El trio ELK - ElasticSearch + Logstash + Kibana

viernes, 15 de mayo de 2015 22:12

ELK

ElasticSearch

'You know, for search'

Logstash

Es una herramienta open source que nos va a permitir recolectar, parsear y almacenar logs en herramientas como ElasticSearch.

Kibana

Kibana es una interfaz web que puede ser utilizada para buscar y ver los logs que logstash ha indexado.

Ejercicio - Haciendo uso de ELK para dar sentido a los logs

Introducción

En este ejercicio, vamos a configurar el trío ELK (ElasticSearch, Logstash y Kibana) sobre un entorno Ubuntu 14.04 (aunque instalarlo en Windows es igualmente sencillo). Nuestro objetivo es configurar Logstash para recopilar logs que pondremos en un determinado folder, indexar esa información en ES y hacer uso de Kibana para visualizar la información proporcionada.

Para todo ello, haremos uso del siguiente software:

- ElasticSearch - donde almacenaremos e indexaremos toda la información.
- Logstash 1.5.0 - encargado de procesar los logs entrantes y de enviarlos a elasticsearch
- Kibana 4.0 - la interfaz web que nos permitirá buscar y visualizar logs.

Al finalizar el ejercicio habremos comprendido cómo instalar, configurar, recopilar información de los logs, indexarlos en ElasticSearch y visualizarlos en un entorno web como Kibana. Esta configuración es muy útil, puesto que nos proporciona un entorno de logs centralizado que puede resultar muy útil cuando se está intentando comprender qué posibles problemas pueden tener las aplicaciones o servicios que estamos monitorizando. Podremos buscar sobre montones de logs haciendo uso del poder de ElasticSearch y de la facilidad visual que presenta Kibana como interfaz web de acceso a la información indexada.

Pre-requisitos

- **Ubuntu 14.04**
- **Java 1.7 o superior**
  Verificamos que tenemos java instalado en nuestra máquina

  ```
  $ java -version
  ```

  En caso de no estar instalado debemos ejecutar los siguientes comandos con permisos de administrador:

  ```
  $ sudo add-apt-repository -y ppa:webupd8team/java
  $ sudo apt-get update
  $ sudo apt-get -y install oracle-java8-installer
  ```

- Configuramos el sistema de directorios

  ```
  /ciff
  /ciff/data
  /ciff/exercises
  /ciff/runtime
  /ciff/software
  /ciff/tarballs

  $ mkdir ./ciff; mkdir ./ciff/data; mkdir ./ciff/exercises; mkdir ./ciff/runtime;
  mkdir ./ciff/software; mkdir ./ciff/tarballs
  ```

- **ElasticSearch software**

  ```
  $ cd ciff
  $ cd tarballs
  $ wget https://download.elasticsearch.org/elasticsearch/elasticsearch/elasticsearch-1.5.1.deb
  $ sudo dpkg -i elasticsearch-1.5.1.deb
  $ sudo vi /etc/elasticsearch/elasticsearch.yml

      network.host: localhost
      cluster.name: "cmdb.elasticsearch"
      node.name: "cmdb"

  $ sudo service elasticsearch restart
  ```

- **Kibana software**

  Descargamos el software necesario

  ```
  $ cd
  $ cd ciff
  ```

```
$ cd tarballs
$ wget https://download.elastic.co/kibana/kibana/kibana-4.0.0-linux-x86.tar.gz
```

Lo descomprimimos en el folder adecuado

```
$ tar xvf kibana-*.tar.gz
$ mv kibana-4.0.0-linux-x86 ../software/
$ cd ../runtime
```

Y creamos los link simbólicos necesarios

```
$ ln -s /home/adminuser/ciff/software/kibana-4.0.0-linux-x86/ kibana
```

Configuramos el servicio para que el host sea nuestra máquina local

```
$ vi ~/kibana-4*/config/kibana.yml

    host: "localhost"
```

Y lo ejecutamos…

```
$ cd kibana
$ cd bin
$ ./kibana
```

Verificando que el servicio está escuchando en la siguiente URL
http://localhost:5601/

