INFO-H-415 : Advanced Databases
Logs collection and analysis with Elasticsearch

Rémy Detobel – ***000 408 013***

Alexis Reynouard – ***000 410 614***

December 2017

Contents

In this work we aim to show a practical approach to log collection and analysis. We would like the solution presented to meet the usual needs of today's businesses and companies, both in terms of quality requirements and implementation costs.

1 Case study

1.1 Context

Every day students and professors of the ULB use the IT services as part of studies, teaching and research. This is done through various services like ULBPodcast, MonULB, mobile app, *etc.*

The ULB IT service wish to gather as much data as possible from these various sources for the usual purpose of ***logging*** (like security, debugging and profiling applications) but also in order to ***extract*** and ***visualise statistical informations*** for use by teachers and authorities of the ULB.

All of these services record lot of data, several hundred log lines per minute. Thus it is several tens of mega recorded and may be couple of giga every day. So the system must be highly ***scalable***.

The action required by the system maintainers in response to an ***update*** of these data sources should be as minimal as possible in order to keep extracting consistent statistical informations without requiring a lot of work from maintainers.

Some data should also be visible in near ***real time,*** for example the status of a service or the detection of an error. Therefore these data have to be present in search results quickly and it should be possible to automatically trigger ***events*** from them.

Replication must also be possible in order to not loose informations.

1.2 Current system

Today lot of these data are recorded in files. Thus it is easy to make some basic backups, however it is difficult to access and analyse these data.

Some test were done on a MySQL database with 1 000 000 entries (number of logs after 10 day's) but some query takes more than 5 minutes. It is too long to create some graph or to display some information on the website. Currently the statistic are pre-process (one process per day).

With the solution we propose here, the same query runs in less than 5 seconds, more than 60 times faster.

2 Desired system properties

2.1 Robustness

The solution provided must ensure data persistence. Therefore the solution must be scalable and have the ability to realise data replication, fast or instant crash recovery, *etc.*

2.2 Solution lifetime

To avoid having to rebuild the system or part of it frequently, the solution should provide not only a robust must also an adaptive system. Otherwise, each update of a service whose logs are collected may require an update of this log collection and analysis service. One should also be able to (re)configure only part of the system to adapt it to his needs. For example, the set up of a new way to visualise data should not interfere with the collection of these data. Thus we will prefer a modular system.

2.3 Responsiveness

It seems obvious that data have to be accessible through the database very quickly after their generation by services. When we say "quickly", its means that we must not wait 10 minutes to have a result or a chart. Some information also need to be “ near real time”, to see if there is a crash or any other problem.

3 Comparison of different solution

The most famous database to store logs are: MongoDB, Cassandra and Elasticsearch. MongoDB is one of the most popular document stores and offer a com-

plete flexibility on the data stored. Cassandra is a Wide column store database optimised to write access and to hold huge amount of data. Elasticsearch is an analytic and search engine.
All three are open source, works on lot of OS and support lot of programming languages. They all provide a more flexible way to store data than usual relational database (by the way they manage the schema). However, the datastructure on MongoDB is completely flexible while on Elasticsearch a minimal structure is required on the document indexed. Cassandra on his side is said “schema-optional”. Thus, MongoDB is great to deal with unstructured data. However Elasticsearch provide much better performances to search and analyse high volume of data and Cassandra is far better to write data.

For the database manager, Cassandra handle queries written in a SQL like syntax in opposite with MongoDB or Elasticsearch which use more JSON format. However MongoDB use its own protocol and Elasticsearch provide a REST API. The two have their pro and cons.

Note that all data stored in these databases are persistent and may be replicated and distributed, but Elasticsearch handle these configuration almost automatically. This way Elasticsearch is able to respond to local failure without maintainer actions nor any prior configuration. However, it worth to note that Elasticsearch works on top of Lucene which does not store checksum of all data. So data integrity is mostly ensured by the replication itself or the underlying OS.

For our purposes we have chosen to use Elasticsearch because it offers modularity, adaptivity, ease of use and a lot of analytical aspects. However we see that Elasticsearch need lot of resources and requires several servers to work well.

4 Elasticsearch and the Elastic Stack

4.1 Elasticsearch overview

Among many other solutions, Elasticsearch seems to fit our needs best.

> Elasticsearch is a highly scalable open-source full-text search and analytic engine.

Elasticsearch is a database used to store, search and analyse big volumes of data.

First of all, Elasticsearch is a document oriented database, as opposed to a relational database. In a few words, this means that the data structure is more flexible. We will describe in more detail what this means when presenting the key concepts of Elasticsearch below (see 4.3). Let us already notice that, as logs are poorly structured data, this seems really appropriate for logs storage. For example, if a software update add a new datum in its log lines, it is not required to update the datastructure itself. Sometimes one will have to (re)configure some modules or other programs Elasticsearch work with. But in general this is less heavy and more maintainable because the updates to performs are well encapsulated.

This allows to manage a lot of logs from various services with the minimum effort, as well as to adapt to update of any of these in the shortest possible time.

4.2 The Elastic Stack

Actually, Elasticsearch is the core of a groups of softwares called the "Elastic Stack".

While Elasticsearch handle the data themselves, the other main components perform tasks to allow to include Elasticsearch in any realistic environments. Most of these components are highly modular and configurable. Modules are here to provide specifics functionalities.

Let us very quickly present these components and their respective role, from the official documentation:

Kibana Kibana is an analytic and visualization platform designed to work with Elasticsearch. You use Kibana to search, view, and interact with data stored in Elasticsearch. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps. More details are given in 4.5.

Beats Platform The Beats are data shippers that one can install as agents on his servers to send data from different sources to Elasticsearch. Beats can send data directly to Elasticsearch or send it to Elasticsearch via Logstash, which can be used to parse and transform the data (see figure 1). More details are given in 4.6.

Logstash Logstash is a data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize the data into arbitrary destinations. More details are given in 4.6.

Logstash is able to collect data not only from log or other usual data sources, but from a wide variety of events like HTTP events, messages queue services (*e.g.* Amazon SQS) . . . .

X-Pack X-Pack is an Elastic Stack extension that bundles security, alerting, monitoring, reporting, and graph capabilities. It provide functionalities that span over the other components. X-Pack come also with machine learning capabilities useful to detect (and eventually fix) anomalies in the data.

4.3 Key concepts

Like any system, Elasticsearch has its own specific properties and its own vocabulary. These specific terms will be explained as they occur through the document. However lets already introduce the main terms and concepts of the Elastic Stack.

4.3.1 Document

A ***document*** may be view as a record of traditional relational database systems. In Elasticsearch, document are viewed as JSON object [1], although it is often automatically created from another type of text, like a line from a log file.

4.3.2 Index

An ***index*** corresponds roughly to a table in an usual relational database, in the sense that it groups together documents with similar characteristics.

