      "mappings" : {          "users" : {              "properties" : {                  "index" : { "type" : "integer" },                  "email" : { "type" : "string" },                  "nickName" : { "type" : "string", "analyzer" : "korean\_analyzer" },                  "tag" : { "type" : "string", "analyzer" : "korean\_analyzer"                  }              }          }      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

도큐먼트 생성

두 인덱스 모두 도큐멘트를 생성한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X POST http://ip:9200/test/uesrs/1 | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X POST http://ip:9200/test/enjun/1 | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | {      "email":"test@gmail.com",      "nickName":"지니",      "tag":"보도자료"  } | [cs](http://colorscripter.com/info#e) |

도큐먼트 검색

태그가 '보도v자료' 와 일치하는 사용자를 찾을 것이다. 우선 형태소분석기가 적용되지 않은 인덱스부터

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X GET http://ip:9200/test/uesrs/\_search?q=tag:'보도 자료' | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | {      "took": 8,      "timed\_out": false,      "\_shards": {          "total": 5,          "successful": 5,          "failed": 0      },      "hits": {          "total": 0,          "max\_score": null,          "hits": []                 // 인식하지 못함      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

'보도자료'를 하나의 키워드로 인식하여 검색결과가 하나도 없는것을 볼 수 있다.

이번엔 형태소 분석기가 적용된 인덱스에서 검색한다.

|  |  |  |
| --- | --- | --- |
| 1 | $ curl -X GET http://ip:9200/enjun/uesrs/\_search?q=tag:'보도 자료' | [cs](http://colorscripter.com/info#e) |

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | {      "took": 8,      "timed\_out": false,      "\_shards": {          "total": 5,          "successful": 5,          "failed": 0  },      "hits": {          "total": 1,          "max\_score": 0.028130025,          "hits": [{              "\_index": "enjun",              "\_type": "users",              "\_id": "1",              "\_score": 0.028130025,              "\_source": {                  "email": "test@gmail.com",                  "nickName": "지니",                  "tag": "보도자료"                       // ‘보도’ + ‘자료’ 로 인식              }          }]      }  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

