6️⃣ [5주차] 6장 로그스태시

로그 수집, 가공, 전송 일련의 과정을 간편하게 해준다.

6.1 로그스태시란

- 플러그인 기반의 오픈소스 데이터 처리 ***파이프라인*** 도구
- 데이터를 저장하기 전에 원하는 형태로 ***가공*** 하는 역할
- 특징
  - 플러그인 기반: 로그스태시의 파이프라인은 플러그인 형태의 요소들로 이뤄져있다.
  - 모든 형태의 데이터 처리: 이벤트 데이터(시간에 따라 발생하는 데이터)를 처리하는 데최적화되어있다. 시간과 무관한 데이터를 처리하는 작업도 유연하게 사용된다.
  - 성능: 자체적으로 내장된 메모리와 파일 기반의 큐를 사용한다. → 처리속도와 안정성이 높다.
    - 인덱싱할 도큐먼트의 수와 용량을 종합적으로 고려해 벌크 인덱싱을 수행한다.
    - 파이프라인 배치 크기 조정을 통해 병목현상을 방지하고 성능을 최적화한다.
  - 안정성: ES의 장애 상황에 대응하기 위한 재시도 로직이나 오류가 발생한 도큐먼트를 따로 보관하는 `데드 레터 큐`를 내장한다.
    - 파일 기반의 큐를 사용할 경우 뜻하지 않은 로그스태시의 장애 상황에도 도큐먼트 유실을 최소화할 수 있다.

6.2 설치

- 지금은 jdk 도 같이 설치되어 jdk를 신경쓰지 않아도 된다. 다만 OS는 참고해야 한다.

```
brew install logstash
cd /usr/local
bin/logstash -e '
input { stdin {} }
output { stdout{} }'
```

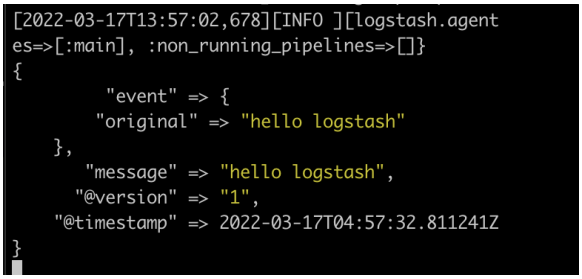
```
songi@MacBook-Pro-2 local % ./bin/logstash -e '
quote> input { stdin { } }
quote> output { stdout { } }'
Using LS_JAVA_HOME defined java: /usr/local/opt/openjdk@11/libexec/openjdk.jdk/Contents/Home.
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.
Sending Logstash logs to /usr/local/Cellar/logstash/8.0.1/libexec/logs which is now configured via log4j2.properties
[2022-03-17T13:34:54,957][INFO ][logstash.runner          ] Log4j configuration path used is: /usr/local/Cellar/logstash/8.0.1/libexec/config/log4j
2.properties
[2022-03-17T13:34:54,967][WARN ][logstash.runner          ] The use of JAVA_HOME has been deprecated. Logstash 8.0 and later ignores JAVA_HOME and
uses the bundled JDK. Running Logstash with the bundled JDK is recommended. The bundled JDK has been verified to work with each specific version of
 Logstash, and generally provides best performance and reliability. If you have compelling reasons for using your own JDK (organizational-specific
compliance requirements, for example), you can configure LS_JAVA_HOME to use that version instead.
[2022-03-17T13:34:54,969][INFO ][logstash.runner          ] Starting Logstash {"logstash.version"=>"8.0.1", "jruby.version"=>"jruby 9.2.20.1 (2.5.8
) 2021-11-30 2a2962fbd1 OpenJDK 64-Bit Server VM 11.0.14.1+0 on 11.0.14.1+0 +indy +jit [darwin-x86_64]"}
[2022-03-17T13:34:54,971][INFO ][logstash.runner          ] JVM bootstrap flags: [-Xms1g, -Xmx1g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupan
cyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djruby.compile.invokedynamic=true, -Djruby.jit
.threshold=0, -Djruby.regexp.interruptible=true, -XX:+HeapDumpOnOutOfMemoryError, -Djava.security.egd=file:/dev/urandom, -Dlog4j2.isThreadContextMa
pInheritable=true, --add-opens=java.base/java.security=ALL-UNNAMED, --add-opens=java.base/java.io=ALL-UNNAMED, --add-opens=java.base/java.nio.chann
els=ALL-UNNAMED, --add-opens=java.base/sun.nio.ch=ALL-UNNAMED, --add-opens=java.management/sun.management=ALL-UNNAMED]
[2022-03-17T13:34:55,065][WARN ][logstash.config.source.multilocal] Ignoring the 'pipelines.yml' file because modules or command line options are s
pecified
[2022-03-17T13:34:55,092][INFO ][logstash.agent           ] No persistent UUID file found. Generating new UUID {:uuid=>"b19be04a-ea1a-451e-9adc-f80
c8090323c", :path=>"/usr/local/Cellar/logstash/8.0.1/libexec/data/uuid"}
[2022-03-17T13:34:56,026][INFO ][logstash.agent           ] Successfully started Logstash API endpoint {:port=>9600, :ssl_enabled=>false}
[2022-03-17T13:34:56,382][INFO ][org.reflections.Reflections] Reflections took 68 ms to scan 1 urls, producing 120 keys and 417 values
[2022-03-17T13:34:57,158][INFO ][logstash.javapipeline    ] Pipeline `main` is configured with `pipeline.ecs_compatibility: v8` setting. All plugin
s in this pipeline will default to `ecs_compatibility => v8` unless explicitly configured otherwise.
[2022-03-17T13:34:57,261][INFO ][logstash.javapipeline    ][main] Starting pipeline {:pipeline_id=>"main", "pipeline.workers"=>8, "pipeline.batch.s
ize"=>125, "pipeline.batch.delay"=>50, "pipeline.max_inflight"=>1000, "pipeline.sources"=>["config string"], :thread=>"#<Thread:0x30be41d8 run>"}
[2022-03-17T13:34:58,076][INFO ][logstash.javapipeline    ][main] Pipeline Java execution initialization time {"seconds"=>0.81}
[2022-03-17T13:34:58,145][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[2022-03-17T13:34:58,196][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
```

```
hello
{
      "@version" => "1",
         "event" => {
        "original" => "hello"
    },
       "message" => "hello",
    "@timestamp" => 2022-03-17T04:35:36.135406Z,
          "host" => {
        "hostname" => "MacBook-Pro-2.local"
    }
}
```

