

## Elastic Stack 을 활용한 Data Dashboard 만들기

Week 5 - Logstash로 데이터를 전처리하고 전송하자



Fast Campus

내용	페이지
Logstash 개요	4
Logstash workflow	9
Input Plugins	
stdin	15
file	20
jdbc	39
elasticsearch	56
Output Plugins	
csv	61
elasticsearch	73
<i>multi output plugins</i>	77
Conditional	81
Filter Plugins	
csv	85
mutate	101
grok	116
date	130
drop	137
ruby	145
elasticsearch	157

**지금까지 시각화도 했고, 검색도 해봤는데 뭔가 2% 부족하다**

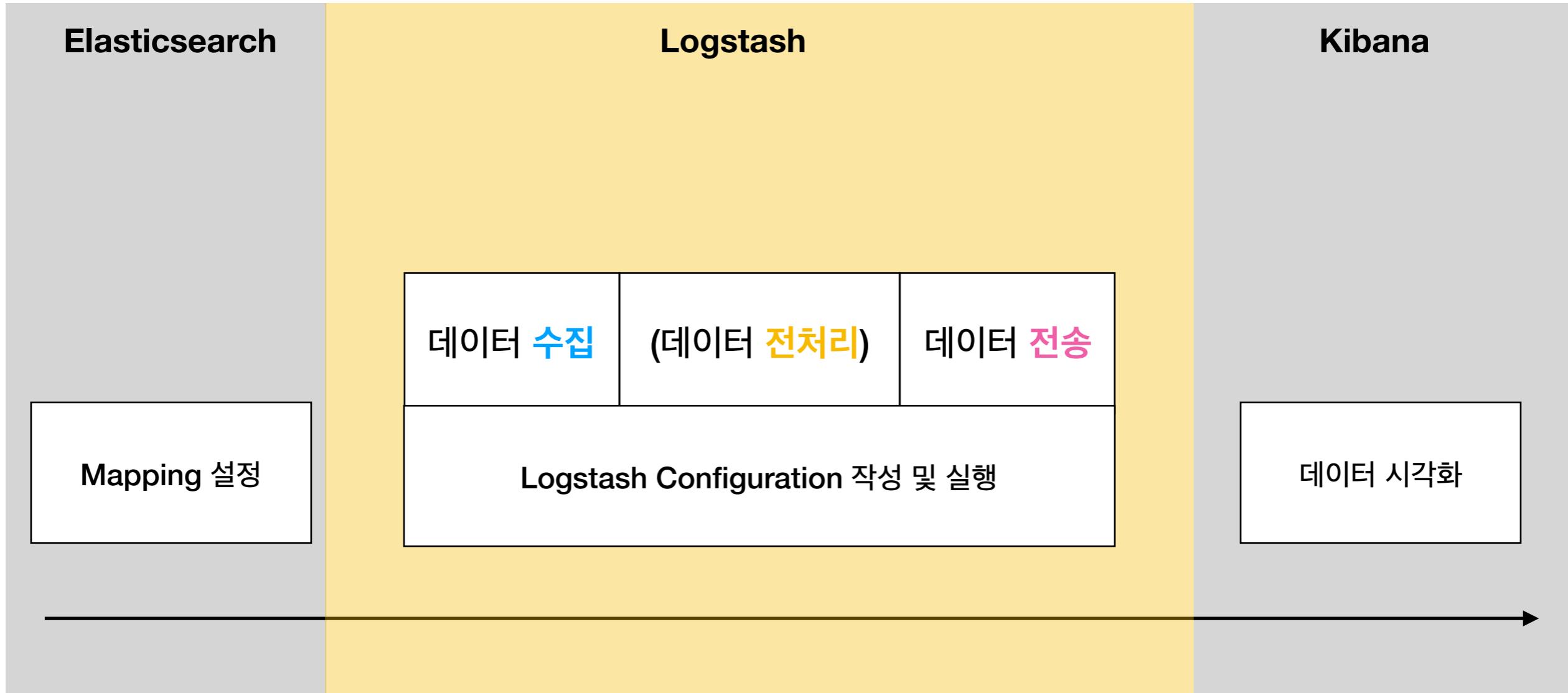
**이번에는 내가 가지고 있는 데이터를 시각화하고 싶다**

**내가 가진 모든 데이터를 Elastic Stack으로 분석할 수 있나?**

**원본 데이터를 변형할 수도 있나?**



이 때 **데이터를** 필요한 게 **Logstash!**



(분석가 workflow)

## logstash 6.20이 공식적으로 지원하는 plugins 중에서...

<-> community plugins 예) mongodb input plugin 

	용어	종류	예시
수집 	<b>input plugin</b>	48	stdin, file(log, csv ...), jdbc(mysql, ...), elasticsearch, s3, ...
전처리 	<b>filter plugin</b>	41	mutate, csv, date, ruby, if, grok, drop, kv, range, ...
전송 	<b>output plugin</b>	50	stdout, elasticsearch, csv, kafka, mongodb, redis, s3, ...

## ... 목적에 맞게 선택해서 사용하면 된다 (example)

	예시	해석
<b>input plugin</b> 	<b>jdbc</b>	mysql database에서 데이터를 수집해서...
<b>filter plugin</b> 	<b>mutate</b>	데이터에 mutate filter를 적용하고...
<b>output plugin</b> 	<b>elasticsearch</b>	elasticsearch에 데이터를 전송해라

input

filter

output

product	quantity	sales
Onepiece	2	39000
Cardigan	1	37000
Knit	3	69000
Jeans	1	78000
T-Shirt	1	89000

다음 상품 '10% off' tag 추가  
- Onepiece  
- Cardigan

```
"hits": {  
  "total": 2,  
  "max_score": 1,  
  "hits": [  
    {  
      "_index": "week5_higee",  
      "_type": "week5_higee",  
      "_id": "AWFaqPo-PloSIAlpN8yg",  
      "_score": 1,  
      "_source": {  
        "product": "Onepiece",  
        "quantity": 2,  
        "sales": 39000,  
        "tags": [  
          "10% off"  
        ]  
      }  
    },  
    {  
      "_index": "week5_higee",  
      "_type": "week5_higee",  
      "_id": "AWFaqQIiPloSIAlpN8yh",  
      "_score": 1,  
      "_source": {  
        "product": "Knit",  
        "quantity": 3,  
        "sales": 69000  
      }  
    }  
  ]  
}
```

**그렇다면 logstash를 실제로 활용할 경우 어떤 flow로 생각해야 할까?**

## 1. HOME

Logstash Home Directory 이동

```
$ docker exec -it logstash bash
```

## 2. conf 생성

Logstash conf 파일 생성



```
$ vi exercise.conf
```

(파일 이름 예시 : exercise.conf)

## 3. conf 편집

Logstash conf 파일 편집모드



i

## 4. conf 작성

Logstash conf 파일 편집

문제 정의

- 데이터가 어디에 있는지 (=input) 명확히 한다.
- 데이터를 어디로 전송할지 (=output) 명확히 한다.
- 데이터를 어떻게 변형할지 (=filter) 명확히 한다.

코드 작성

- 적절한 input plugin 선택
- 적절한 output plugin 선택
- 적절한 filter plugin 선택

## 5. conf 저장

Logstash conf 파일 저장



```
ESC 입력 후 :wq
```



conf 작성이 logstash 핵심

## 6. 실행

Logstash (conf 파일) 실행



```
$ bin/logstash -f exercise.conf
```

1. HOME

Logstash Home Directory 이동

\$ docker exec -it logstash bash

2. conf 생성

Logstash conf 파일 생성

\$ vi exercise.conf

(파일 이름 예시 : exercise.conf)

3. conf 편집

Logstash conf 파일 편집모드

i

큰 흐름은 같고 **conf 작성하는 부분만 바꾼다.**

4. conf 작성

Logstash conf 파일 편집

(오늘 실습은 **이 과정의 반복이다.**)

문제 정의

1. 데이터가 어디에 있는지 (=input) 명확히 한다.

코드 작성

2. 데이터를 어디로 전송할지 (=output) 명확히 한다.

3. 데이터를 어떻게 변형할지 (=filter) 명확히 한다.

4. 적절한 input plugin 선택

5. 적절한 output plugin 선택

6. 적절한 filter plugin 선택

5. conf 저장

Logstash conf 파일 저장

\$ ESC 입력 후 :wq



conf 작성이 logstash 핵심

6. 실행

Logstash (conf 파일) 실행

\$ bin/logstash -f exercise.conf

sample logstash configuration은 아래와 같다

```
1 input {
2     stdin { }
3 }
4
5 filter {
6     mutate {
7         split => { "message" => ":" }
8         add_field => {
9             "ip" => "%{message[0]}"
10            "port" => "%{message[1]}"
11        }
12    }
13 }
14
15 output {
16     stdout {
17     }
18 }
```

## sample logstash configuration은 아래와 같다

```
1 input {  
2     stdin { }  
3 }  
4  
5 filter {  
6     mutate {  
7         split => { "message" => ":" }  
8         add_field => {  
9             "ip" => "%{message[0]}"  
10            "port" => "%{message[1]}"  
11        }  
12    }  
13 }  
14  
15 output {  
16     stdout {  
17 }  
18 }
```

1. 데이터가 어디에 있는지 (=input) 명확히 한다.
2. 데이터를 어디로 전송할지 (=output) 명확히 한다.
3. 데이터를 어떻게 변형할지 (=filter) 명확히 한다.
4. 적절한 **input** plugin 선택 **stdin**
5. 적절한 **output** plugin 선택 **stdout**
6. 적절한 **filter** plugin 선택 **mutate**

## 실습에 앞서

- 수업 시간에 logstash configuration을 직접 작성하지 않음
- 대신, 미리 만들어 놓은 logstash configuration을 실행
- docker container 내부와 host를 구분하기 위해 docker container 내부 작업은  표시

## **Input - `stdin`**

- `console`에 직접 입력한 데이터 수집
- 간단한 `logstash` 테스트 할 때 혹은 디버깅 할 때 주로 사용

## logstash configuration을 확인하자



```
1 input {
2   stdin { }
3 }
4
5 output {
6   stdout { }
7 }
```

- **input** plugin과 **output** plugin은 최소 1개씩 있어야 한다 (filter는 옵션)
- 그러므로 input 학습 실습이지만 기본 output인 stdout을 설정했다

## logstash를 실행하자



```
$ bin/logstash -f code/input/stdin/stdin.conf
```

## (약 1분 후) 아래와 같이 나오면...



```
bash-4.2$ bin/logstash -f code/input/stdin/stdin.conf
warning: Ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:AssumeMP; pass JVM parameters via LS_JAVA_OPTS
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you should configure the number of parallel GC threads appropriately using -XX:ParallelGCThreads=N
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-07-15T09:12:13,379][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache", :directory=>/usr/share/logstash/modules/fb_apache/configuration"}
[2018-07-15T09:12:13,429][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow", :directory=>/usr/share/logstash/modules/netflow/configuration"}
[2018-07-15T09:12:14,542][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because modules or command line options are specified
[2018-07-15T09:12:15,692][INFO ][logstash.runner] ] Starting Logstash {"logstash.version"=>"6.2.4"}
[2018-07-15T09:12:16,456][INFO ][logstash.agent] ] Successfully started Logstash API endpoint {:port=>9600}
[2018-07-15T09:12:20,668][INFO ][logstash.pipeline] ] Starting pipeline {:pipeline_id=>"main", "pipeline.workers"=>1, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50}
[2018-07-15T09:12:20,862][INFO ][logstash.pipeline] ] Pipeline started successfully {:pipeline_id=>"main", :thread=>"#<Thread:0x6c2895 run>"}
The stdin plugin is now waiting for input:
[2018-07-15T09:12:21,029][INFO ][logstash.agent] ] Pipelines running {:count=>1, :pipelines=>["main"]}
```



```
The stdin plugin is now waiting for input:
[2018-07-15T09:18:16,175][INFO ][logstash.agent] ] Pipelines running {:count=>1, :pipelines=>["main"]}
hi
```

stdin으로 **hi** 입력하고 **Enter** 를 누르자

## logstash 실행 결과

- **stdin** 입력값 (= 사용자가 직접 입력)
- input을 stdin으로 설정했기에 "hi" 입력하자 결과 출력
- event 1개



```
{  
  "message" => "hi",  
  "host" => "72f070e6b6a4",  
  "@timestamp" => 2018-07-15T09:18:20.484Z,  
  "@version" => "1"  
}
```



- **stdout** 출력값 (= 화면에 직접 출력)
- stdin 입력값인 "hi" 이외의 메타데이터는 시스템에서 제공

## **Input - file**

- file 형태 데이터를 수집할 때 사용
- test.log, test.csv 등 일반적인 파일형태는 모두 file plugin을 사용

## logstash configuration을 확인하자



```
1 input {  
2   file {  file에서 데이터를 읽을 것이기에 file 입력  
3     path => "/usr/share/logstash/data/titanic.csv"  
4   }  파일 경로 입력  
5 }  
6  
7 output {  
8   stdout {  
9   }  
10 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/input/file/file.conf
```

