Design and simulation of an ELK-based logging infrastructure

Josep Brugués i Pujolràs

Abstract

En un món on cada vegada els serveis al *cloud*, Internet, els sistemes automatizats, la Intel·ligència Artificial, els supercomputadors, etc. són una peça cada vegada més fonamental per a la vida dels éssers humans, es fa necessari tenir una eina capaç d'analitzar tota l'activitat registrada en els computadors. Cal un sistema de gestió de *logs*.

Paraules clau Sistema de gestio de logs, Elasticsearch, Kibana, Logstash, ELK, Python

Abstract

In a world where cloud services, Internet, automatized systems, Artificial Intelligence, supercomputers, etc. play a big role in the life of the human species, there is the necessity of having a tool capable of analyzing all the activity registered on these computers. There is the need of a log management system.

Paraules clau Log Management System, Elasticsearch, Kibana, Logstash, ELK, Python

---

I. INTRODUCTION

124,689 *petabytes*. That is the quantity of information that Cisco has predicted[1] will be sent through Internet only this year. Services based on the *cloud* are being used more and more, and this has unimaginable consequences on both processing and storage. On the corporate level, this creates the necessity of having an infrastructure capable of giving service to all its users in real time. In 2017 only, Google processed almost 1,200 trillions of searches, equivalent to 10 *exabytes* of information.

The increase stated above on the *cloud* services, together with the boom of *big data* in the latest years, has provoked the rise on the number of logs –documentation produced automatically on each relevant event on a particular system— produced by the servers. These logs include information on each operation that has been done in the system[1] and they allow us to understand how complex systems work, and that makes their analysis very

• Contact email: bruguesjosep@outlook.com
• Bachelor's Degree in Computer Engineering (Specialization in Information Technologies)
• Project tutored by Angel Elbaz
• Course Year 2017-2018

[1]Logs contain information on whether the operation has been completed correctly or, on the contrary, there has been any error. Additionally, they also include information about the operation: *timestamp*, message, client, etc.

important. There's only one problem: How can we –as a medium-size company or a corporation—, analyze that enormous quantity of logs? How can we analyze them in an organized, fast and effective way in order to detect errors or mis-configurations? With a **log management system**.

A log management system gathers up all the logs generated in the different components of the system –DNS servers, web servers, mail servers or the system itself— in order to analyze them, extract information and display that information in a simple –and understandable— way. Furthermore, another important aspect is that log management systems allow the correlation of information from various sources so that new information can be obtained, which could not be obtained from any other way or it just would be hidden from us. In that context, one of the most important log management systems is **ELK**[2], acronym for *Elasticsearch[3], Logstash[4]* i *Kibana[5].*

This project has been divided in two big parts: the first stage will be an analysis of the commercial and open source alternatives for a log management system, as well as how they work and how they can be configured. Apart from that, the main focus will be on ELK, specially on the more technical details about it, since it will be the base for the second part, which will be the implementation in Python of my own log management system based on ELK. Furthermore, I will explain the approach a small company with a given network should take in order to implement my management system, taking into account that the log analysis will be centralized in one computer, instead of doing the analysis on the different components of the network.

II. STATE OF THE ART

Log management systems have been an important tool to understand how complex systems work and to find possible errors and mis-configurations. Any management system needs to be able to handle **large amounts of logs**, **normalize the information**, be **fast** and **precise**, as well as to be **secure**. Focusing on ELK, this system is based in 3 **modules** that interact with each other:

- *Logstash* is responsible for reading the logs from different sources and normalize them. That is, give them a common format.
- *Elasticsearch* is the log indexer, search engine and analysis tool that lets the user to perform searches and analyze big quantities of data in a fast and efficient way.
- *Kibana* is the tool that visualizes the information through a graphical interface. To do so, Kibana performs searches with the API provided by Elasticsearch.

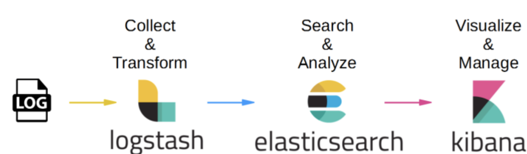

Fig. 1. Interaction between ELK modules

Some log management systems, including ELK, are designed with modularity in mind, and they allow the user to expand capabilities with **plug-ins**. Those plug-ins can be used in any of the 3 modules, and can be either created by the ELK developers or the community.

Over the past few years, and with the momentum gained by **Machine Learning**, companies have brought log management to the next level, being able to predict, for example, system performance and usage based on previous statistics. That enables companies to improve their scalability and reduce costs. It also allows the prediction of anomalies in the system by making a correspondence between the current system status and events from the past.