Note however that an index do not define any attributes for its documents. In other words, there is no “columns”. An index is more like a collection of data that have somewhat similar characteristics. This is a key concept as it allows a great flexibility needed to adapt quickly to new configurations. However, the structure is kept by the mean of the common attributes of the documents inside the index.

[1]To describe briefly, a JSON object is a set of ⟨key, value⟩ tuples. Each key is a `string`, each value may be a `string`, a `number`, a `boolean`, a `null`, an array of these types or another JSON object

Thus, Elasticsearch does not restrict you about the number or the name of column. You could send different document with different format and Elasticsearch will automatically try to recognise and interpret them. However, datatype values with the same key have to be coherent. This is required by Elasticsearch in opposite to other NoSQL database, to allows better search and analyse performances.

Each documents must be indexed. The indexation is done only once: when the document first arrive in the database.

4.3.3 Near Real-time

Near real-time is a core feature of Elasticsearch. This means that it take about one second (on a realistic database) between the time a document is indexed and the moment it become searchable.

4.3.4 Cluster and nodes

Elasticsearch is a distributed database organised in ***clusters***. Thus a single database is a group of (one or more) servers connected together and identified by a unique name. A cluster is useful to have load balancing and more CPU power.

Each server in the cluster is called a ***node***. Usually, each node runs an Elasticsearch instance. It participate in the cluster's indexing and search capabilities. The configuration is transparent to the user. Internally, each node may be responsible for some data and keeping a backup of other data. There is also a (automatically or user-defined) master which receive and redirect queries.

4.3.5 Shards

Even if it is out of the scope of this presentation, it worth noting that an index does not have to fit on a single node (main or/and secondary) memory. Indeed, Elasticsearch is able to subdivide the indexes into small pieces called ***shards***. Shards are fully-functional and independent by itself. Thus sharding not only allows you to split your indexes into several nodes, but also to distribute operations across shards (potentially on multiple nodes) thus increasing performance.

4.4 Main properties

Elasticsearch and the Elastic Stack are highly configurable.

It is well integrated with a great number of systems. For example installation may be done with Homebrew on macOS, MSI installer on Windows, rpm or deb packages on linux and on docker. Large deployment may be done with tools like Ansible.

To communicate, Elasticsearch provide a REST API. Thus you could simply make a HTTP request (with `curl` for example) or use Elasticsearch API (Java, C#, Python, JavaScript, PHP, Perl, Ruby). For instance, to see the cluster health, you could make the following command:

```
curl -XGET 'localhost:9200/_cat/health?v&pretty'
```

And the response will be something like that:

```
epoch      timestamp cluster       status node.total node.data
1512122482 11:01:22  elasticsearch yellow          1         1

shards pri relo init unassign pending_tasks max_task_wait_time
    31     31  0    0    31        0             -

active_shards_percent
50.0%
```

Here we could see that the status of the server is yellow. It is because there are only one node on this network/cluster. Thus they have not replication of the data and then no security.

4.5 Kibana

Kibana is an interface to view data contained into Elasticsearch database. It may be compared to phpMyAdmin / phpPgAdmin for mySQL / PostgreSQL but offer more graphical tools and other capabilities due to the document based architecture of Elasticsearch.

Kibana is used to manage the databases easily, and quickly analyse its content.

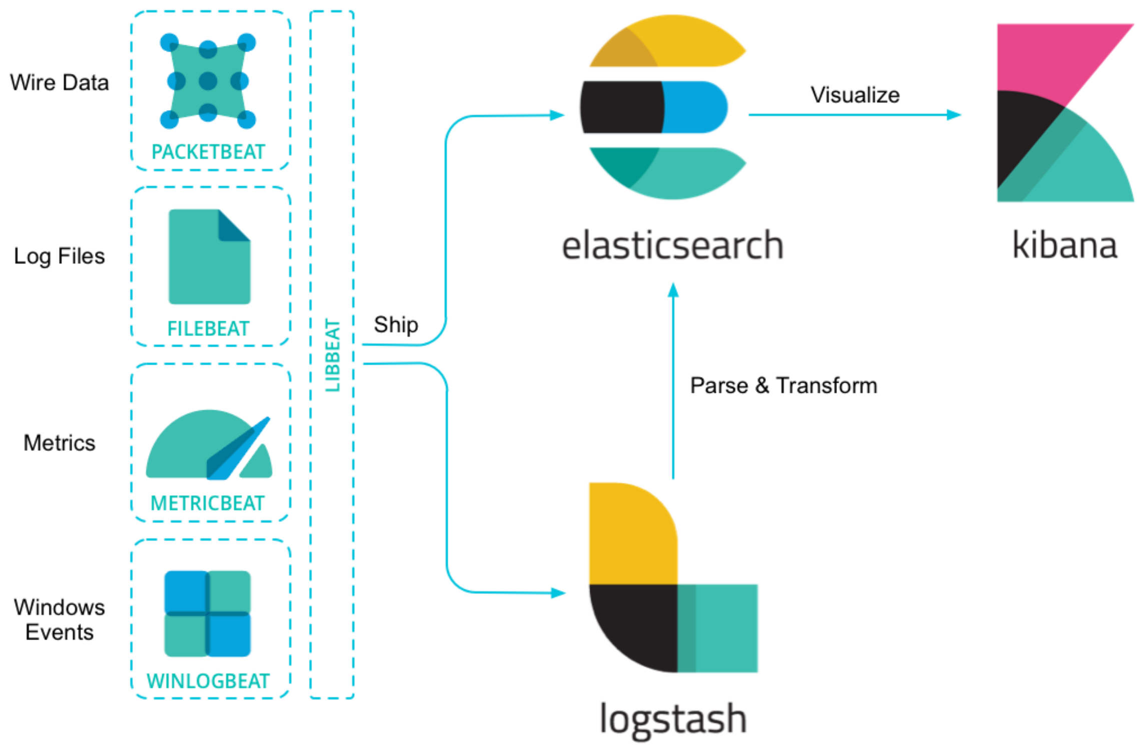


Figure 1: Communication between beats, Logstash and Elasticsearch (from the official documentation)

You can create indexes, query for specific data, update them, delete, realise arbitrary queries, visualize and analyse through graphs, create graph and compose them to create boards, automatically create reports (thanks to an X-Pack add-on) *etc.*

In practice Kibana is very user friendly. You create first an index in the ***Management*** part and then you can ***Discover*** the data, ***Visualize*** them with different charts, group this charts on a ***Dashboard***. Finally you have the ***Dev Tools*** where you can directly execute arbitrary queries.

4.6 Logstash & Beats

To fill the database one can use the REST API or type commands in the Kibana interface. But Elasticsearch was designed with ease of use in mind. So, to send data to the database, you should use beats to collect them and Logstash to structure them and send them to Elasticsearch.

In practical there is no interface with Logstash. You can just edit the configuration. The configuration depend a lot of the beats that you use and of the plugin that you use.