## 아무 것도 안 나온다 (정상이다)



```
bash-4.2$ bin/logstash -f code/input/file/file.conf
warning: ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:-AssumeMP; pass JVM
OpenJDK 64-Bit Server VM warning: If the number of processors is exposed to the Java VM from
Sending Logstash's logs to /usr/share/logstash/logs which is now considered via logstash.conf
[2018-07-15T09:39:16,011][INFO ][logstash.modules.scaffold] Initialising module [file]
[2018-07-15T09:39:16,053][INFO ][logstash.modules.scaffold] Initialising module [logstash]
[2018-07-15T09:39:17,208][WARN ][logstash.config.source.multilocal]
[2018-07-15T09:39:18,456][INFO ][logstash.runner      ] Starting Logstash
[2018-07-15T09:39:19,294][INFO ][logstash.agent       ] Successfully started Logstash pipeline
[2018-07-15T09:39:24,321][INFO ][logstash.pipeline    ] Starting pipeline {"id": "main", "workers": 1}
[2018-07-15T09:39:25,005][INFO ][logstash.pipeline    ] Pipeline start successful {"pipeline.id": "main", "workers": 1,
[2018-07-15T09:39:25,249][INFO ][logstash.agent       ] Pipeline main started
```

- **start\_position**이라는 옵션이 default로 **end**로 설정되어 있기 때문이다.
- 즉, 새로 추가되는 데이터만 보겠다는 것인데 이걸 **beginning**으로 해줘야 된다.
- **다른 파일**로 다시 실행해보자

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/titanic2.csv"
4     start_position => "beginning"
5   }
6 }
7
8 output {
9   stdout {
10 }
12 }
```



## logstash를 실행하자



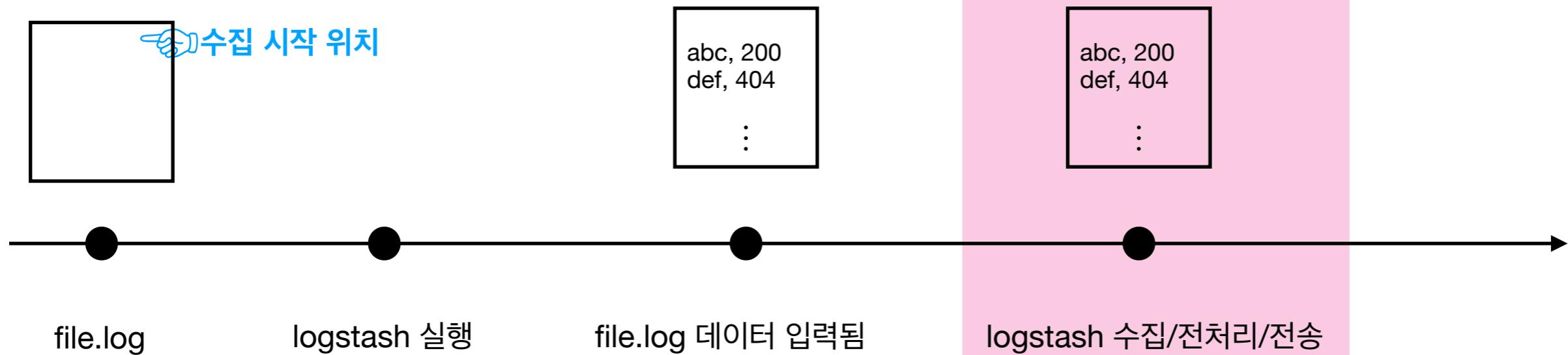
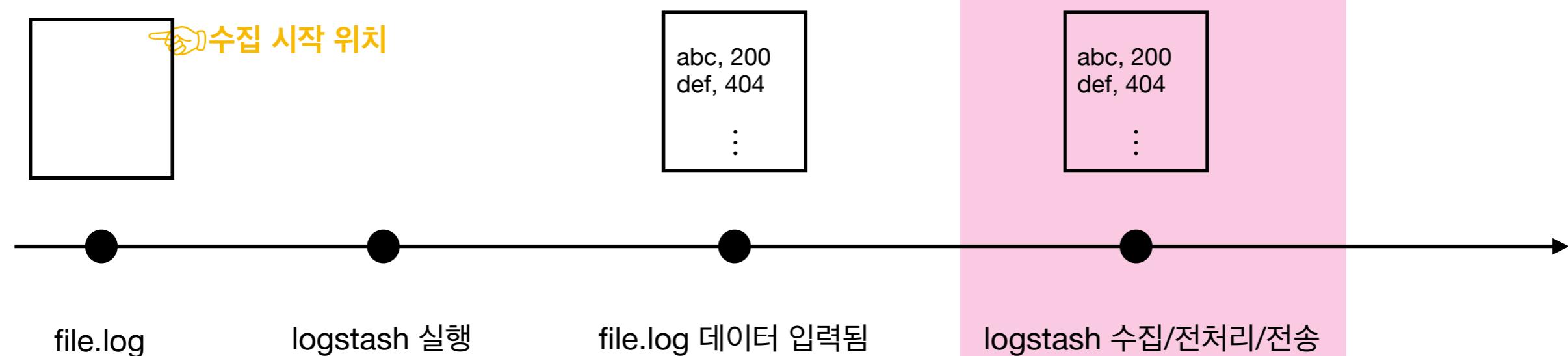
```
$ bin/logstash -f code/input/file/file-start-position.conf
```

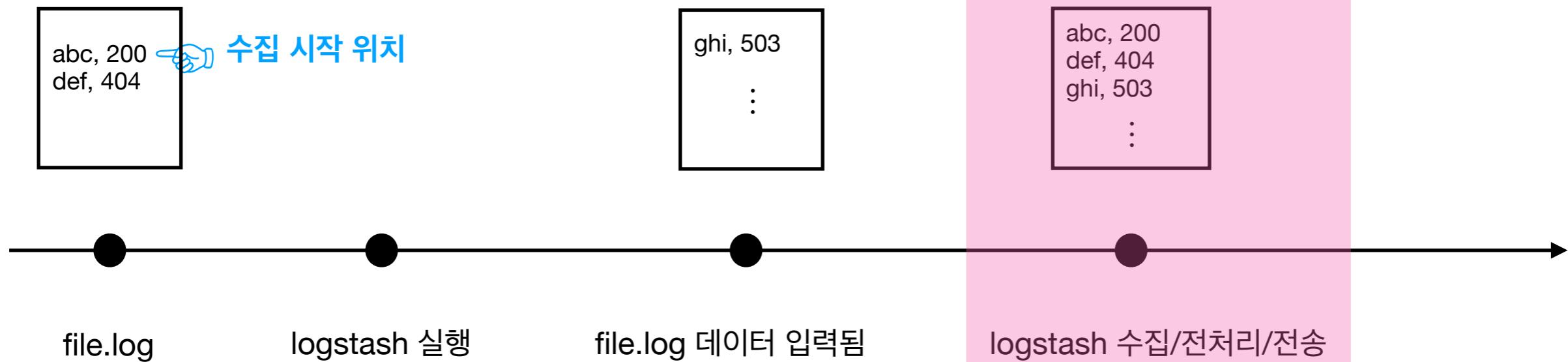
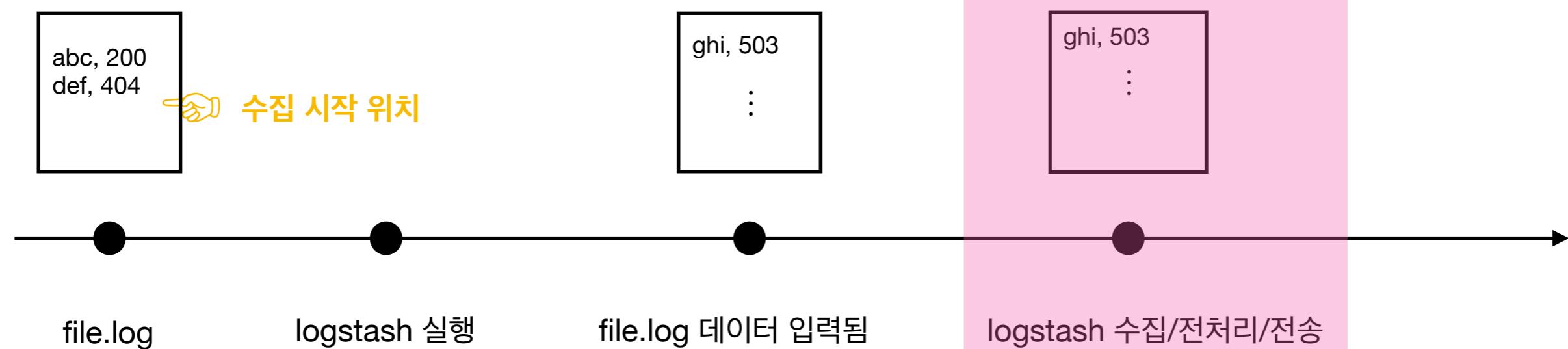
## 파일을 처음부터 읽으므로 결과가 제대로 출력

```
bash-4.2$ bin/logstash -f code/input/file/file-start-position.conf
warning: ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:-AssumeMP; pass JVM parameters via LS_JAVA_OPTS
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you sh
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-07-15T10:07:18,372][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache"
[2018-07-15T10:07:18,413][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow",
[2018-07-15T10:07:19,490][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because it
[2018-07-15T10:07:20,688][INFO ][logstash.runner] Starting Logstash {"logstash.version"=>"6.2.4"}
[2018-07-15T10:07:21,477][INFO ][logstash.agent] Successfully started Logstash API endpoint {:po
[2018-07-15T10:07:26,973][INFO ][logstash.pipeline] Starting pipeline {:pipeline_id=>"main", "pipe
[2018-07-15T10:07:27,724][INFO ][logstash.pipeline] Pipeline started successfully {:pipeline_id=>"ma
[2018-07-15T10:07:27,952][INFO ][logstash.agent] Pipelines running {:count=>1, :pipelines=>["ma
{
    "@version" => "1",
    "host" => "d0d2cd6bcf0a",
    "path" => "/usr/share/logstash/data/titanic2.csv",
    "message" => "51,Panula,0,3,male,7,4,1,3101295,39.6875,S",
    "@timestamp" => 2018-07-15T10:07:28.562Z
}
{
    "@version" => "1",
    "host" => "d0d2cd6bcf0a",
    "path" => "/usr/share/logstash/data/titanic2.csv",
    "message" => "52,Nosworthy,0,3,male,21,0,0,A/4. 39886,7.8,S",
    "@timestamp" => 2018-07-15T10:07:28.641Z
}
{
    "@version" => "1",
    "host" => "d0d2cd6bcf0a",
    "path" => "/usr/share/logstash/data/titanic2.csv",
    "message" => "53,Harper,1,1,female,49,1,0,PC 17572,76.7292,C",
    "@timestamp" => 2018-07-15T10:07:28.647Z
}
```

Q. 그래서 start\_position option를 어떻게 하라는건지?

A. logstash 실행 전에 file에 이미 데이터가 있는지 아닌지가 중요하다

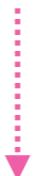
**start\_position : beginning****start\_position : end**

**start\_position : beginning****start\_position : end**

단, `start_position`은 특정 파일을 처음 읽을 때에만 효과가 있다.  
그 이후에는 `sincedb`에 기록이 되어 효과가 없다.

Q. 왜 한 번 **titanic.csv**를 logstash로 읽은 후에는 **titanic2.csv**로 수정했을까?

A. logstash는 기본적으로, 같은 데이터는 한 번만 조회하는 정책을 가지고 있기 때문이다



**since db**

- path : <\$path.data>/plugins/inputs/file



```
bash-4.2$ pwd  
/usr/share/logstash/data/plugins/inputs/file  
bash-4.2$ ls -lah  
total 16K  
drwxr-xr-x 2 logstash logstash 4.0K Jul 15 12:33 .  
drwxr-xr-x 3 logstash logstash 4.0K Jul 15 06:25 ..  
-rw-r--r-- 1 logstash logstash 20 Jul 15 12:28 sincedb_b503666ae29e95ab8024b1fd6cbc694c  
-rw-r--r-- 1 logstash logstash 20 Jul 15 12:33 .sincedb_dae92a8984be95dc3377c3b9394a87f
```



file input plugin에서 설정한 path 별로 1개씩 생긴다

- data



```
bash-4.2$ cat .sincedb_dae92a8984be95dc3377c3b9394a87f  
137722 0 51713 2232
```

파일정보



current byte offset

- path : <\$path.data>/plugins/inputs/file