ELK is not the only system out there, with lots of alternatives –opensource and commercial— available for both particulars and companies:

- **Opensource:** Graylog & Grafana
- **Commercial:** Scalyr & Splunk

All of them are also based on three modules like ELK, being the only difference the extra

services that offer, such as cloud analysis and custom support for companies.

III. METHODOLOGY AND DESIGN

The implementation of my own log management system has been based on the structure and methodology used in ELK, and has been written in Python. Despite ELK being open source and its modules available as packages in Python, I decided not to use those packages. Instead, I made the choice to start my implementation from scratch based on ELK but without using parts of their implementation, not only to have more freedom in the design process but also for learning purposes. In general terms, my implementation will comply with all the objectives a log management system has to do, although it will not implement any Machine Learning characteristics.

In terms of the design, I took into the account the modularity of ELK, and despite my design not supporting plug-ins, it can be configured in a dynamic way thanks to the use of JSON configuration files, in which you can configure different parameters, as will later be seen in this paper.

A. Logstash

The Logstash module is in charge of both **collecting** the logs from different sources in order to centralize them into one place and **normalize** the different log formats into a common and easy-to-use format. In my implementation, I have chosen JSON as the output format since it is widely supported.

Another important aspect in my design is the **dynamic configuration**, accomplished by using a simple JSON file where the user can introduce the logs that have to be scanned, as well as the directory where those logs can be found. In Fig. 4 in the appendix the functionality of Logstash's dynamic configuration can be seen.

In terms of speed and reliability, I have designed Logstash to take advantage of multi-thread architectures, as each type of logs are normalized in different threads. This ensures that the normalization takes places almost in **real-time**.

For a generic type of log, this is the algorithm used in the normalization process:

Algorithm 1 Logstash Normalizer

procedure NORMALIZER($directory$)
  $normalizedLogs \leftarrow newDict$
  $logs \leftarrow logFile$
  for log in $logs$ **do**
    $parsedLog \leftarrow parse$(log)
    $normalizedLogs \leftarrow normalizedLogs$ + $parsedLog$
  copyToFile($normalizedLogs$)

As of now, my implementation supports 3 types of logs: HTTP, DNS and all application's logs that use the *syslog* format. In the first two cases –HTTP and DNS— if used my own algorithm to convert them to JSON format. For the *syslog* logs I have used a third-party open-source parser[2], that is able to identify different *syslog* formats, so that more applications can be supported. In Figures 5,6 & 7 it can be seen the different JSON format used for the 3 types of log.

B. Elasticsearch

The Elasticsearch module performs two core tasks: the **indexation** of the logs, and the implementation of a **Search API**. The two tasks are independent from each other, so any client application can perform queries while the indexation process is taking place.

The main task of the indexer is to put each log into its correct index. The name of each index is a time-stamp, so each index will contain only the logs generated on that particular day. In addition, the indexer has to make sure that any index is not older than 30 days. The process takes place both before and after the indexation task. For more detail on the operation of the indexer task, the algorithm works as follows:

[2]https://gist.github.com/leandrosilva/3651640

Algorithm 2 Indexer Controller

```
1: procedure INDEXCONTROLLER(directory)
2:     while True do
3:         check-if-old-indexes(directory)
4:         index-all-logs(directory)
5:         wait
```

In more detail, we can see how the process of indexing all the logs works:

Algorithm 3 Log Indexer

```
    procedure INDEXER(directory)
2:      while True do                                   ▷ Keep indexing
            timestamps ← timestampsFile
4:          logs ← logFile
            newTimestamps ← newList
6:          timestampMap ← newDict
            newTimestamps ← False
8:          for log in logs do
                logTimestamp ← log[Timestamp]
10:             if logTimestamp is in timestamps then
                    if logTimestamp is in timestampMapKeys then
12:                     data ← indexLogs
                        data ← data + log
14:                     map[timestamp] ← data
                else                                    ▷ Create new Index
16:                 newTimestamps ← True
                    if logTimestamp not in timestampMapKeys then
18:                     data ← map[timestamp] + log
                        map[timestamp] ← data
20:         updateIndexes()
```

The Search API is implemented independently of Kibana's graphic interface, so that the API can be used by any client application.

In its current state, the search API supports 3 types of queries, very similar to those found in SQL:

- **COUNT queries** obtain the number of logs that comply the requirements specified in the query.
- **GROUP queries** group (by a value of a particular field) the logs that comply with requirements specified in the query, so that we can obtain the number of logs for each value of the field selected.
- **GROUP queries** order the logs by a value of a particular field and, at the same time, they obtain information on another field.