Let us present on figure 2 a basic configuration of logstash.

```
input {
    beats {
        port => "5043"
    }
}
# filter {
#
# }
output {
  elasticsearch {
    hosts => "localhost:9200"
    manage_template => false
    index => "test-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
  }
  stdout { codec => rubydebug }
}
```

Figure 2: Logstash configuration

It is quite intuitive: the configuration is composed of two parts: ***input*** and ***output***. There is also an optional section *filter*, here commented out. This configuration will listen on the port *5043* for beats connections. They are here two outputs: Elasticsearch and stdout. The first connect to Elasticsearch and send data to the *"test-YYYY.MM.dd"* index, the second is only here to debug purposes: it displays on the screen the data sent. One can use the filter part to filter out some data or performs transformations on data, for example in order normalise it. The filter part allow you to select data which must be send or change the format of these data.

Beats could send data directly to Elasticsearch or send it to Logstash which will parse, filter and structure them to transfer them to Elasticsearch. There are lot of Beats ready to use in the Elastic Stack: Filebeat (send file content), Winlogbeat (send Windows event logs), Metricbeat (send data about your operating systems and services running on your servers), apachebeat (send stats from Apache), amazonbeat (send data about a specific product), mysqlbeat (run SQL command and send result)...

4.6.1 Logstash plugin

Logstash get data from beats (see point 4.6.2) and send them (usually) to Elasticsearch. But it could filter the data and also parse them. To make this second point, Logstash need some plugin. For instance there exist a plugin to parse csv data or to check duplicates event, to perform DNS reverse or to parse any unstructured data.

4.6.2 Beats

Beats are designed to be lightweight, easy to develop and to not require a lot of resources. Usually the configuration of beats is not complex. Indeed, the beat send the data to further processing and persistent storage. The computing part is not on the beat.

Here you have for instance the Filebeat[2] configuration used to send the file `testfile.log` (located in a home) to Logstash

```
filebeat.prospectors:
- type: log
  enabled: true
  paths:
    - /home/user/testfile.log

output.logstash:
  # The Logstash hosts
  hosts: ["localhost:5043"]
```

Figure 3: Example of configuration of Filebeat

4.7 X-Pack

X-Pack is an extension that can be installed on each module of Elastic Stack. This add-on contains security, machine learning, graph drawing, alert, monitoring, ... All the functionalities added by X-pack are accessible through Kibana.

[2]`https://www.elastic.co/products/beats/filebeat`

5 Concrete example: EZCast

5.1 Situation

To spread video, ULBPodcast uses the open source solution EZCast[3]. This application store videos to serve and data about user interactions. By default ezcast does not record user interaction, to enable this user interaction recording, you just need to change one line in the file `config.inc` in the commons folder (figure 4).

```
// -- Traces -- //
$trace_on = true; // determines whether we want to enable traces on actions or
    not
```

Figure 4: EZCast configuration: trace enabled

These data are called "traces". Currently, each user interaction is translated into a trace and each trace is store in a file. There are one file per day. The figure 5

```
2017-11-16-00:01:23 | b25v5ik93cnrc4mr2jh0emmoi5 | 109.131.47.227 | nologin | 4
    | video_play | PHYS-H-100-pub | 2013_01_17_12h52 | 1827 | 416 | cam | high |
    from_shortcut
```

Figure 5: Example of trace

shows you the structure of a trace. Data fields are delimited by the `"|"` symbol. The six first elements are fixed: time, PHP session, ip address, netid (or nologin if not know), level and action. The other elements depend on the action. The "level" information let us know where was the user when he makes this action (more info in the table 1).

0	unknown
1	login page
2	course list
3	list of videos (for one course)
4	video player

Table 1: Description on each level of EZCast

[3]https://github.com/ulbpodcast/ezcast

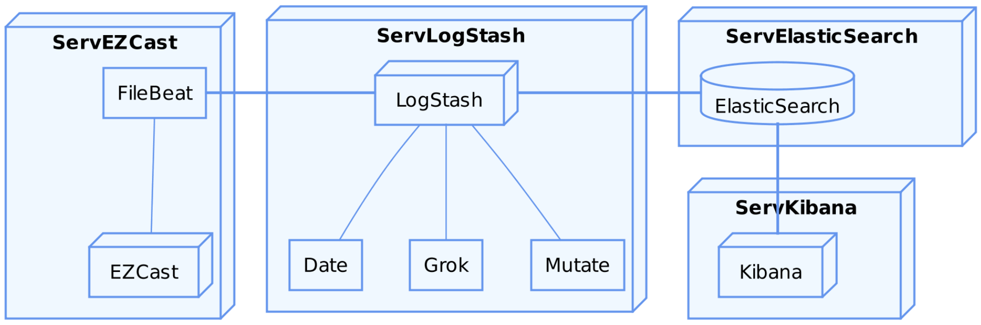


Figure 6: Deployment of Elasticsearch with EZCast

5.2 Problem

To analyse and summarize all of these data, we must load each file and count, sum, *etc.* all the information needed to make a report. This take lot of time and required lot of memory. With a SQL database the problem is that there are too much data of various forms and that the structure may change with new versions of EZCast. There are more or less 100,000 lines per day. EZCast was launched four years ago and this system exists since three years. Thus, there are now: $3 \times 365 \times 100,000 = 109,500,000$ lines. And each year it is 36500000 lines that we must add (maybe more because we can collect more informations).

For these reasons and the observations made before we think that Elasticsearch is a good alternative.

5.3 Configuration of Elasticsearch for EZCast

Elasticsearch can be distributed on different servers (see point 4.3.5). One possibility is to separate all different components, like show on the figure 6. We have four servers, one with EZCast (where the students watch the videos) and Filebeat which send trace data to another server which runs Logstash for further processing. On the Logstash server, various plugins are configured. Logstash send the processed data to Elasticsearch (on another server). To view the data we use one last server with Kibana.

This configuration does not overload the servers but we must have four of these. Thus for this report we test another configuration: one server with Logstash, Elasticsearch, Kibana and another with EZCast and Filebeat. Note that all these product may require a lot of resources.

5.3.1 Filebeat

First of all we must configure Filebeat and indicate where are the EZCast traces and how to send them to Logstash. See point 4.6.2 for more informations and figure 3 for an example of how to specify this port in Filebeat.

5.3.2 Logstash

We will configure Logstash to listen to all the messages sent by Filebeat. Like shown in figure 2, we just have to specify the port. We also have to tell Logstash where to send the resulting data. In this case we want that he send it to Elastic-search, the example configuration (on point 2) already do this.

In the figure 5 we can see that the EZCast traces contains multiple informations separated by the symbol `"|"`. Thus we must configure Logstash to parse these data. All the traces in EZCast start with the same informations: timestamp, session, ip, netid, level, action and then optional informations (maximum 10).

