Building a real-world logging infrastructure with Logstash, Elasticsearch and Kibana

Patrick Kleindienst
Stuttgart Media University/Bertsch Innovation GmbH
pk070@hdm-stuttgart.de

Abstract

Talking about highly scalable and reliable systems, issues like logging and monitoring are often disregarded. However, being able to manage today's software systems absolutely requires deep knowledge about the current state of applications as well as the underlying infrastructure. Extracting and preparing debug information as well as various metrics in a fast and clearly arranged manner is an essential precondition in order to handle this task.

Since we at *Bertsch Innovation GmbH* also face increasing requirements concerning *Media-Cockpit* as one of our core products, we decided to establish a centralized logging infrastructure in order to come up to the application's evolution towards a more and more distributed system.

In this paper, I want to describe the steps that I have taken in order to setup a functioning logging tool stack consisting of Elasticsearch [16], Logstash [29] and Kibana [27] (usually abbreviated as *ELK stack*). Besides outlining proper setup and configuration, I will also discuss possible pitfalls as well as custom adjustments made when ELK did not meet our demands.

1 Introduction

1.1 Why ELK?

Searching for available solutions which meet our requirements, we finally decided to give ELK a try. ELK by *Elastic* [11] turned out to be a well-documented and established option, representing an integral platform that offers virtually everything that it takes to collect (Logstash), administer (Elasticsearch) and visualize (Kibana) large amounts of data in nearly real-time. Although this may sound like a vendor lock-in, we rather considered this an advantage, thinking that a collection of homogeneous software components saves us the need to install a bunch of potentially incompatible tools or - even worse - end up in painfully implementing an in-house solution which has to be maintained by ourselves. Moreover, a large community has clustered around Elasticsearch as well as its companions Logstash and Kibana over the last years, representing a good contact point in case of difficulties.

1.2 Goals

With these arguments in mind, we defined the following goals we wanted to achieve using ELK:

- Setting up Elasticsearch, Logstash and Kibana on a dedicated VM
- Configure Logstash to consume logging messages from log4j [3]
- Extract the root cause of incoming stack traces to speed up the identification of exception origins
- Automating the process of backing up Elasticsearch contents and removing outdated records
- Configure an alerting mechanism which sends a notification (e.g. an email) if certain anomalies (e.g. high exception rate, high average response time etc.) are detected

1.3 Prerequisites

The following explanations assume Ubuntu 14.04 as OS with Java 8 installed, since the whole ELK stack is implemented in Java and therefore requires a JRE. Setting up such an environment is beyond the scope of this paper and will not be covered below.

2 Logstash

2.1 Overview and setup

Logstash is a real-time data collection engine that is able to consume messages from many different sources like HTTP, messaging queues or several logging frameworks. Since the inputs produced by these sources have their own inherent structure, Logstash performs the task of normalizing them and bringing them in a consistent form. As soon as a message arrives, it is transformed into a JSON-like event, consisting of key-value pairs. Subsequently, any incoming event can be enriched with additional information, e.g. a simple timestamp, and its contents can also be modified. Once a message has been received and processed, it gets dispatched to one or more destinations. As for possible targets, a wide range of data stores, e.g. MongoDB, Amazon S3, Hadoop or of course Elasticsearch, are supported [29].

Figure 1 descriptively outlines the typical Logstash workflow.

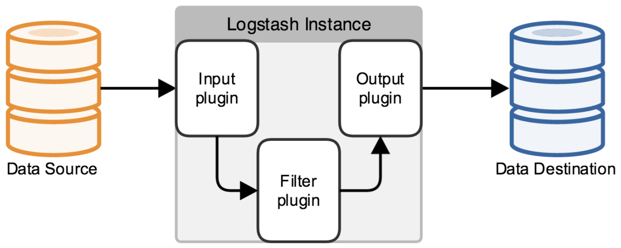

Figure 1: Logstash processing pipeline [35]

Getting Logstash up and running under Ubuntu is straightforward and can be done in three steps:

```
$ wget −qO − https://packages.elastic.co/GPG− \
  KEY−elasticsearch | sudo apt−key add −

$ echo 'deb http://packages.elastic.co/logstash/ \
  2.2/debian stable main' | \
  sudo tee /etc/apt/sources.list.d/logstash−list

$ sudo apt−get update &&
  sudo apt−get install logstash
```

Listing 1: Installing Logstash [33]

What we have to do is download the repositoy's public key, add the repository to our sources and install Logstash as soon as we have refreshed our package lists. Once the installation process has finished, the Logstash service starts automatically [33].

