

Elastic Stack 을 활용한 Data Dashboard 만들기

Week 4 - Logstash로 데이터를 전처리하고 전송하자



Fast Campus

목차

- 개요	3
- Input Plugins	12
- file (csv)	15
- file (log)	18
- database	20
- elasticsearch	25
- Output	
- slack	30
- elasticsarch	32
- Filter	
- csv	37
- date	42
- mutate	45
- grok	53

지금까지 시각화도 했고, 검색도 해봤는데 뭔가 2% 부족하다

과연 Elastic Stack으로 어떤 데이터를 분석할 수 있지?

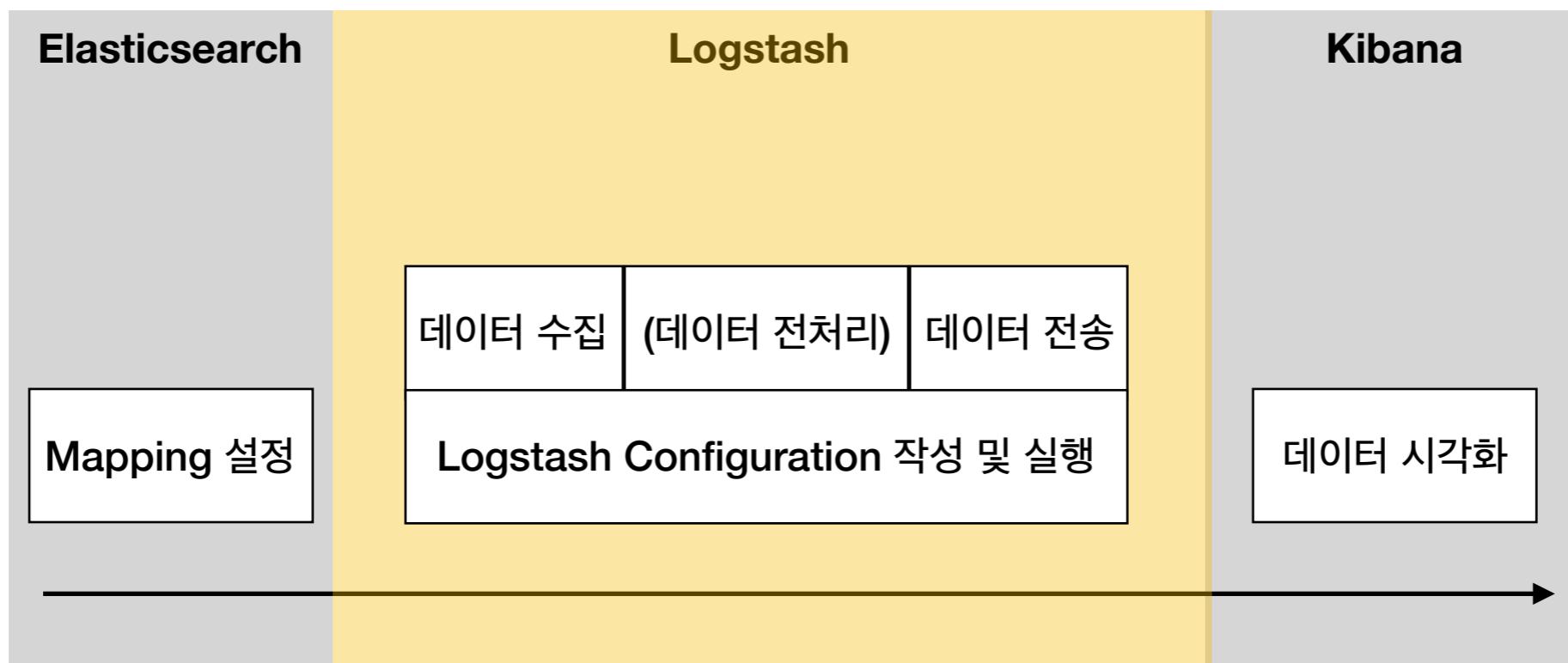
당장 컴퓨터에 있는 csv 파일부터 DB에 있는 정보까지

그리고 요즘에는 실시간으로 모니터링 하는 세상인데...

이 때 필요한 게 Logstash!

Logstash 5.6 기준 **54개** Input Plugin 존재

Logstash Workflow



(분석가 작업의 흐름)

Logstash Workflow

작업	어디에서?	상세	필수
데이터 수집	Logstash Input Plugin	File, Web, Db, Streaming, Sensors 등에서부터 데이터 수집	o
데이터 전처리	Logstash Filter Plugin	분석하기에 가장 좋은 형태로 데이터 전처리	x
데이터 전송	Logstash Output Plugin	전처리한 데이터를 분석하기에 최적의 저장소로 전송	o

**현재 사용하고 있는 AWS EC2 Instance는 t2.micro라서
Elasticsearch 및 Logstash 사양을 낮춰야 합니다**

Elasticsearch 종료

Elasticsearch process id (pid) 확인

\$ ps -ef | grep elasticsearch

```
[ec2-user@ip-172-31-15-62 logstash-5.6.4]$ ps -ef | grep elasticsearch
ec2-user 12510 1 0 Dec03 ? 00:05:23 /usr/bin/java -Xms128m -Xmx128m -XX:-
atingOccupancyOnly -XX:+AlwaysPreTouch -server -Xss1m -Djava.awt.headless=true -Dfil
e -Dio.netty.noUnsafe=true -Dio.netty.noKeySetOptimization=true -Dio.netty.recycler.
jmx=true -Dlog4j.skipJansi=true -XX:+HeapDumpOnOutOfMemoryError -Des.path.home=/home
/lib/* org.elasticsearch.bootstrap.Elasticsearch -d
ec2-user 23296 23265 0 03:09 pts/0 00:00:00 grep --color=auto elasticsearch
[ec2-user@ip-172-31-15-62 logstash-5.6.4]$
```