5.3.2.1 Grok

To parse these traces we will use the plugin *Grok*[4]. This plugin allows to use regex to parse the log. For example one can add the following configuration in the `filter` part of the Logstash configuration file:

```
grok {
  patterns_dir => ["/etc/logstash/patterns"]
  match => {
    "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level:int} \| %{WORD:action}( \| %{ANY:elem1} ?)?(\| %{ANY:elem2} ?)?(\| %{ANY:elem3} ?)?(\| %{ANY:elem4} ?)?(\| %{ANY:elem5} ?)?(\| %{ANY:elem6} ?)?(\| %{ANY:elem7} ?)?(\| %{ANY:elem8} ?)?(\| %{ANY:elem9} ?)?(\| %{ANY:elem10} ?)?"
  }
}
```

Grok use regex to parse the data. Here we use default patterns offered by grok, like `IP`, `WORD` or `NUMBER`. But we have also defined custom patterns like `ANY` or `CUSTOM_TIMESTAMP` in the file `/etc/logstash/patterns`.

[4]`https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html`

Thus, this will create 16 fields: timestamp, session, ip, netid, level, ...
Note that there is an option in Kibana to directly test regex (X-Pack need to be enabled).

5.3.2.2 Date

In EZCast the timestamp is saved with the following encoding: `YYYY-MM-DD-HH-mm-ss` but this encoding is not supported natively by Elasticsearch. To convert this timestamp one can use the *Date*[5] plugin. With the following configuration (still in the "filter" section) we convert the "timestamp" string to a normal timestamp for Elasticsearch:

```
date {
  match => ["timestamp", "yyyy-MM-dd-HH:mm:ss"]
}
```

5.3.2.3 Mutate

Now there is a problem with integer data. Indeed, Grok just split strings into substrings. For example the level is a number and it may be interesting to perform numerical comparisons on this field. It is not possible, as long as the file "level" is a string. To change this, we will use the *Mutate*[6] plugin:

```
mutate {
  convert => {
    "level"                 => "integer"
  }
}
```

5.3.2.4 Enhance configuration

This final configuration is given in the annexe A.1.

However in this configuration the optional fields are all stored as strings. Another problem is that the field *elem1* could contain an album name for a specific action and an asset name for another action or the album name could be in the field *elem3*.
To avoid this problem we must define more regex filter, one regex by action.

[5] https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html
[6] https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html

```
grok {
  patterns_dir => ["/etc/logstash/patterns"]
  match => {
    "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip}
    \| %{WORD:netid} \| %{NUMBER:level} \| video_seeked \| %{ANY:album} \| %{ANY:
    asset} \| %{NUMBER:video_duration} \| %{NUMBER:previous_video_time} \| %{
    NUMBER:video_time} \| %{ANY:type} \| %{ANY:quality} ?"
  }
  add_field => { "action" => "video_seeked" }
}
```

This part of the configuration parse log that have the action `video_seeked`. This regex create 6 new fields: `album, asset, video_duration, previous_video_time, vic` and `quality`. To convert `video_duration, previous_video_time` and `video_time` we use the plugin *mutate* (see point 5.3.2.3).
We could make it for all the possible actions but in this case we just consider the useful ones. Thus we keep the general regex explained above for the others and we use the following syntax to aplly it only when needed. Otherwise some data may pass through many filter and be sent many times to Elasticsearch. This is useful in some cases but not what we want to do here.

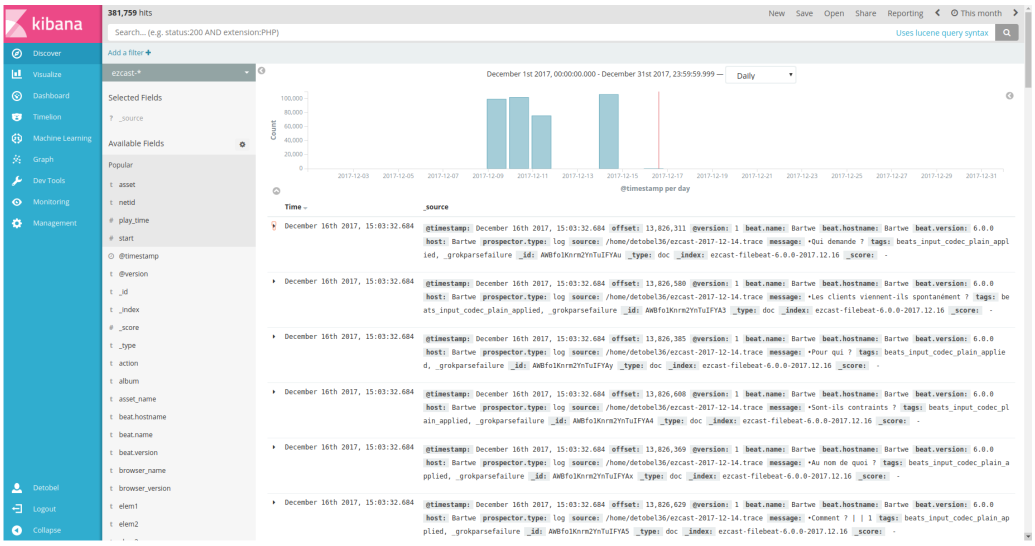
```
if ![action] {
  grok {
    ...
  }
}
```

5.4 Data visualisation

Kibana is the friendly user interface of the Elastic stack. Among other it allow to see the stored data (see point 4.5). The figure 7 shows us theses data.

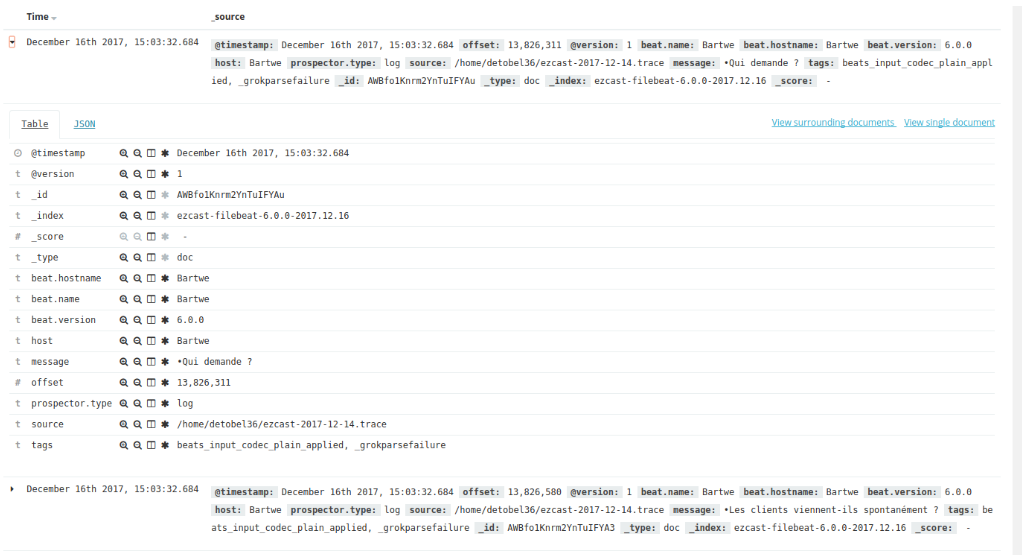
We can also view a specific document as shown by the figure 8. It is also possible to view this information in JSON format. There are also four buttons: zoom, dezoom, window and stars. Each of them allows us to filter the result. The zoom add a filter such as resulting documents have the selected key and value. The dezoom icon make the invert of the zoom (resulting documents could not have this key with this value). The window button allow us to filter the column view and finally the stars select only documents where the clicked field exists.