Please consider that, in contrast to what is shown in Listing 1, our ELK stack is based on Logstash 2.1. The installation instructions above refer to the latest Logstash documentation and therefore make use of the most recent 2.2 repository.

2.2 Consuming input messages

The running instance can now be configured according to our needs. This requires the definition of a configuration file inside the */etc/logstash/conf.d* directory. The following listing illustrates such a file's skeletal structure [1]:

```
input {
}

filter {
}

output {
}
```

Listing 2: Logstash configuration skeleton [35]

Every Logstash configuration is made up of three fundamental parts, reflecting the overall high-level principle of gathering (input), processing (filter) and forwarding (output) data as described in the previous section. A special feature of Logstash is that it completely relies on a plugin architecture for defining a message processing pipeline. This enables users to develop and integrate their own plugins. It is no accident that the configuration is completely described in JSON. This runs like a thread through the Elasticsearch ecosystem, avoiding additional complexity and promoting comprehension of what exactly is going on [35].

In our scenario, we want our Logstash instance to receive and process application logs generated by log4j. Fortunately, there is already a corresponding input plugin available which gets along with a minimal setup:

```
input {
  log4j {
    "port" => 4560
  }
}
```

Listing 3: log4j input plugin [28]

Although the log4j plugin offers lots of configuration options, Listing 3 shows everything we need as a start. As a consequence, a TCP-based server is launched at port 4560. For the rest of the parameters, the plugin already provides a set of default values which is generally convenient. [28].

The next step is to make log4j pass its messages to the freshly established TCP end point. In order to send logs over the network, log4j makes use of a SocketAppender, which is able to communicate via TCP. Listing 4 shows an excerpt of the *log4j2.xml* file, defining the SocketAppender appropriate to our Logstash settings [4].

```
<Appenders>
  <Socket name="socketAppender"
          host="{logstash-host}" port="4560">
        <SerializedLayout />
  </Socket>
</Appenders>
```

Listing 4: log4j SocketAppender configuration [4]

Even though the introduced settings worked fine at first glance, for us they came with a great drawback. Since we were focused on narrowing down incoming stack traces to only the root cause (i.e. the last appearance of *Caused by*) of a chain of exceptions, log4j's standard SocketAppender highly complicated this task. The reason is that having used this appender, we could only send exceptions and their complete stack traces in chunks instead of transmitting them in a single message. Subsequently, Logstash created an extra event for every stack trace fragment it received, therefore splitting up a logical units in disjointed events.

This behavior was not acceptable and made it very hard to retrace what went wrong on our systems, since the correlation of the individual exceptions got lost. Of course this is not just a issue induced by log4j, since the underlying transport protocols define limited package sizes (e.g. 8192 bytes for UDP datagrams) anyway [38].

To bypass this problem, we decided to switch to GELF (Graylog Extended Log Format) [38] as our log format. In short, GELF can be used as a log4j appender that creates JSON messages from application or server logs, applying a fixed set of predefined fields. Though GELF also has to go along with the limitations of the underlying transport protocol, one main characteristic is that messages exceeding the maximum package size are labeled with a message ID as well as a sequence number and therefore can be correctly reassemlbed after reception, which is exactly what we needed [38].

At this point, a in-depth view on GELF as well as its configuration as an appender and Logstash input is omitted, since there is not so much difference to what has already been shown. For further details consider [22] and [39].

2.3 Filtering logging messages

If Logstash's capabilities were limited to only receive and forward messages, integrating it into a logging infrastructure would not make so much sense. Its real power appears when it comes to alter the internal structure of incoming events, which can mean adding, removing or changing information. This can be done by using available filter plugins in our confuguration file. It is important to know that Logstash does not delimit the number of filters that shall be applied. However, consider that the filters are executed in descending order, analogous to their arrangement in the configuration file. During this process, the input of a filter is defined by the output of the preceding filter, if available [21].