- **Logstash software**

Descargamos el software necesario

```
$ cd
$ cd ciff
$ cd tarballs
$ wget http://download.elastic.co/logstash/logstash/logstash-1.5.0.tar.gz
```

Lo descomprimimos en el folder adecuado

```
$ tar xvf logstash-*.tar.gz
$ mv logstash-1.5.0 ../software/
$ cd ../runtime
```

Y creamos los link simbólicos necesarios

```
$ ln -s /home/adminuser/ciff/software/logstash-1.5.0/ logstash
```

Los logs de IIS

Para el ejercicio, vamos a analizar un conjunto de tres ficheros de logs que han sido generados por un servicio de IIS en una máquina cualquiera. El formato y los campos que tiene estos ficheros de logs son los siguientes:

#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) sc-status sc-substatus sc-win32-status time-taken

Logstash

¿Cómo podemos arrancar logstash y procesar información?

Podemos verificar que logstash está funcionando correctamente ejecutando la siguiente línea de comando:

```
$ cd logstash-{logstash_version}
$ ./logstash -e 'input { stdin { } } output { stdout {} }'
```

El cursor quedará esperando nuestro input. Si tecleamos

```
Esto es un test
```

Recibiremos una respuesta de logstash similar a esta:

```
2015-05-15T22:01:39.867Z adminuser-VirtualBox Esto es un test
```

Logstash ha sido configurado con un input de línea de comando, y le hemos peidido que únicamente redireccione la información a la salida estándar.

Podemos ejecutar ahora este otro comando y obtener el mismo comportamiento pero con un formato de visualización mejorado:

```
$ ./logstash -e 'input { stdin { } } output { stdout { codec => rubydebug } }'
Esto es un test

{
       "message" => "Esto es un test",
      "@version" => "1",
    "@timestamp" => "2015-05-15T22:05:09.755Z",
          "host" => "adminuser-VirtualBox"
}
```

En este caso, logstash ha trabajado un poco mejor la respuesta y ahora nos entrega un documento en formato "JSON" con información adicional sobre el mensaje que ha procesado.

Si seguimos jugando con el output, podemos ahora enviar la información que logstash procese desde la línea de comando directamente a un índice y un tipo de ElasticSearch. Para ello, nos aseguramos primero que nuestra instancia de ElasticSearch está arrancada y procedemos a arrancar logstash haciendo uso de la siguiente línea de comando

```
$ ./logstash -e 'input { stdin { } } output { elasticsearch { host => localhost } }'
Esto es un test
```

En este otro caso, logstash ha redireccionado la salida a un elasticsearch que está escuchando en la IP 127.0.0.1. Si ahora hacemos una búsqueda sobre cualquier índice en ese elasticsearch, deberíamos encontrar el documento con el contenido que hemos posteado

```
$ curl 'http://localhost:9200/_search?pretty'

{
      "_index" : "logstash-2015.05.15",
      "_type" : "logs",
      "_id" : "AU1ZohnDZwGDwUDTCrAd",
      "_score" : 1.0,
      "_source":{"message":"Esto es un test",
         "@version":"1",
         "@timestamp":"2015-05-15T22:11:44.840Z","host":"adminuser-VirtualBox"}
}
```

Es muy interesante fijarse en cómo ha indexado el documento ElasticSearch. El proceso, ha creado un índice nuevo llamado logstash-2015.05.15, con un tipo llamado logs. Es muy importante este dato para poder trabajar posteriormente con Kibana puesto que las métricas de tiempo y los filtrados los llevará a cabo por estas fechas en los nombres de los índices.

Si quisiéramos más información sobre lo que logstash está haciendo durante su ejecución podemos solicitar un poco más de información de log agregando --verbose a la línea de comando o con --debug si queremos información de depuración en el standard output.

```
$ ./logstash -e 'input { stdin { } } output { elasticsearch { host => localhost } }' --debug
```

Más información sobre las opciones de líneas de comando de logstash aquí.

¿Cómo configuramos logstash?