Elasticsearch process 종료

\$ kill -9 12510

JVM Options 조정

elasticsearch JVM options 조정

- \$ vim ~/fc/elasticsearch-5.6.4/config/jvm.options
- 편집 모드 : i 입력
- 아래 부분 수정
 - 기존

```
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
```

```
-Xms512m
-Xmx512m
```

- 수정

```
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
```

```
-Xms128m
-Xmx128m
```

- 저장 : ESC 누르고 :wq 입력

logstash JVM options 조정

- \$ vim ~/fc/logstash-5.6.4/config/jvm.options
- 편집 모드 : i 입력
- 아래 부분 수정
 - 기존

```
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
```

```
-Xms512m
-Xmx512m
```

- 수정

```
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
```

```
-Xms128m
-Xmx128m
```

- 저장 : ESC 누르고 :wq 입력

Logstash 맛보기



[ec2-user@ip-172-31-15-62 logstash-5.6.4]\$ █

Logstash Input Plugin은 정말 많다.
그 중에서 `stdin`, `file`, `es`, `db`를 살펴보자



Input - file

우선 실습에 사용할 데이터를 다운 받자 !

Logstash Home Directory 이동 `$ cd /home/ec2-user/fc/logstash-5.6.4/`

데이터 저장할 디렉토리 생성 `$ mkdir sample`

데이터 다운로드 `$ cd sample`

```
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/titanic.csv
```

```
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/titanic2.csv
```

```
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/titanic3.csv
```

```
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/test.log
```

```
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/apache.log
```

Input - file (csv)

File에서 기본적인 csv 파일을 읽어보자

우선 아래와 같이 입력하고 실행해보자

위키

```
1 input {  
2   file {  
3     path => "/home/ec2-user/fc/logstash-5.6.4/sample/titanic.csv"  
4   }  
5 }  
6  
7 output {  
8   stdout {  
9     codec => rubydebug  
10  }  
11 }
```

아무 것도 안 나온다.

Input - file (csv)

그 이유는 **start_position**이라는 옵션이 end로 설정되어 있기 때문이다.

즉, 뒤에서부터 보겠다는 것인데 이걸 **beginning**으로 해줘야 된다.

왜 그럼 이렇게 불편하게 default가 end로 되어 있을까?

그건 바로 streaming data에 대응하기 위해서다.

어쨌든 제대로 아래와 같이 코드를 변경해서 다시 해보자.

코드

```
1 input {  
2   file {  
3     path => "/home/ec2-user/fc/logstash-5.6.4/sample/titanic2.csv"  
4     start_position => "beginning"  
5   }  
6 }  
7  
8 output {  
9   stdout {  
10    codec => rubydebug  
11  }  
12 }
```

Input - file (log)

이번에는 log 파일이다

데이터를 다운 받고 한 번 콘솔에 출력시켜보자.

확장자가 .log 이기에 .csv와 달라보이나 logstash에게는 둘 다 file이므로 차이가 없다.

start_position => “beginning” 설정에 주의해서 한 번 Logstash configuration을 작성해보자.

[코드](#)

```
1 input {  
2   file {  
3     path => "/home/ec2-user/fc/logstash-5.6.4/example/apache.log"  
4     start_position => "beginning"  
5   }  
6 }  
7  
8 output {  
9   stdout {  
10    codec => rubydebug  
11  }  
12 }
```

클릭

File Input Plugin을 넘어가기 전에 옵션 몇 가지만 보자.
(단, Logstash를 처음 접할 경우 가볍게!)

이번에는 인기 Plugin 중 하나인 **jdbc**를 보자

Input - jdbc

Description



This plugin was created as a way to ingest data in any database with a JDBC interface into Logstash. You can periodically schedule ingestion using a cron syntax (see `schedule` setting) or run the query one time to load data into Logstash. Each row in the resultset becomes a single event. Columns in the resultset are converted into fields in the event.

...라고 하니 안심하고 database 데이터를 읽고 싶을 땐 jdbc를 사용하자

단, 사용하기에 앞서 몇 가지 설치를 해야 한다.

아래 링크에 있는 절차를 따라해보자.

<https://github.com/higee/elastic/wiki/Logstash-Input#logstash-db>

Input - jdbc

코드

```
1 input {  
2   jdbc {  
3     jdbc_validate_connection => true  
4     jdbc_connection_string => "jdbc:mysql://13.125.21.52:3306/fc"  
5     jdbc_driver_library => "/home/ec2-user/fc/logstash-5.6.4/drive  
6     jdbc_driver_class => "com.mysql.jdbc.Driver"  
7     jdbc_user => "fc"  
8     jdbc_password => "fc"  
9     statement => "SELECT * FROM fc"  
10    }  
11  }  
12  
13 output {  
14   stdout {  
15     codec => rubydebug  
16   }  
17 }
```

~



실질적으로 Database에서 데이터를 읽어오는 부분

Input - jdbc

database 데이터입니다.

다음 결과를 콘솔에 출력해보세요

1. 국가별 평균 연봉
2. employment_status 별 평균 연봉
3. 연봉이 10,000 이상인 사람의 수
4. 30세 이하의 평균 연봉

id age salary name location employment_status
1 33 4000 Josh US employed
2 33 45000 Tom US employed
3 23 3000 Kirk US employed
4 27 3500 Ken US employed
5 38 4500 Jessie US employed
6 31 5200 Jennifer US unemployed
7 33 22200 Bob US unemployed
8 25 2200 John US unemployed
9 45 4200 Aaron UK unemployed
10 45 6200 Hanks UK unemployed
11 43 8200 Gordon UK employed
12 47 11200 Searl UK employed
13 37 9900 Waterhouse UK employed
14 25 3900 Lisa UK employed
15 28 4500 Kelly UK employed
16 38 47000 Tim UK employed
17 38 3500 Tanaka JP employed
18 28 2500 Kogawa JP unemployed
19 31 3500 Oota JP employed
20 30 3700 Yamada JP employed
21 44 4500 Saeki JP employed
22 44 4500 Abe JP employed
23 42 4200 Murata JP unemployed
24 46 5600 Yamauchi JP unemployed
25 43 8600 Siranui JP unemployed
26 48 8200 Kawakami JP unemployed
27 48 8200 Kim KR unemployed
28 48 3200 Lee KR unemployed
29 48 3500 Kwon KR unemployed
30 28 3300 Kang KR employed
31 23 3500 Yoon KR unemployed
32 23 3600 Yang KR unemployed
33 26 3800 Park KR employed

JDBC Input Plugin을 넘어가기 전에 옵션 몇 가지만 보자.
(단, Logstash를 처음 접할 경우 가볍게!)