Having GELF configured properly, Logstash's filter mechanism can now be applied to extract the desired information about an exception's root cause. The basic idea is to access a stack trace included in a GELF message's *StackTrace* field (where GELF lodges it by default), parse it and store the leached root cause in an additional *RootCause* field that will be added to the JSON message on the fly.

For parsing purposes, the Logstash ecosystem offers the predefined *grok* filter plugin [24]. It takes a pattern as an argument and matches it against the input it receives. Grok is built upon regular expressions and can be regared as some kind of wrapper around them. This way, the plugin can supply ready-made expressions for common message structures like HTTP communications or certain server logs, preventing its users from reinventing the wheel again and again. Nevertheless, it is still possible to pass plain regular expressions to Grok in case they are no predefined patterns that suit a special use case [24].

Because grok is based on *Oniguruma* regex library [42], it is essential to keep this engine's pecularities in mind when it comes to custom patterns. Otherwise your own regex pattern will not work. This is exactly the issue I ran into when searching for a pattern which satisfied our special requirements. The regular expression we ended up with looked like this:

```
\A[\s\S]*\nCaused by:\s* (?<RootCause>.*)\Z
```

Listing 5: Regex for root cause extraction

The pattern matches the last line of any stack trace that starts with *Caused by* and stops at the end of the corresponding line (maching the text after the colon). Anything that is captured by the expression is stored in the additional *RootCause* field.

Finally, this custom grok pattern can be used to conclude the intended filtering. This is what the following excerpt of our filter configuration illustrates:

```
filter {
  # we check if there's a stack trace available
  if [StrackTrace] != "" {
    grok {
      # Everything that is matched by the pattern is
      # stored in the new 'RootCause' field
      match => {
        "StackTrace" => "[\s\S]*\nCaused by: \
              (?<RootCause>.*:)[\s\S]*"
      }
    }
  }
}
```

Listing 6: Using a grok filter

At the beginning, it is made sure that the message received from the previously defined input actually is an exception. By the way, this demonstrates that if-clauses can be applied to restrict the execution of filter plugins to scenarios where a certain condition is met. This saves the need for executing unnecessary regex operations, which can be very expensive [19].

Since an exception does not necessarily have to be caused by another one, we agreed that in such a case, the particular exception should be treated as the root cause. Listing 7 describes the corresponding grok filter. The special thing here is that this filter is only executed if the message's *tags* field contains the *_grokparsefailure* value. This tag is added by the grok plugin to indicate that a message has been parsed, but did not match the provided pattern. In our case that can happen if a message is identified as an exception (i.e. the *StackTrace* field is not empty), but does not contain any *Caused by* clauses.

```
# check for '_grokparsefailure' tag
if "_grokparsefailure" in [tags] {
  grok {
    # assign the first line of the stack trace to the
    # 'RootCause' field
    match => {
     "StackTrace" =>
              "(?<RootCause>Exception.*?:)"
    }
  }
}
```

Listing 7: Grok filter in case of missing *Caused by* clauses

Aside from parsing exceptions, we established additional filters for removing white space or tags we did not need. For clarity, these will not be presented at this point.

2.4 Passing messages to Elasticsearch

After having received and processed a message, all that is left to do is to choose one or more destinations to store the results. As for our team, for now we were satisfied accumulating the output in a single Elasticsearch data store. Because this is a common scenario, it is not a big surprise that there is already a suitable Elasticsearch output plugin available [14].

With the aid of this plugin, establishing a running Elasticsearch server as an output does not take much more than specifying the server's host and port:

```
output {
  elasticsearch {
    hosts => "localhost:9200"
  }
}
```

Listing 8: Forwarding messages to Elasticsearch

For debugging purposes, it is also an option to make use of the stdout output plugin which prints the filter results on the command line [36]. Our report will come back to this later, since we had to rethink the plugin's settings when we started discussing about how to deal with outdated logging messages residing in our Elasticsearch instance.

3 Elasticsearch fundamentals

3.1 Elasticsearch in a mini nutshell

Diving deeper into Elasticsearch, it is important to forestall that this is probably the most complex element of the whole logging tool stack. Therefore, this section solely focuses on its very basics and fundamental functions.