Obviamente, no podemos hacer uso de la línea de comando cuando necesitamos configurar logstash de una forma más complicada. Loogstash permite suministrarle las opciones de configuración mediante el uso de un fichero externo. Para ello,

- Creamos el fichero de configuración con el siguiente formato:

```
$ cd ~/ciff/runtime/logstash/bin
$ nano logstash.config

    input {
        stdin { }
    }
    output {
        elasticsearch {
            host => localhost
        }
    }
```

- Una vez creado, arrancamos logstash haciendo uso de la siguiente línea de comando:

```
$ ./logstash --config logstash.config --debug
```

El efecto de la configuración que hemos suministrado a logstash es el mismo que cuando lo hicimos desde la línea de comando. Pero en este caso, tenemos la opción de controlar más cómodamente la información que suministramos al procesador de logs.

Qué estructura tiene el fichero de configuración de logstash

```
input {
    # Zona donde haremos uso de los diferentes inputs que permite logstash
}
filter {
    # Zona donde podemos aplicar transformaciones a los datos enviados por cada input
}
output {
    # Zona donde publicaremos cada evento procesado en el destino seleccionado
}
```

Inputs

Logstash utiliza los inputs para procesar el formato de entrada que hayamos seleccionado. En nuestro caso, vamos a procesar ficheros de texto que contienen información de logs de un servidor IIS. El input plugin tipo *file* será el que utilicemos para procesar la información que necesitamos.

```
input {
     file {
          path => "/home/adminuser/ciff/logs"
          type => "mislogs"
          start_position => "beginning"
          exclude => "*.gz"
     }
}
```

Estamos explicando a logstash que debe buscar ficheros en el path /home/adminuser/ciff/logs que no tengan la extensión .gz. Que para cualquiera de los ficheros encontrados empiece siempre a analizarlos desde el principio y que agrege, para cada línea procesada, un tag llamado "mislogs". Este tag podrmos utilizarlo despues para filtrar y procesar la información que de aquí nos llegue.

File es uno de los múltiples plugins de entrada que permite logstash. En este enlace, podéis encontrar todos y cada uno de los inputs que permite configurar logstash por defecto. Podéis agregar muchos más custom bajándolos desde su repositorio git o haciéndolos vosotros mismos trabajando en ruby.

Outputs

Logstash utiliza los plugins de output para decidir donde envía cada uno de los tokens que le llegan desde los plugin de input. Tal y como estamos definiendo el proceso ahora, logstash leerá línea a línea los ficheros que tengamos bajo el folder *logs* y los enviará al output que vamos a definir en el siguiente párrafo:

```
output {
     elasticsearch {
          host => localhost
          type => "stdin-type"
          embedded => false
          port => "9200"
          cluster => "tu-cluster-name"
          node_name => "tu-node-name"
     }
}
```

Pongamos lo que tenemos todo junto

```
$ cd ~/ciff/runtime/logstash/bin
$ nano logstash.config

     input {
             file {
                     path => "/home/adminuser/ciff/logs/*"
                     type => "mislogs"
                     start_position => "beginning"
                     exclude => "*.gz"
             }
     }
     output {
             elasticsearch {
                     host => localhost
             }
     }

$ ./logstash --config logstash.config --debug
```

Si dejamos que termine el proceso (podemos pararlo antes con un CTRL+C), logstash habrá procesado los logs debajo del folder indicado y enviado la información a nuestro elasticsearch local. Podemos ver el resultado haciendo un:

```
$ curl -XGET "http://localhost:9200/_search"
O
$ curl -XGET "http://localhost:9200/logstash-*/_search"

     {
        "_index": "logstash-2015.05.16",
        "_type": "mislogs",
        "_id": "AU1c_yKiZm3v_n-kPT8l",
        "_score": 1,
        "_source": {
           "message": "2014-03-31 07:39:01 10.114.22.69 GET / - 81 - 10.114.22.69 Mozilla/5.0
+(compatible;+MSIE+9.0;+Windows+NT+6.1;+WOW64;+Trident/5.0;+managedpc) 200 0 0 173\r",
          "@version": "1",
          "@timestamp": "2015-05-16T13:52:00.460Z",
          "type": "mislogs",
          "host": "adminuser-VirtualBox",
          "path": "/home/adminuser/ciff/logs/u_ex140331.log"
```

```
            }
        }
```

ElasticSearch contiene ahora la información de los logs indexada en un índice llamado logstash-{fecha}. El problema es que toda la información ha sido volcada en el campo message sin haber sido dividida y parseada para una mejor utilización.