참고자료

ElasticSearch Search API Documents

<https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html>

ElasticSearch - MySQL Import

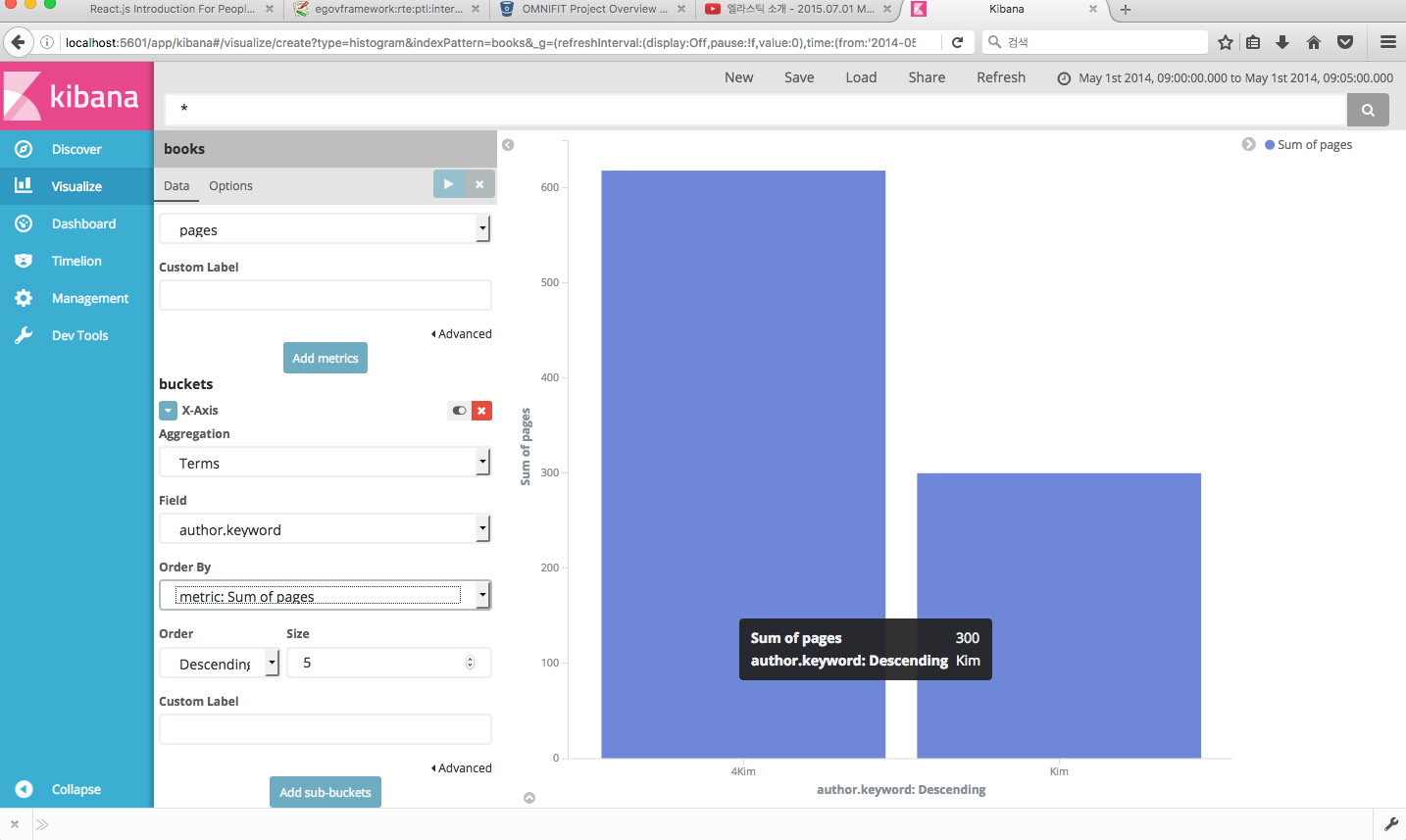
<http://digndig.kr/elasticsearch/756/>

<https://github.com/jprante/elasticsearch-jdbc>

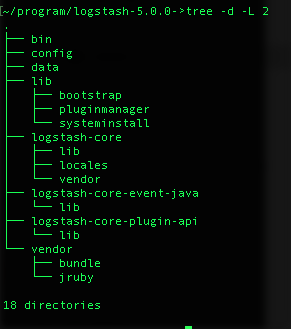
ElasticSearch - 개요

<http://www.slideshare.net/seunghyuneom/elastic-search-52724188>

키바나



로그스태시 로 로그파일 엘라스틱에 넣기



|  |  |
| --- | --- |
| logstash -f configfile | 로그스태시 실행 |
|  | |
| input{  stdin{}  }  output{  stdout{}  } | |

json으로 바꿔서보내기

|  |
| --- |
| input{  stdin{}  }  output{  stdout{  codec=>json  }  } |
| {"@timestamp":"2016-11-06T11:02:25.105Z","@version":"1","host":"HYUNHA-KIMui-MacBook-Pro.local","message":"show me the money"} |

입력, 출력 json으로

|  |  |
| --- | --- |
| {"title": "Elasticsearch Guide","author": "4Kim","date": "2014-05-05","pages": 305} | {"date":"2014-05-05","pages":305,"@timestamp":"2016-11-06T11:05:35.396Z","author":"4Kim","@version":"1","host":"HYUNHA-KIMui-MacBook-Pro.local","title":"Elasticsearch Guide"} |
|  | |
| input{  stdin{  codec=>json  }  }  output{  stdout{  codec=>json  }  } | |
|  | |

엘라스틱에게 보내기

|  |
| --- |
| output { elasticsearch {  cluster => "elasticsearch"  node\_name => "node-logstash"  index => "tests"  document\_type => "test- %{+YYYY.MM.dd}"  id => "%{id}"  } } |
| input{  stdin{  codec=>json  }  }  output{  elasticsearch {  hosts => ["localhost:9200"]  index => "books"  }  stdout { codec => rubydebug }  }    이렇게 보내지며    보내진걸 확인해보면 \_type이 log로 들어간다  아이디값도 입력이 안되어서 자동으로 들어간다 하지만 |
|  |
| 데이터값 파싱해서 처리하기 DATE 날짜로  date { match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"] target => "@timestamp" }    보내고 확인해보니.  ~/program->curl -XGET localhost:9200/\_search?q=author:vva  {"took":2,"timed\_out":false,"\_shards":{"total":11,"successful":11,"failed":0},"hits":{"total":1,"max\_score":0.2876821,"hits":[{"\_index":"books","\_type":"book","\_id":"%{id}","\_score":0.2876821,"\_source":{"date":"2014-05-03","pages":303,"@timestamp":"2016-11-06T11:37:33.032Z","author":"vva","@version":"1","host":"HYUNHA-KIMui-MacBook-Pro.local","title":"Elasticsearch Guide"}}]}}  ~/program→  인덱스와 타입이 잘들어간걸 확인할수 있다  아이디값이 안들어갔는데.  아이디값이 입력된 아이디값으로 취한하는건데 그것이 없으니깐 메타스트링이 그대로들어갔다 그래서 넣고싶다면  {"id" : "4", "title": "Elasticsearch Guide","author": "vva","date": "2014-05-03","pages": 303}  이렇게 입력을 하면된다. |
| #http://stackoverflow.com/questions/25156517/logstash-date-parsing-as-timestamp-using-the-date-filter  date {  locale => "en"  match => ["logTimestamp", "YYYY-MM-dd;HH:mm:ss.SSS", "ISO8601"]  timezone => "Europe/Vienna"  target => "@timestamp"  add\_field => { "debug" => "timestampMatched"}  } |