```
bash-4.2$ pwd  
/usr/share/logstash/data/plugins/inputs/file  
bash-4.2$ ls -lah  
total 16K  
drwxr-xr-x 2 logstash logstash 4.0K Jul 15 12:33 .  
drwxr-xr-x 2 logstash logstash 4.0K Jul 15 12:33 ..  
-rw-r--r-- 1 logstash logstash 20 Jul 15 12:33 .sinceDB_dae92a8984be95dcb3377c3b9394a87f
```

# 한 마디로, logstash는 sinceDB를 이용해서

1) file (혹은 file patterns) 별로

2) 어디까지 조회했는지



```
bash-4.2$ cat .sinceDB_dae92a8984be95dcb3377c3b9394a87f
```

# 기록하고 있다고 알고 넘어가자

파일정보

current byte offset

그러므로, 한 번 끝까지 읽은 파일을 logstash로 재실행해도 반응이 없는 이유는

이미 해당 파일에 대해 끝까지 봤다는 기록이 남아있기 때문이다.

다시 조회하고 싶으면 sincedb\_path option을 변경하면 된다.

## logstash configuration을 확인하자



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/titanic.csv"
4     start_position => "beginning"
5     sincedb_path => /dev/null
6   }
7 }
8
9 output {
10   stdout {
11 }
13 }
```



- 이미 읽은 데이터도 처음 보는 데이터인 것처럼 설정
- 단, 이번에 실행한 결과에 대한 기록은 sincedb에 저장하지 않는다.
- 주로 디버깅/테스트 용도로 사용

## logstash를 실행하자



```
$ bin/logstash -f code/input/file/file-sinceDB-path.conf
```



```
bash-4.2$ bin/logstash -f code/input/file/file-sincedb-path.conf
warning: ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:-AssumeMP; pass JVM parameters via LS_JAVA_OPTS
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, th
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-07-15T13:11:10,496][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb
[2018-07-15T13:11:10,542][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"ne
[2018-07-15T13:11:11,625][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' f
[2018-07-15T13:11:12,829][INFO ][logstash.runner] Starting Logstash {"logstash.version"=>
[2018-07-15T13:11:13,560][INFO ][logstash.agent] Successfully started Logstash API endpoint
[2018-07-15T13:11:19,264][INFO ][logstash.pipeline] Starting pipeline {:pipeline_id=>"main"
[2018-07-15T13:11:19,982][INFO ][logstash.pipeline] Pipeline started successfully {:pipeli
[2018-07-15T13:11:20,218][INFO ][logstash.agent] Pipelines running {:count=>1, :pipelin

{
    "@timestamp" => "2018-07-15T13:11:20.744Z",
    "@version" => "1",
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",
    "path" => "/usr/share/logstash/data/titanic.csv",
    "host" => "fef3405d41af"
}
{
    "@timestamp" => "2018-07-15T13:11:20.815Z",
    "@version" => "1",
    "message" => "2,Cumings,1,1,female,38,1,0,PC 17599,71.2833,C",
    "path" => "/usr/share/logstash/data/titanic.csv",
    "host" => "fef3405d41af"
}
{
    "@timestamp" => "2018-07-15T13:11:20.823Z",
    "@version" => "1",
    "message" => "3,Heikkinen,1,3,female,26,0,0,STON/O2. 3101282,7.925,S",
    "path" => "/usr/share/logstash/data/titanic.csv",
    "host" => "fef3405d41af"
}
```

## Input - jdbc

- jdbc를 지원하는 database 데이터를 수집할 때 사용하는 plugin
- 실습 database 정보
  - ▶ uri : mysql://52.78.134.20:3306/fc
  - ▶ database : fc
  - ▶ table : fc
  - ▶ user : fc
  - ▶ password : fc

## 원본 데이터 (총 33 row)

```
mysql> use fc;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from fc;
+----+----+----+----+----+----+
| 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       |
+----+----+----+----+----+----+
```

## logstash configuration을 확인하자



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10   }
11 }
12
13 output {
14   stdout {
15   }
16 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/input/jdbc/jdbc.conf
```

## logstash를 실행하면...



```
bash-4.2$ bin/logstash -f code/input/jdbc/jdbc.conf
warning: ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:-AssumeMP; pass JVM parameters via LS_JAVA_OPTS
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-07-15T14:05:53,013][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apa
[2018-07-15T14:05:53,059][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflo
[2018-07-15T14:05:54,205][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file
[2018-07-15T14:05:55,466][INFO ][logstash.runner] ] Starting Logstash {"logstash.version"=>"6.
[2018-07-15T14:05:56,240][INFO ][logstash.agent] ] Successfully started Logstash API endpoint
[2018-07-15T14:06:02,578][INFO ][logstash.pipeline] ] Starting pipeline {:pipeline_id=>"main", "t
[2018-07-15T14:06:03,008][INFO ][logstash.pipeline] ] Pipeline started successfully {:pipeline_i
[2018-07-15T14:06:03,254][INFO ][logstash.agent] ] Pipelines running {:count=>1, :pipelines=>
[2018-07-15T14:06:06,344][INFO ][logstash.inputs.jdbc] ] (0.036580s) SELECT * FROM fc
{
    "age" => 33,
    "salary" => 4000,
    "name" => "Josh",
    "employment_status" => "employed",
    "@timestamp" => 2018-07-15T14:06:06.393Z,
    "id" => 1,
    "location" => "US",
    "@version" => "1"
}
{
    "age" => 33,
    "salary" => 45000,
    "name" => "Tom",
    "employment_status" => "employed",
    "@timestamp" => 2018-07-15T14:06:06.439Z,
    "id" => 2,
    "location" => "US",
    "@version" => "1"
}
{
    "age" => 23,
    "salary" => 3000,
    "name" => "Kirk",
    "employment_status" => "employed",
    "@timestamp" => 2018-07-15T14:06:06.440Z,
    "id" => 3,
    "location" => "US",
    "@version" => "1"
}
```

Q. jdbc input도 logstash가 어디까지 조회했는지 기록을 하나?

A. sql로 조회한 마지막 row의 특정 column 데이터를 기록하는 방식으로 한다.



**sql\_last\_value**

다음 중 기준 column으로 적합한 column은?

심화

```
{  
    "@timestamp" => 2018-02-03T19:03:01.883Z,  
    "name" => "Kwon",  
    "@version" => "1",  
    "location" => "KR",  
    "id" => 29,  
    "employment_status" => "unemployed",  
    "salary" => 3500,  
    "age" => 48  
}  
{
```

```
    "@timestamp" => 2018-02-03T19:03:01.884Z,  
    "name" => "Kang",  
    "@version" => "1",  
    "location" => "KR",  
    "id" => 30,  
    "employment_status" => "employed",  
    "salary" => 3300,  
    "age" => 28  
}
```

```
{  
    "@timestamp" => 2018-02-03T19:03:01.884Z,  
    "name" => "Yoon",  
    "@version" => "1",  
    "location" => "KR",  
    "id" => 31,  
    "employment_status" => "unemployed",  
    "salary" => 3500,  
    "age" => 23  
}
```



마지막 row 였다면



마지막 row 였다면



실제 마지막 row

기준 column	sql_last_value
id	29
salary	3500
age	48

기준 column	sql_last_value
id	30
salary	3300
age	28

기준 column	sql_last_value
id	31
salary	3500
age	23

1. 기준 column 선정은 다음과 같은 성격을 가진 column을 권장

- (일정하게) **incremental** 하는 numeric type
- created date 또는 updated date 같은 date type

2. sql 작성할 때 기준으로 정한 column으로 **order by** 할 것

- sql last value는 가장 마지막 row만을 기억한다
- 그러므로 꼭 order by를 해서 **마지막 row에 최대값이 (혹은 최소값)** 오도록 설정



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     use_column_value => true
10    tracking_column => id
11    last_run_metadata_path => "/usr/share/logstash/id-under-5.db"
12    statement =>
13      "SELECT *
14       FROM fc
15       WHERE id < 5
16      "
17  }
18 }
19
20 output {
21   stdout {
22   }
23 }
```

☞ 어디까지 조회했는지를 id라는 column으로 추적

☞ 어디까지 조회했는지에 대한 기록을 id-under-5.db라는 파일에 저장

(Step 1) id < 5 인 데이터를 조회하고 id column을 sql\_last\_value로 저장하자

## logstash를 실행하자



```
$ bin/logstash -f code/input/jdbc/jdbc-sql-last-value-1.conf
```

## logstash를 실행하면...



```
{  
    "id" => 1,  
    "salary" => 4000,  
    "age" => 33,  
    "location" => "US",  
    "employment_status" => "employed",  
    "@version" => "1",  
    "name" => "Josh",  
    "@timestamp" => 2018-07-15T14:55:53.248Z  
}  
{  
    "id" => 2,  
    "salary" => 45000,  
    "age" => 33,  
    "location" => "US",  
    "employment_status" => "employed",  
    "@version" => "1",  
    "name" => "Tom",  
    "@timestamp" => 2018-07-15T14:55:53.300Z  
}  
{  
    "id" => 3,  
    "salary" => 3000,  
    "age" => 23,  
    "location" => "US",  
    "employment_status" => "employed",  
    "@version" => "1",  
    "name" => "Kirk",  
    "@timestamp" => 2018-07-15T14:55:53.302Z  
}  
{  
    "id" => 4,  
    "salary" => 3500,  
    "age" => 27,  
    "location" => "US",  
    "employment_status" => "employed",  
    "@version" => "1",  
    "name" => "Ken",  
    "@timestamp" => 2018-07-15T14:55:53.306Z  
}  
[2018-07-15T14:55:54,686][INFO ][logstash.pipeline ] Pipeline has terminated {:pipeline_id=>"main", :thread=>"#<Thread:0x23464788 run>"}  
bash-4.2$ cat id-under-5.db  
--- 4
```



- id < 5 인 데이터를 조회했으므로 sql\_last\_value는 4가 된다
- (last\_run\_metadata\_path를 변경하지 않는다면) 이 값을 이용해서 연이은 작업을 할 수 있다

```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     use_column_value => true
10    tracking_column => id
11    last_run_metadata_path => "/usr/share/logstash/id-under-5.db"
12    statement =>
13      "SELECT *
14      FROM fc
15      WHERE id > :sql_last_value"
16    "
17  }
18 }  이 부분을 수정했다
19
20 output {
21   stdout {
22   }
23 }
```

(Step2) **sql\_last\_value**를 이용해서 데이터를 조회하자

## logstash를 실행하자

```
 $ bin/logstash -f code/input/jdbc/jdbc-sql-last-value-2.conf
```

```
bash-4.2$ bin/logstash -f code/input/jdbc/jdbc-sql-last-value-2.conf
warning: ignoring JAVA_OPTS=-Xms128m -Xmx128m -XX:-AssumeMP; pass JVM parameters via LS_JAVA_OPTS
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-07-15T15:55:20,248][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache"
[2018-07-15T15:55:20,292][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow",
[2018-07-15T15:55:21,465][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file bec
[2018-07-15T15:55:22,768][INFO ][logstash.runner] Starting Logstash {"logstash.version"=>"6.2.4"
[2018-07-15T15:55:23,567][INFO ][logstash.agent] Successfully started Logstash API endpoint {:_
[2018-07-15T15:55:30,507][INFO ][logstash.pipeline] Starting pipeline {:pipeline_id=>"main", "pip
[2018-07-15T15:55:31,064][INFO ][logstash.pipeline] Pipeline started successfully {:pipeline_id=>
[2018-07-15T15:55:31,307][INFO ][logstash.agent] Pipelines running {:count=>1, :pipelines=>["m
[2018-07-15T15:55:34,559][INFO ][logstash.inputs.jdbc] (0.037203s)

    SELECT *
    FROM fc
    WHERE id > 4
```

☞ 내부적으로 코드가 수정됐다

```
{
    "age" => 38,
    "@version" => "1",
    "@timestamp" => 2018-07-15T15:55:34.621Z,
    "location" => "US",
    "salary" => 4500,
    "name" => "Jessie",
    "id" => 5,
    "employment_status" => "employed"
}
{
    "age" => 31,
    "@version" => "1",
    "@timestamp" => 2018-07-15T15:55:34.676Z,
    "location" => "US",
    "salary" => 5200,
    "name" => "Jennifer",
    "id" => 6,
    "employment_status" => "unemployed"
}
```

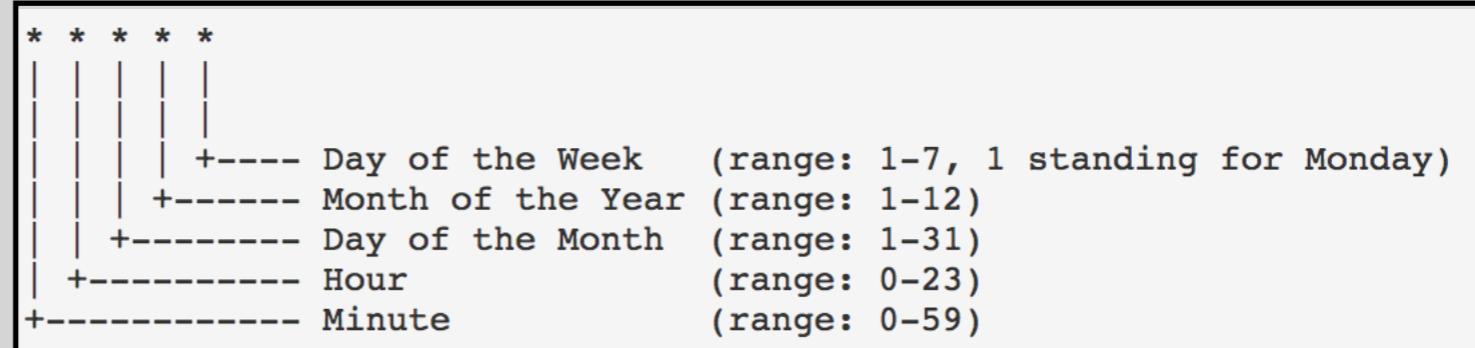
☞ id = 5 부터 조회

Q. 데이터베이스에서 정기적으로 데이터를 조회할 수 없나?

A. scheduling을 이용해서 가능하다

```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10    schedule => "0 6 * * *"  
11  }
12 }
13
14 output {
15   stdout {
16   }
17 }
```

☞ 스케줄 시간 설정



logstash process는 항상 떠있으며 매일 오전 6시에 위의 task를 수행한다

단, logstash는 가볍지 않아 서버 성능에 부담을 줄 수 있다.