Filter

El fitro es el módulo de logstash que nos permite analizar nuestros datos en crudo que provienen del input y hacer algo que tenga sentido para nuestra indexación. Logstash tiene un montón de filtros que podemos aplicar para transformar la información. Uno de los más útiles es grok. Grok permite parsear la información que proviene del input haciendo uso de expresiones regulares y patrones.

Si recordamos el formato de los logs que estamos parseando:

```
date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent) sc-status
sc-substatus sc-win32-status time-taken
```

Podemos buscar un filtro grok que nos perrmita trocear esta información de una forma coherente. Cuando hagamos filtros grok, podemos hacer uso de esta página web para verificar que estamos haciendo lo correcto:

https://grokdebug.herokuapp.com/

%{DATESTAMP:timestamp} %{IPORHOST:clientip} (?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest}) %{IPORHOST:client2ip} %{IPORHOST:client}

```
$ curl -XDELETE "http://localhost:9200/_all"
$ cd ~/ciff/runtime/logstash/bin
$ nano logstash.config

input {
        file {
                path => "/home/adminuser/ciff/logs/*"
                type => "mislogs"
                start_position => "beginning"
                exclude => "*.gz"
        }
}
filter {
        grok {
                match => [ "message", "%{DATESTAMP:timestamp} %{IPORHOST:clientip}
(?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpver$
        }
}
output {
        elasticsearch {
                host => localhost
        }
}
$ ./logstash --config logstash.config --debug
```

Si ahora ejecutamos un _search contra elasticsearch, podremos observar que el documento que se ha almacenado ahora tiene mejor pinta:

```
$ curl -XGET "http://localhost:9200/_all/_search"
{
        "_index": "logstash-2015.05.16",
        "_type": "mislogs",
        "_id": "AU1dE-1kKA2surtV_Jar",
        "_score": 1,
        "_source": {
           "message": "2014-03-30 16:04:01 10.130.40.113 GET /config.js - 81 - 10.130.40.113
Mozilla/5.0+(compatible;+MSIE+9.0;+Windows+NT+6.1;+WOW64;+Trident/5.0;+managedpc) 200 0 0 2\r",
           "@version": "1",
           "@timestamp": "2015-05-16T14:14:42.847Z",
           "type": "mislogs",
           "host": "adminuser-VirtualBox",
           "path": "/home/adminuser/ciff/logs/u_ex140330.log",
           "timestamp": "14-03-30 16:04:01",
           "clientip": "10.130.40.113",
           "rawrequest": "GET /config.js - 81 -",
           "client2ip": "10.130.40.113",
           "client": "Mozilla"
      }
}
```

Alguno de los filtros más interesantes que podréis encontrar en logstash son:

- **grok**: parsea y estructura cualquier tipo de texto. Es la mejor manera en logstash de parsear logs no estructurados en algo estructurado e indexable por elasticsearch.

- **mutate**: ejecuta transformaciones de cualquier tipo en campos del mensaje: eliminar, modificar, renombrar, reemplazar campos que entran por campos que salen.
- **drop**: permite eliminar un evento entrante por completo. Útil cuando los eventos han de cumplir determinadas condiciones.
- **clone**: crea una copia exacta del evento.
- **geoip**: agrega información geolocalizada por IP.

Expresiones condicionales

```
input {
        file {
                path => "/home/adminuser/ciff/logs/*"
                type => "mislogs"
                start_position => "beginning"
                exclude => "*.gz"
        }
}
filter {
        grok {
                match => [ "message", "%{DATESTAMP:timestamp} %{IPORHOST:clientip} (?:%{WORD:verb}
%{NOTSPACE:request}(?: HTTP/%{NUMBER:httpver$
         }
         if [type] == "mislogs" {
             mutate { remove => "secret" }
         }
}
output {
        elasticsearch {
                host => localhost
        }
}
```

Lectura recomendada

Inputs -
https://www.elastic.co/guide/en/logstash/current/input-plugins.html
Outputs -
https://www.elastic.co/guide/en/logstash/current/output-plugins.html