But we can make more things with Kibana. For example one can create charts in the "visualise" tab to view evolution or repartition.



```
Time    _source
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,311 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Qui demande ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYAu _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,580 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Les clients viennent-ils spontanément ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYA3 _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,385 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Pour qui ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYAy _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,608 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Sont-ils contraints ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYA4 _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,369 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Au nom de quoi ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYAx _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,629 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Comment ? | | 1 tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYA5 _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
```

Figure 7: List of data in Kibana

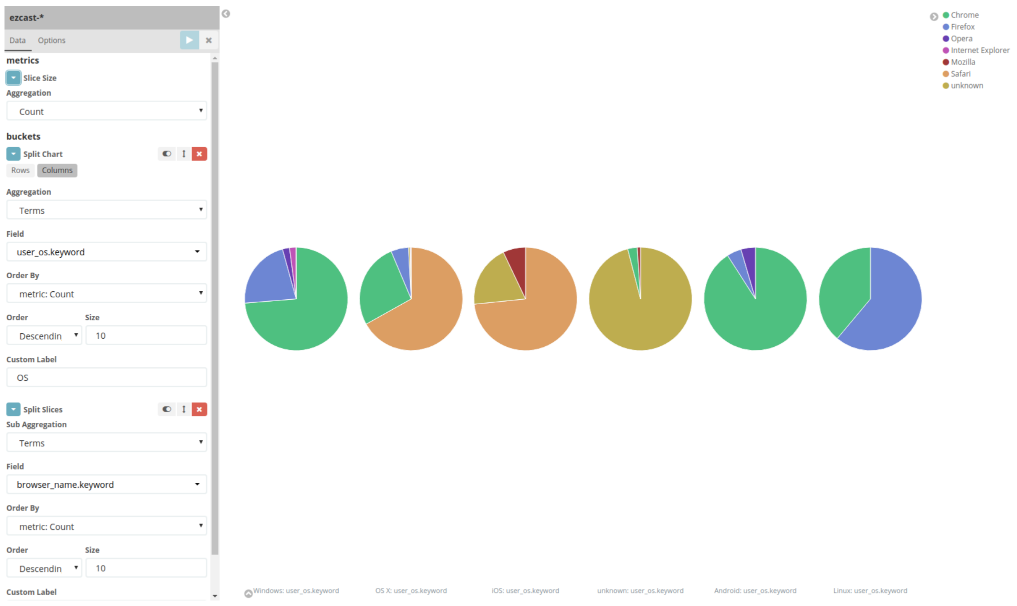


```
Time    _source
December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,311 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Qui demande ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYAu _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -

@timestamp       December 16th 2017, 15:03:32.684
@version         1
_id              AWBfo1Knrm2YnTuIFYAu
_index           ezcast-filebeat-6.0.0-2017.12.16
_score           -
_type            doc
beat.hostname    Bartwe
beat.name        Bartwe
beat.version     6.0.0
host             Bartwe
message          •Qui demande ?
offset           13,826,311
prospector.type  log
source           /home/detobel36/ezcast-2017-12-14.trace
tags             beats_input_codec_plain_applied, _grokparsefailure

December 16th 2017, 15:03:32.684  @timestamp: December 16th 2017, 15:03:32.684 offset: 13,826,580 @version: 1 beat.name: Bartwe beat.hostname: Bartwe beat.version: 6.0.0 host: Bartwe prospector.type: log source: /home/detobel36/ezcast-2017-12-14.trace message: •Les clients viennent-ils spontanément ? tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWBfo1Knrm2YnTuIFYA3 _type: doc _index: ezcast-filebeat-6.0.0-2017.12.16 _score: -
```

Figure 8: One document in Kibana

5.4.1 Data visualisation Example: User's OS and browser

In the login trace, EZCast store the OS and the Browser of the user. We can make a pie graph to see which are the more used. As show on figure 9 we have to select the type of aggregation: "count" in this case. Then we may adapt the split parameters. In this case we have specified that the aggregation concern the term `user_os` and that the order depend on the count. We also need to limit the number of results, here to 10. Then we split the charts between the different OS. It is the reason why there are six pie charts.



ezcast-*

Data Options

metrics

Slice Size

Aggregation: Count

buckets

Split Chart

Rows Columns

Aggregation: Terms

Field: user_os.keyword

Order By: metric: Count

Order: Descendin Size: 10

Custom Label: OS

Split Slices

Sub Aggregation: Terms

Field: browser_name.keyword

Order By: metric: Count

Order: Descendin Size: 10

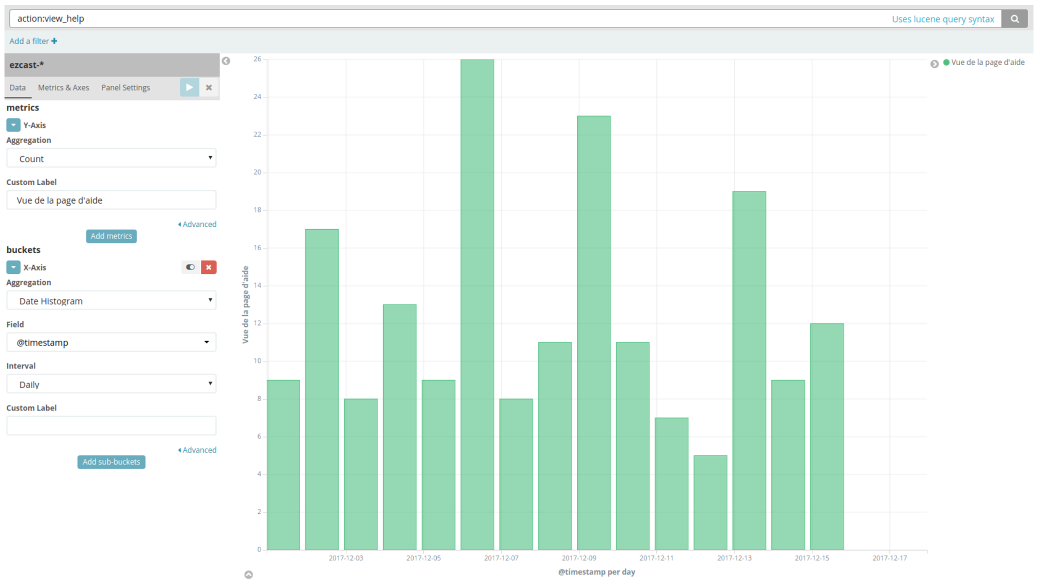
Custom Label

Figure 9: Pie charts in Kibana

5.4.2 Data visualisation Example: help page views

The view frequency of the help page is a relevant piece of information about the user experience with the system.