standard input-output

- @ 기호는 로그스태시에 의해 생성된 필드
- @가없는 필드: 수집을 통해 얻어진 정보

```
input {
  { 입력 플러그인 }
}
filter {
  { 필터 플러그인 }
}
output {
  { 출력 플러그인 }
}
```



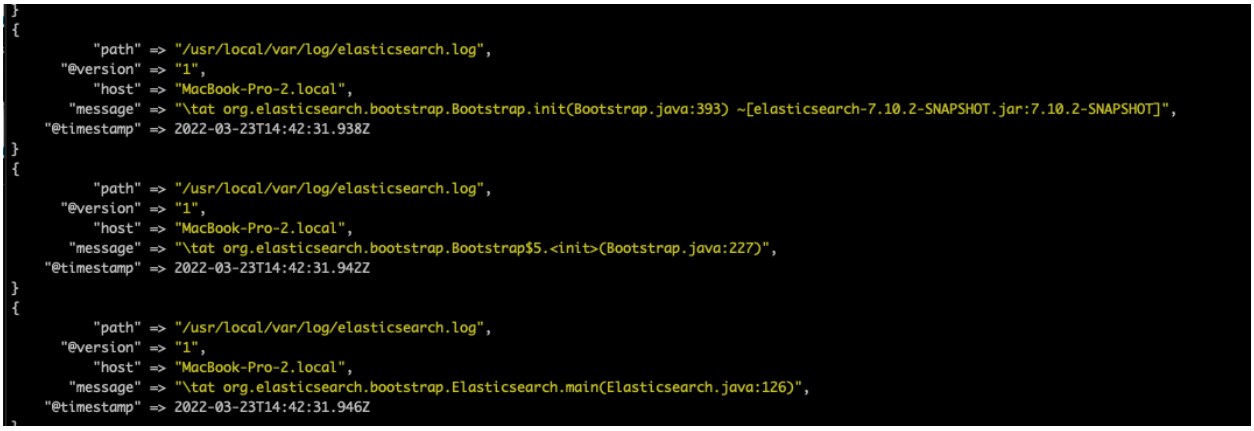
input { tcp { port =⇒9900} } 일 때, echo 'hello logstash' | nc localhost 9900 을 보내면

```
input {
    tcp {
        port => 9900
    }
}
output {
    elasticsearch {
        hosts => ["localhost:9200"]
        user => "" # security
        password => ""
    }

}
```

```
input {
  file {
    path => "/usr/local/var/log/elasticsearch.log"
  }
}

output {
  stdout { }
}
```