At its core, Elasticsearch is a full-text search and analytics engine based on Apache Lucene [2]. Its probably greatest advantage is that it is not only a high-performance data store. Moreover, it allows searching and analyzing large amounts of data in almost real time, what answers the question of why not simply using a conventional SQL database instead. Although Elasticsearch can also be employed for recording application data, in our case it serves as a storage for logging messages it gets delivered by Logstash. This way, we are able to watch out for various patterns in our logs (e.g. high exception frequency), discovering potential problems as soon as they occur instead of noticing them after hours or even days [17].

In order to achieve fault tolerance and reliability, Elasticsearch comes with built-in mechanisms which facilitate sharding, replication and clustering. We narrowed down our experiment to a single node, since the main focus of our work lied on how to supervise growing systems rather than scaling Elasticsearch itself. Nevertheless, characteristics like robustness may not be constrained to only the applications bringing the money. Hence, this topic is one of the next points on our agenda [6].

3.2 Installation

Getting Elasticsearch up and running is just a few steps away. The first pace is to add the corresponding repository to our apt sources, like we did when setting up Logstash. Afterwards, the actual installation process can be triggered. Listing 9 summarizes the necessary commands.

```
$ wget -qO - https://packages.elastic.co/GPG-
    KEY-elasticsearch | sudo apt-key add -

$ echo "deb http://packages.elastic.co/elasticsearch
    /2.x/debian stable main" | sudo tee -a /etc/apt
    /sources.list.d/elasticsearch-2.x.list

$ sudo apt-get update && sudo apt-get install
    elasticsearch
```

Listing 9: Installing Elasticsearch [1]

Like Logstash, Elasticsearch immediately starts running after the installation procedure has finished.

3.3 Indices, types and documents

The smallest unit of information in the context of Elasticsearch is a document. A document is constituted by a single JSON object, representing a concrete instance of an arbitrary domain object. Once again, we see that JSON supersedes the need for a specific and unconversant data format [6].

For being able to group documents, Elasticsearch introduces the concept of indices. An index can be considered a collection of documents with similar characteristics. [6].

Because arranging documents under different indices is not a very granular approach, each index can besides be split up by defining one or more so-called types. A type is nothing but a logical partition of an index which allows a subtle classification of an index' documents [6].

3.4 Introducing the RESTful-API

A nice feature that makes working with Elasticsearch highly comfortable is its intuitive RESTful API, offering a clean and handy interface for creating, reading, updating and deleting indices or documents. Additionaly, rather than being restricted to only operate the data residing inside Elasticsearch, the API also enables users to query the health of an Elasticsearch instance or taking snapshots of its current state. The latter plays an imortant role in the next section, when it comes to how to backup logging data and clean up your storage by removing outdated data sets. Since every operation is just HTTP and JSON, one can choose an arbitrary REST client or even command line tools like *curl* for talking to the API [20].

As the feature set of the REST API as well as of Elasticsearch itself is highly extensive, the paper will focus on presenting a handful of common operations which will provide a basic understanding of how working with Elasticsearch feels like. For everyone who is interested in the details, the official documentation [20] is the place to go.

The first thing to be covered is creating an index and adding a sample document. Listing 10 shows the structure of a command that generates an index named *heroes*. In this scenario, *curl* is used for building and sending HTTP requests.

```
$ curl -XPUT 'localhost:9200/heroes?pretty'
```

Listing 10: Creating a sample index [7]

Any index must be given a lower case name, which requires the request to be a HTTP PUT request. It becomes apparent that Elasticsearch strictly follows the specification of a RESTful interface, since the name and the location of the desired resource (here: the newly created index) must be specified. The *pretty* URI parameter instructs Elasticsearch to return its JSON response in a properly formatted manner. Assumed that an operation has been executed successfully, this response at least contains an ACK flag [7]:

```
{
  "acknowledged" : true
}
```

Listing 11: Elasticsearch response in case of successfully executed operations [7]

The next step is to create one or more documents and add them to the new index (Listing 12).

```
$ curl -XPUT 'localhost:9200/heroes/dc-universe
     /1?pretty' -d '
{
  "name": "Batman"
}'
```

Listing 12: Creating a document [25]

The URI describes the target index (*heroes*), a mandatory type (*dc-universe*) as well as the ID (here: 1) that should be assigned to our new document. If everything worked fine, Elasticsearch sends an ACK along with a response body that reflects what exactly has been persisted:

```
{
  "_index" : "heroes",
  "_type" : "dc-universe",
  "_id" : "1",
  "_version" : 1,
  "created" : true
}
```

Listing 13: Response if storing a document has been successful [25]

Conformable to the REST specification, the ID value can be left out if a POST request is used instead of PUT. In this case, it is generated by the data store itself and published in the related response [25].