To view its evolution one may create a simple graph as shown on figure 10.



action:view_help

Add a filter +

ezcast-*

Data Metrics & Axes Panel Settings

metrics

Y-Axis

Aggregation: Count

Custom Label: Vue de la page d'aide

buckets

X-Axis

Aggregation: Date Histogram

Field: @timestamp

Interval: Daily

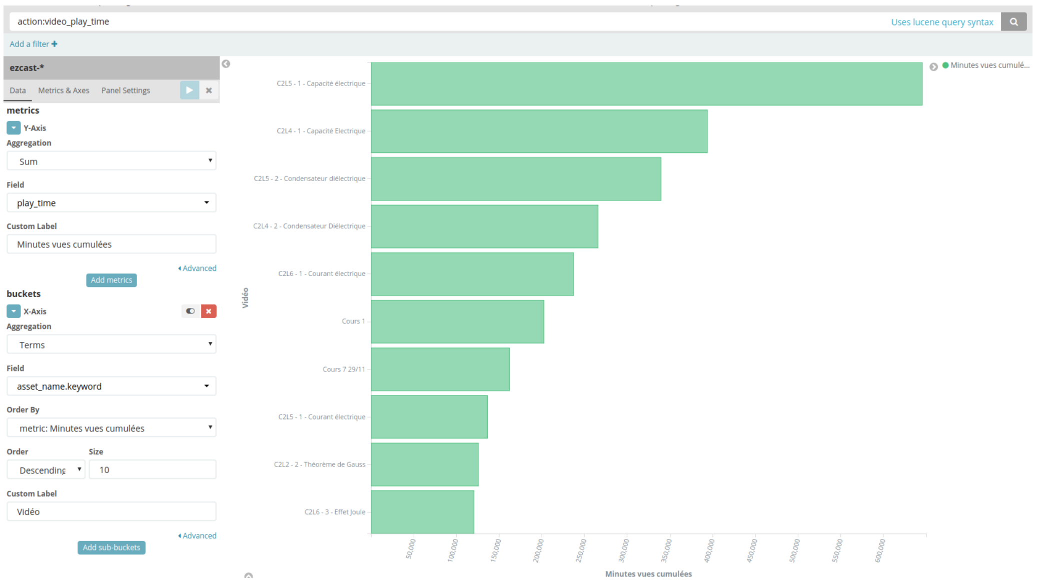
Custom Label

Figure 10: Access on the help page

On this graph we must keep only `view_help` action (as you can see on the left top) and just count the number of actions during this time.

5.4.3 Data visualisation Example: Most watched video

We can also make a sum of viewed minute per video. The figure 11 show us the "most viewed video".


action:video_play_time

ezcast-*

metrics
Y-Axis
Aggregation: Sum
Field: play_time
Custom Label: Minutes vues cumulées

buckets
X-Axis
Aggregation: Terms
Field: asset_name.keyword
Order By: metric: Minutes vues cumulées
Order: Descending
Size: 10
Custom Label: Vidéo

Figure 11: Cumulative time on video

5.4.4 Dashboard

Note that we can merge all of these charts on one page. The figure 12 group the charts previously described. It is an interactive dashboard. If one selects a browser, the number of connection (charts on the top) will be updated to show data only for this browser. You can also select the range of the date with a scroll on the top charts.

5.5 Queries

Kibana does not only offer to visualise and analyse the data stored in Elasticsearch. We can directly send arbitrary requests to the database. For example with the following command:

```
curl -XGET 'localhost:9200/_search?pretty' -H
  'Content-Type: application/json' -d'<JsonQuery>'
```

where `<JsonQuery>` is the request.
It is also possible to execute the query in the "Dev Tools" tab of Kibana.

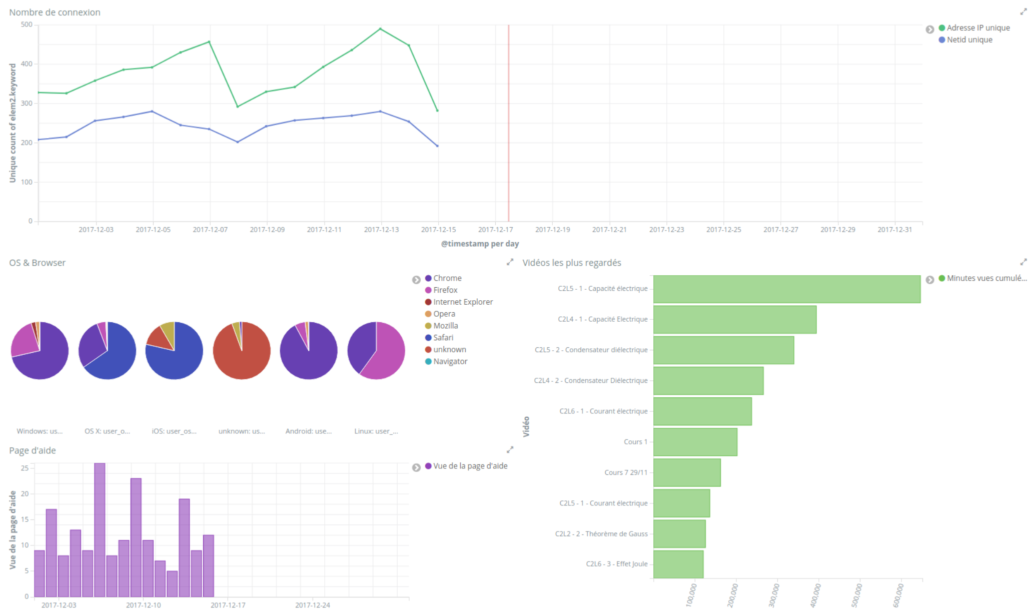


Figure 12: Dashboard of Kibana

5.5.1 Simple filter

A simple query is to select all document with a specific action. In the query 13 we select any document that concern the action `video_play_time`.

```
POST /ezcast-*/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term": { "action": "video_play_time" }
        }
      ]
    }
  }
}
```

Figure 13: Simple query with a filter

To limit the number of results, we can add `"size" : x` and we can also have an offset with `"from" : 0`. Sorting may be realised with `"sort" : [{"start" : "desc"}]`. Thus with the following query we have the 10 first documents after 20, ordered according to the "start" field:

Note that it is also possible to make far more complex sorting, for example with location or average on a field with multiple values (array). See the documentation

```
POST /ezcast-*/_search
{
  "from" : 20,
  "size" : 10,
  sort" : [
    {
      "start" : "desc"
    }
  ],
  "query": {
    "bool": {
      "must": [
        {
          "term": { "action": "video_play_time" }
        }
      ]
    }
  }
}
```

Figure 14: Simple query with a limit

of Elasticsearch[7].