파일입력으로 처리하기

|  |
| --- |
| input{  file{  codec=>json  path => "/Users/visualkhh/program/logstash-5.0.0/temp/\*.log"  }  }  output{  elasticsearch {  hosts => ["localhost:9200"]  index => "books"  document\_type => "book"  document\_id => "%{id}"  }  stdout { codec => rubydebug }  } |
| {"title": "Elasticsearch Guide","author": "4Kim","date": "2014-05-01","pages": 301}  {"title": "Elasticsearch Guide","author": "rrm","date": "2014-05-02","pages": 302}  {"title": "Elasticsearch Guide","author": "vva","date": "2014-05-03","pages": 303}  {"title": "Elasticsearch Guide","author": "gaga","date": "2014-05-04","pages": 304}  {"title": "Elasticsearch Guide","author": "vovo","date": "2014-05-05","pages": 305} |
| 그런뒤 로그스태시를 실행시킨다 그런데 기존에 있던 log파일내용을 읽지 않고  변화가 생겼을때 그때 만 바뀐내용만 정송한다..  즉  log 에 {"title": "Elasticsearch Guide","author": "vovv","date": "2014-05-06","pages": 305}  추가하면 이것만 보낸다.. |
| 파일 읽어드릴때 시작할위치 포지션 어떻게 잡을지  start\_position => "beginning" |

필터

|  |
| --- |
| #멀티 매칭.  input { stdin { } }  filter {    grok {      patterns\_dir => ["/tmp/patterns"]      match => { message => [        "%{ERRORCODE:error\_code}%{SPACE}%{GREEDYDATA:error\_message}",        "%{MESSAGETYPE:message\_type}%{SPACE}%{GREEDYDATA:other\_message}"        ]      }    }  }  output {    stdout {      codec => rubydebug    }  }  cite@aphrodite:~$ logstash-1.4.0/bin/logstash agent -f /tmp/logstash.conf < /tmp/example.input  {            "message" => "Error1234 no space available",           "@version" => "1",         "@timestamp" => "2014-05-24T11:31:08.768Z",               "host" => "aphrodite",         "error\_code" => "Error1234",      "error\_message" => "no space available"  }  {            "message" => "ONS4711 nice smell",           "@version" => "1",         "@timestamp" => "2014-05-24T11:31:08.769Z",               "host" => "aphrodite",       "message\_type" => "ONS4711",      "other\_message" => "nice smell" |
| grok {  patterns\_dir => "patterns"  match => [  "message", "%{SYSLOGBASE} %{POSTFIXBOUNCE}"  ]  add\_tag => ["BOUNCED"]  remove\_tag => ["\_grokparsefailure"]  named\_captures\_only => true  }  grok {  patterns\_dir => "patterns"  match => [  "message", "%{SYSLOGBASE} %{POSTFIXCLEANUP}"  ]  add\_tag => ["INTIALIZATION"]  remove\_tag => ["\_grokparsefailure"]  named\_captures\_only => true  } |
| filter { if [type] == "zimbra\_audit" { grok { break\_on\_match => false patterns\_dir => [ "/etc/logstash/patterns.d" ] match => [ "message", "%{ZIMBRA\_AUDIT\_IMAPPOP}", "message", "%{ZIMBRA\_AUDIT\_WEB}", "message", "%{ZIMBRA\_AUDIT\_WEBFAIL}", "message", "%{ZIMBRA\_AUDIT\_ACTIVESYNC}", "message", "%{ZIMBRA\_AUDIT\_CALDAV}" ] } } } |
| 최종  input {  file {  #codec=>json  path => "/home/elastic/logstash-5.0.1/\*.log"  }  }  filter {  grok {  match => { "message" => "%{WORD:rcv\_time}>##>%{GREEDYDATA:request}"}  }  date {  match => ["rcv\_time", "YYYYMMddHHmmss"]  target => "@timestamp"  }  json {  source => "request"  remove\_field=>["request","message"]  }  }  output {  elasticsearch {  hosts => ["localhost:9200"]  index => "maindcare"  document\_type => "personal"  #document\_id => "%{id}"  }  stdout { codec => rubydebug }  #stdout {  # codec => json  #}  } |
| 기존거 날리자  curl -XDELETE <http://127.0.0.1:9200/maindcare> |
| query range  GET maindcare/personal/\_search  {  "query": {  "range" : {  "rcv\_time" : {  "gt" : "20161116163554",  "lt" : "20161116172337"  }  }    }  } |