The queries follow the following structure:

index type_logs field value [field value] OPERATION [field]

The components of the query mean:

- **Index:** Indexes, separated by a comma, where we want to perform the query. If the search has to be done in all indexes, the keyword **all** can be used.
- **Logs:** Types of logs we want to search. As we seen before, the implementation supports *HTTP, syslog* and *DNS*. If we want to perform the query in all of them, the keyword **all** can be used.
- **Field:** It can be any of the fields of the types of logs that are being searched. For instance, in HTTP, that can be textitclientIP, identity, username, request, status_code, size_response, url and *useragent*.
- **Value:** Value of the field that is being searched. Note that multiple pairs of field-value can be searched in the same query
- **Operation:** Operation that wants to be performed in the query. Currently, the implementation supports **COUNT**, **GROUP** and **ORDER**.

As it has been stated before, the API is independent from the user interface, so it can be used by any client application. In order to do so, the API gives the results in JSON format. I will use a set of examples –in Fig. 8 in the Appendix it can be seen the logs used for this— to show how it works:

For the **count query** "*27052018 http clientIP 11.234.205.99 sizeresponse all COUNT*", the API returns the result "*'http-2': 'clientIP':*

'11.234.205.99', 'sizeresponse': '5034'". As it can be seen, it just returns the logs that match the query –in JSON format– but only with the fields specified by the user. With that information, any application can display the result as they wish.

For the **group query** "*27052018 http clientIP all GROUP*", the API returns the result "*'130.109.141.177': 1, '26.130.80.179': 1, '11.234.205.99': 1, '110.112.114.234': 1, '188.228.2.144': 1, '243.177.243.74': 1, '23.59.179.134': 1, '160.11.63.101': 1, '91.245.28.250': 1, '38.122.225.45': 1*". That is, it counts the different occurrences of each clientIP value found in the HTTP logs. With that information, any client application can, for instance, create a bar plot to visualize the result. Again, the result comes in JSON format, where the keys are the different clientIPs and the values are the number of times they appear.

For the **order query** "*27052018 http clientIP all sizeresponse all ORDER clientIP*", the API returns the result "*'130.109.141.177': 'count': 1, 'sizeresponse': 4992, '26.130.80.179': 'count': 1, 'sizeresponse': 4946, '11.234.205.99': 'count': 1, 'sizeresponse': 5034, '110.112.114.234': 'count': 1, 'sizeresponse': 4941, '188.228.2.144': 'count': 1, 'sizeresponse': 5002, '243.177.243.74': 'count': 1, 'sizeresponse': 5092, '23.59.179.134': 'count': 1, 'sizeresponse': 5050, '160.11.63.101': 'count': 1, 'sizeresponse': 4948, '91.245.28.250': 'count': 1, 'sizeresponse': 4955, '38.122.225.45': 'count': 1, 'sizeresponse': 5072*". That is, it orders –in a descendant way— the logs for the number of time they appear. The keys are the values of the field that has been asked to order in the query, while the values are a dictionary with information on the occurrences and other fields introduced in the query.

C. Kibana

For the development of Kibana's Graphic Interface I have used Tkinter[6] framework –already included in Python's installation— since it is easy to use and includes different layout managers[3] that facilitate the design. Throughout the development, I have used the following:

- **Grid Layout:** This manager distributes the components of the UI in rows and columns. Moreover, for each component, it can specify if the component is going to be small or large. That allows, for example, the obtention of a column larger than the other one. The manager has been used in the main view, formed by 2 rows and 2 columns:
  - **Search Bar:** occupies all the columns in the first row. The bar is used to perform searches using Elasticsearch's API.
  - **Left panel:** occupies the first column of the last row, and the size is set to be small. It shows the user the indexes available for search.
  - **Right panel:** occupies the second column of the last row and, at the same time, it contains a frame that has a grid layout too –1 column and 2 rows—, with 2 components:
    - **Graphic:** occupies the first rows, and displays a graphic with the results of the query.
    - **Log List:** occupies the second row, and displays all the logs that are a match with the query performed.
- **Pack Layout:** This manager enables the internal distribution of the components in their parent and has facilitated the alignment of the components inside their parents.

The resulting UI can be seen in Fig. 9 in the appendix. It is a simple but intuitive and easy-to-use interface.

Finally, I will describe the methodology used in the module to display results to the user, following the 3 examples used in the previous subsection.

[3]Software components that have the ability to control interface's graphic elements in a smart way and without using distance units, such as pixels.

For the **count queries**, I just count the number of elements in the JSON response and display it in a bar plot. The result can be seen in Fig. 10 in the appendix.

For the **group queries**, I have decided to use a bar plot in which the labels in the X axis are the keys in the JSON response –that is the different clientIP values— and the values in the Y axis are the number of appearances.

For the **order queries**, I again user a bar plot, in which the labels in the X axis are the keys in the JSON response, while the left Y axis is the number of occurrences. If there is a second pair of field-value in the query –such as in the example—, I create a second Y axis, on the right, where the labels are dynamic, depending on the values.

The graphics result of these previous 3 examples are in the Appendix as Fig. 10, 11 & 12.

