6.3 Caso de Uso - Apache

En este caso de uso se va a solucionar un problema que apareció al final de la práctica de Filebeats, en el que se vio que los logs que se estaban generando terminaban en el campo `message` pero sin parsear, todos los datos en una única línea sin que puedan aprovecharse y filtrarse.

En esta práctica se buscará enviar a Logstash estos eventos en vez de a Elasticsearch, y mostrar la potencia y el verdadero objetivo de Logstash **parseando** estos eventos antes de indexarlos.

Cambio en Filebeat

La primera modificación es cambiar el *output* que se configuró en Filebeats que enviaba los eventos a Elasticsearch y ahora deberá apuntar a Logstash. Será necesario *comentar* la salida de Elasticsearch y descomentar las de Logstash añadiendo la siguiente configuración:

```
output.logstash:
  hosts: ["192.168.1.30:5044"]
```

Configuración de Logstash

Sería una buena práctica, antes de redirigir los eventos directamente a Elasticsearch, imprimirlos por pantalla hasta que veamos que se encuentran correctamente parseados y así evitar tener que estar indexando y borrando datos continuamente tras cada prueba. Para ello, podemos generar un archivo de parseo de Logstash simple que escuche lo que le llega y lo imprima por la salida estándar, `vi /etc/logstash/conf.d/apache.conf` :

```
input {
  beats {
    port => 5044
  }
}

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

Si se quiere hacer una pequeña prueba para confirmar que todo está conectado correctamente debemos asegurarnos que Filebeat se encuentra funcionando y solo faltaría arrancar Logstash y generar algún evento con los siguientes comandos:

```
/usr/share/logstash/bin/logstash --path.settings /etc/logstash
python apache-fake-log-gen.py -n 1 -o LOG
```

COMBINEDAPACHELOG

Conectado todo correctamente, es hora de ponerse manos a la obra con el procesamiento en Logstash. Para un formato de Logs tan utilizado hoy día y conocido como son los Logs de Apache, Logstash ya dispone de un template para parsear estos logs y solo será necesario indicarle que el campo `message` debe hacer *match* con este formato, modificaremos el `/etc/logstash/conf.d/apache.conf` de la siguiente forma:

```
beats {
    port => 5044
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

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

Dicho formato se logs se puede encontrar en:

```
vi /usr/share/logstash/vendor/bundle/jruby/2.3.0/gems/logstash-patterns-core-4.1.2
/patterns/httpd
```

De la misma forma que se ha hecho antes, se volverá a arrancar el productor de logs y Logstash para comprobar el correcto funcionamiento.

Redirección a Elasticsearch

Una vez se comprueba que el campo se está parseando correctamente, se pueden redirigir los eventos a Elasticsearch. Para ello solo será necesario añadir el plugin de salida en nuestro archivo de configuración `apache.conf`:

```
input {
  beats {
    port => 5044
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch { hosts => ["192.168.1.31:9200"] }
  stdout { codec => rubydebug }
}
```

El único problema es que esta información comienza a indexarse en el índice `logstash-*` y probablemente, la mayoría de los casos se trabajará con más tipos de información y querremos organizar cada tipo de información en su correspondiente índice. Para ello añadimos el parámetro dentro del Plugin Elasticsearch de salida y se le indica el índice donde se quiere almacenar:

```
input {
  beats {
    port => 5044
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch {
    hosts => ["192.168.1.31:9200"]
    index => "apache-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug }
}
```

Enriquecimiento del flujo con Geolocalización

Como se ha comentado en este módulo, uno de los mayores objetivos de Logstash además de preprocesar la información que por él pasa, será enriquecer la misma. Por ejemplo, en este caso aprovechando que aparecen IPs públicas podemos añadir a los eventos la geolocalización de la misma, lo que nos aportará más visibilidad sobre qué países son los que están accediendo a los recursos:

```
input {
  beats {
    port => 5044
  }
}

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
  geoip {
       source => "clientip"
  }
}

output {
  elasticsearch {
    hosts => ["192.168.1.31:9200"]
    index => "apache-%{+YYYY.MM.dd}"
  }
  # stdout { codec => rubydebug }
}
```