The previous query request any elements of the selected documents. Most of the time however, getting all the fields is not required. To limit the results to some specific fields we must add `"_source": [<field>]`.
We can also compute some “not stored” fields. To make it, we must use `script_fields`. The followed query instance will filter all documents and keep only 3 which have action `video_play_time`. The result will contains only the field `start`, `play_time` and a new field `end_time` which is just the sum of `start` and `play_time`. The result of this query is shown in annex A.3.

Note that we could have more filter and more complex, for example the following query (16) make the same thing that the previous but check that the `play_time` is between 4 and 20. You can find a possible result in annexe A.4

5.5.2 Aggregation

Sometimes we will have to group selected data. For example if we want to know the most used browser to connect to EZCast we can make the query 17. In this query we select action `login` or `login_as_anonymous` and we group the results on `browser_name`. It could be interesting to see if this usage grow or not. We can

[7]https://www.elastic.co/guide/en/elasticsearch/reference/5.6/search.html

```
POST /ezcast-*/_search
{
  "size": 3,
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "action": "video_play_time"
          }
        }
      ]
    }
  },
  "_source": ["start", "play_time"],
  "script_fields" : {
    "end_time" : {
      "script" : {
        "source": "doc['start'].value + doc['play_time'].value"
      }
    }
  }
}
```

Figure 15: Simple query which select return field

for instance check this evolution week by week like show in the query 18.

5.6 Machine Learning

X-Pack come with ready-to-use machine learning systems, for example predictions and anomaly detections. We will not explain how it works in this document. Instead we will focus on how to configure the anomaly detection for a simple usage and how to interpret the results for EZCast.

The figure 19 show us the number of action with a unique netid. In other words, it is a good indication on the number of user that are connected to EZCast. Anomaly detection algorithm run to find some anomalies. When it finds one, it evaluates the severity. Red is a critical anomaly while blue is just a warning. In this case we can see that the 10th of December at 2 pm they have been a big drop in activity. May be a maintenance or a problem with another service of the ULB. The blue point on the 15th of December can also be the reflection of a problem but here may be just limited to a special video or course. We can have more information about the anomalies with the table just under the charts. This table contains the precise date, the severity but also some various numerical data. These results can be linked to an alert add-on.

```
POST /ezcast-*/_search
{
  "size": 4,
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "action": "video_play_time"
          }
        },
        {
          "range": {
            "play_time": {
              "gte": 4,
              "lte" : 20
            }
          }
        }
      ]
    }
  },
  "_source": [
    "start",
    "play_time"
  ],
  "script_fields": {
    "end_time": {
      "script": {
        "source": "doc['start'].value + doc['play_time'].value"
      }
    }
  }
}
```

Figure 16: Request which limit the result data set

Note that, for now, the machine learning algorithms work on new data and do not take into account any value inserted before they were enabled or values referring to anteriors events: the training is continuous.

6 Conclusion

Elasticsearch provide a good solution for EZCast and other services of ULB but need a lot of resources and multiple servers to works well. Kibana offer a very convenient interface that allows quickly visualise and analyse the data. There is a lot of modules and documentation to create our own. Works with module help to spread the workload, to install only what we need and to keep the development

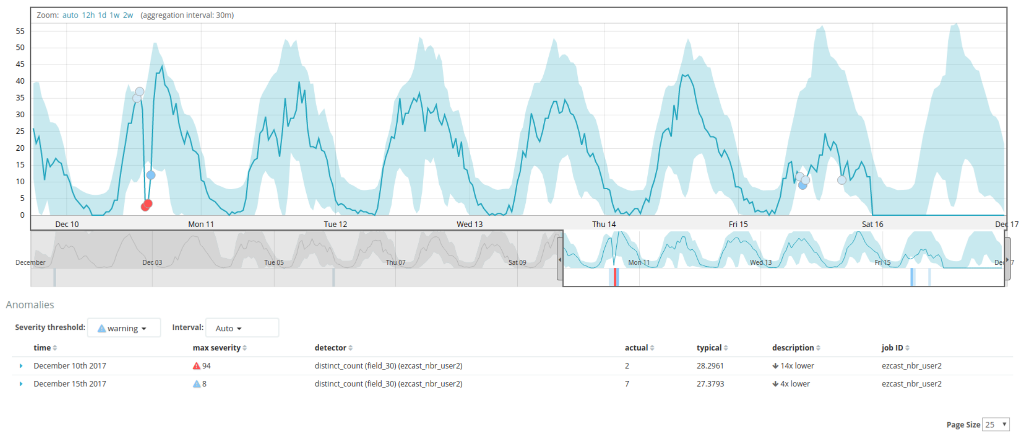
```
POST /ezcast-*/_search
{
  "size": 4,
  "query": {
    "bool": {
      "should": [
        {
          "term": {
            "action": "login"
          }
        },
        {
          "term": {
            "action": "login_as_anonymous"
          }
        }
      ]
    }
  },
  "_source": false,
  "aggs": {
    "browsers": {
      "terms": {
        "field": "browser_name.keyword"
      }
    }
  }
}
```

Figure 17: Aggregation request with simple condition

and the maintenance easy. Unfortunately there are to much modules to present all of them in this document.
Note that the error message are not always very easy to understand. However there is a great active community.

```
POST /ezcast-*/_search
{
  "size": 4,
  "query": {
    ...
  },
  "_source": false,
  "aggs": {
    "browsers": {
      "terms": {
        "field": "user_os.keyword"
      },
      "aggs": {
        "incidents_per_week": {
          "date_histogram": {
            "field": "@timestamp",
            "interval": "week"
          }
        }
      }
    }
  }
}
```

Figure 18: Aggregation request with timestamp



Anomalies

Severity threshold: warning Interval: Auto

	time	max severity	detector	actual	typical	description	job ID
▸	December 10th 2017	▲ 94	distinct_count (field_30) (ezcast_nbr_user2)	2	28.2961	↓ 14x lower	ezcast_nbr_user2
▸	December 15th 2017	▲ 8	distinct_count (field_30) (ezcast_nbr_user2)	7	27.3793	↓ 4x lower	ezcast_nbr_user2

Page Size 25

Figure 19: Machine learning

References

[DB-Engines, 2017] DB-Engines (2017). Db-engines; detailed side-by-side view of cassandra and elasticsearch and mongodb.

[Elasticsearch, 2013] Elasticsearch (2013). Blog elasticsearch: Elasticsearch from the bottom up.

[Elasticsearch, 2017] Elasticsearch (2017). Elasticsearch: official web site.

[ULBPodcast, 2017a] ULBPodcast (2017a). Github: Ezcast.

[ULBPodcast, 2017b] ULBPodcast (2017b). Website of ulbpodcast.