elasticsearch.log 파일이 업데이트 될 때마다 출력 플러그인으로 출력된다.

6.3 파이프라인

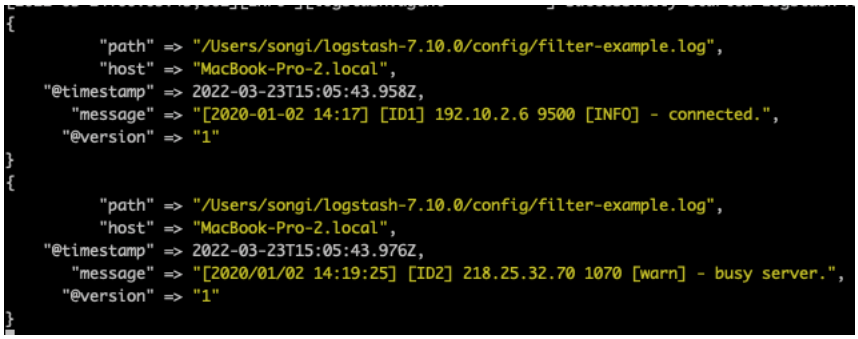
- 데이터를 입력 받아 실시간으로 변경하고 다른 시스템에 전달하는 역할을 하는 로그스태시의 핵심 기능
- 자주 사용하는 입력 플러그인
  - file: 리눅스의 tail -f 명령처럼 파일을 스트리밍하여 이벤트를 읽어 들인다.
  - syslog: 네트워크를 통해 전달되는 시스로그를 수신한다.
  - kafka: 카프카의 토픽에서 데이터를 읽어 들인다.

- jdbc: JDBC 라이브러리로 지정한 일정마다 쿼리를 실행해 결과를 읽어 들인다.

- sincedb 데이터베이스 파일

파일이 마지막으로 읽힌 오프셋값을 알 수 있다.

  - 파일을 어디까지 읽었는지 기록하는 파일
  - beginning → 파일의 처음부터 읽어온다.
  - end → 파일의 끝에서부터 읽어온다.



- 자주 사용되는 필터 플러그인
  - gork: gork패턴을 사용해 메시지를 구조화된 형태로 분석한다.
  - dissect: 간단한 패턴을 사용해 메시지를 구조화된 형태로 분석한다. 정규식을 사용하지 않아 gork에 비해 자유도는 조금 떨어지지만 더 빠른 처리가 가능하다.
  - mutate: 필드명을 변경하거나 문자열 처리 등 일반적인 가공 함수들을 제공한다.
  - date: 문자열을 지정한 패턴의 날짜형으로 분석한다.

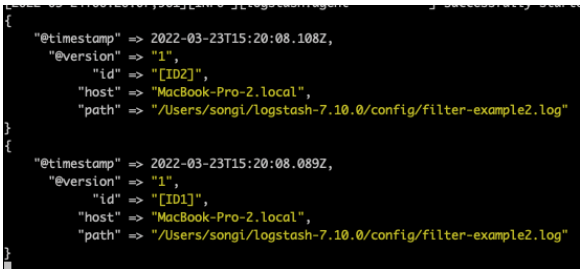
- 문자열 자르기

```
input {
  file {
    path => "/Users/songi/logstash-7.10.0/config/filter-example.log"
    start_position => "beginning"
    sincedb_path => "nul"
  }
}
filter {
  mutate {
    split => { "message" => " " }
  }
}

output {
  stdout { }
}
```

  - mutate 옵션
    - split : 쉼표 같은 구분 문자를 기준으로 문자열을 배열로 나눈다.