As soon as a document is no longer needed, it can be removed with a DELETE request. Listing 14 illustrates how to do this for the sample document as well as the index.

```
$ curl -XDELETE 'localhost:9200/heroes/dc- \
  universe/1?pretty'
$ curl -XDELETE 'localhost:9200/heroes?pretty'
```

Listing 14: Deleting a document as well as an index [10]

As usual, both commands are ACKed if the document and the index could have been removed without errors [10].

3.5 Elasticsearch Query DSL

Besides creating and deleting indices or documents, they can of course be accessed with GET requests by specifying the name of an index or a document's ID in the URI, which is not so different as far as the syntax is concerned and will therefore not be skipped. In fact, the true power of Elasticsearch lies in a sophisticated Search API, built on a catchy and powerful JSON-based Query DSL. Although its fundamentals are far from being complicated, it provides a rich set of features facilitating advanced and sometimes obfuscating querying. Hence, this paragraph limits itself to the very basics insofar they are relevant to understand the subsequent explanations [34].

The Search API can be addressed by sending well-formatted queries to the `http://{host:port}/{index}/_search` URI, whereupon the index part is optional. The request must be a GET request, transfering the query as its payload. If we wanted to search for the document we have created in the previous section and all we know is the value assigned to its name attribute, an appropriate query might look like this:

```
$ curl -XGET http://localhost:9200/_search?pretty
    -d
'{
   "query": {
      "match": {
         "name": "Batman"
      }
   }
}'
```

Listing 15: Retrieving a document with the Query DSL [34]

In order to live up to the amount of possible queries and search options, there is a whole chapter dedicated to this topic in the official docs [34]. For the scenario described in this paper, having a basic idea of the big picture shall be sufficient.

4 Converting data into knowledge

4.1 Data != information

So far, the paper only covered how to store and retrieve data in Elasticsearch using REST interfaces and search queries. But solely having available a vast number of application logs in a data store does not help anybody to supervise the health of applications or servers.

Imagine a scenario where an upset client is calling, complaining about his app being crashed. It seems clear that hectically firing arbitrary querys to Elasticsearch will probably not lead to fast success while searching for the cause of a failure. In order to prevent such an incident, our team agreed that its essential to be permanently informed about what is currently going on in our systems. If something goes wrong, our goal is to become aware of this in seconds. Thereby, we are able to rapidly identify the reason of an anomaly and take steps before the whole system goes down.

4.2 Elasticsearch Watcher plugin

Since implementing such a tool from scratch might take weeks, coming back to *Watcher* [15] was a logical choice. Watcher is an Elasticsearch plugin which allows to define actions that should be triggered as soon as certain criteria are met. To install the plugin, we have to run the plugin installation script that ships with Elasticsearch:

```
$ {ELASTIC_HOME}/bin/plugin install license
$ {ELASTIC_HOME}/bin/plugin install watcher
```

Listing 16: Installing Watcher [18]

Notice that a special license has to be installed for being able to use the full range of Watcher features. We will pick that up and highlight the consequences when talking about the plugin's advantages and disadvantages [18].