그러므로 **linux cron tab**을 사용해서 특정한 시점에 한 번  
logstash를 실행하고 종료하는 방법도 고려할만하다.

(주기가 긴 경우)

## **Input - elasticsearch**

- elasticsearch에 저장된 데이터를 수집할 때 사용
- 어떤 use case가 있을까?
  - ▶ 데이터를 csv로 export 할 경우
  - ▶ 서로 다른 cluster 상의 elasticsearch 간 데이터 전송
- 실습 elasticsearch 정보
  - ▶ host: http://52.78.134.20:9200
  - ▶ index : exercise1

## 원본 데이터

Dev Tools      History    Settings    Help

Console

```
1 GET exercise1/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
```

▶ 🔍

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 33,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "exercise1",
16        "_type": "exercise1",
17        "_id": "D5X0nmQByNsCKuKnLCTX",
18        "_score": 1,
19        "_source": {
20          "@version": "1",
21          "salary": 4000,
22          "age": 33,
23          "location": "US",
24          "@timestamp": "2018-07-15T17:19:20.289Z",
25          "id": 1,
26          "employment_status": "employed",
27          "name": "Josh"
28        }
29      },
30      {
31        "_index": "exercise1",
32        "_type": "exercise1",
33        "_id": "FJX0nmQByNsCKuKnLCTX",
34        "_score": 1,
35        "_source": {
36          "@version": "1",
37          "salary": 5200,
38          "age": 31,
39          "location": "US",
40          "@timestamp": "2018-07-15T17:19:20.342Z",
41          "id": 6,
42          "employment_status": "unemployed",
43        }
44      }
45    ]
46  }
47 }
```



```
1 input {
2   elasticsearch {
3     hosts => ["52.78.134.20:9200"]
4     index => "exercise1"
5     query => '{
6       "query" : {
7         "match_all" : { }
8       }
9     }'
10   }
11 }
```

 **Query DSL**을 알면 좋은 이유가 하나 더 늘었다

```
12
13 output {
14   stdout {
15   }
16 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/input/elasticsearch/elasticsearch.conf
```

## logstash를 실행하면...



```
bash-4.2$ bin/logstash -f code/input/elasticsearch/elasticsearch.conf
warning: ignoring JAVA_OPTS=-Xms512m -Xmx512m -XX:-AssumeMP; pass JVM parameters via LS_JAVA_OPTS
Sending Logstash's logs to /usr/share/logstash/logs which is now configured via log4j2.properties
[2018-09-11T15:27:49,100][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"fb_apac"}
[2018-09-11T15:27:49,118][INFO ][logstash.modules.scaffold] Initializing module {:module_name=>"netflow"}
[2018-09-11T15:27:49,620][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file be
[2018-09-11T15:27:50,247][INFO ] [logstash.runner] Starting Logstash {"logstash.version"=>"6.2
[2018-09-11T15:27:50,659][INFO ] [logstash.agent] Successfully started Logstash API endpoint
[2018-09-11T15:27:53,679][INFO ] [logstash.pipeline] Starting pipeline {:pipeline_id=>"main", "p
[2018-09-11T15:27:54,190][INFO ] [logstash.pipeline] Pipeline started successfully {:pipeline_id=>
[2018-09-11T15:27:54,306][INFO ] [logstash.agent] Pipelines running {:count=>1, :pipelines=>[]

{
    "age" => 38,
    "employment_status" => "employed",
    "location" => "JP",
    "@version" => "1",
    "salary" => 3500,
    "name" => "Tanaka",
    "id" => 17,
    "@timestamp" => 2018-09-11T15:26:15.257Z
}
{
    "age" => 44,
    "employment_status" => "employed",
    "location" => "JP",
    "@version" => "1",
    "salary" => 4500,
    "name" => "Abe",
    "id" => 22,
    "@timestamp" => 2018-09-11T15:26:15.259Z
}
{
    "age" => 23,
    "employment_status" => "unemployed",
    "location" => "KR",
    "@version" => "1",
    "salary" => 3500,
    "name" => "Yoon",
    "id" => 31,
    "@timestamp" => 2018-09-11T15:26:15.268Z
}
{
    "age" => 38,
    "employment_status" => "employed",
    "location" => "US",
    "@version" => "1",
    "salary" => 4500,
    "name" => "Jessie",
    "id" => 5,
    "@timestamp" => 2018-09-11T15:26:15.248Z
}
```

## **Output - csv**

- **Input (Input + Filter)**를 거친 데이터를 csv 형태로 저장
- 실습 **elasticsearch** 정보
  - ▶ **host: http://52.78.134.20:9200**
  - ▶ **index : exercise1**



```
1 input {  
2   elasticsearch {  
3     hosts => ["52.78.134.20:9200"]  
4     index => "exercise1"  
5     query => '  
6       "query" : {  
7         "match_all" : {}  
8       }  
9     }'  
10   }  
11 }  
12  
13 output {  
14   csv {  
15     fields => ["name", "location", "employment_status", "salary", "age"]  
16     path => "/usr/share/logstash/data/exercise1.csv"  
17   }  
18 }
```

 csv 출력을 위해 csv output plugin 사용

 csv file을 저장할 경로 설정

- event에서 csv에 저장할 Field 설정
- csv 각 row의 column 또한 저 순서로 저장

## logstash를 실행하자



```
$ bin/logstash -f code/output/csv/csv.conf
```

## logstash를 실행하면...



```
bash-4.2$ head -10 data/exercise1.csv
Josh,US,employed,4000,33
Jennifer,US,unemployed,5200,31
John,US,unemployed,2200,25
Hanks,UK,unemployed,6200,45
Waterhouse,UK,employed,9900,37
Lee,KR,unemployed,3200,48
Tom,US,employed,45000,33
Bob,US,unemployed,22200,33
Lisa,UK,employed,3900,25
Yamauchi,JP,unemployed,5600,46
```

## **Output - elasticsearch**

- 수집하고 전처리한 결과를 elasticsearch에 전송
- dashboard를 만드는데 있어 가장 중요한 output plugin



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10   }
11 }
12
13 output {
14   elasticsearch {  es에 데이터를 전송하기 위해 es output plugin 사용
15     hosts => ["elasticsearch:9200"]
16     index => "exercise1"
17     document_type => "exercise1"
18   }
19 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/output/elasticsearch/elasticsearch.conf
```

## Kibana에서 확인하자

Dev Tools      History    Settings    Help

Console

```
1 GET exercise1/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
```

▶ 🔒

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 33,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "exercise1",
16        "_type": "exercise1",
17        "_id": "D5X0nmQByNsCKuKnLCTX",
18        "_score": 1,
19        "_source": {
20          "@version": "1",
21          "salary": 4000,
22          "age": 33,
23          "location": "US",
24          "@timestamp": "2018-07-15T17:19:20.289Z",
25          "id": 1,
26          "employment_status": "employed",
27          "name": "Josh"
28        }
29      },
30      {
31        "_index": "exercise1",
32        "_type": "exercise1",
33        "_id": "FJX0nmQByNsCKuKnLCTX",
34        "_score": 1,
35        "_source": {
36          "@version": "1",
37          "salary": 5200,
38          "age": 31,
39          "location": "US",
40          "@timestamp": "2018-07-15T17:19:20.342Z",
41          "id": 6,
42          "employment_status": "unemployed",
43        }
44      }
45    ]
46  }
47 }
```

**index를 생성할 때 index 이름에 날짜 정보를 넣을 수 있나?**



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10   }
11 }
12
13 output {
14   elasticsearch {
15     hosts => ["elasticsearch:9200"]
16     index => "exercise1-%{+YYYY.MM.dd}"
17     document_type => "exercise1"
18   }
19 }
```



@timestamp에서 연도, 월, 일을 추출해서 index이름을 생성할 때 추가해준다.

## logstash를 실행하자



```
$ bin/logstash -f code/output/elasticsearch/elasticsearch-dynamic-field-name-1.conf
```

## logstash를 실행하면...

Dev Tools      History    Settings    Help

Console

```
1 GET /exercise1-2018.07.15/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
```

다음과 같은 조합으로 index 이름 생성

- static : exercise
- dynamic : 2018.07.15

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 33,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "exercise1-2018.07.15",
16        "_type": "exercise1",
17        "_id": "kZUZn2QByNsCKuKnhCRB",
18        "_score": 1,
19        "_source": {
20          "age": 25,
21          "location": "UK",
22          "salary": 3900,
23          "employment_status": "employed",
24          "@version": "1",
25          "name": "Lisa",
26          "id": 14,
27          "@timestamp": "2018-07-15T18:00:07.473Z"
28        }
29      },
30      {
31        "_index": "exercise1-2018.07.15",
32        "_type": "exercise1",
33        "_id": "lJUZn2QByNsCKuKnhCRB",
34        "_score": 1,
35        "_source": {
36          "age": 38,
37          "location": "JP",
38          "salary": 3500,
39          "employment_status": "employed",
40          "@version": "1",
41          "name": "Tanaka",
42          "id": 17,
43        }
44      }
45    ]
46  }
47 }
```

**es document id를 event의 특정 field data를 이용해서 설정할 수 있나?**



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10   }
11 }
12
13 output {
14   elasticsearch {
15     hosts => ["elasticsearch:9200"]
16     index => "exercise2-%{+YYYY.MM.dd}"
17     document_type => "exercise2"
18     document_id => "%{@timestamp}-%{name}"
19   }
20 }
```



%{@timestamp}와 %{name} 부분에 각 event에서 수집한 실제 value가 입력된다

## logstash를 실행하자



```
$ bin/logstash -f code/output/elasticsearch/elasticsearch-dynamic-field-name-2.conf
```

## Kibana에서 확인하자

Dev Tools      History   Settings   Help

Console

```
1 GET exercise2-2018.07.15/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
```

▶ 🔍

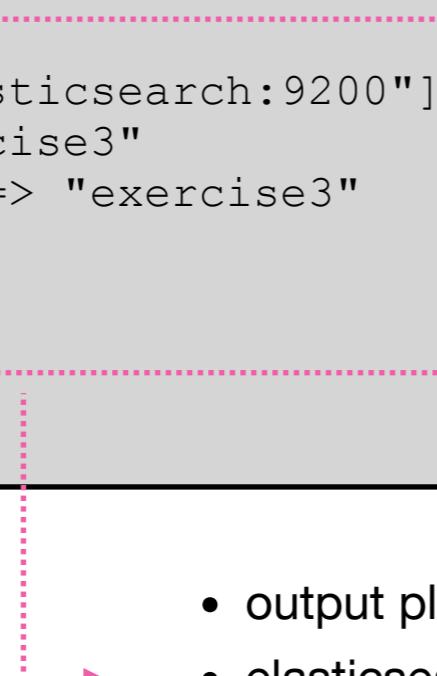
```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 33,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "exercise2-2018.07.15",
16        "_type": "exercise2",
17        "_id": "2018-07-15T18:08:19.785Z-Ken",
18        "_score": 1,
19        "_source": {
20          "@version": "1",
21          "id": 4,
22          "name": "Ken",
23          "salary": 3500,
24          "@timestamp": "2018-07-15T18:08:19.785Z",
25          "location": "US",
26          "age": 27,
27          "employment_status": "employed"
28        }
29      },
30      {
31        "_index": "exercise2-2018.07.15",
32        "_type": "exercise2",
33        "_id": "2018-07-15T18:08:19.786Z-Jessie",
34        "_score": 1,
35        "_source": {
36          "@version": "1",
37          "id": 5,
38          "name": "Jessie",
39          "salary": 4500,
40          "@timestamp": "2018-07-15T18:08:19.786Z",
41          "location": "US",
42          "age": 38,
43        }
44      }
45    ]
46  }
47 }
```