Moreover, in all cases I print all the logs that are a match with the query, so users can have more perspective and additional information.

D. *File and directory structure*

In order to accomplish all the work and to achieve communication between the 3 modules, the project follows a strict folder structure that cannot be changed –if done, the functionality of the whole system would be broken—.

In the following diagram it can be observed the relationship between the modules, while on Fig. 13 in the Appendix there is the whole file and directory scheme:

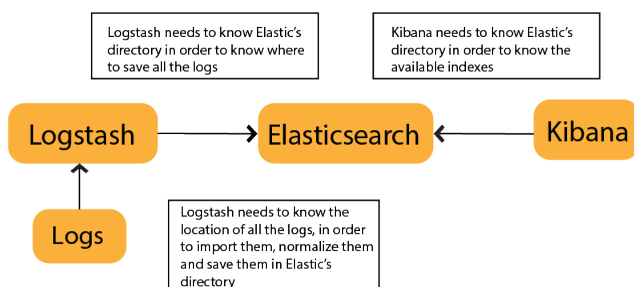

Fig. 2. Relationship between the modules

IV. INSTALLATION AND CONFIGURATION

The installation and configuration of my log management system is quite easy, since it only requires **Python 3.6** and certain packages[4] installed in the machine that will perform all the operations.

Let's suppose that a corporation that has the network from Fig. 2 wants to implement my development in their systems:

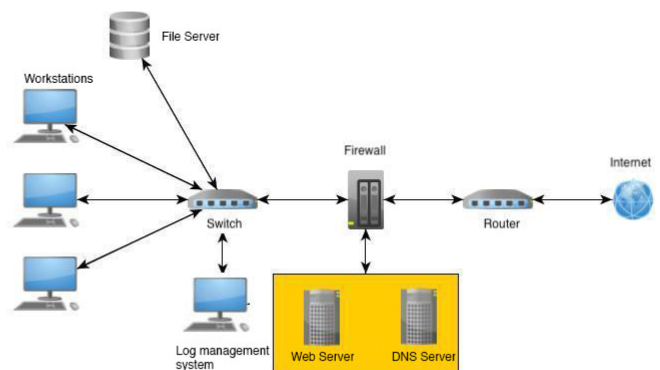

Fig. 3. Network Diagram

In order to centralize all the log management into the machine 'Log Management System' from Fig. 2, we first need to send all the logs from the different components –Web Server, for the HTTP logs; DNS Server, for the DNS logs; Workstations, for the logs from applications using the syslog format— to that computer. In order to do so, we can either configure a **syslog server** using UDP over a VLAN[5] or create a **custom mechanism** to send the logs to the desired machine. If we use the latter, that can be achieved, for instance, using Python[6][4]. That mechanism can be configured as a one-time sender or a permanent sender, so logs can be indexed in a permanent way.

[4]threading, queue, json, pyparsing, time, datetime, os, abc, tkinter, numpy, matplotlib and sys.

[5]The configuration of the VLAN is necessary since the syslog server sends the logs using UDP without any type of encryption. That means anyone could read the logs if a VLAN is not used.

[6]The implementation would be based on sockets. In each machine, we would need to open an UDP connection with the central machine and send all the logs that we read from the log file.

Once we have the log configuration ready, it is time to configure the management system. In order to do so, we need to have the 3 folders with the Logstash, Elasticsearch and Kibana implementations on the desired directory. After that, we need to configure the 3 services with their respective JSON configuration files, and run the services:

For the **Logstash service**, there is only one script to run: '*indexer.py*'. If the configuration is correct, the service will start normalizing the desired logs and send them to the Elasticsearch module.

For the **Elasticsearch service**, again there is only one script to run: '*indexer.py*'. The script will both index all the already normalized logs and keep the indexes maintained –that is, remove the old indexes from the system, as per 30 day policy—. The search API does not need to be run, since it acts as a module that will be imported by my Kibana implementation or any other 3rd party client.

For the **Kibana service**, only '*kibana_ui.py*' need to run. As the name implies, it is just the UI that enables the user to perform queries using the search API.

V. RESULTS AND CONCLUSIONS

First of all, in the examples I have performed along this paper, I have used a **small sample** of 10 logs HTTP, generated on the same day –so that all of them are in the same index—. The number of indexes does not matter in the execution time, it is the number of logs that matter. Despite that, and with current microprocessors, a query takes, at least for the purposes of my project, less than a second. Despite that, if a large number of logs is analyzed –circa 1M logs and beyond—, if multiple queries at once are required, then several instances of the UI are needed, since as of now my implementation of Kibana cannot handle multiple queries.