Input - elasticsearch

Logstash Input Plugin에 Elasticsearch가 있다?

Index를 다른 Elasticsearch cluster에 옮기거나 csv로 export할 때 이렇게 사용하기도 한다

jdbc와 달리 따로 설치할 게 없으니 바로 사용해보자

```
1 input {  
2   elasticsearch {  
3     hosts => ["13.125.21.52:9200"]  
4     index => "test"  
5     query => '{ "query": { "match_all": {} }}'  
6   }  
7 }  
8  
9 output {  
10   stdout {  
11     codec => rubydebug  
12   }  
13 }
```

실질적으로 데이터를
읽어오는 부분

Input - elasticsearch

넘어가기 전에 다음 query를 작성해서 콘솔에 결과를 출력해보자

1. 구매사이트가 “옵션” 또는 “11번가”인 Documents
2. 위의 고객들의 평균 나이
3. 판매자 평점이 2~4 사이인 고객의 평균 상품개수

Input - exercise

1. 사용자 입력 (stdin)을 받아서 콘솔에 출력 (stdout)해보자
2. (이미 읽은) titanic.csv를 강제로 다시 읽어서 콘솔에 출력해보자
3. db에서 id > 5인 data를 sql_last_value를 이용해서 구해보자
 - db host : 13.125.21.52:3306
 - jdbc user : fc
 - jdbc password : fc
 - table name : fc
 - 기본 statement 예시 : “SELECT * FROM fc”

Logstash Output Plugin도 정말 많다.
여기서는 Elasticsearch와 Slack (비공식)을 살펴보자.

Logstash 결과와 관련된 알림을 받을 수 없을까?

Elastic에서 email Output Plugin을 제공하나 다른 알림과
마찬가지로 Slack에서 알림을 받고 싶은 생각 강함

1. Slack Webhook 설정
2. Logstash Slack Output Plugin 설치



incoming-webhook APP

축하드립니다! US 국적의 23살님의 연봉이 1억원을 돌파했습니다

축하드립니다! JP 국적의 55살님의 연봉이 1억원을 돌파했습니다

축하드립니다! KR 국적의 33살님의 연봉이 1억원을 돌파했습니다

축하드립니다! KR 국적의 40살님의 연봉이 1억원을 돌파했습니다

```
input {  
  jdbc {  
    jdbc_validate_connection => true  
    jdbc_connection_string => "jdbc:mysql://13.125.21.52:3306/fc"  
    jdbc_driver_library => "/home/ec2-user/fc/logstash-5.6.4/driver/mysql-connector-java-5.1.36/r  
    jdbc_driver_class => "com.mysql.jdbc.Driver"  
    jdbc_user => "fc"  
    jdbc_password => "fc"  
    statement => "SELECT * FROM fc"  
  }  
}  
  
output {  
  if [salary] > 10000  
  {  
    slack {  
      url => ["https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXX"]  
      format => "축하드립니다! %{location} 국적의 %{age}살님의 연봉이 1억원을 돌파했습니다"  
    }  
  }  
  stdout {  
    codec => rubydebug  
  }  
}
```

**사실 바로 이 Elasticsearch Output Plugin을
사용하는 부분이 Elastic Stack의 핵심이다**

큰 구조는 Input Plugin 실습 할 때와 유사하다

Output Plugin에 elasticsearch를 사용한다는 점이다.

우선 jdbc input을 받아서 elasticsearch로 전송해보자.

이 때, 디버깅용으로 stdout output plugin도 병행해서 사용하자.

미리 mapping을 설정하고
전송하면 깔끔하게 들어간다

```
1 input {  
2   jdbc {  
3     jdbc_connection_string => "jdbc:mysql://13.125.21.52:3306/fc"  
4     jdbc_validate_connection => true  
5     jdbc_user => "fc"  
6     jdbc_password => "fc"  
7     jdbc_driver_library => "/home/ec2-user/fc/logstash-5.6.4/driver/r  
8     jdbc_driver_class => "com.mysql.jdbc.Driver"  
9     statement => "SELECT * FROM fc"  
10    }  
11  }  
12  
13 output {  
14   elasticsearch {  
15     hosts => ["13.125.35.175:9200"]  
16     index => "fc2"  
17     document_type => "fc2"  
18     document_id => "%{salary}_%{name}"  
19   }  
20   stdout {  
21     codec => rubydebug  
22   }  
23 }
```

?

Output - elasticsearch

```
13 output {  
14   elasticsearch {  
15     hosts => ["13.125.35.175:9200"]  
16     index => "fc2"  
17     document_type => "fc2"  
18     document_id => "%{salary}_%{name}"  
19   }  
20   stdout {  
21     codec => rubydebug  
22   }  
23 }
```

```
13 output {  
14   elasticsearch {  
15     hosts => ["13.125.35.175:9200"]  
16     index => "fc2"  
17     document_type => "fc2"  
18     document_id => "%{name}_%{@version}"  
19   }  
20   stdout {  
21     codec => rubydebug  
22   }  
23 }
```

Field Value를 받아서 사용하는 경우

- 일반 Field : %{필드명}
- Metadata Field : %{@필드명}

특정 Document의 값이 다음과 같다고 하자

- salary : 500
- name : John
- @version : 1

이 때 elasticsearch의 Document ID는 다음과 같다

- 파란색처럼 설정한 경우 : “500_John”
- 핑크색처럼 설정한 경우 : “John_1”