## **Output - multiple output plugins**

- (수집하고 전처리한 결과를) 여러 output에 전송하는 방법
- 개발 단계에서 `stdout+main output`을 함께 쓰면 빠르게 확인할 수 있다



```
1 input {
2   jdbc {
3     jdbc_validate_connection => true
4     jdbc_connection_string => "jdbc:mysql://52.78.134.20:3306/fc"
5     jdbc_user => "fc"
6     jdbc_password => "fc"
7     jdbc_driver_library => "/usr/share/logstash/driver/mysql-connector-java-5.1.36-bin.jar"
8     jdbc_driver_class => "com.mysql.jdbc.Driver"
9     statement => "SELECT * FROM fc"
10  }
11 }
12
13 output {
14   elasticsearch {
15     hosts => ["elasticsearch:9200"]
16     index => "exercise3"
17     document_type => "exercise3"
18   }
19   stdout {
20   }
21 }
```

- 
- output plugin에 2개(elasticsearch, stdout) 사용
  - elasticsearch와 stdout에 모두 전송
  - input/output/filter 모두 복수개 사용 가능

## logstash를 실행하자



```
$ bin/logstash -f code/output/elasticsearch/elasticsearch-stdout.conf
```

## logstash를 실행하면...

### stdout

```
{  
    "name" => "Josh",  
    "@timestamp" => 2018-07-15T18:53:15.758Z,  
    "@version" => "1",  
    "salary" => 4000,  
    "id" => 1,  
    "age" => 33,  
    "location" => "US",  
    "employment_status" => "employed"  
}  
  
{  
    "name" => "Tom",  
    "@timestamp" => 2018-07-15T18:53:15.796Z,  
    "@version" => "1",  
    "salary" => 45000,  
    "id" => 2,  
    "age" => 33,  
    "location" => "US",  
    "employment_status" => "employed"  
}  
  
{  
    "name" => "Kirk",  
    "@timestamp" => 2018-07-15T18:53:15.802Z,  
    "@version" => "1",  
    "salary" => 3000,  
    "id" => 3,  
    "age" => 23,  
    "location" => "US",  
    "employment_status" => "employed"  
}
```

### elasticsearch

```
1 {  
2     "took": 0,  
3     "timed_out": false,  
4     "_shards": {  
5         "total": 5,  
6         "successful": 5,  
7         "skipped": 0,  
8         "failed": 0  
9     },  
10    "hits": {  
11        "total": 33,  
12        "max_score": 1,  
13        "hits": [  
14            {  
15                "_index": "exercise3",  
16                "_type": "exercise3",  
17                "_id": "FJVKn2QByNsCKuKnKiU-",  
18                "_score": 1,  
19                "_source": {  
20                    "name": "Jennifer",  
21                    "@timestamp": "2018-07-15T18:53:15.812Z",  
22                    "@version": "1",  
23                    "salary": 5200,  
24                    "id": 6,  
25                    "age": 31,  
26                    "location": "US",  
27                    "employment_status": "unemployed"  
28                }  
29            },  
30            {  
31                "_index": "exercise3",  
32                "_type": "exercise3",  
33                "_id": "FZVKn2QByNsCKuKnKiU-",  
34                "_score": 1,  
35                "_source": {  
36                    "name": "Bob",  
37                    "@timestamp": "2018-07-15T18:53:15.812Z",  
38                    "@version": "1",  
39                    "salary": 22200,  
40                    "id": 7,  
41                    "age": 33,  
42                    "location": "US",  
43                }  
44            }  
45        ]  
46    }  
47 }
```

## Conditionals

- 조건에 따라 filter 또는 output을 선별적으로 적용하고 싶을 때
- 옵션
  - ▶ if
  - ▶ else if
  - ▶ else



```
1 input {
2     stdin {}
3 }
4
5 output {
6     if [message] in ["Seoul", "Busan"] {
7         elasticsearch {
8             hosts => ["elasticsearch:9200"]
9             index => "korea"
10    }
11   }
12  else if [message] == "Osaka" {
13      elasticsearch {
14          hosts => ["elasticsearch:9200"]
15          index => "japan"
16      }
17  }
18  else {
19      elasticsearch {
20          hosts => ["elasticsearch:9200"]
21          index => "else"
22      }
23  }
24 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/output/conditional.conf
```



```
The stdin plugin is now waiting for input:  
[2018-07-15T19:05:05,972] [INFO ] [logstash.agent  
Osaka  
Seoul  
New Yor  
New York
```



입력

## logstash를 실행하면...

Dev Tools

Console

The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. On the left, there is a sidebar with various icons for Dev Tools. The main area displays three search requests and their results:

```
1 GET korea/_search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
7
8 GET japan/_search
9 {
10   "query": {
11     "match_all": {}
12   }
13 }
14
15 GET else/_search
16 {
17   "query": {
18     "match_all": {}
19   }
20 }
```

On the right, the results for the 'korea/\_search' request are shown:

```
1 {
2   "took": 0,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 1,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "korea",
16        "_type": "doc",
17        "_id": "TZVVn2QByNsCKuKnoyVW",
18        "_score": 1,
19        "_source": {
20          "message": "Seoul",
21          "@timestamp": "2018-07-15T19:05:48.031Z",
22          "@version": "1",
23          "host": "fef3405d41af"
24        }
25      }
26    ]
27  }
28 }
```

## **Filter - csv**

- 특정한 separator로 연결된 데이터 (예: csv) parsing할 때 유용

이런 데이터가 있다고 하자

```
1 Index, Name, Survival, Pclass, Sex, Age, SibSp, Parch, Ticket, Fare, Embarked
2 1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S
3 2,Cumings,1,1,female,38,1,0,PC 17599,71.2833,C
4 3,Heikkinen,1,3,female,26,0,0,STON/O2. 3101282,7.925,S
```

## 아래와 같은 형태로 변형하려면 어떻게 해야 할까?

```
1  {
2      "Index" : 1,
3      "Name" : "Braud",
4      "Survival" : 0,
5      "Pclass" : 3,
6      "Sex" : "male",
7      "Age" : 22,
8      "Sibsp" : 1,
9      "Parch" : 0,
10     "Ticket" : "A/5 21171",
11     "Fare" : 7.25,
12     "Embarked" : "S"
13 }
```



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/titanic-header.csv"
4     start_position => "beginning"           ↪👉 test용 설정
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   csv {
11     separator => ","
12   }
13 }
14
15 output {
16   stdout {
17 }
18 }
```

👉👉 filter plugin의 전반적인 구조는 input/output과 유사

,(comma)로 구분된 csv 데이터이기에 ',' 입력

## logstash를 실행하자



```
$ bin/logstash -f code/filter/csv/separator.conf
```

## logstash를 실행하면...

Header에 해당하는 데이터



```
{  
    "host" => "fef3405d41af",  
    "column1" => "Index",  
    "column4" => "Pclass",  
    "@version" => "1",  
    "column3" => "Survival",  
    "column5" => "Sex",  
    "column2" => "Name",  
    "@timestamp" => 2018-07-15T19:20:15.795Z,  
    "column6" => "Age",  
    "column8" => "Parch",  
    "column9" => "Ticket",  
    "column11" => "Embarked",  
    "message" => "Index,Name,Survival,Pclass,Sex,Age,SibSp,Parch,Ticket,Fare,Embarked",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "column7" => "SibSp",  
    "column10" => "Fare"  
}  
  
{  
    "host" => "fef3405d41af",  
    "column1" => "1",  
    "column4" => "3",  
    "@version" => "1",  
    "column3" => "0",  
    "column5" => "male",  
    "column2" => "Braund",  
    "@timestamp" => 2018-07-15T19:20:15.868Z,  
    "column6" => "22",  
    "column8" => "0",  
    "column9" => "A/5 21171",  
    "column11" => "S",  
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "column7" => "1",  
    "column10" => "7.25"  
}
```

잘 구분되었으나 column 이름으로는



어떤 데이터를 나타내는지 파악이 어렵다

실제 데이터

## logstash를 실행하면...

```
{  
    "host" => "fef3405d41af",  
    "column1" => "Index",  
    "column4" => "Pclass",  
    "@version" => "1",  
    "column3" => "Survival",  
    "column5" => "Sex",  
    "column2" => "Name",  
    "@timestamp" => 2018-07-15T19:20:15.795Z,  
    "column6" => "Age",  
    "column8" => "Parch",  
    "column9" => "Ticket",  
    "column11" => "Embarked",  
    "message" => "Index,Name,Survival,Pclass,Sex,Age,SibSp,Parch,Ticket,Fare,Embarked",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "column7" => "SibSp",  
    "column10" => "Fare"  
}  
  
{  
    "host" => "fef3405d41af",  
    "column1" => "1",  
    "column4" => "3",  
    "@version" => "1",  
    "column3" => "0",  
    "column5" => "male",  
    "column2" => "Braund",  
    "@timestamp" => 2018-07-15T19:20:15.868Z,  
    "column6" => "22",  
    "column8" => "0",  
    "column9" => "A/5 21171",  
    "column11" => "S",  
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "column7" => "1",  
    "column10" => "7.25"  
}
```



Header

실제 데이터

column1, column2 등으로는 의미 파악이 어렵다

**csv의 Header 값이 Field 이름으로 오도록 설정하자**



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/titanic-header.csv"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   csv {
11     separator => ","
12     autodetect_column_names => true
13   }
14 }
15
16 output {
17   stdout {
18 }
19 }
```



이 옵션을 이용해서 첫번째 row의 데이터를 column으로 사용하도록 설정

## logstash를 실행하자



```
$ bin/logstash -f code/filter/csv/autodetect-column-names.conf
```

## logstash를 실행하면...

```
{  
    "Index" => "1",  
    "Ticket" => "A/5 21171",  
    "Fare" => "7.25",  
    "@timestamp" => 2018-07-16T02:11:47.106Z,  
    "host" => "fef3405d41af",  
    "SibSp" => "1",  
    "Parch" => "0",  
    "Embarked" => "S",  
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",  
    "@version" => "1",  
    "Sex" => "male",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "Survival" => "0",  
    "Age" => "22",  
    "Pclass" => "3",  
    "Name" => "Braund"  
}  
{  
    "Index" => "2",  
    "Ticket" => "PC 17599",  
    "Fare" => "71.2833",  
    "@timestamp" => 2018-07-16T02:11:47.112Z,  
    "host" => "fef3405d41af",  
    "SibSp" => "1",  
    "Parch" => "0",  
    "Embarked" => "C",  
    "message" => "2,Cumings,1,1,female,38,1,0,PC 17599,71.2833,C",  
    "@version" => "1",  
    "Sex" => "female",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "Survival" => "1",  
    "Age" => "38",  
    "Pclass" => "1",  
    "Name" => "Cumings"  
}
```



column name이 올바르게 매칭된 걸 볼 수 있다

그런데 모든 field가 string 처리 되었다.

적절히 integer, float, date type 등으로 변경해보자 



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/titanic-header.csv"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10  csv {
11    separator => ","
12    autodetect_column_names => true
13    convert => {
14      "Pclass" => "integer"
15      "Index" => "integer"
16      "Survival" => "integer"
17      "Fare" => "float"
18      "@timestamp" => "date"
19    }
20  }
21 }
22
23 output {
24   stdout {
25   }
26 }
```

"Pclass" Field의 Type을 integer로  
"Index" Field의 Type을 integer로  
"Survival" Field의 Type을 integer로  
"Fare" Field의 Type을 float로  
"@timestamp" Field의 Type을 date로

## logstash를 실행하자



```
$ bin/logstash -f code/filter/csv/convert.conf
```

## logstash를 실행하면...

**Before**

```
{  
    "Index" => "1",  
    "Ticket" => "A/5 21171",  
    "Fare" => "7.25",  
    "@timestamp" => 2018-07-16T02:11:47.106Z,  
    "host" => "fef3405d41af",  
    "SibSp" => "1",  
    "Parch" => "0",  
    "Embarked" => "S",  
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",  
    "@version" => "1",  
    "Sex" => "male",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "Survival" => "0",  
    "Age" => "22",  
    "Pclass" => "3",  
    "Name" => "Braund"  
}
```

**After**

```
{  
    "Parch" => "0",  
    "@version" => "1",  
    "Survival" => 0,  
    "host" => "fef3405d41af",  
    "SibSp" => "1",  
    "Ticket" => "A/5 21171",  
    "path" => "/usr/share/logstash/data/titanic-header.csv",  
    "Name" => "Braund",  
    "Embarked" => "S",  
    "Index" => 1,  
    "Pclass" => 3,  
    "Sex" => "male",  
    "message" => "1,Braund,0,3,male,22,1,0,A/5 21171,7.25,S",  
    "@timestamp" => 2018-07-16T02:43:50.871Z,  
    "Fare" => 7.25,  
    "Age" => "22"  
}
```

Q. elasticsearch output과 함께 사용시 mapping과 convert type이 다르면 어떻게 되는지?

A.

- mapping (o), convert (o) : mapping 우선
- mapping (x), convert (o) : convert에서 설정한 data type 우선
- mapping (x), convert (x) : dynamic mapping (elasticsearch 자동 설정)

## Filter - mutate

- Field data의 값을 변형할 때 사용하는 강력한 plugin
- 여러가지 option을 혼합해서 같이 사용한다

아래와 같은 데이터가 있다고 하자

1	<b>13.124.230.195:5601</b>
2	<b>13.124.230.195:3306</b>
3	<b>13.124.230.195:9200</b>
4	<b>13.124.230.195:9300</b>

아래와 같이 변경하고 싶다면?