A Annex

A.1 Simple Logstash configuration

```
input {
  beats {
    port => "5043"
  }
}
filter {

  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
      "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level:int} \| %{WORD:action}( \| %{ANY:elem1} ?)?(\| %{ANY:elem2} ?)?(\| %{ANY:elem3} ?)?(\| %{ANY:elem4} ?)?(\| %{ANY:elem5} ?)?(\| %{ANY:elem6} ?)?(\| %{ANY:elem7} ?)?(\| %{ANY:elem8} ?)?(\| %{ANY:elem9} ?)?(\| %{ANY:elem10} ?)?"
    }
  }

  mutate {
    convert => {
      "level" => "integer"
    }
  }

  date {
    match => ["timestamp", "yyyy-MM-dd-HH:mm:ss"] # 2017-11-16-00:13:32
  }
}
output {
  elasticsearch {
    hosts => "localhost:9200"
    manage_template => false
    index => "ezcast-%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
  }
}
```

A.2 Simple Logstash configuration

```
input {
    beats {
        port => "5043"
    }
}
filter {

  # video_play_time
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| video_play_time \| %{ANY:album} \| %{ANY:asset} \| %{ANY:asset_name} \| %{ANY:type} \| %{NUMBER:start} \| %{NUMBER:play_time} ?"
    }
    add_field => { "action" => "video_play_time" }
  }

  # video_play | video_pause
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| video_(?<part_action>play|pause) \| %{ANY:album} \| %{ANY:asset} \| %{NUMBER:video_duration} \| %{NUMBER:video_time} \| %{ANY:type} \| %{ANY:quality} \| %{ANY:origin} ?"
    }
    add_field => { "action" => "video_%{part_action}" }
    remove_field => [ "part_action" ]
  }

  # video_seeked
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| video_seeked \| %{ANY:album} \| %{ANY:asset} \| %{NUMBER:video_duration} \| %{NUMBER:previous_video_time} \| %{NUMBER:video_time} \| %{ANY:type} \| %{ANY:quality} ?"
    }
    add_field => { "action" => "video_seeked" }
  }

  # login | login_as_anonymous | login_from_anonymous
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| login(?<other>|_as_anonymous|_from_anonymous) \| %{ANY:browser_name} \| %{ANY:browser_version} \| %{ANY:user_os} \| %{ANY:user_agent} ?"
    }
    add_field => { "action" => "login%{other}" }
    remove_field => [ "other" ]
  }

  # video_fullscreen_enter | video_fullscreen_exit
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| video_fullscreen_(?<other>enter|exit) \| %{ANY:album} \| %{ANY:asset} \| %{NUMBER:video_duration} \| %{NUMBER:video_time} \| %{ANY:type} \| %{ANY:quality} \| %{ANY:origin} ?"
    }
    add_field => { "action" => "video_fullscreen_%{other}" }
    remove_field => [ "other" ]
  }

  # video_forward | video_rewind
  grok {
    patterns_dir => ["/etc/logstash/patterns"]
    match => {
        "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level} \| video_(?<other>forward|rewind) \| %{ANY:album} \| %{ANY:asset} \| %{NUMBER:video_duration} \| %{NUMBER:video_time} \| %{ANY:type} \| %{ANY:quality} \| %{ANY:origin} ?"
    }
    add_field => { "action" => "video_%{other}" }
    remove_field => [ "other" ]
  }

  # All other think
  if ![action] {
    grok {
      patterns_dir => ["/etc/logstash/patterns"]
      match => {
          "message" => "%{CUSTOM_TIMESTAMP:timestamp} \| %{WORD:session} \| %{IP:ip} \| %{WORD:netid} \| %{NUMBER:level:int} \| %{WORD:action}( \| %{ANY:elem1} ?)?(\| %{ANY:elem2} ?)?(\| %{ANY:elem3} ?)?(\| %{ANY:elem4} ?)?(\| %{ANY:elem5} ?)?(\| %{ANY:elem6} ?)?(\| %{ANY:elem7} ?)?(\| %{ANY:elem8} ?)?(\| %{ANY:elem9} ?)?(\| %{ANY:elem10} ?)?"
      }
    }
  }

  mutate {
    convert => {
        "level"               => "integer"
        "play_time"           => "integer"
        "start"               => "integer"
        "video_duration"      => "integer"
        "video_time"          => "integer"
        "previous_video_time" => "integer"
    }
  }
}

output {
  elasticsearch {
    hosts => "localhost:9200"
    manage_template => false
    index => "ezcast-%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd
   }"
    document_type => "%{[@metadata][type]}"
    user => "logstash_user"
    password => "password"
  }
  stdout { codec => rubydebug }
}
```

A.3 Result of a simple request

```
{
  "took": 10,
  "timed_out": false,
  "_shards": {
    "total": 35,
    "successful": 35,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 162368,
    "max_score": 1.1393828,
    "hits": [
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS1X",
        "_score": 1.1393828,
        "_source": {
          "start": 120,
          "play_time": 30
        },
        "fields": {
          "end_time": [
            150
          ]
        }
      },
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS1e",
        "_score": 1.1393828,
        "_source": {
          "start": 2395,
          "play_time": 30
        },
        "fields": {
          "end_time": [
            2425
          ]
        }
      },
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS10",
        "_score": 1.1393828,
        "_source": {
          "start": 2306,
          "play_time": 30
        },
        "fields": {
          "end_time": [
            2336
          ]
        }
      }
    ]
  }
}
```

A.4 Result of a simple request 2

```
{
  "took": 15,
  "timed_out": false,
  "_shards": {
    "total": 35,
    "successful": 35,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 31987,
    "max_score": 2.1393828,
    "hits": [
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS16",
        "_score": 2.1393828,
        "_source": {
          "start": 5710,
          "play_time": 7
        },
        "fields": {
          "end_time": [
            5717
          ]
        }
      },
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS2X",
        "_score": 2.1393828,
        "_source": {
          "start": 2147,
          "play_time": 5
        },
        "fields": {
          "end_time": [
            2152
          ]
        }
      },
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS2u",
        "_score": 2.1393828,
        "_source": {
          "start": 2225,
          "play_time": 8
        },
        "fields": {
          "end_time": [
            2233
          ]
        }
      },
      {
        "_index": "ezcast-filebeat-6.0.0-2017.12.09",
        "_type": "doc",
        "_id": "AWBRiGhd3Hy1K0oppS3W",
        "_score": 2.1393828,
        "_source": {
          "start": 943,
          "play_time": 6
        },
        "fields": {
          "end_time": [
            949
          ]
        }
      }
    ]
  }
}
```

A.5 Result of a Aggregation

```
{
  "took": 56,
  "timed_out": false,
  "_shards": { ... },
  "hits": { ...  },
  "aggregations": {
    "browsers": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Chrome",
          "doc_count": 3071
        },
        {
          "key": "Safari",
          "doc_count": 1700
        },
        {
          "key": "Firefox",
          "doc_count": 826
        },
        {
          "key": "unknown",
          "doc_count": 410
        },
        {
          "key": "Opera",
          "doc_count": 85
        },
        {
          "key": "Internet Explorer",
          "doc_count": 79
        },
        {
          "key": "Mozilla",
          "doc_count": 73
        }
      ]
    }
  }
}
```