로그스태시 (Logstash) - Filter  
• 입력데이터를분해,추가,삭제,변형등의과정을거친뒤

출력으로 전송

• grok, mutate, date ...  
• 입력한 순서 대로 위에서 부터 차례대로 적용됨

로그스태시 (Logstash)

• 공통  
• add\_field => { "comment" => "My name is %{name}" } • remove\_field => [ "name", "age" ]

• grok  
• match => { "message" => "Duration: %{NUMBER:duration}" }

• mutate  
• convert => { “age" => "integer" } • lowercase => [ "name" ]  
• split => { "fieldname" => "," }

로그스태시 (Logstash) • grok

|  |
| --- |
| filter.conf |
| filter { grok {  match => { "message" =>  "%{IP:client} %{WORD:method} %{ URIPATHPARAM:request} %{NUMB ER:bytes} %{NUMBER:duration}"  } }  } |

특정 필드를 지울수도 있다

remove\_field => [“message”,”@versionb”]

config속의 주석은 # 이다. 샾

filter매칭으로 값을 처리하는거

|  |
| --- |
| input{  stdin{}  }  filter{  grok {  match => {"message"=>"%{WORD:client} %{WORD:method} %{WORD:request} %{NUMBER:bytes} %{NUMBER:duration}" }  #add\_field => { "name" => %{client}}  }  }  output{    stdout { codec => json }  } |
| 본것처럼 show me the 500 10 했을때엔 잘들어가지만 아닐때에는  그냥 들어간다 message 쪽에. 이렇게 필터링할수 있다.  grok {  match => {"message"=>"<<%{WORD:client}>> %{WORD:method} %{WORD:request} %{NUMBER:bytes} %{NUMBER:duration}" }  #add\_field => { "name" => %{client}}  }  <<show>> me the 44 33  {"duration":"33","request":"the","@timestamp":"2016-11-06T13:21:03.636Z","method":"me","bytes":"44","@version":"1","host":"HYUNHA-KIMui-MacBook-Pro.local","client":"show","message":"<<show>> me the 44 33"} |

문자열과 Json이 썩여있을때.

<http://stackoverflow.com/questions/33937936/how-to-parse-json-in-logstash-grok-from-a-text-file-line>

|  |
| --- |
| input {  file {  path => "C:/logstashold/bin/input.txt"  }  }  filter {  grok {  match => { "message" => "%{WORD:MyWord} data=%{GREEDYDATA:request}"}  }  json{  source => "request"  target => "parsedJson"  remove\_field=>["request"]  }  }  output {  file{  path => "C:/logstashold/bin/output.txt"  }  } |
| input {  stdin{  add\_field => [ "filetype", "stdin" ]  }  }  filter {  grok {  match => { "message" => "<<%{WORD:systime}>> %{GREEDYDATA:request}"}  }  json{  source => "request"  target => "parsedJson"  remove\_field=>["request"]  }  } output{  stdout { codec => json }  } |
| 여러가지패턴  input{  #stdin{}  file{  path => "/home/omnifit\_rest/log/rest/\*.log"  }  }  filter{  grok {  match => {"message"=>[  "%{TIMESTAMP\_ISO8601:date}\|%{LOGLEVEL:log\_level}\|%{JAVACLASS:java\_class}\|%{WORD:java\_method}\|%{NUMBER:java\_line}\|X-omnifit=%{GREEDYDATA:api\_header}\|RequestBody=%{GREEDYDATA:api\_body}",  "%{TIMESTAMP\_ISO8601:date}\|%{LOGLEVEL:log\_level}\|%{JAVACLASS:java\_class}\|%{JAVAMETHOD:java\_method}\|%{NUMBER:java\_line}\|%{GREEDYDATA:msg}"  ]}  }  #remove\_field=>["request","message"]  #add\_field => { "name" => %{client}}  date {  match => ["date", "YYYYMMdd HH:mm:ss"]  target => "@timestamp"  }  json{  source => "api\_header"  target => "api\_header"  }  json{  source => "api\_body"  target => "api\_body"  }  }  output {  if [log\_level] == "ERROR" {  elasticsearch {  hosts => ["119.206.205.172:80"]  index => "mindcare\_rest"  document\_type => "rest"  #document\_id => "%{id}"  }  stdout { codec => rubydebug }  }  #stdout {  # codec => json  #}  } |

<http://grokdebug.herokuapp.com/> 에서 패턴확인가능하다.