```
{  
    "ip" : 13.124.230.195,  
    "port" : 5601  
},  
{  
    "ip" : 13.124.230.195,  
    "port" : 3306  
},  
{  
    "ip" : 13.124.230.195,  
    "port" : 9200  
},  
{  
    "ip" : 13.124.230.195,  
    "port" : 9300  
}
```

**우선 하나의 field를 특정 separator를 기준으로 나누자**

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/ip-address.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   mutate {
11     split => {
12       "message" => ":" 
13     }
14   }
15 }
16
17 output {
18   stdout {
19 }
20 }
```



☞ message field를 : separator로 나누기

## logstash를 실행하자



```
$ bin/logstash -f code/filter/mutate/split.conf
```

## logstash를 실행하면...

**Before**

```
{  
    "host" => "fef3405d41af",  
    "message" => "13.124.230.195:5601",  
    "@timestamp" => 2018-07-16T04:54:44.018Z,  
    "path" => "/usr/share/logstash/data/ip-address.log",  
    "@version" => "1"  
}
```

**After**

```
{  
    "@version" => "1",  
    "@timestamp" => 2018-07-16T04:49:25.645Z,  
    "message" => [  
        [0] "13.124.230.195",  
        [1] "5601"  
    ],  
    "path" => "/usr/share/logstash/data/ip-address.log",  
    "host" => "fef3405d41af"  
}
```

**split 결과를 활용해서 새로운 field를 생성하자**



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/ip-address.log"
4     start_position => "beginning"
5     since_db_path => "/dev/null"
6   }
7 }
8
9 filter {
10   mutate {
11     split => {
12       "message" => ":"}
13     }
14     add_field => {
15       "ip" => "%{message[0]}" split된 message의 첫 번째 데이터
16       "port" => "%{message[1]}" split된 message의 두 번째 데이터
17     }
18   }
19 }
20
21 output {
22   stdout {
23   }
24 }
```

## logstash를 실행하자



```
$ bin/logstash -f code/filter/mutate/add_field.conf
```

## logstash를 실행하면...

**Before**

```
{  
    "@version" => "1",  
    "@timestamp" => 2018-07-16T04:49:25.645Z,  
    "message" => [  
        [0] "13.124.230.195",  
        [1] "5601"  
    ],  
    "path" => "/usr/share/logstash/data/ip-address.log",  
    "host" => "fef3405d41af"  
}
```

**After**

```
{  
    "@version" => "1",  
    "host" => "fef3405d41af",  
    "path" => "/usr/share/logstash/data/ip-address.log",  
    "port" => "5601",  
    "message" => [  
        [0] "13.124.230.195",  
        [1] "5601"  
    ],  
    "@timestamp" => 2018-07-16T05:07:07.955Z,  
    "ip" => "13.124.230.195"  
}
```

**불필요한 field를 제거하자**



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/ip-address.log"
4     start_position => "beginning"
5     since_db_path => "/dev/null"
6   }
7 }
8
9 filter {
10   mutate {
11     split => {
12       "message" => ":"}
13     }
14     add_field => {
15       "ip" => "%{message[0]} "
16       "port" => "%{message[1]} "
17     }
18     remove_field => ["path", "@timestamp", "@version", "host", "message"]
19   }
20 }
21
22 output {
23   stdout {
24   }
25 }
```



제거할 field 이름을 직접 나열

## logstash를 실행하자



```
$ bin/logstash -f code/filter/mutate/remove_field.conf
```

## logstash를 실행하면...

**Before**

```
{  
    "@version" => "1",  
    "host" => "fef3405d41af",  
    "path" => "/usr/share/logstash/data/ip-address.log",  
    "port" => "5601",  
    "message" => [  
        [0] "13.124.230.195",  
        [1] "5601"  
    ],  
    "@timestamp" => 2018-07-16T05:07:07.955Z,  
    "ip" => "13.124.230.195"  
}
```

**After**

```
{  
    "ip" => "13.124.230.195",  
    "port" => "5601"  
}
```

## Filter - grok

- 구조화되어 있지 않은 데이터를 처리하는데 사용
- 실제 production log 분석할 때 grok + 다른 plugin 조합 활발히 사용

아래와 같은 로그가 생긴다고 하자

심화

	time	client_ip	res_time	status_code	req
1	2018-02-05T05:49:53.859060Z	172.30.39.133	0.079	200	"GET https://helloworld.com/users/1 HTTP/1.1"
2	2018-02-05T06:49:53.859060Z	172.29.43.253	0.1	404	"GET https://helloworld.com/users/3 HTTP/1.1"
3	2018-02-05T07:49:53.859060Z	172.30.40.210	0.052	503	"GET https://helloworld.com/users/2 HTTP/1.1"
4	2018-02-05T08:49:53.859060Z	172.31.40.131	0.038	200	"POST https://helloworld.com/users/5 HTTP/1.1"

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

	time	client_ip	res_time	status_code	req
1	2018-02-05T05:49:53.859060Z	172.30.39.133	0.079	200	"GET https://helloworld.com/users/1 HTTP/1.1"
2	2018-02-05T06:49:53.859060Z	172.29.43.253	0.1	404	"GET https://helloworld.com/users/3 HTTP/1.1"
3	2018-02-05T07:49:53.859060Z	172.30.40.210	0.052	503	"GET https://helloworld.com/users/2 HTTP/1.1"
4	2018-02-05T08:49:53.859060Z	172.31.40.131	0.038	200	"POST https://helloworld.com/users/5 HTTP/1.1"

- time 날짜/시간
- client\_ip ip 주소
- res\_time → 0보다 큰 소수점
- status\_code 정수
- req HTTP 메서드 + protocol://host:port/uri + HTTP 버전

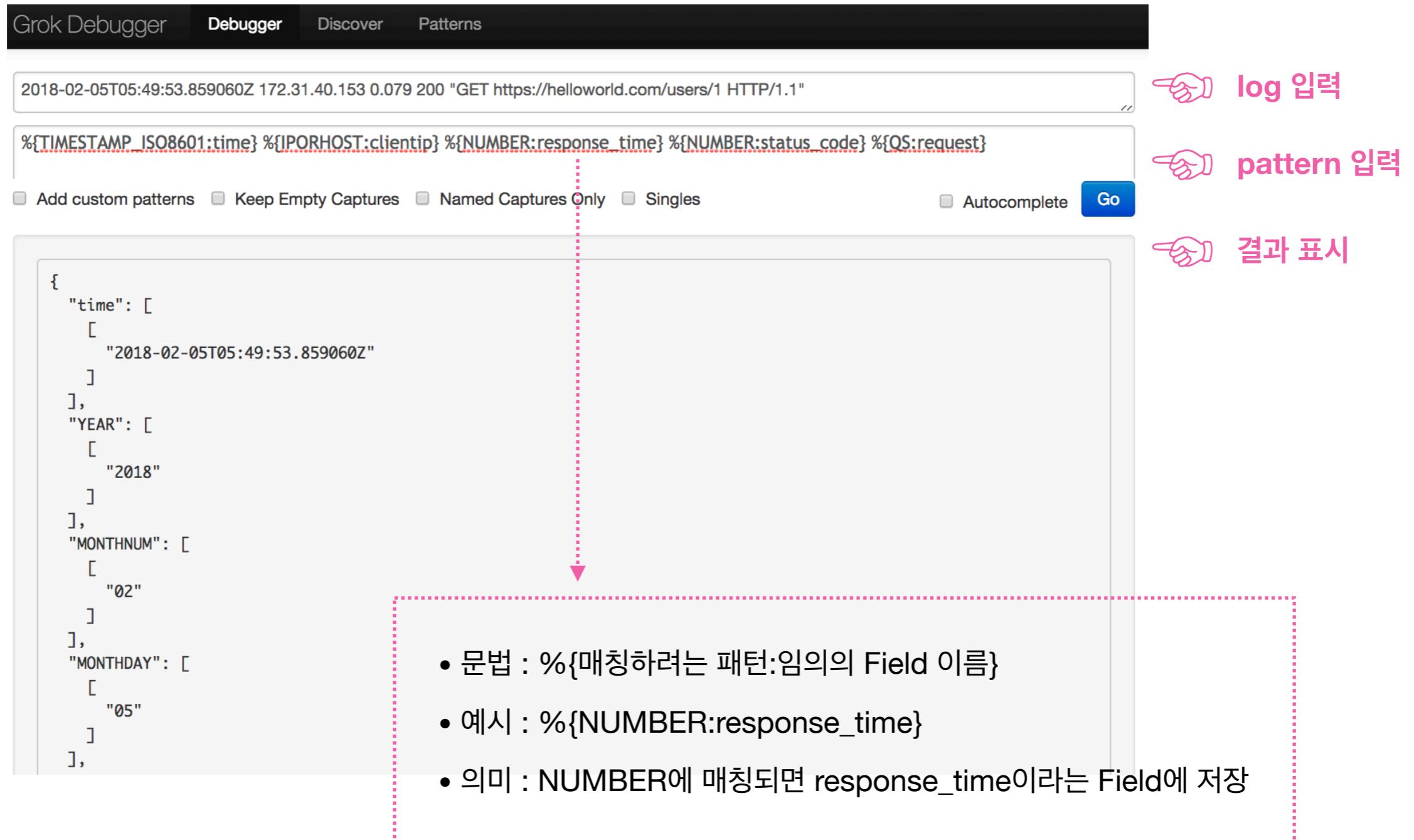
(각 pattern이 무엇을 의미하는지 사전 학습 필요)

	time	client_ip	res_time	status_code	req
1	2018-02-05T05:49:53.859060Z	172.30.39.133	0.079	200	"GET https://helloworld.com/users/1 HTTP/1.1"
2	2018-02-05T06:49:53.859060Z	172.29.43.253	0.1	404	"GET https://helloworld.com/users/3 HTTP/1.1"
3	2018-02-05T07:49:53.859060Z	172.30.40.210	0.052	503	"GET https://helloworld.com/users/2 HTTP/1.1"
4	2018-02-05T08:49:53.859060Z	172.31.40.131	0.038	200	"POST https://helloworld.com/users/5 HTTP/1.1"

- time 날짜/시간 TIMESTAMP\_ISO8601
- client\_ip ip 주소 IPORHOST
- res\_time → 0보다 큰 소수점 → NUMBER
- status\_code 정수 NUMBER
- req HTTP 메서드 + protocol://host:port/uri + HTTP 버전 QS

### 3. grok debugger를 이용해 시도해본다

심화



The screenshot shows the Grok Debugger interface. At the top, there are tabs: Grok Debugger (selected), Debugger, Discover, and Patterns. Below the tabs, a log entry is displayed: "2018-02-05T05:49:53.859060Z 172.31.40.153 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"".

In the middle section, a pattern is being defined: "%{TIMESTAMP\_ISO8601:time} %{IPORHOST:clientip} %{NUMBER:response\_time} %{NUMBER:status\_code} %{OS:request}". Below this, several checkboxes are available: Add custom patterns, Keep Empty Captures, Named Captures Only, Singles, Autocomplete (unchecked), and a blue "Go" button.

The bottom section shows the results of the pattern matching. A pink arrow points from the pattern definition down to the results table. The results table contains a single row of JSON objects:

```
{  
  "time": [  
    ["2018-02-05T05:49:53.859060Z"]  
  ],  
  "YEAR": [  
    ["2018"]  
  ],  
  "MONTHNUM": [  
    ["02"]  
  ],  
  "MONTHDAY": [  
    ["05"]  
  ]  
}
```

A pink dashed box highlights the "MONTHDAY" field in the results table. To the right of this box, three bullet points explain the pattern definition:

- 문법 : %{매칭하려는 패턴:임의의 Field 이름}
- 예시 : %{NUMBER:response\_time}
- 의미 : NUMBER에 매칭되면 response\_time이라는 Field에 저장

## logstash configuration을 확인하자



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13       '%{TIMESTAMP_ISO8601:time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16 }
17
18 output {
19   stdout {
20   }
21 }
```



grok debugger에서 작성했던 코드를 입력한다

## logstash를 실행하자



```
$ bin/logstash -f code/filter/grok/access.conf
```

## logstash를 실행하면...