1. 사용자 입력 (stdin)을 받아서 사용자 입력값을 id로 갖는 Documents를 elasticsearch에 전송해보자
 - elasticsearch host : 13.125.21.52:9200
 - elasticsearch index : wee4_exercise_stdin_{id}
2. titanic3.csv를 elasticsearch에 전송하고 콘솔에도 출력해보자
 - elasticsearch host : 13.125.21.52:9200
 - elasticsearch index : wee4_exercise_titanic_{id}
3. db에서 id > 5인 모든 데이터를 sql_last_value를 이용해서 elasticsearch에 전송해보자
 - db host : 13.125.21.52:3306
 - jdbc user : fc
 - jdbc password : fc
 - table name : fc

**Filter는 Logstash로 들어온 데이터를
가공해서 분석하기 쉽게 도아준다.**

그 중에서 add_field, remove_field, csv, date, mutate, grok를 살펴보자

Filter - csv

csv는 ,로 구분된 값의 집합이다.
이 값을 의미 있는 단위로 변환할 수 있지 않을까?

1	1,Panula,0,3,male,7,4,1,3101295,39.6875,S
2	52,Nosworthy,0,3,male,21,0,0,A/4. 39886,7.8,S
3	53,Harper,1,1,female,49,1,0,PC 17572,76.7292,C
4	54,Faunthorpe,1,2,female,29,1,0,2926,26,S
5	55,Ostby,0,1,male,65,0,1,113509,61.9792,C
6	56,Woolner,1,1,male,29.69911765,0,0,19947,35.5,S
7	57,Rugg,1,2,female,21,0,0,C.A. 31026,10.5,S
8	58,Novel,0,3,male,28.5,0,0,2697,7.2292,C
9	59,West,1,2,female,5,1,2,C.A. 34651,27.75,S
10	60,Goodwin,0,3,male,11,5,2,CA 2144,46.9,S
11	61,Sirayanian,0,3,male,22,0,0,2669,7.2292,C
12	62,Icard,1,1,female,38,0,0,113572,80,C
13	63,Harris,0,1,male,45,1,0,36973,83.475,S
14	64,Skoog,0,3,male,4,3,2,347088,27.9,S
15	65,Stewart,0,1,male,29.69911765,0,0,PC 17605,27.7208,C
16	66,Moubarek,1,3,male,29.69911765,1,1,2661,15.2458,C
17	67,Nye,1,2,female,29,0,0,C.A. 29395,10.5,S
18	68,Crease,0,3,male,19,0,0,S.P. 3464,8.1583,S
19	69,Andersson,1,3,female,17,4,2,3101281,7.925,S
20	70,Kink,0,3,male,26,2,0,315151,8.6625,S
21	71,Jenkin,0,2,male,32,0,0,C.A. 33111,10.5,S
22	72,Goodwin,0,3,female,16,5,2,CA 2144,46.9,S
23	73,Hood,0,2,male,21,0,0,S.O.C. 14879,73.5,S
24	74,Chronopoulos,0,3,male,26,1,0,2680,14.4542,C
25	75,Bing,1,3,male,32,0,0,1601,56.4958,S
26	76,Moen,0,3,male,25,0,0,348123,7.65,S

Filter - csv

- 특정 구분자를 통해서 event message를 나눈다.

```
1 input {  
2   file {  
3     path => "/home/ec2-user/fc/logstash-  
4     start_position => "beginning"  
5     since_db_path => "/dev/null"  
6   }  
7 }  
8  
9 filter {  
10   csv {  
11     separator => ","  
12   }  
13 }  
14  
15 output {  
16   stdout {  
17     codec => rubydebug  
18   }  
19 }
```



```
}  
{  
  "column1" => "48",  
  "column11" => "Q",  
  "column10" => "7.75",  
  "column5" => "female",  
  "column4" => "3",  
  "column3" => "1",  
  "column2" => "O'Driscoll",  
  "message" => "48,O'Driscoll,1,3,female,29.69911765",  
  "path" => "/home/ec2-user/fc/logstash-5.6.1.log",  
  "@timestamp" => 2017-12-05T07:52:24.139Z,  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "column9" => "14311",  
  "column8" => "0",  
  "column7" => "0",  
  "column6" => "29.69911765"  
}  
{  
  "column1" => "49",  
  "column11" => "C",  
  "column10" => "21.6792",  
  "column5" => "male",  
  "column4" => "3",  
  "column3" => "0",  
  "column2" => "Samaan",  
  "message" => "49,Samaan,0,3,male,29.69911765",  
  "path" => "/home/ec2-user/fc/logstash-5.6.1.log",  
  "@timestamp" => 2017-12-05T07:52:24.140Z,  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "column9" => "2662",  
  "column8" => "0",  
  "column7" => "2",  
  "column6" => "29.69911765"  
}  
{  
  "column1" => "50",  
  "column11" => "S",  
  "column10" => "17.8",  
}
```

Filter - csv

- 특정 구분자를 통해서 event message를 나눈다.
- column 이름 생성

```
input {  
  file {  
    path => "/home/ec2-user/fc/logstash-5.6.4/sample/  
    sincedb_path => "/dev/null"  
    start_position => "beginning"  
  }  
}  
  
filter {  
  csv {  
    columns => ["Index", "Name", "Survival", "Pclass"]  
    separator => ","  
  }  
}  
  
output {  
  stdout {  
    codec => rubydebug  
  }  
  elasticsearch {  
    hosts => ["13.125.35.175:9200"]  
    index => "fc"  
  }  
}
```



```
"@version" => "1",  
"host" => "ip-172-31-15-62",  
"Age" => "27"  
}  
}  
  
"Embarked" => "C",  
"Pclass" => "3",  
"Ticket" => "349253",  
"Sex" => "male",  
"Index" => "43",  
"SibSp" => "0",  
"message" => "43,Kraeff,0,3,male,29.69911765,0,0,3",  
"Survival" => "0",  
"Name" => "Kraeff",  
"Fare" => "7.8958",  
"path" => "/home/ec2-user/fc/logstash-5.6.4/san",  
"@timestamp" => 2017-12-05T08:03:26.251Z,  
"Parch" => "0",  
"@version" => "1",  
"host" => "ip-172-31-15-62",  
"Age" => "29.69911765"  
}  
}  
  
"Embarked" => "C",  
"Pclass" => "2",  
"Ticket" => "SC/Paris 2123",  
"Sex" => "female",  
"Index" => "44",  
"SibSp" => "1",  
"message" => "44,Laroche,1,2,female,3,1,2,SC/Paris",  
"Survival" => "1",  
"Name" => "Laroche",  
"Fare" => "41.5792",  
"path" => "/home/ec2-user/fc/logstash-5.6.4/san",  
"@timestamp" => 2017-12-05T08:03:26.251Z,  
"Parch" => "2",  
"@version" => "1",  
"host" => "ip-172-31-15-62",  
"Age" => "3"
```