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

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

nohup ./bin/logstash -f log-config &

nohup ./logstash-5.0.1/bin/logstash -f ./logstash-5.0.1/log-rest-config & >/dev/null

한번 문자열과 json이 썩여있는걸 한번에 로그에서 추출해보자

최종

|  |
| --- |
|  |
| input {  file{  #codec=>json  path => "/Users/visualkhh/program/logstash-5.0.0/temp/\*.log"  }  }  filter {  grok {  match => { "message" => "<<%{WORD:systime}>> %{GREEDYDATA:request}"}  }  json{  source => "request"  target => "parsedJson" #여기 필드로 들어간다 안들어가게 1depth로 하고싶으면 target을 지워주면된다.  remove\_field=>["request"]  }  }  output{    elasticsearch {  hosts => ["localhost:9200"]  index => "books"  document\_type => "book"  document\_id => "%{id}"  }  stdout { codec => rubydebug }  } |
| <<zz>> {"title": "Elastics3earch Guide","author": "14Kim","date": "2014-05-08","pages": 321}  <<xx>> {"title": "Elastics2earch Guide","author": "2rrm","date": "2014-05-09","pages": 312}  <<xx>> {"title": "Elasticsearch Guide","author": "3vva","date": "2014-05-10","pages": 313}  <<xx>> {"title": "Elast4ics3earch Guide","author": "4gaga","date": "2014-05-11","pages": 314}  <<xx>> {"title": "Elasti4csearch Guide","author": "5vovo","date": "2014-05-12","pages": 315}  <<xx>> {"title": "Elasti4csearch Guide","author": "6vovv","date": "2014-05-13","pages": 315} |
| input {  file{  #codec=>json  path => "/Users/visualkhh/program/logstash-5.0.0/temp/\*.log"  }  }  filter {  grok {  match => { "message" => "<<%{WORD:id}>> %{GREEDYDATA:request}"}  }  json{  source => "request"  #target => "parsedJson"  remove\_field=>["request"]  }  }  output{    elasticsearch {  hosts => ["localhost:9200"]  index => "books"  document\_type => "book"  document\_id => "%{id}"  }  stdout { codec => rubydebug }  }  <<5>> {"title": "Elastics3earch Guide","author": "14Kim","date": "2014-05-08","pages": 321}  <<6>> {"title": "Elastics2earch Guide","author": "2rrm","date": "2014-05-09","pages": 312}  <<7>> {"title": "Elasticsearch Guide","author": "3vva","date": "2014-05-10","pages": 313}  <<8>> {"title": "Elast4ics3earch Guide","author": "4gaga","date": "2014-05-11","pages": 314}  <<9>> {"title": "Elasti4csearch Guide","author": "5vovo","date": "2014-05-12","pages": 315}  <<10>> {"title": "Elasti4csearch Guide","author": "6vovv","date": "2014-05-13","pages": 315} |

[https://evilimp79.wordpress.com/2014/10/23/logstash-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0/](https://evilimp79.wordpress.com/2014/10/23/logstash-이해하기/)

<http://jpmens.net/2012/08/09/i-grok-how-to-mutate-a-file-with-logstash/>

일단 filter에서 사용하는 경우에 match 또는 add\_field를 이용하여 E/S쪽의 인덱스에 추가적인 컬럼을 생성할 수 있다. grok 패턴처리중 %{NAME:name} 과 같은 형식은 NAME은 grok내 정의된 패턴의 ID값이고, name은 패턴으로 추출된 실제 값이다. 즉, NAME패턴으로 추출된 값이 john이라면 grok 태그 내에서 %{name} 을 사용하면 john이라는 값이 쓰여진다. 물론 match의 경우 내부적인 hash에 name = john 과 같은 key/value 형태로 적재되기 때문에 별도의 작업을 해줄필요는 없다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | grok {  match => "${NAME:name}"  add\_field => { "name" => %{name}  }  }  } |

file을 사용할 때와 log4j를 사용할 때는 기본적으로 생성되는 색인구조가 달라지는데, 이는 logstash내부적으로 사용하는 filter가 별도로 정의되어 있기 때문이다. 기본적으로 class, method 등의 정보를 추출하고, message에 실제 log message(%m) 만 적재된다.

grok의 경우 다음[URL](http://grokdebug.herokuapp.com/)에서 데이터와 패턴을 체크해볼 수 있음.

grok에서 쓰는 [RegEx](http://www.geocities.jp/kosako3/oniguruma/doc/RE.txt)

##### 주의사항

1. logstash의 file tailing은 제일 마지막 발생되는 이벤트는 catch가 안된다.
2. 최초 application load 시에 불필요한 로그가 찍히는 경우가 있음.
3. 이는 log4j 설정 등으로 해소해야 할 사항으로 판단됨.