Secondly, as it can be seen in the examples, the results obtained –and its display in the Kibana interface— are **quite simple**, and no more complex queries are available due to the **limits of the search API**. Despite that, I don't think that the fact the the results are too simple mean that my work is bad, not even close: I think that this will push me into working more in it. Moreover, I don't think that simple results mean useless results, since big part of the job is know what you are looking for.

In the next lines I expose the conclusions to which I have arrived by developing the log management system and analyzing its behavior:

- On the one hand, I think that with this log management system –and taking into account that it has not been developed by a full team, full-time— important –but basic— information can be found in a small company or a particular. The results obtain throughout this development have been quite positive on this aspect.
- On the other hand, and with the experience not only with the development, but also understanding how other log management systems work –ELK, for instance—, I have understood not only the importance of this tools, but their most important characteristics. Creating a good service from scratch in a tight period of time is hard, and that's why my software will be incomplete and might have some bugs. Despite that, in my opinion, the most important thing will be that my design will be able to be improved over time without changing its core.

VI. FUTURE IMPROVEMENTS

My log management system is okay for simple and specific queries configured in a small environment, but as I stated before, it would not perform quite well in big environments. Improvements could be made in future versions in all of the 3 modules.

Logstash could be improved by supporting **more types of logs** and **more outputs** apart from JSON. This can be accomplished with 2

approaches: the first one would be the support for plugins, as ELK does: that would allow the quick configuration and support from modules developed by third parties. The second one would be keep the development private and implement all the options all by myself.

Elasticsearch could be improved with **support for more queries**. Right now, the queries have to be in the right format, and only 3 types of query are allowed. With more queries, any user could go deeper in the logs to find hidden –and more valuable— information. Another option would be that each index is a SQL, so the queries could be performed directly to those databases, without the need to develop and support more queries.

Finally, the main limit in Kibana's UI is the fact that it can only display information on a certain query. I think it would be appropriate to bring support for **multiple views** in the UI, apart from query results.

ACKNOWLEDGMENT

I just wanted to thank Angel for his supervising and guidance throughout the development of this project and for his ideas and corrections that have helped me during these past months.

REFERENCES

[1] CISCO SYSTEMS, *Cisco Visual Networking Index: Forecast and Methodology*, 2016-2021. [ONLINE]

[2] ELASTIC, *ELK Stack*, 2018. [ONLINE]

[3] ELASTIC, *Elasticsearch Documentation*, 2018. [ONLINE]

[4] ELASTIC, *Logstash Documentation*, 2018. [ONLINE]

[5] ELASTIC, *Kibana Documentation*, 2018. [ONLINE]

[6] PYTHON SOFTWARE FOUNDATION, *Tkinter Documentation*, 2018. [ONLINE]

[7] PYTHON SOFTWARE FOUNDATION, *Sending and receiving logging events across a network*, 2018. [ONLINE]

APPENDIX

In the following figure you can see Logstash's configuration file, where you can configure which type of logs have to be normalized and where to find them. You can also configure the location of Elasticsearch's directory.

```
{
  "syslog":{
    "active": false,
    "directory": "/Users/Josep/Desktop/LogGenerators/syslog-generator-master/"
  },

  "dns": {
    "active": false,
    "directory": "/Users/Josep/Desktop/LogGenerators/dns-generator/"
  },

  "http":{
    "active": true,
    "directory": "/Users/Josep/Desktop/LogGenerators/Fake-Apache-Log-Generator-master/"
  },

  "elasticsearch":{
    "directory": "/Users/Josep/Documents/GitHub/log-gestor/elasticsearch/"
  }
}
```

Fig. 4. Dynamic configuration in Logastash using JSON

In the following figures the different JSON formats used for the logs are displayed:

```
{
    "dns_id":{
        "type": "dns",
        "timestamp": ["day", "month", "year", "time"]
        "clientIP": "",
        "protocol": "",
        "send_receive": "",
        "record_type": "",
        "flag": "",
        "domain": ""
    }
}
```

Fig. 5. JSON format used in DNS logs

```
{
    "http_id":{
        "type": "http",
        "clientIP": "",
        "identity": "",
        "username": "",
        "timestamp": ["day", "month", "year", "time"],
        "resquest": "","
        "status_code": "",
        "size_response": "",
        "url": "",
        "useragent": "",
    }
}
```

Fig. 6. JSON format used in HTTP logs

```
{
    "syslog_id":{
        "type": "syslog",
        "timestamp": ["day", "month", "year", "time"],
        "hostname": "",
        "appname": "",
        "message": "",
    }
}
```