- 특정 구분자를 통해서 event message를 나눈다.
- column 이름 생성
- Field data type 변경

```

1 input {
2   file {
3     path => "/home/ec2-user/fc/logstash/conf.d/filter.conf"
4     since_db_path => "/dev/null"
5     start_position => "beginning"
6   }
7 }
8
9 filter {
10   csv {
11     columns => ["Index", "Name", "Survival"]
12     separator => ","
13     convert => {
14       "Pclass" => "integer"
15       "Index" => "integer"
16       "Survival" => "integer"
17       "Age" => "integer"
18       "Fare" => "float"
19     }
20   }
21 }
22
23 output {
24   stdout {
25     codec => rubydebug
26   }
27   elasticsearch {
28     hosts => ["13.125.35.175:9200"]
29     index => "fc"
30   }
31 }
```



```

"@version" => "1",
"host" => "ip-172-31-15-62",
"Age" => "29.69911765"
}

{
  "Embarked" => "S",
  "Pclass" => 3,
  "Ticket" => "349237",
  "Sex" => "female",
  "Index" => 50,
  "SibSp" => "1",
  "message" => "50,Arnold-Franchi,0,3,female,29.69911765,S,3,349237,female,50,1,2017-12-05T08:08:23.507Z,0,Arnold-Franchi,17.8,/home/ec2-user/fc/logstash/conf.d/filter.conf,17.8,@timestamp,2017-12-05T08:08:23.507Z,0",
  "Survival" => 0,
  "Name" => "Arnold-Franchi",
  "Fare" => 17.8,
  "path" => "/home/ec2-user/fc/logstash/conf.d/filter.conf"
}
}

{
  "Embarked" => "S",
  "Pclass" => 3,
  "Ticket" => "349237",
  "Sex" => "female",
  "Index" => 51,
  "SibSp" => "1",
  "message" => "51,Arnold-Franchi,0,3,female,29.69911765,S,3,349237,female,51,1,2017-12-05T08:08:23.508Z,0,Arnold-Franchi,17.8,/home/ec2-user/fc/logstash/conf.d/filter.conf,17.8,@timestamp,2017-12-05T08:08:23.508Z,0",
  "Survival" => 0,
  "Name" => "Arnold-Franchi",
  "Fare" => 17.8,
  "path" => "/home/ec2-user/fc/logstash/conf.d/filter.conf"
}
}
```

1. 데이터를 다운 받자

```
$ cd /home/ec2-user/fc/logstash-5.6.4/sample  
$ wget https://raw.githubusercontent.com/higee/elasticsearch/master/Week4_Logstash/data/sample.csv
```

2. logstash로 다음 작업을 하자

- logstash로 읽어서 모든 row를 ,로 구분하자
- 각 column 이름을 앞에서부터 idx, item, name, int1, float1, float2, float3, location, item2, float4로 하자
- int1, float1, float2, float3, float4는 각각 integer, float, float, float, float로 datatype을 변형하자
- 그리고 console에 출력해보자

일반적으로 @timestamp는 logstash가 데이터를 처음 본 시간을 기록한다

만약 input data의 일부분이 시간 데이터를 포함하고 있고 그 값으로 @timestamp를 대체하고 싶으면 어떻게 할까?

```
hi
{
    "@version" => "1",
    "host" => "ip-172-31-15-62",
    "@timestamp" => 2017-12-05T08:27:04.384Z,
    "message" => "hi"
}
hi
{
    "@version" => "1",
    "host" => "ip-172-31-15-62",
    "@timestamp" => 2017-12-05T08:27:05.449Z,
    "message" => "hi"
}
```