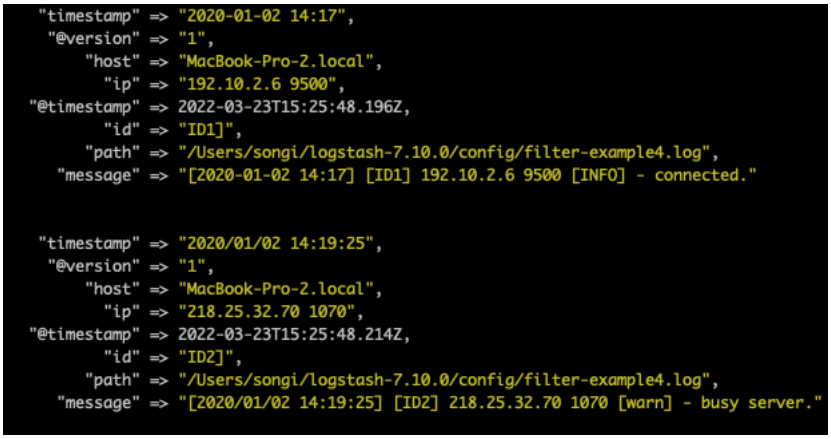
    - rename: 필드 이름을 바꾼다.
    - replace: 해당 필드값을 특정 값으로 바꾼다.
    - uppercase: 문자를 대문자로 변경한다.
    - lowercase: 문자를 소문자로 변경한다.
    - join: 배열을 쉼표같은 구분 문자로 연결해 하나의 문자열로 합친다.
    - gsub: 정규식이 일치하는 항목을 다른 문자열로 대체한다.
    - merge: 특정 필드를 다른 필드에 포함시킨다.
    - coerce: null인 필드값에 기본값을 넣어준다.
    - strip: 필드 값의 좌우 공백을 제거한다.
    - coerce → rename → update → replace → convert → gsub → upppercase → capitalize → lowercase → strip → remove → split → join → merge → copy 순으로 옵션이 적용된다.
- 필터 플러그인 공통 옵션




```
filter {
  mutate {
    split => { "message" => " " }
    add_field => {"id" => "%{[message][2]}"}
    remove_field => "message"
  }
}
```

    - add_field: 새로운 필드를 추가할 수 있다.
    - add_tag: 성공한 이벤트에 태그를 추가할 수 있다.
    - enable_metric: 메트릭 로깅을 활성화하거나 비활성화할 수 있다. 기본적으로 활성화되어 있고, 수집된 데이터는 로그스태시 모니터링에서 해당 필터의 성능을 분석할 때 사용된다.
    - id: 플러그인의 아이디를 설정한다. 모니터링 시 아이디를 이용해 특정 플러그인을 쉽게 찾을 수 있다.
    - remove_field: 필드를 삭제할 수 있다.
    - remove_tag: 성공한 이벤트에 붙은 태그를 제거할 수 있다.
- 문자열 파싱 - dissect

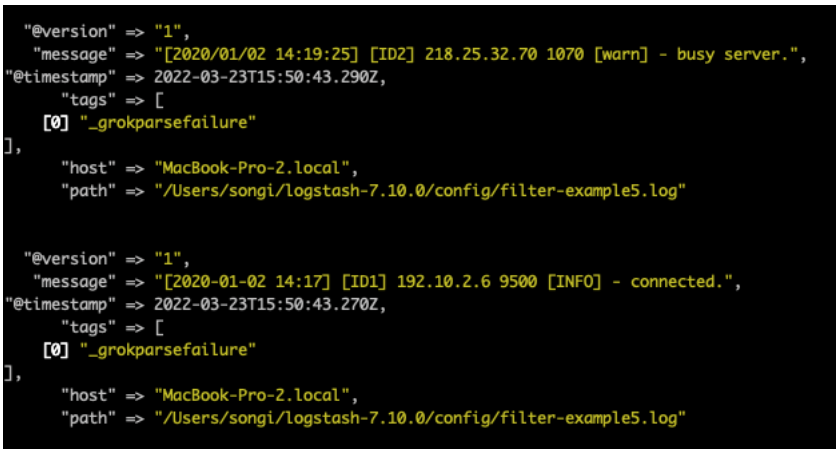
```
filter {
  dissect {
    mapping => { "message" => "[%{timestamp}] [%{id} %{ip} %{port} [%{level}] - %{message}."}
  }
}
```