Fig. 7. JSON format used in syslog logs

In the following figure they appear the logs used for the query examples in Methodology section, subsection B:

access_log_20180527-193436.log

Mostrar Ara Esborrar Tornar a carregar Compartir Buscar

```
130.109.141.177 - - [27/May/2018:19:36:39 +1000] "PUT /posts/posts/explore HTTP/1.0" 200 4992 "http://www.potts.com/main/" "Mozilla/5.0 (X11; Linux x86_64; rv:1.9.5.20) Gecko/2014-03-17
22:46:48 Firefox/3.8"
26.130.80.179 - - [27/May/2018:19:39:02 +1000] "GET /wp-content HTTP/1.0" 200 4946 "http://www.farrell.com/main/" "Mozilla/5.0 (Windows 98) AppleWebKit/5312 (KHTML, like Gecko) Chrome/
45.0.870.0 Safari/5312"
11.234.205.99 - - [27/May/2018:19:42:19 +1000] "PUT /explore HTTP/1.0" 200 5034 "http://douglas.info/app/post/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8; rv:1.9.6.20) Gecko/
2010-05-10 08:41:05 Firefox/3.6.14"
110.112.114.234 - - [27/May/2018:19:43:28 +1000] "PUT /app/main/posts HTTP/1.0" 200 4941 "http://www.johnson.com/" "Mozilla/5.0 (X11; Linux i686; rv:1.9.6.20) Gecko/2016-12-21 15:48:33
Firefox/3.6.15"
188.228.2.144 - - [27/May/2018:19:44:47 +1000] "GET /wp-content HTTP/1.0" 200 5002 "https://www.lopez-hart.org/search/index/" "Mozilla/5.0 (Windows; U; Windows NT 5.01) AppleWebKit/
533.45.6 (KHTML, like Gecko) Version/4.1 Safari/533.45.6"
243.177.243.74 - - [27/May/2018:19:46:40 +1000] "POST /wp-admin HTTP/1.0" 200 5092 "http://www.suarez.com/list/about/" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_12_1; rv:1.9.3.20)
Gecko/2016-11-05 18:28:49 Firefox/3.6.4"
23.59.179.134 - - [27/May/2018:19:51:33 +1000] "GET /app/main/posts HTTP/1.0" 200 5050 "https://www.pruitt-lewis.com/main/wp-content/privacy.htm" "Mozilla/5.0 (Windows NT 6.1; as-IN; rv:
1.9.1.20) Gecko/2013-05-20 04:47:25 Firefox/3.8"
160.11.63.101 - - [27/May/2018:19:55:43 +1000] "GET /apps/cart.jsp?appID=5054 HTTP/1.0" 200 4948 "https://www.tate.net/author.htm" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4; rv:
1.9.2.20) Gecko/2010-03-28 04:06:27 Firefox/3.8"
91.245.28.250 - - [27/May/2018:19:59:29 +1000] "GET /app/main/posts HTTP/1.0" 200 4955 "https://www.carter-anderson.org/posts/wp-content/search/home.html" "Mozilla/5.0 (Macintosh; PPC
Mac OS X 10_9_0) AppleWebKit/5310 (KHTML, like Gecko) Chrome/40.0.842.0 Safari/5310"
38.122.225.45 - - [27/May/2018:20:00:21 +1000] "DELETE /app/main/posts HTTP/1.0" 404 5072 "http://coleman-nichols.com/main/" "Mozilla/5.0 (X11; Linux i686; rv:1.9.5.20) Gecko/2018-03-12
12:53:08 Firefox/3.6.7"
```

Fig. 8. HTTP logs used in the example before being processed by the Logstash module

In the following figure you can observe Kibana's user interface with a sample query already performed in one of the 2 available indexes.