<http://grokdebug.herokuapp.com/>

여기서 검사할수있다.

filter {

       kv {

# '&', '?', ' ' 을 기준으로 field가 쪼개진다.

                field\_split => "&? "

# split 대상은 "request" field의 값

                source => "request"

# '='를 기준으로 좌 field, 우 value

value\_split => "="

        }

}

"p\_dt" => "145796399024", "p\_lfg" => "load", "p\_sfg" => "junior", "p\_ifg" => "ctn", "p\_un" => "92698", "p\_ws" => "11", "p\_ts" => "1008", "p\_sid" => "352461072109240", "p\_rf" => "com.olleh.olltoon",

-------------------------------------------------------------------------------------------

"p\_cd" => "고양이^강아지^33"

filter {

        mutate {

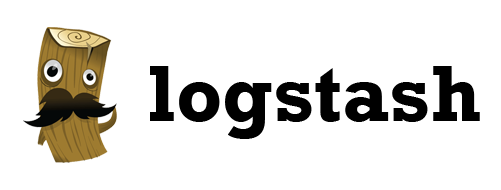
# "p\_cd" field의 값을 '^'로 구분

split => { "p\_cd" => "^" }

        }

}

"p\_cd" => [ [0] "고양이", [1] "강아지", [2] "33" ]



**로그 포맷**

[*timestmp*] [*log\_level*] [*task\_id*] *message*

**로그 예제**

[2016-07-21 05:27:19,818] [INFO] [....................................] Received task: qservice.devicemgmt.worker.GetServices[f770306e-1582-4b1e-9a6b-b5e09ca5f924]

[2016-07-21 05:27:19,819] [INFO] [f770306e-1582-4b1e-9a6b-b5e09ca5f924] message...

[2016-07-21 05:27:19,761] [INFO] [....................................] Task qservice.devicemgmt.worker.GetCapabilities[a4616504-2677-4e19-8b8e-a57180100d14] succeeded in 0.961237121024s: message...

**patterns/celerylog 작성**

task\_id 포맷의 문자열 파싱을 위한 정규식 패턴을 TASKID라는 이름으로 정의합니다.

TASKID [0-9a-z]{8}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{4}-[0-9a-z]{12}|\.{36}

**logstash.conf 작성**

1. input

input {

  file {

    path => "/var/log/celery.log"

    codec => multiline {

      pattern => "^\s|^Traceback"

      what => "previous"

    }

  }

}

celery.log 파일로부터 로그를 읽어옵니다. 이 파일에는 의도하지 않은 포맷의 로그도 포함되어 있을 수 있습니다. 예를 들어 오류가 발생한 경우 아래와 같이 여러 행의 로그가 기록됩니다. 저는 이것을 하나의 로그 단위로 저장하기 위해 multiline 코덱을 적용하였습니다. Traceback으로 시작(^Traceback)하거나 공백으로 시작(^\s)하는 행은 이전(previous) 행에 붙습니다.

[2016-07-21 05:27:20,547] [ERROR] [....................................] Task qservice.devicemgmt.worker.GetServices[f770306e-1582-4b1e-9a6b-b5e09ca5f924] raised unexpected: TypeError("'NoneType' object has no attribute '\_\_getitem\_\_'",)

**Traceback**(most recent call last):

  File "/usr/local/lib/python2.7/site-packages/celery/app/trace.py", line 240, in trace\_task

    R = retval = fun(\*args, \*\*kwargs)

  File "/usr/local/lib/python2.7/site-packages/celery/app/trace.py", line 438, in \_\_protected\_call\_\_

    return self.run(\*args, \*\*kwargs)

  File "/data/qworker/qservice/devicemgmt/worker.py", line 13, in GetServices

    return self.call("GetServices", param)

  File "/data/qworker/qservice/service.py", line 25, in call

    return func(param)

  File "/data/qworker/qservice/device/driver/plugin/onvif/devicemgmt.py", line 44, in GetServices

    print (err['test'])