Putting Watcher into operation requires the definition of a so-called *watch*. Every watch consists of four substantial components:

- **Schedule:** The time interval a certain condition should be checked
- **Query:** The query that is executed on every interval and which defines the input for the condition
- **Condition:** The criterion the input is evaluated against
- **Actions:** What should be done in case the defined criterion is met (e.g. sending an email)

For testing purposes, we started with a simple use case: If an exception or error is thrown, a notification mail should be sent to a support account. Registering a watch that performs this task might look like this:

```
$ curl −XPUT 'http://localhost:9200/_watcher/ \
  watch/exception_watch' −d
'{
  "trigger" : {
    "schedule" : { "interval" : "10s" }
  },

  "input" : {
    "search" : {
      "request" : {
        "body" : {
          "query" : {
            "bool" : {
              "must" : [
                {
                  "match" : { "Severity" : "Error" }
                },
                {
                  "range" : {
                    "timestamp" : {
                      "gte" : "now−10s"
                    }
                  }
                }
              ]
            }
          }
        }
      }
    }
  },

  "condition" : {
    "compare" : {
      "ctx.payload.hits.total" : { "gt" : 0 }
    }
  },

  "actions" : {
    "send_mail" : {
      "to" : "support@mediacockpit.com"
      "subject": "Hey, something's going wrong here
          .."
    }
  }
}'
```

Listing 17: Registering a sample watch [37]

In short, Listing 17 creates a watch that is triggered every ten seconds. It queries Elasticsearch for records with *Error* severity that appeared since the last execution of the watch (*now-10s*). If the result count is greater than zero, a warning email gets dispatched. The PUT request has to address the `/_watcher` REST interface, along with the JSON payload transfering the actual definition [37].

4.3 Nothing comes for free

Disregarding that watch definitions quickly become kind of extensive, we were very excited about Watcher. It offered us easy configuration along with rich set of possible actions out of the box. But the disillusion came quick: It turned out that the license we had to install right at the beginning was limited to 30 days. After this period, a commercial license has to be purchased. As there is no other way than buying a quite expensive license which comprises the full range of available Elasticsearch plugins, we are currently working on a custom solution which shall support a small subset of Watcher's functionalities [31].

5 Managing logging data

5.1 Log retention with Curator

Another thing we had to think of was how to deal with persisted logging messages that are no longer up to date. Since a bulk of outdated information slows down the searching process and most likely is of secondary importance relating to the current state of our applications, we decided that every data set should only be stored by Elasticsearch for a customizable transitory period.

Indeed, there is no built-in feature in Elasticsearch to achieve this by e.g. setting a single parameter which tells it to drop records that are older than x days. However, when we discovered *Curator* we experienced that we were not the first team dealing with that issue. Curator is a command-line tool written in Python and available under the Apache License, Version 2.0 instead of a commercial one (see 4.3). One the one hand, it was developed in favor of offering a convenient possibility to prevent Elasticsearch from getting messed up with old data, convincing us to give it a try. As another domain, Curator aims at simplifying the creation of backups, which will be covered in the next section [13].

Curator is a standalone application rather than an Elasticsearch plugin and the recommended way of installing it is via Pythons pip package management tool [9]:

```
$ pip install elasticsearch−curator
$ curator −−version
```

Listing 18: Installing Curator [9]

5.2 Rethinking our indices

While exploring the Curator API, we became aware that it does not support the appliance of any Query DSL statements to explicitly select entries which shall be deleted. Instead of being that granular, Curator's approach is purely index-based, meaning that an index is the smallest unit of data it can handle. This behavior might be sufficient in most cases, but for us, problems arose concerning the default pattern used by Logstash's Elasticsearch output plugin to build indices. Before an event gets forwarded, it creates the *_index* field and assigns it a value following the pattern *logstash-{YYYY.MM.dd}*. As a consequence, all the logging messages passing Logstash at the same day are collected under the same index, regardless of which application or server they originally came from. In the context of Curator, this lead to the implication that there would be no option for us to define a dedicated expiration date per application, because our indices were completely date-based [14].

As we considered altering the index naming pattern much simpler than implementing our own Curator clone, we watched out for how to put that into practice. Fortunately, the GELF application logs already provided information about the origin app by themselves, storing unique identifier inside the *facility* field.

```
output {
  elasticsearch {
    index => "%{facility}−%{+YYYY.MM.dd}"
  }
}
```

Listing 19: Adapting the Elasticsearch index pattern

The plugin's *index* option allows the Elasticsearch index to be manually configured. So we defined an index based on the app's denomination and the current date and were finally ready to focus on our cleanup efforts with Curator.