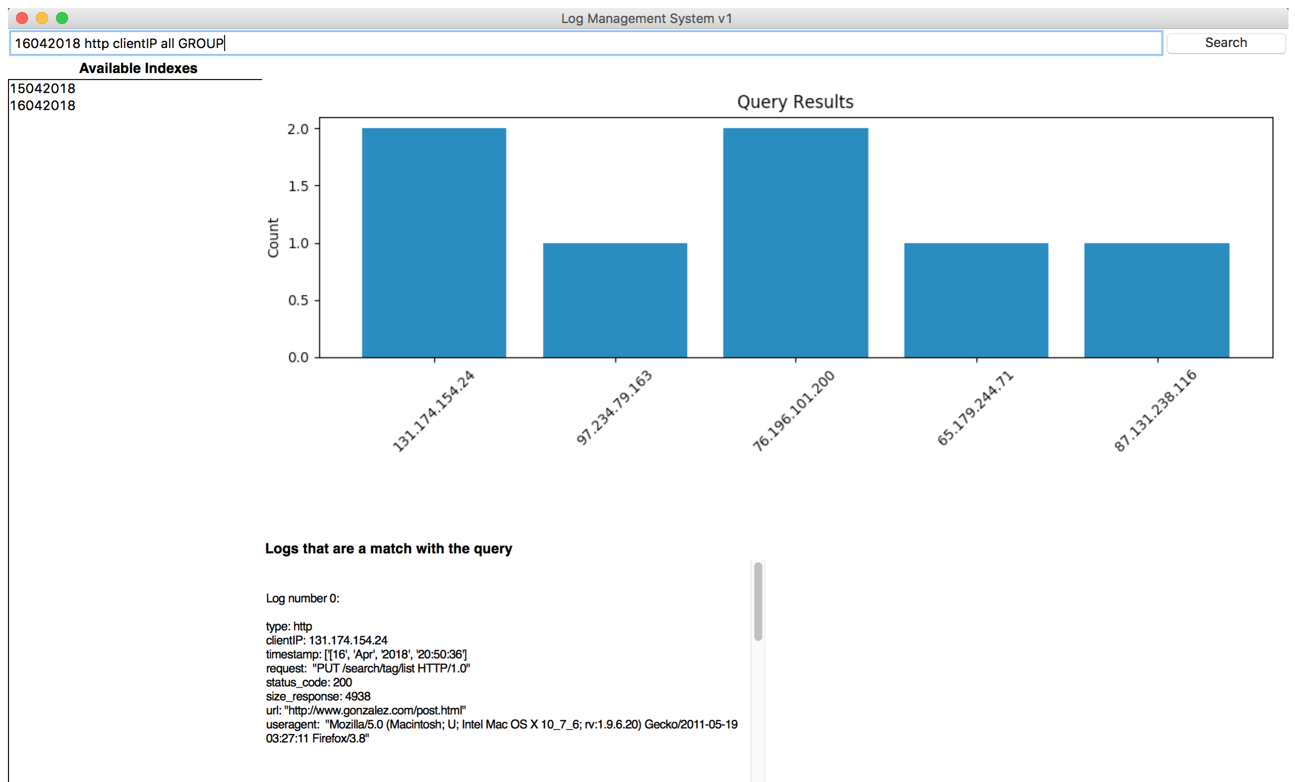

Fig. 9. Kibana's user interface

In the following figures it can be seen the graphic results for the examples performed:

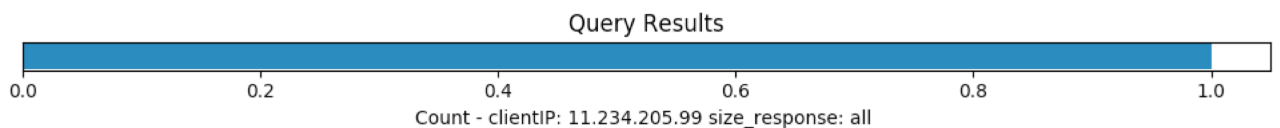

Fig. 10. Results for the COUNT query

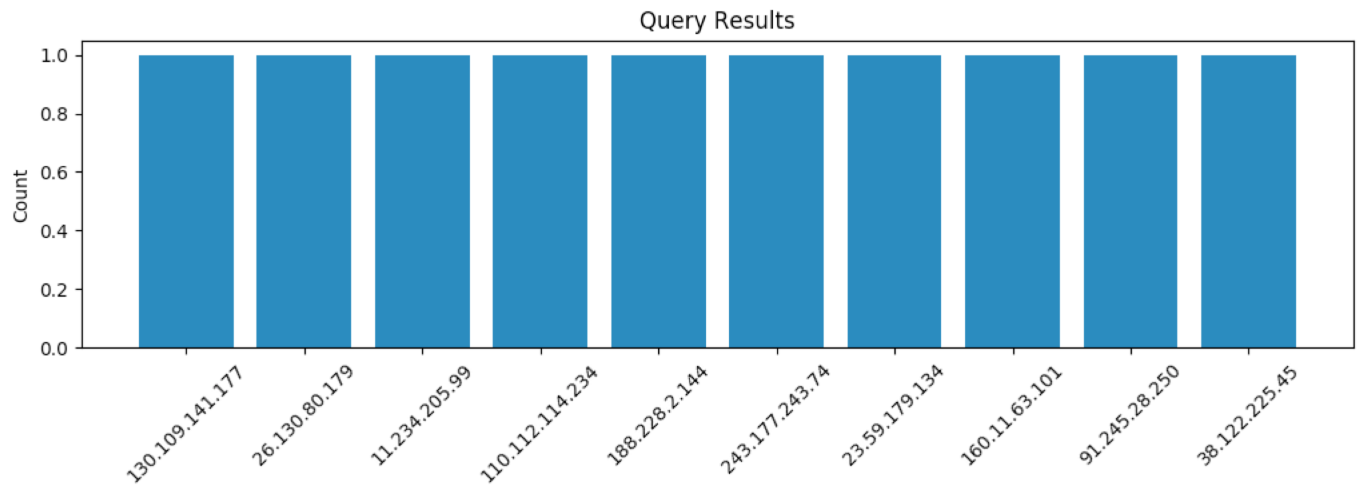

Fig. 11. Results for the GROUP query

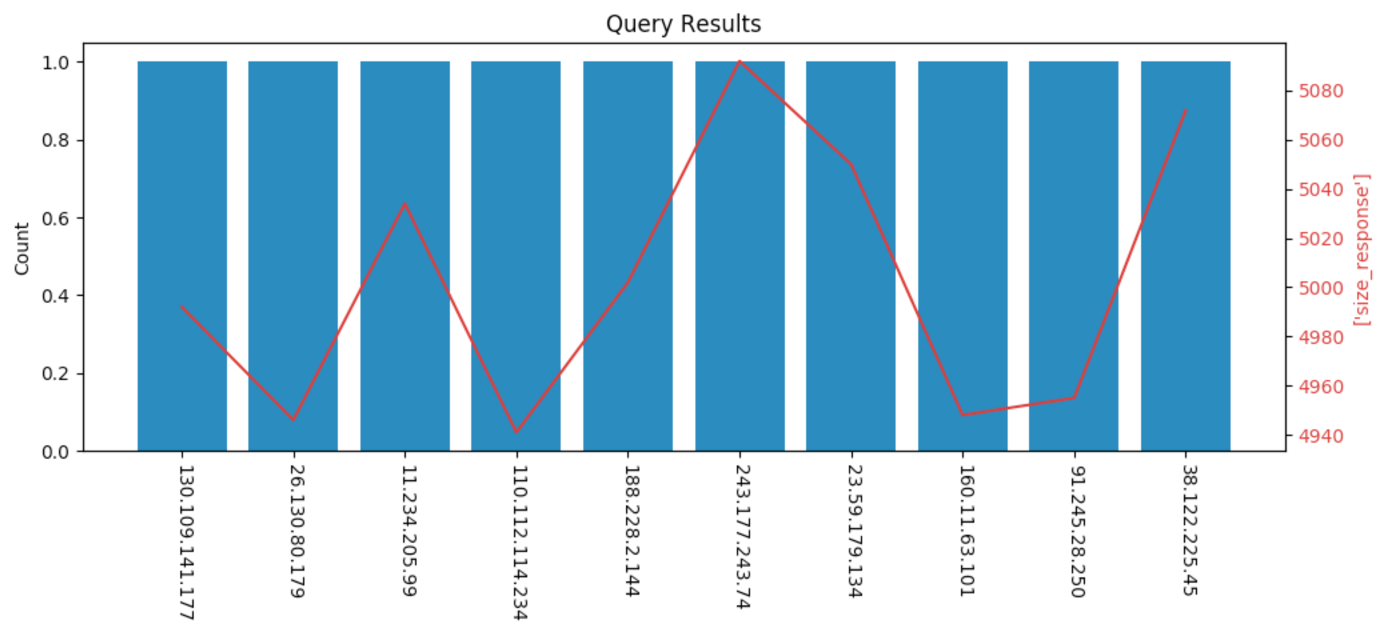

Fig. 12. Results for the ORDER query

Name	Date Modified	Size	Kind
▼ elasticsearch	avui, 12:35 p. m.	--	Carpeta
configuration.json	12 juny 2018, 4:54 a. m.	358 bytes	JSON
indexer.py	10 juny 2018, 9:10 a. m.	6 KB	Python Source
▼ indexes	27 maig 2018, 12:12 p. m.	--	Carpeta
27052018.json	27 maig 2018, 12:11 p. m.	4 KB	JSON
logs.json	27 maig 2018, 12:11 p. m.	Zero bytes	JSON
search_api.py	15 maig 2018, 6:22 a. m.	11 KB	Python Source
timestamps.json	27 maig 2018, 12:11 p. m.	52 bytes	JSON
▼ kibana	avui, 12:35 p. m.	--	Carpeta
configuration.json	11 maig 2018, 5:08 a. m.	92 bytes	JSON
kibana_ui.py	12 juny 2018, 8:10 a. m.	13 KB	Python Source
▼ logstash	avui, 12:35 p. m.	--	Carpeta
configuration.json	10 juny 2018, 9:20 a. m.	459 bytes	JSON
constants.py	14 maig 2018, 11:56 a. m.	79 bytes	Python Source
dns_normalizer.py	22 abr 2018, 3:47 a. m.	2 KB	Python Source
http_normalizer.py	21 abr 2018, 10:31 a. m.	4 KB	Python Source
logstash.py	13 juny 2018, 3:47 a. m.	3 KB	Python Source
normalizer.py	13 juny 2018, 3:47 a. m.	627 bytes	Python Source
syslog_normalizer.py	15 abr 2018, 8:50 a. m.	3 KB	Python Source

Fig. 13. Directory scheme of the whole project