2. filter

filter {

  grok {

    patterns\_dir => "/var/log/patterns"

    match => { "message" => [

     "%{TIMESTAMP\_ISO8601:timestamp}%{DATA}%{LOGLEVEL:log\_level}%{DATA}Task %{DATA:method}\[%{TASKID:task\_id}\] succeeded in (?<proc\_time>[0-9]+\.[0-9]+)s: (?<body>.+)",

     "%{TIMESTAMP\_ISO8601:timestamp}%{DATA}%{LOGLEVEL:log\_level}%{DATA}(T|t)ask(:|) %{DATA:method}\[%{TASKID:task\_id}\](?<body>.+|)",

     "%{TIMESTAMP\_ISO8601:timestamp}%{DATA}%{LOGLEVEL:log\_level}%{DATA}%{TASKID:task\_id}\] (?<body>.+)"]

    }

  }

  date {

    match => [ "timestamp", "yyyy-MM-dd HH:mm:ss,SSS" ]

    timezone => "UTC"

  }

}

filter 플러그인을 적용하여 읽어들인 로그를 재가공합니다. 먼저 grok을 사용해서 timestamp, log\_level, method, task\_id, proc\_time, body 필드를 추출하였습니다. TIMESTAMP\_ISO8601, LOGLEVEL, DATA는 [LogStash에 선언되어 있는 정규식 패턴](https://github.com/logstash-plugins/logstash-patterns-core/blob/master/patterns/grok-patterns)입니다.

파일에서 로그를 읽어들이면 해당 라인을 읽어들인 시점이 ElasticSearch의 @timestamp 필드에 기록됩니다. 저는 grok을 사용해서 추출한 timestamp를 @timestamp에 적용하기 위해 date 플러그인을 사용하였습니다.

3. output

output {

  elasticsearch {

    hosts => ["127.0.0.1:9200"]

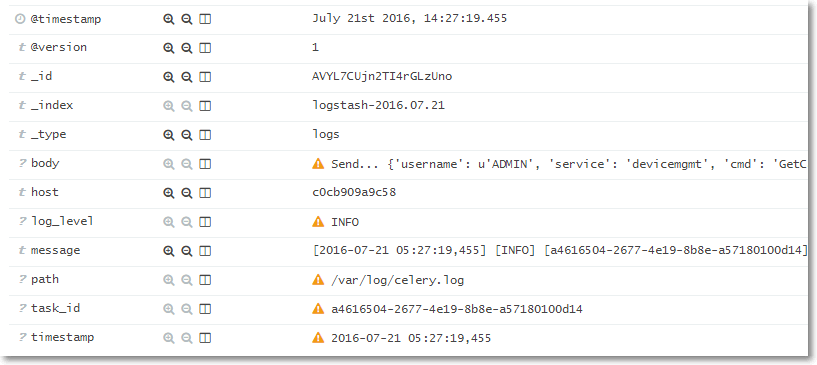
  }

  stdout {}

}

재가공된 데이터를 ElasticSearch에 저장하고 터미널에 출력되도록 설정하였습니다. 만약 ElasticSearch가 여러개의 클러스터로 구성되어 있다면 Dedicated Master Node가 아닌 Client나 Data 노드에 전달되도록 설정해야 클러스터의 안정성을 유지할 수 있다고 하네요.

**ElasticSearch에 저장된 결과 확인**



**고려사항**

Filter 플러그인을 적용하면 성능에 많은 영향을 미칠 수 있습니다. 특히 grok 필터는 정규식 계산을 위해 많은 리소스를 사용합니다. 이를 조금이나마 개선하기 위해서는 멀티코어 기반에서 병렬 처리를 수행하도록 설정해야 합니다. -w 옵션을 사용하여 스레드 개수를 설정할 수 있습니다. 예를 들어 logstash -w 8 명령어로 실행한 경우 필터 처리를 위해 8개의 다른 스레드를 사용합니다.

**참고**

* [LogStash Grok 패턴](https://github.com/logstash-plugins/logstash-patterns-core/blob/master/patterns/grok-patterns)
* [grok 정규식 테스트 사이트](http://grokconstructor.appspot.com/do/match)
* [https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html](https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.htmlhttps://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html)

<http://m.blog.naver.com/indy9052/220615243591>

### LogStash 사용하기

LogStash 는 Java 기반의 오픈소소 툴로 구조화되지 않은 일반 로그 데이터를 특정 필터로 가공하여 ElasticSearch 와 같은 검색 시스템에 전송해주는 일종의 로그 수집기다.  
input - filter - output 의 파이프 라인 구조로 플러그인을 통해 다양한 데이터 가공 및 입출력이 가능하다.  
  