특정 Field 값을 logstash event 날짜로 활용 가능

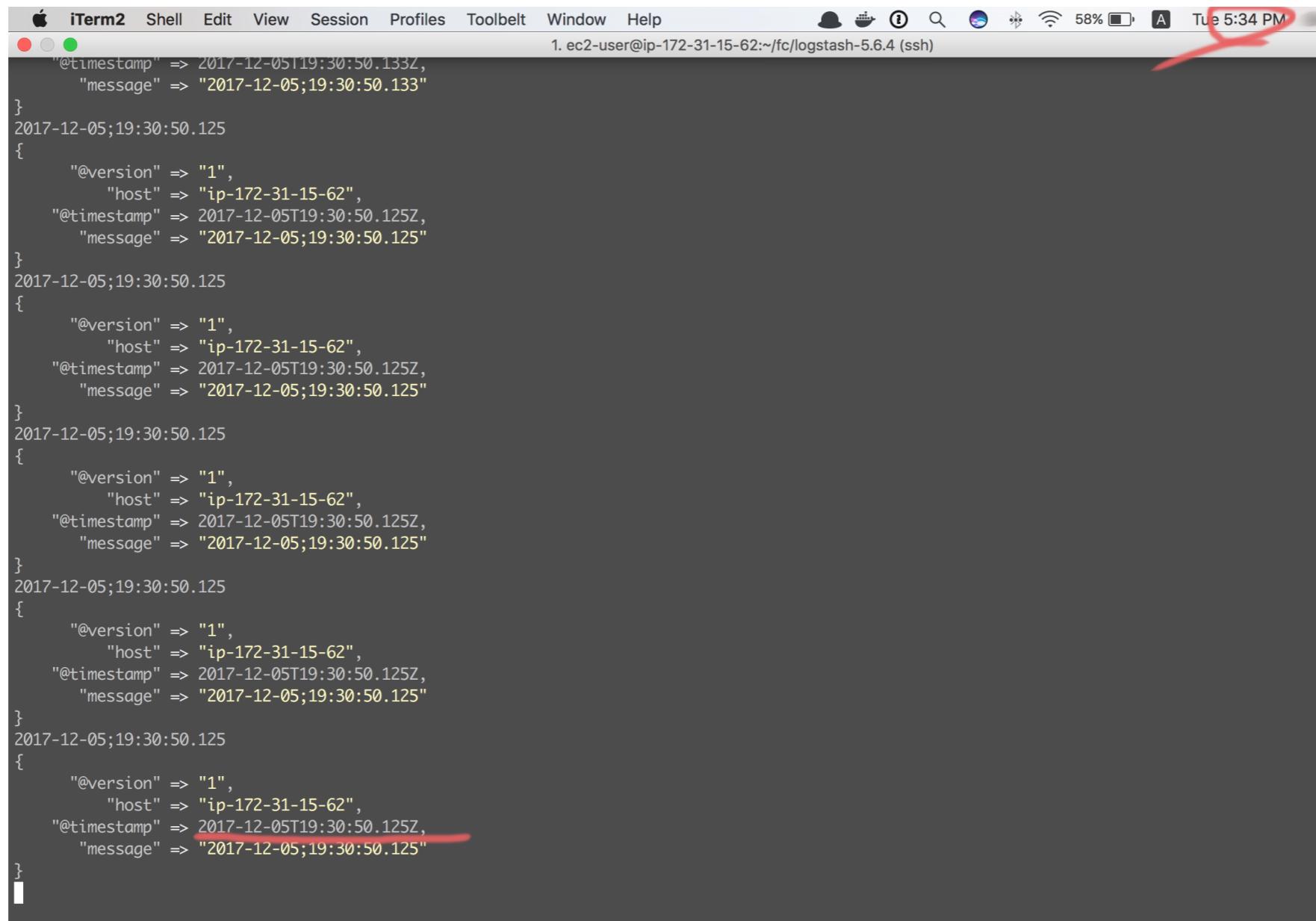
```
1 input {
2   stdin{}
3 }
4
5 filter {
6   date {
7     match => ["message", "YYYY-MM-dd;HH:mm:ss.SSS"]
8     target => "@timestamp"
9   }
10 }
11
12 output {
13   stdout {
14     codec => "rubydebug"
15   }
16   elasticsearch {
17     hosts => ["13.125.21.52:9200"]
18     index => "date_filter"
19   }
20 }~
```

logstash에서 정의한 패턴에 맞는 시간 값을 stdin으로 입력해보자

2017-12-05;19:30:50.125

2015-11-01;11:25:43.335

2016-03-30;03:23:17.553



```
iTerm2 Shell Edit View Session Profiles Toolbelt Window Help
1. ec2-user@ip-172-31-15-62:~/fc/logstash-5.6.4 (ssh)
Tue 5:34 PM

"@timestamp" => 2017-12-05T19:30:50.133Z,
  "message" => "2017-12-05;19:30:50.133"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
2017-12-05;19:30:50.125
{
  "@version" => "1",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T19:30:50.125Z,
  "message" => "2017-12-05;19:30:50.125"
}
```

- 입력값을 복사해서 새로운 Field에 할당한다

```
1 input {  
2   stdin {}  
3 }  
4  
5 filter {  
6   mutate {  
7     copy => { "message" => "copied message" }  
8   }  
9 }  
10  
11 output {  
12   stdout {  
13     codec => rubydebug  
14   }  
15 }
```



```
that  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:43:02.071Z,  
  "copied message" => "that",  
  "message" => "that"  
}  
elastic stack  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:43:05.416Z,  
  "copied message" => "elastic stack",  
  "message" => "elastic stack"  
}  
fast campus  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:43:08.531Z,  
  "copied message" => "fast campus",  
  "message" => "fast campus"  
}  
2017  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:43:11.506Z,  
  "copied message" => "2017 ",  
  "message" => "2017 "  
}
```

- 입력값을 구분자로 나누어준다

```
1 input {  
2   stdin {}  
3 }  
4  
5 filter {  
6   mutate {  
7     split => { "message" => " " }  
8   }  
9 }  
10  
11 output {  
12   stdout {  
13     codec => rubydebug  
14   }  
15 }
```



```
hi my name is  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:47:38.073Z,  
  "message" => [  
    [0] "hi",  
    [1] "my",  
    [2] "name",  
    [3] "is"  
  ]  
}  
안녕 내 이름은  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:47:43.149Z,  
  "message" => [  
    [0] "안녕 ",  
    [1] "내 ",  
    [2] "이름은 "  
  ]  
}  
패스트캠퍼스는 신사역 근처에 있다  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:47:54.262Z,  
  "message" => [  
    [0] "패스트캠퍼스는 ",  
    [1] "신사역 근처에 ",  
    [2] "있다 "  
  ]  
}
```

- 여러 Field를 merge해서 하나의 Field (배열)로 처리한다

```
1 input {  
2   stdin {}  
3 }  
4  
5 filter {  
6   mutate {  
7     merge => ["@version", "host"]  
8   }  
9 }  
10  
11 output {  
12   stdout {  
13     codec => rubydebug  
14   }  
15 }
```



```
hi  
{  
  "@version" => [  
    [0] "1",  
    [1] "ip-172-31-15-62"  
  ],  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:58:54.851Z,  
  "message" => "hi"  
}  
you  
{  
  "@version" => [  
    [0] "1",  
    [1] "ip-172-31-15-62"  
  ],  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T08:59:00.953Z,  
  "message" => "you"  
}
```