message 형식

- dissect 플러그인에서는 공백 한 칸과 세 칸을 다르게 인식한다. → `%{?→}` 로 입력하면 공백들을 하나의 필드로 인식하게 만든 후 무시한다.
- `%{+필드명}` 여러 개의 필드를 하나의 필드로 합쳐서 표현한다.

- grok 를 이용한 문자열 파싱
  - 정규 표현식을 이용해 문자열을 파싱한다.




```
filter {
  grok {
    match => { "message" => "\[%{TIMESTAMP_ISO8601:timestamp}\] [ ]*\[%{DATA:id}\] %{IP:ip} %{NUMBER:port:int} \[%{LOGLEVEL:level}\
  }
}
```

  - %{패턴명:변수명}
  - grok 패턴
    - NUMBER: 십진수를 인식, 부호와 소수점을 포함할 수 있다.
    - SPACE: 스페이스, 탭 등 하나 이상의 공백을 인식한다.
    - URI: URI를 인식한다. 프로토콜, 인증 정보, 호스트, 경로, 파라미터를 포함할 수 있다.
    - IP: IP주소를 인식한다. IPv4, IPv6 모두 인식.
    - SYSLOGBASE: 시스로그의 일반적인 포맷에서 타임스탬프, 중요도, 호스트, 프로세스 정보까지 메시지외의 헤더 부분을 인식한다.

    - TIMESTAMP_ISO8601: ISO8601 포맷의 타임스탬프를 인식한다. 타임존까지 정확한 정보를 기록한다.
    - DATA: 직전 패턴부터 다음 패턴 사이를 모두 인식한다. 특별히 인식하고자 하는 값의 유형을 신경 쓸 필요가 없으니 특별히 값이 검증될 필요가 없다면 가장 많이 쓰이는 패턴 중 하나다.
    - GREEDYDATA: DATA 타입과 동일, 표현식의 가장 뒤에 위치시킬 경우 해당 위치로부터 이벤트의 끝까지를 값으로 인식한다.
- 대소문자 변경

```
filter {
  dissect {
    mapping => { "message" => "[%{?timestamp}]%{?->}[%{?id}] %{?id} %{?port} [%{level}] - %{?msg}."}
  }
  mutate {
    uppercase => ["level"]
  }
}
```

- 날짜/시간 문자열 분석

```
filter {
  dissect {
    mapping => { "message" => "[%{?timestamp}]%{?->}[%{?id}] %{?id} %{?port} [%{level}] - %{?msg}."}
  }
  mutate {
    stript => "timestamp"
  }
  date {
    match => ["timestamp", "YYYY-MM-dd HH:mm", "yyyy/MM/dd HH:mm:ss"]
    target => "new_timestamp"
    timezone => "UTC"
  }
}
```

- 조건문

```
filter {
  dissect {
    mapping => { "message" => "[%{?timestamp}]%{?->}[%{?id}] %{?id} %{?port} [%{level}] - %{?msg}."}
  }
  if [level] == "INFO" {
    drop { }
  }
  else if [level] == "warn" {
    mutate {
      remove_field => [ "ip", "port", "timestamp", "level"]
    }
  }
}
```

- 출력 플러그인
    - elasticsearch: 가장많이 사용, bulk API를 사용해 엘라스틱 서치에 인덱싱을 수행한다.
    - file: 지정한 파일의 새로운 줄에 데이터를 기록한다.
    - kafka:카프카 토픽에 데이터를 기록한다.
- 엘라스틱서치 플러그인 옵션
    - hosts: 이벤트를 전송할 엘라스틱서치의 주소
    - index: 이벤트를 인덱싱할 대상 인덱스
    - document_id: 인덱싱될 문서의 아이디를 직접 지정할 수 있는 옵션
    - user/password: 엘라스틱서치에 보안 기능이 활성화되어 있을 때 인증을 위한 사용자 이름과 비밀번호
    - pipeline: 엘라스틱서치에 등록된 인제스트 파이프라인을 활용하기 위한 옵션

- template,template_name: 로그스태시에서 기본 제공되는 인덱스 템플릿 외에 커스텀 템플릿을 사용하기 위한 옵션. template에는 정의한 인덱스 템플릿 파일의 경로, template_name에는 엘라스틱서치에 어떤 이름으로 등록할지를 설정한다.