# 설치  
# Java(JDK or JRE) 설치가 되어 있어야 한다.  
# 다운로드 후 압축 풀기  
# <https://www.elastic.co/downloads/logstash>  
  
# 윈도우 기준 설명  
# 설정을 커맨드에 명시하여 실행하는 경우  
# 표준 입력으로 데이터를 입력 받아 필터없이 표준 출력으로 데이터를 출력하는 설정  
bin\logstash.bat -e 'input { stdin {} } output { stdout {} }'  
  
# 설정 파일 만들기  
vim logstash.conf  
input {   
# 파일 내용이 변경되는 경우  
# file 의 필드 별 설정(@version 및 @timestamp 필드는 기본 적용)  
file {  
# type 필드 => 정해진 값은 없고 정해서 사용하면 됨  
type => "textlog"  
  
# tag 필드 => 태그를 달 수 있다.  
tags => ["tag1", "tag2", "tag3"]  
  
# 커스텀 필드를 추가 할 수 있다.  
# hash(key=>value 형태)  
add\_field => {  
"name" => "mypc"  
"ip" => "127.0.0.1"  
}  
  
# path 필드 => 로그 파일이 있는 절대 경로 명시  
path => "C:/ysoftman/\*.log"  
  
# 파일 처음부터 변경 부분 검색(디폴트 end)  
start\_position => "beginning"  
  
# 파일 상태 체크 주기 설정 (디폴트1초)  
stat\_interval => 1  
  
# 1시간으로 이전 수정된 파일은 수집에서 제외 (디폴트 86400 (하루))  
ignore\_older => 3600  
  
# 인코딩 명시(기본 UTF-8 로 취급)  
codec => plain {  
charset => "EUC-KR"  
}  
}  
}  
  
filter {  
grok {  
# 메시지 필드에 test 나 testing 스트링이 있다면  
match => { "message" => ["test" , "testing"] }  
  
# 위 조건이 참이면 ysoftman 태그 추가  
# 위 조건에 맞지 않으면 \_grokparsefailure 태그가 자동으로 추가된다.  
add\_tag => [ "ysoftman" ]  
}   
  
# \_grokparsefailure 태그가 있다면 drop (output 으로 전달되지 않는다.)  
#if "\_grokparsefailure" in [tags] {  
# drop {}  
#}  
  
# \_grokparsefailure 태그만 지우기  
mutate {  
remove\_tag => ["\_grokparsefailure"]  
}  
}  
  
output {  
# elasticsearch 서버의 주소 명시  
elasticsearch {  
# 해당 호스트로 전송  
hosts => ["localhost:9200"]  
}  
# 디버깅을 위해 표준 출력 표시  
stdout {  
# rubydebug 형식의 보기 좋은 json 포맷으로 표시  
codec => rubydebug  
}  
}

# 설정 파일 사용하여 실행하는 경우  
bin\logstash.bat agent -f logstash.conf

# 이제 지정 경로의 파일에 변화가 있다면 변경된 내용이 elasticsearch 전송되고, 콘솔에 다음과 같이 출력된다.  
{  
       "message" => "ysoftman test message...\r",  
      "@version" => "1",  
    "@timestamp" => "2016-06-01T07:32:46.249Z",  
          "path" => "C:/ysoftman/ysoftman.log",  
          "host" => "ysoftman",  
          "type" => "textlog",  
        "name" => "mypc",  
              "ip" => "127.0.0.1",  
          "tags" => [  
        [0] "tag1",  
        [1] "tag2",  
        [2] "tag3",  
        [3] "ysoftman"  
    ]  
}  
  
# ElasticSearch(Kibana) 를 통해 추가된 로그 확인해보기  
http://localhost:5601

# 플러그인 관련  
# 설치된 플러그인 버전정보 보기  
bin\plugin.bat list --verbose  
  
# 설치된 플러그인 모두 업데이트(잘못되면 logstash 다시 설치해야함)  
bin\plugin.bat update  
  
# 예) file 플러그인 설치하기  
bin\plugin.bat install logstash-input-file  
  
참고  
<https://www.elastic.co/guide/en/logstash/current/config-examples.html>  
<https://www.elastic.co/guide/en/logstash/current/plugins-inputs-file.html>  
[https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#plugins-filters-grok-match](https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#_blank)  
<https://www.elastic.co/guide/en/logstash/current/plugins-filters-drop.html>  
<https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html>

-----

키바나

\_type:rest\_dev AND log\_level:ERROR