- Field 이름을 변경한다

```
input {  
  stdin {}  
}  
  
filter {  
  mutate {  
    rename => { "@timestamp" => "what time is it?" }  
  }  
}  
  
output {  
  stdout {  
    codec => rubydebug  
  }  
}
```



```
hi  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "message" => "hi",  
  "what time is it?" => 2017-12-05T08:52:54.711Z  
}  
  
hi  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "message" => "hi",  
  "what time is it?" => 2017-12-05T08:52:56.413Z  
}
```

- 특정 Field의 값을 변경한다

```
input {
  stdin {}
}

filter {
  mutate {
    replace => { "@version" => "Used to be @version" }
  }
}

output {
  stdout {
    codec => rubydebug
  }
}
```



```
hi
{
  "@version" => "Used to be @version",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T09:01:38.575Z,
  "message" => "hi"
}
no more version
{
  "@version" => "Used to be @version",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T09:01:43.264Z,
  "message" => "no more version"
}
again?
{
  "@version" => "Used to be @version",
  "host" => "ip-172-31-15-62",
  "@timestamp" => 2017-12-05T09:01:47.531Z,
  "message" => "again?"}
```

원본 데이터에 없던 Field를 추가한다

```
1 input {  
2   stdin {  
3   }  
4 }  
5  
6 filter {  
7   mutate {  
8     split => {"message" => ","}  
9     add_field => {  
10       "first" => "%{message[0]}"  
11       "second" => "%{message[1]}"  
12     }  
13   }  
14 }  
15  
16 output {  
17   stdout {  
18     codec => rubydebug  
19   }  
20 }
```



```
hi, you  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T09:10:12.785Z,  
  "message" => [  
    [0] "hi",  
    [1] " you"  
,  
    "first" => "hi",  
    "second" => " you"  
]  
}  
message is separated by comma  
{  
  "@version" => "1",  
  "host" => "ip-172-31-15-62",  
  "@timestamp" => 2017-12-05T09:10:39.745Z,  
  "message" => [  
    [0] "message is separated by comma"  
,  
    "first" => "message is separated by comma",  
    "second" => "%{message[1]}"  
]
```

원본 데이터의 특정 Field를 삭제한다

```
1 input {  
2   stdin {  
3   }  
4 }  
5  
6 filter {  
7   mutate {  
8     remove_field => ["@version", "host", "@timestamp"]  
9   }  
10 }  
11  
12 output {  
13   stdout {  
14     codec => rubydebug  
15   }  
16 }
```



```
hi  
{  
  "message" => "hi"  
}  
every other field is gone  
{  
  "message" => "every other field is gone"  
}  
now I only have message field  
{  
  "message" => "now I only have message field"  
}  
Nice!  
{  
  "message" => "Nice!"  
}
```

Filter - mutate

콘솔에 “Seoul”, “Tokyo”, “Beijing” 입력하면
아래처럼 나오도록 Logstash config를 작성해보자

```
Seoul,Tokyo,Beijing
{
    "original" => [
        [0] "Seoul",
        [1] "Tokyo",
        [2] "Beijing"
    ],
    "Koera" => "Seoul",
    "Japan" => "Tokyo",
    "China" => "Beijing"
}
```

구조화되어 있지 않은 데이터의 경우 어떻게 할까?

Filter - grok

예를 들면 아래와 같은 log는 어떻게 처리할까?

```
1 127.0.0.1 - - [13/Dec/2015:03:02:45 -0800] "GET /xampp/status.php HTTP/1.1" 200 891 "http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"
2 123.222.333.123 HOME - [01/Feb/1998:01:08:46 -0800] "GET /bannerad/ad.htm HTTP/1.0" 200 2808 "http://www.referrer.com/bannerad/ba_intro.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
3 daum.net HOME - [01/Apr/1998:01:09:14 -0800] "GET /bannerad/click.htm HTTP/1.0" 200 2070 "http://www.referrer.com/bannerad/menu.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
4 111.222.333.123 AWAY - [01/Apr/1998:01:09:14 -0800] "GET /bannerad/click.htm HTTP/1.0" 200 2070 "http://www.referrer.com/bannerad/menu.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
5 unicomp6.unicomp.net AWAY - [01/Apr/1998:01:09:14 -0800] "GET /bannerad/click.htm HTTP/1.0" 200 2070 "http://www.referrer.com/bannerad/menu.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
6 123.222.333.123 AWAY - [01/Feb/2003:01:08:53 -0800] "GET /bannerad/ad7.gif HTTP/1.0" 200 332 "http://www.referrer.com/bannerad/ba_ad.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
```

~

또는 이런 log는 어떻게 할까?

```
1 2017-11-28 12:11:23, app.py, main, 91, INFO, ValueError
2 2017-09-13 23:13:03, item.py, get, 31, DEBUG, ZeroDivisionError
3 2017-12-18 12:25:53, security.py, auth, 51, DEBUG, KeyError
4 2017-10-08 22:31:35, app.py, main, 51, DEBUG, ZeroDivisionError
5 2017-11-22 03:49:55, security.py, auth, 21, DEBUG, SyntaxError
6 2017-10-31 13:21:43, db.py, main, 51, DEBUG, KeyError
7 2017-09-29 20:35:39, item.py, post, 21, CRITICAL, NameError
8 2017-11-20 02:05:11, item.py, put, 61, INFO, SyntaxError
```

~

Filter - grok

regular expression처럼 사용 가능하며 큰 틀은 다음과 같다

- 가지고 있는 로그 파일 패턴을 발견한다 (~~최소한 노력한다~~)
- **grok-patterns**(클릭)에서 가서 match 할 수 있는 pattern을 찾는다
- **grok debugger**(클릭)를 이용해서 시도해본다
- grok debugger에서 발견한 pattern을 활용해서 logstash grok filter를 작성한다