- 코덱
  - 독립적으로 동작하지 않고 입력, 출력 과정에 사용되는 플러그인
  - 입/출력시 메시지를 적절한 형태로 변환하는 스트림 필터
  - 코덱 플러그인
    - json: 입력시 JSON 형태의 메시지를 객체로 읽어 들인다. 출력시에 이벤트 객체를 다시 JSON 형태로 변환한다.
    - plain: 메시지를 단순 문자열로 읽어들인다. 출력시 원하는 포맷 지정 가능
    - rubydebug: 로그스태시의 설정을 테스트하거나 예기치 못한 파이프라인 설정 오류를 디버깅하기 위한 목적으로 주로 사용되며 출력시 Ruby언어 해시 형태로 이벤트를 기록한다. 입력시엔 사용되지 않는다.

6.4 다중 파이프라인

- 하나의 파이프라인으로 처리할 수 있지만, 모니터링이 어렵고 관리가 어렵다.
  - if,else 조건문 많아지므로 지저분해진다.
- config/pipelines.yml

```
- pipeline.id: mypipe1
  path.config: "/logstash-7.10.0/config/mypipe1.conf"
- pipeline.id: mypipe2
  path.config: "/logstash-7.10.0/config/mypipe2.conf"
```

  - pipeline.id: 파이프라인의 고유한 아이디
  - path.config: 파이프라인 설정 파일의 위치
  - pipeline.workers: 필터와 출력을 병렬로 처리하기 위한 워커 수. 기본적으로 호스트의 CPU 코어수와 동일하게 설정된다
  - pipeline.batch.size: 입력 시 하나의 워커당 최대 몇 개 까지의 이벤트를 동시에 처리할지를 결정한다. 배치 처리가 된 이벤트들은 엘라스틱서치 출력에서 하나의 벌크 요청으로 묶이기 때문에, 이 수치가 클수록 요청 수행 횟수가 줄어들어 인덱싱 성능 개선 효과가 있지만, 그만큼 단일 요청이 커지므로 1000,2000과 같이 적당히 조절해가며 튜닝할 필요가 있다.
  - queue.type: 파이프라인에서 사용할 큐의 종류를 정할 수 있다. memory타입이 기본이고 persisted 타입을 선택하면 이벤트 유실을 최소화할 수 있다.

6.5 모니터링

1. 로그스태시가 제공하는 API를 활용해 특정 시점의 통계 정보를 얻는 방법
   - 노드정보: curl -XGET “localhost:9600/_node?pretty”
   - 플러그인 정보: curl -XGET ‘localhost:9600_node/plugins?pretty’
   - 노드 통계 정보: curl -XGET ‘localhost:9600_node/stats?pretty’
   - 핫 스레드 정보: curl -XGET ‘localhost:9600_node/hot_threads?pretty’
   - 타입별 모니터링 노드 통계 API
     - jvm: 스레드, 메모리 사용량, GC등 JVM 사용 통계
     - process: 사용중인 파일 디스크립터 수= 열어둔 파일 수와 메모리, CPU 사용량 등 프로세스의 사용통계
     - events: 실행중인 로그스태시에 인입된, 필터링된, 출력된 총 이벤트의 수와 이를 처리하기 위해 소요된 시간 등의 통계
     - pipeplines: 파이프라인과 그 하위에 구성된 플러그인별 이벤트 통계
     - reloads: 로그스태시에서 자동/수동으로 설정을 리로드했을 때 성공/실패 수 통계
     - os: 로그스태시가 도커와 같은 컨테이너에서 실행될 때 cgroup에 대한 통계

2. 모니터링 기능을 활성화해서 지속적인 통계 정보를 수집하고 키바나를 통해 대시보드 형태로 모니터링을수행하는 방법