```
{  
    "res_time" => "0.079",  
    "@timestamp" => 2018-07-16T05:47:08.357Z,  
        "req" => "\"GET https://helloworld.com/users/1 HTTP/1.1\"",  
    "clientip" => "172.30.39.133",  
    "@version" => "1",  
        "time" => "2018-02-05T05:49:53.859060Z",  
    "message" => "2018-02-05T05:49:53.859060Z 172.30.39.133 0.079 200 \"GET https://helloworld.com/users/1 HTTP/1.1\"",  
        "path" => "/usr/share/logstash/data/access.log",  
    "status_code" => "200",  
        "host" => "fef3405d41af"  
}  
  
{  
    "res_time" => "0.1",  
    "@timestamp" => 2018-07-16T05:47:08.431Z,  
        "req" => "\"GET https://helloworld.com/users/3 HTTP/1.1\"",  
    "clientip" => "172.29.43.253",  
    "@version" => "1",  
        "time" => "2018-02-05T06:49:53.859060Z",  
    "message" => "2018-02-05T06:49:53.859060Z 172.29.43.253 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\"",  
        "path" => "/usr/share/logstash/data/access.log",  
    "status_code" => "404",  
        "host" => "fef3405d41af"  
}
```

(이번에는) 아래와 같은 로그가 생긴다고 하자

심화

```
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 OSX 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)"
```

```
{  
    "clientip" => "123.222.333.123",  
    "host" => "fef3405d41af",  
    "path" => "/usr/share/logstash/data/apache.log",  
    "@version" => "1",  
    "bytes" => "332",  
    "ident" => "AWAY",  
    "referrer" => "\"http://www.referrer.com/bannerad/ba_ad.htm\"",  
    "@timestamp" => 2018-07-16T05:52:52.890Z,  
    "verb" => "GET",  
    "auth" => "-",  
    "httpversion" => "1.0",  
    "response" => "200",  
    "request" => "/bannerad/ad7.gif",  
    "timestamp" => "01/Feb/2003:01:08:53 -0800",  
    "message" => "123.222.333.123 AWAY - [01/Feb/2003:01:08:53 -0800] \"GET /bannerad/ad7.gif HTTP/1.0\" 200 332  
    "agent" => "\"Mozilla/4.01 (Macintosh; I; PPC)\""  
}
```

**(다행히) 자주/많이 사용되는 패턴은 등록되어 있다**

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/apache.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => { "message" => "%{COMBINEDAPACHELOG}" }
12   }
13 }
14
15 output {
16   stdout {
17   }
18 }
```



## logstash를 실행하자



```
$ bin/logstash -f code/filter/grok/apache.conf
```

## Filter - date

- elasticsearch는 기본적으로 입력된 시간을 utc로 생각한다
- 만약에 log data에 현지 시간으로 값이 입력되어 있으면 어떻게 될까?
- date filter를 사용하면 쉽게 해결할 수 있다

아래와 같은 로그가 생긴다고 하자

1	2018-02-12T16:03:38	Ben
2	2018-02-13T03:25:31	John
3	2018-02-14T13:31:11	Leo



한국 시간 기준

**date filer**를 사용한 것과 그렇지 않은 경우를 비교하자

## mapping 설정

### default

```
1 PUT exercise4
2 {
3   "mappings" : {
4     "exercise4" : {
5       "properties" : {
6         "time" : {
7           "type" : "date"
8         },
9         "name" : {
10           "type" : "keyword"
11         }
12       }
13     }
14   }
15 }
```

### date filter

```
1 PUT exercise5
2 {
3   "mappings" : {
4     "exercise5" : {
5       "properties" : {
6         "time" : {
7           "type" : "date"
8         },
9         "name" : {
10           "type" : "keyword"
11         }
12       }
13     }
14   }
15 }
```

## 아래와 같은 logstash.conf로 데이터를 전송하자

### default

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/date.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" => '%{TIMESTAMP_ISO8601:time} %{WORD:name}'
13     }
14     remove_field =>
15       ["path", "@timestamp", "@version", "host", "message"]
16   }
17 }
18
19 output {
20   elasticsearch {
21     hosts => ["elasticsearch:9200"]
22     index => "exercise4"
23     document_type => "exercise4"
24   }
25 }
```

### date filter

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/date.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:time} %{WORD:name}'
14     }
15     remove_field =>
16       ["path", "@timestamp", "@version", "host", "message"]
17   }
18   date {
19     match => ["time", "YYYY-MM-dd'T'HH:mm:ss"]
20     timezone => "Asia/Seoul"
21     target => "time"
22   }
23 }
24
25 output {
26   elasticsearch {
27     hosts => ["elasticsearch:9200"]
28     index => "exercise5"
29     document_type => "exercise5"
30   }
31 }
```

## logstash를 실행하자

**default**



```
$ bin/logstash -f code/filter/date/default.conf
```

**date filter**



```
$ bin/logstash -f code/filter/date/date.conf
```

## date filter 적용 전/후 elasticserch 저장된 값을 비교하자

입력값	default	date filter
2018-02-12 16:03:38	<pre>{   "_index": "exercise4",   "_type": "exercise4",   "_id": "AWL0hVoszMqVnr-9MyRS",   "_score": 1,   "_source": {     "name": "Ben",     "time": "2018-02-12T16:03:38"   } }, {   "_index": "exercise4",   "_type": "exercise4",   "_id": "AWL0hVoszMqVnr-9MyRT",   "_score": 1,   "_source": {     "name": "John",     "time": "2018-02-13T03:25:31"   } }, {   "_index": "exercise4",   "_type": "exercise4",   "_id": "AWL0hVoszMqVnr-9MyRU",   "_score": 1,   "_source": {     "name": "Leo",     "time": "2018-02-14T13:31:11"   } }</pre>	<pre>{   "_index": "exercise5",   "_type": "exercise5",   "_id": "AWL0hnY-zMqVnr-9MyRV",   "_score": 1,   "_source": {     "name": "Ben",     "time": "2018-02-12T07:03:38.000Z"   } }, {   "_index": "exercise5",   "_type": "exercise5",   "_id": "AWL0hnZ9zMqVnr-9MyRW",   "_score": 1,   "_source": {     "name": "John",     "time": "2018-02-12T18:25:31.000Z"   } }, {   "_index": "exercise5",   "_type": "exercise5",   "_id": "AWL0hnZ9zMqVnr-9MyRX",   "_score": 1,   "_source": {     "name": "Leo",     "time": "2018-02-14T04:31:11.000Z"   } }</pre>
2018-02-13 03:25:31		
2018-02-14 013:31:11		

## Filter - drop

- 특정한 event 자체를 무시할 경우 사용
- 예를 들어 log를 수집할 때 INFO 레벨은 무시하고 싶을 때 사용 가능

## logstash configuration을 확인하자



```
1 input {
2   stdin { }
3 }
4
5 filter {
6   if [message] == "hello" {
7     drop { }
8   }
9 }
10
11 output {
12   stdout {
13 }
14 }
```



message가 정확히 hello이면 해당 event 무시

## logstash를 실행하자



```
$ bin/logstash -f code/filter/drop/drop1.conf
```

## logstash를 실행하면...

```
The stdin plugin is now waiting for input:  
[2018-07-16T06:37:56,988][INFO ][logstash.agent] ]  
hello  
hi  
{  
    "message" => "hi",  
    "@timestamp" => 2018-07-16T06:38:28.286Z,  
    "@version" => "1",  
    "host" => "fef3405d41af"  
}  
bye  
{  
    "message" => "bye",  
    "@timestamp" => 2018-07-16T06:38:34.552Z,  
    "@version" => "1",  
    "host" => "fef3405d41af"  
}
```

message가 hello인 event는 무시된다

**access.log에서 status code가 200인 event는 무시해보자**

## logstash configuration을 확인하자



```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16   if [status_code] == "200" {
17     drop {}
18   }
19 }
20
21 output {
22   stdout {
23   }
24 }
```

status code가 200인, 즉 정상인 결과는 수집하지 않고 에러 로그만 수집

## logstash를 실행하자



```
$ bin/logstash -f code/filter/drop/drop2.conf
```

## logstash를 실행하면...

```
{  
    "res_time" => "0.1",  
    "req" => "\"GET https://helloworld.com/users/3 HTTP/1.1\"",  
    "path" => "/usr/share/logstash/data/access.log",  
    "@timestamp" => "2018-07-16T06:43:56.669Z",  
    "status_code" => "404",  
    "@version" => "1",  
    "message" => "2018-02-05T06:49:53.859060Z 172.29.43.253 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\"",  
    "host" => "fef3405d41af",  
    "time" => "2018-02-05T06:49:53.859060Z",  
    "clientip" => "172.29.43.253"  
}  
{  
    "res_time" => "0.052",  
    "req" => "\"GET https://helloworld.com/users/2 HTTP/1.1\"",  
    "path" => "/usr/share/logstash/data/access.log",  
    "@timestamp" => "2018-07-16T06:43:56.680Z",  
    "status_code" => "503",  
    "@version" => "1",  
    "message" => "2018-02-05T07:49:53.859060Z 172.30.40.210 0.052 503 \"GET https://helloworld.com/users/2 HTTP/1.1\"",  
    "host" => "fef3405d41af",  
    "time" => "2018-02-05T07:49:53.859060Z",  
    "clientip" => "172.30.40.210"  
}
```

## Filter - ruby

- ruby code를 이용해 custom filter 작성
- logstash는 ruby로 되어 있음 
- 종류
  - Inline ruby code 
  - ~~ruby script file~~ (수업)

**단, ruby code를 다 볼 순 없으므로  
Event API의 기본문법만 익히고  
몇 가지 예시를 통해 어떻게 사용하는지 보자**

## Event API - SET

문법	예시
<code>event.set('필드명', 값)</code>	<code>event.set('new field', 100)</code>
<code>event.set(' [상위필드] [하위필드] ', 값)</code> (※ object 형태)	<code>event.set(' [foo] [bar] ', 'hello')</code>

**새로운 필드 생성하고 값 부여**

## Event API - GET

문법	예시
<code>event.get('필드명')</code>	<code>event.get('new field')</code>
<code>event.get(' [상위필드] [하위필드] ')</code> (※ object 형태)	<code>event.get(' [foo] [bar] ')</code>

100

'hello'

이미 존재하는 특정 Field의 값 조회

**예시1. time object에 시간을 더하거나 빼기**

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/date.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {"message" => '%{TIMESTAMP_ISO8601:time} %{WORD:name}'}
12     remove_field => ["path", "@timestamp", "@version", "host", "message"]
13   }
14   date {
15     match => ["time", "YYYY-MM-dd'T'HH:mm:ss"]
16     timezone => "Asia/Seoul"
17     target => "time2"
18     remove_field => ["time"]
19   }
20   ruby {
21     code =>
22       "event.set('_time2', event.get('time2').time.localtime('+09:00').strftime('%Y-%m-%dT%H:%M:%S'))"
23   }
24 }                                ▶ _time2라는 field를 생성하고
25                                         • time2 field를
26                                         • time object로 변환하고
27                                         • 9시간을 더한 뒤
28                                         • '%Y-%m-%dT%H:%M:%S' format으로 변환한 값을 갖도록 하자
output {
  stdout {}
}
```

## logstash를 실행하자



```
$ bin/logstash -f code/filter/ruby/example1.conf
```

## logstash를 실행하면...