이처럼 customized된 형태는 직접 하나하나 짜야 된다.

```
1 2017-11-28 12:11:23, app.py, main, 91, INFO, ValueError
```

```
%{TIMESTAMP_ISO8601:date}, %{USERNAME:file}, %{WORD:function}, %{INT:line}, %{LOGLEVEL:level}, %{WORD:messsage}
```



```
}
{
    "date" => "2017-10-31 13:21:43",
    "path" => "/home/ec2-user/fc/logstash-5.6.4/sample/test.log",
    "@timestamp" => 2017-12-05T09:44:29.631Z,
    "file" => "db.py",
    "level" => "DEBUG",
    "line" => "51",
    "function" => "main",
    "@version" => "1",
    "host" => "ip-172-31-15-62",
    "messsage" => "KeyError",
    "message" => "2017-10-31 13:21:43, db.py, main, 51, DEBUG, KeyError"
}
```

가만히 보면 apache log 형태이므로 grok-patterns에서 **COMBINEDAPACHELOG** 선택

```
1 [127.0.0.1 - - [13/Dec/2015:03:02:45 -0800] "GET /xampp/status.php HTTP/1.1" 200 891 "http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"
```

%{COMBINEDAPACHELOG:log}



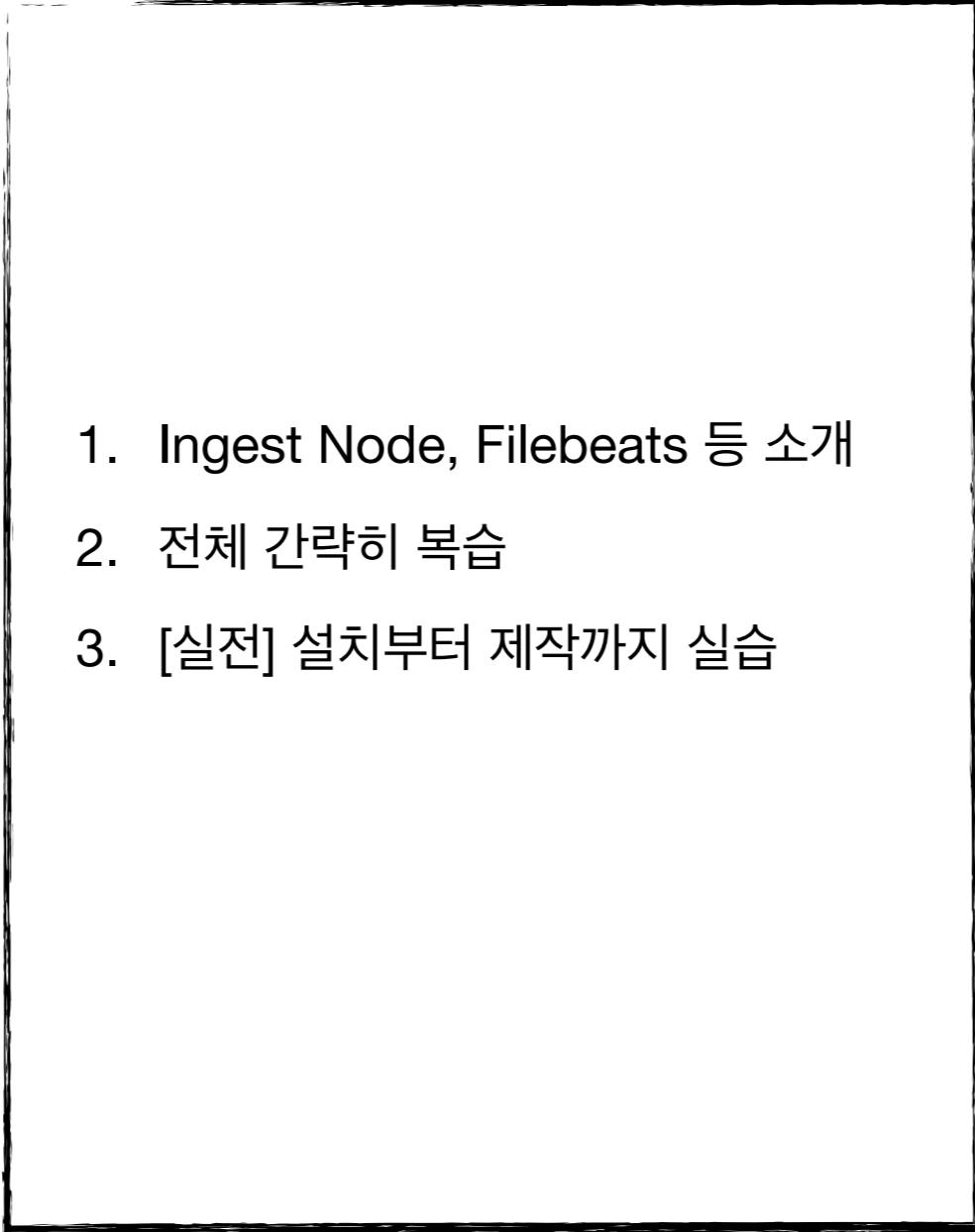
```
{  
    "request" => "/xampp/status.php",  
    "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\"",  
    "auth" => "-",  
    "ident" => "-",  
    "verb" => "GET",  
    "path" => "/home/ec2-user/fc/logstash-5.6.4/sample/apache.log",  
    "referrer" => "\"http://cadenza/xampp/navi.php\"",  
    "@timestamp" => 2017-12-11T13:26:13.112Z,  
    "response" => "200",  
    "bytes" => "891",  
    "clientip" => "127.0.0.1",  
    "@version" => "1",  
    "host" => "ip-172-31-21-251",  
    "httpversion" => "1.1",  
    "timestamp" => "13/Dec/2015:03:02:45 -0800"  
}
```

Filter - grok

stdin으로 S.Korea 27, 2017/12 => 5000을 입력하면 아래처럼 출력하는 logstash를 생성해보자

```
S.Korea 27, 2017/12 => 5000
{
    "date" => "2017/12",
    "Country" => "S.Korea",
    "salary" => "5000",
    "age" => "27"
}
```

Week 5

- 
1. Ingest Node, Filebeats 등 소개
 2. 전체 간략히 복습
 3. [실전] 설치부터 제작까지 실습