```
{  
    "_time2" => "2018-02-12T16:03:38",  
    "time2" => 2018-02-12T07:03:38.000Z,  
    "name" => "Ben"  
}  
{  
    "_time2" => "2018-02-14T13:31:11",  
    "time2" => 2018-02-14T04:31:11.000Z,  
    "name" => "Leo"  
}  
{  
    "_time2" => "2018-02-13T03:25:31",  
    "time2" => 2018-02-12T18:25:31.000Z,  
    "name" => "John"  
}
```

## 예시2. 특정 separator로 문장을 split하고 특정 위치의 값 접근하기

```
1 2018-02-05T05:49:53.859060Z 172.30.39.133 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"
2 2018-02-05T06:49:53.859060Z 172.29.43.253 0.1 404 "GET https://helloworld.com/users/3 HTTP/1.1"
3 2018-02-05T07:49:53.859060Z 172.30.40.210 0.052 503 "GET https://helloworld.com/users/2 HTTP/1.1"
4 2018-02-05T08:49:53.859060Z 172.31.40.131 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
```

ex) status code

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   ruby {
11     code => "event.set('status_code', (event.get('message')).split(' ')[3]).to_i)"
12     remove_field => ["path", "@version", "host", "@timestamp"]
13   }
14 }
15
16 output {
17   stdout { }
18 }
```

▶ status\_code라는 field를 생성하고

- message field를
- white space를 기준으로 나눈 후에
- 4번째 (zero indexing) 값을
- integer로 변환한 뒤 (default : string)
- 값을 할당하자

## logstash를 실행하자



```
$ bin/logstash -f code/filter/ruby/example2.conf
```

## logstash를 실행하면...

```
{  
    "status code" => 404,  
    "message" => "2018-02-05T06:49:53.859060Z 172.29.43.253 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\""  
}  
{  
    "status code" => 200,  
    "message" => "2018-02-05T08:49:53.859060Z 172.31.40.131 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\""  
}  
{  
    "status code" => 200,  
    "message" => "2018-02-05T05:49:53.859060Z 172.30.39.133 0.079 200 \"GET https://helloworld.com/users/1 HTTP/1.1\""  
}  
{  
    "status code" => 503,  
    "message" => "2018-02-05T07:49:53.859060Z 172.30.40.210 0.052 503 \"GET https://helloworld.com/users/2 HTTP/1.1\""  
}
```

## **Filter - elasticsearch**

- 현재 event의 데이터를 이용해 elasticsearch에 search/aggregation 수행
- 위에서 수행한 search/aggregation 결과를 현재 event에 field로 추가

## 예시1.

**access log에서 ip 정보를 elasticsearch에서 조회하여**

**(접속 이력이 있는 ip의 경우) 가장 최근 접속 일자를 새로운 field에 추가하자**

```
1 2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"  
2 2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 "GET https://helloworld.com/users/3 HTTP/1.1"  
3 2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 "GET https://helloworld.com/users/2 HTTP/1.1"  
4 2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"  
5 2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
```



이 값으로 elasticsearch에 검색

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access2.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:current_time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16   elasticsearch {
17     hosts => ["elasticsearch.higee.co"]
18     index => "nginx-*"
19     query => "nginx.access.geoip.ip:%{[clientip]}"
20     sort => "@timestamp:desc"
21     fields => {"@timestamp" => "previous_time"}
22     remove_field => ["status_code", "path", "@timestamp", "req", "res_time", "@version", "host"]
23   }
24 }
25
26 output {
27   stdout {
28   }
29 }
```

clientip 값으로 검색이 수행된다

previous\_time field를 생성하고 return된 document에서 @timestamp 값을 할당해라

## logstash를 실행하자

```
🐳 $ bin/logstash -f code/filter/elasticsearch/example1.conf
```

## logstash를 실행하면...

```
① {
    "current_time" => "2018-09-10T16:51:42.119Z",
    "clientip" => "115.127.73.2",
    "previous_time" => "2018-09-09T17:10:09.000Z",
    "message" => "2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\""
}

{
    "current_time" => "2018-09-10T08:09:03.779Z",
    "clientip" => "93.117.2.7",
    "previous_time" => "2018-09-09T15:53:08.000Z",
    "message" => "2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\""
}

{
    "current_time" => "2018-09-10T05:49:22.859Z",
    "clientip" => "178.73.215.171",
    "previous_time" => "2018-09-09T19:34:55.000Z",
    "message" => "2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 \"GET https://helloworld.com/users/1 HTTP/1.1\""
}

{
    "current_time" => "2018-09-10T07:41:39.239Z",
    "clientip" => "207.148.120.201",
    "previous_time" => "2018-09-09T15:11:18.000Z",
    "message" => "2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 \"GET https://helloworld.com/users/2 HTTP/1.1\""
}

{
    "current_time" => "2018-09-10T08:19:13.989Z",
    "message" => "2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\"",
    "clientip" => "93.115.2.7"
}
```

① 기존에 접속한 적이 있는 ip를 가진 event → previous\_time field 추가

② 기존에 접속한 적이 없는 ip를 가진 event

## 예시2.

지난 번 접속 시간과 이번 접속 시간의 차를 구해서

새로운 필드를 생성하자

```
1 2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"
2 2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 "GET https://helloworld.com/users/3 HTTP/1.1"
3 2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 "GET https://helloworld.com/users/2 HTTP/1.1"
4 2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
5 2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
```



이 값으로 elasticsearch에 검색

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access2.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:current_time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16   elasticsearch {
17     hosts => ["elasticsearch.higee.co"]
18     index => "nginx-*"
19     query => "nginx.access.geoip.ip:%{[clientip]}"
20     sort => "@timestamp:desc"
21     fields => {"@timestamp" => "previous_time"}
22   }
23   if [previous_time] {
24     date {
25       match => ["current_time", "YYYY-MM-dd'T'HH:mm:ss.SSSZ"]
26       target => "current_time"
27     }
28     date {
29       match => ["previous_time", "YYYY-MM-dd'T'HH:mm:ss.SSSZ"]
30       target => "previous_time"
31     }
32     ruby {
33       code => "event.set('access_interval', ((event.get('current_time') - event.get('previous_time'))/3600).round(2))"
34       remove_field => ["status_code", "path", "@timestamp", "req", "res_time", "@version", "host"]
35     }
36   }
37 }
38
39 output {
40   stdout {
41   }
42 }
```

현재 접속 시간과 지난 번 접속 시간의 차를 구해 얼마 만에 방문했는지를 기록

## logstash를 실행하자



```
$ bin/logstash -f code/filter/elasticsearch/example2.conf
```

## logstash를 실행하면...

```
{  
    "clientip" => "115.127.73.2",  
    "previous_time" => 2018-09-09T17:10:09.000Z,  
    "message" => "2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\"",  
    "current_time" => 2018-09-10T16:51:42.119Z,  
    "access_interval" => 23.69  
}  
  
{  
    "clientip" => "93.117.2.7",  
    "previous_time" => 2018-09-09T15:53:08.000Z,  
    "message" => "2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\"",  
    "current_time" => 2018-09-10T08:09:03.779Z,  
    "access_interval" => 16.27  
}  
  
{  
    "clientip" => "178.73.215.171",  
    "previous_time" => 2018-09-09T19:34:55.000Z,  
    "message" => "2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 \"GET https://helloworld.com/users/1 HTTP/1.1\"",  
    "current_time" => 2018-09-10T05:49:22.859Z,  
    "access_interval" => 10.24  
}  
  
{  
    "clientip" => "207.148.120.201",  
    "previous_time" => 2018-09-09T15:11:18.000Z,  
    "message" => "2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 \"GET https://helloworld.com/users/2 HTTP/1.1\"",  
    "current_time" => 2018-09-10T07:41:39.239Z,  
    "access_interval" => 16.51  
}  
  
{  
    "clientip" => "93.115.2.7",  
    "message" => "2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\"",  
    "current_time" => "2018-09-10T08:19:13.989Z"  
}
```



첫 방문이므로 access\_interval이 존재하지 않음

### 예제3. Query DSL을 사용해보자

```
1 2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"  
2 2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 "GET https://helloworld.com/users/3 HTTP/1.1"  
3 2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 "GET https://helloworld.com/users/2 HTTP/1.1"  
4 2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"  
5 2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
```



이 값으로 elasticsearch에 검색

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access2.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:current_time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16   elasticsearch {
17     hosts => ["elasticsearch.higee.co"]
18     index => "nginx-*"
19     query_template => "/usr/share/logstash/code/filter/elasticsearch/example3.json" [pink box]
20     fields => {"@timestamp" => "previous_time"}
21     remove_field => ["status_code", "path", "@timestamp", "req", "res_time", "@version", "host"]
22   }
23 }
24
25 output {
26   stdout {
27   }
28 }
```

query DSL을 작성한 파일 참조

## query template을 확인하자

```
1  {
2    "size": 1,
3    "sort" : [ { "@timestamp" : "desc" } ],
4    "_source" : ["@timestamp"],
5    "query": {
6      "term": {
7        "nginx.access.geoip.ip" : "%{[clientip]}"
8      }
9    }
10 }
```

각 event의 clientip로 검색 수행

## logstash를 실행하자

```
🐳 $ bin/logstash -f code/filter/elasticsearch/example3.conf
```

## logstash를 실행하면...

```
{  
    "message" => "2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 \\"GET https://helloworld.com/users/3 HTTP/1.1\\\"",  
    "clientip" => "115.127.73.2",  
    "current_time" => "2018-09-10T16:51:42.119Z",  
    "previous_time" => "2018-09-09T17:10:09.000Z"  
}  
  
{  
    "message" => "2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 \\"POST https://helloworld.com/users/5 HTTP/1.1\\\"",  
    "clientip" => "93.117.2.7",  
    "current_time" => "2018-09-10T08:09:03.779Z",  
    "previous_time" => "2018-09-09T15:53:08.000Z"  
}  
  
{  
    "message" => "2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 \\"GET https://helloworld.com/users/1 HTTP/1.1\\\"",  
    "clientip" => "178.73.215.171",  
    "current_time" => "2018-09-10T05:49:22.859Z",  
    "previous_time" => "2018-09-09T19:34:55.000Z"  
}  
  
{  
    "message" => "2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 \\"GET https://helloworld.com/users/2 HTTP/1.1\\\"",  
    "clientip" => "207.148.120.201",  
    "current_time" => "2018-09-10T07:41:39.239Z",  
    "previous_time" => "2018-09-09T15:11:18.000Z"  
}  
  
{  
    "clientip" => "93.115.2.7",  
    "current_time" => "2018-09-10T08:19:13.989Z",  
    "message" => "2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 \\"POST https://helloworld.com/users/5 HTTP/1.1\\\""  
}
```

## 예시4 Aggregation 결과도 event에서 사용 가능한가?

```
1 2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 "GET https://helloworld.com/users/1 HTTP/1.1"  
2 2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 "GET https://helloworld.com/users/3 HTTP/1.1"  
3 2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 "GET https://helloworld.com/users/2 HTTP/1.1"  
4 2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"  
5 2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 "POST https://helloworld.com/users/5 HTTP/1.1"
```



이 값으로 elasticsearch에 검색

## logstash configuration을 확인하자

```
1 input {
2   file {
3     path => "/usr/share/logstash/data/access2.log"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6   }
7 }
8
9 filter {
10   grok {
11     match => {
12       "message" =>
13         '%{TIMESTAMP_ISO8601:current_time} %{IPORHOST:clientip} %{NUMBER:res_time} %{NUMBER:status_code} %{QS:req}'
14     }
15   }
16   elasticsearch {
17     hosts => ["elasticsearch.higee.co"]
18     index => "nginx-*"
19     query_template => "/usr/share/logstash/code/filter/elasticsearch/example4.json"
20     fields => {"@timestamp" => "previous_time"}
21     remove_field => ["status_code", "path", "@timestamp", "req", "res_time", "@version", "host", "current_time"]
22     aggregation_fields => {
23       "this_is_avg" => "elasticsearch_aggregation"
24     }
25   }
26   if !*[elasticsearch_aggregation][value] { → this_is_avg라는 aggregation 값을 elasticsearch_aggregation field에 저장
27     mutate {
28       remove_field => ["elasticsearch_aggregation"]
29     }
30   } → [elasticsearch_aggregation][value]라는 field가 존재하지 않는 경우 (=검색 결과 없는 경우)
31   else {
32     ruby {
33       code => "event.set('average_byte', event.get('[elasticsearch_aggregation][value]').round(0))"
34       remove_field => ["elasticsearch_aggregation"]
35     }
36   }
37 }
38
39 output {
40   stdout {
41   }
42 }
```

## logstash configuration을 확인하자

```
1  {
2      "size": 0,
3      "sort" : [ { "@timestamp" : "desc" } ],
4      "query": {
5          "term": {
6              "nginx.access.geoip.ip" : "%{[clientip]}"
7          }
8      },
9      "aggs": {
10         "this_is_avg": {
11             "avg": {
12                 "field": "nginx.access.body_sent.bytes"
13             }
14         }
15     }
16 }
```

search

aggregation

aggregation 문법은 수업에 배우지는 않았지만 향후 참고!

## logstash를 실행하자

```
🐳 $ bin/logstash -f code/filter/elasticsearch/example4.conf
```

## logstash를 실행하면...

```
{  
    "message" => "2018-09-10T07:41:39.239Z 207.148.120.201 0.052 503 \"GET https://helloworld.com/users/2 HTTP/1.1\"",  
    "average_byte" => 189.0,  
    "clientip" => "207.148.120.201"  
}  
  
{  
    "message" => "2018-09-10T08:19:13.989Z 93.115.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\"",  
    "clientip" => "93.115.2.7"  
}  
  
{  
    "message" => "2018-09-10T05:49:22.859Z 178.73.215.171 0.079 200 \"GET https://helloworld.com/users/1 HTTP/1.1\"",  
    "average_byte" => 436.0,  
    "clientip" => "178.73.215.171"  
}  
  
{  
    "message" => "2018-09-10T16:51:42.119Z 115.127.73.2 0.1 404 \"GET https://helloworld.com/users/3 HTTP/1.1\"",  
    "average_byte" => 436.0,  
    "clientip" => "115.127.73.2"  
}  
  
{  
    "message" => "2018-09-10T08:09:03.779Z 93.117.2.7 0.038 200 \"POST https://helloworld.com/users/5 HTTP/1.1\"",  
    "average_byte" => 436.0,  
    "clientip" => "93.117.2.7"  
}
```

**질문 및 Feedback은**

**gshock94@gmail.com로 주세요**