By the use of our redefined index, we were able to elaborate individual cleanup strategies for several running applications. Assuming there is a virtual webapp that produces logging messages with *'myWebapp'* as a facility value, invoking Curator with the following command removes every entry under this index which is older than 14 days [8]:

```
$ curator delete indices −−regex 'myWebapp−∗' \
  −−older−than 14 −−time−unit days \
  −−timestring '%Y.%m.%d'
```

Listing 20: Cleaning up a sample index with Curator [8]

A nice secondary effect of the adjusted index pattern is that it scales very well. Any appliaction which wants to transfer its logs to the logging tool stack simply needs to define a facility value in its GELF messages and automatically receives a dedicated Elasticsearch index. So all the logs produced by a certain application can be managed independently.
For the last step, the command shown in Listing 20 was installed as a cronjob on the Ubuntu VM, since relying on spaced manual execution becomes redundant that way.

5.3 Backups

Backing up Elasticsearch contents makes sense for two reasons: First, ereasing data permanently when cleaning up Elasticsearch maybe is not the best idea, since even outdated logs at a later date may shed light on anomalies that could not be detected immediately. Moreover, if a single node crashes for some reason, it can be restared from a stable state by means of up to date snapshots.

In opposition to what we did when discussing the treatment of overage data sets, we did not make use of Curator for our backups, although this is another common use case of this tool. The reason is that Curator lacks the possibility to send notifications (e.g. an email) giving information about at which time the backup process has been triggered and if it has been successfully conducted. However, we did not want to disclaim that feature and therefore determined to develop *ElasticArchiver* [43], our own Python tool, which initiates the backup procedure, stores the result (= snapshot) to the local file system and sends a short email, advising the responsible staff of the outcome.

For convenience, the script uses the *requests* Python library [40] and communicates with Elasticsearch via HTTP. It addresses the Snapshot API available under the `/_snapshot` URI, which is yet another RESTful interface provided by Elasticsearch [5].

Before any backups can be saved, there must at least exist a single repository. A repository can be a distributed file system, a storage service like Amazon S3 or even the local file system, which has been our choice as a start. Listing 21 shows how to create a such a repository by means of a PUT request [5].

```
$ curl -XPUT 'http://localhost:9200/_snapshot/
      my_repo' -d '{
    "type": "fs",
    "settings": {
        "location": "/var/data/elasticsearch/backup"
    }
}'
```

Listing 21: Creating a backup repository [5]

The request body includes information about the repositiy type (*fs* = local file system) as well as an absolute path determining where it should be located [5].

All that is left to do is storing a snapshot through another PUT request, addressing the newly generated repository. The request might look like this [5]:

```
$ curl -XPUT 'http://localhost:9200/_snapshot/
      my_repo/snapshot_A'
```

Listing 22: Saving a snapshot to a repository [5]

In this case, the snapshot named *snapshotA* contains all the records of every existing Elasticsearch index. It might be mentionable that if only certain indices should be considered, this can be specified inside an optional request body [5].

Listing 23 conveys an impression of how ElasticArchiver builds upon the available REST interfaces. It shows one method of its public API which takes care of sending the snapshot request, receives a corresponding HTTP response and transfers it to an evaluation routine, which is responsible for examining the status code of the reponse and triggering the confirmation email sending procedure.

```
def startBackupProcess(self):
    logging.info("Starting backup
        process..")

    """sending a request which
        triggers the backup creation"""
    response = self.
        __sendBackupRequest()

    """check response and send an
        email with the result"""
    self.__evalResponse(response)
```

Listing 23: ElasticArchiver public API

The *sendBackupRequest* method simply assembles the URI (Listing 24) and sends the PUT request after checking if the target repository is ready (i.e. the repository already exists). This is also a nice feature we have been missing in Curator, since it implements no fallback behavior in case a repository is not available yet. If the addressed repository is missing, ElasticArchiver creates it on the fly and then stores a snapshot to it.

```
def __sendBackupRequest(self):
    repoStatus = self.
        __isRepositoryReady()
    if(repoStatus):
        response = requests.put(
        self.__elasticURI
        + "/_snapshot/"
        + self.__repositoryName
        + "/backup_"
        + self.__getCurrentDateTime(),
        headers=self.__requestHeader)
        return response
```

Listing 24: ElasticArchiver public API

For an in-depth insight into ElasticArchiver, consider the source code which is availbale on Github under `https://github.com/PaddySmalls/elastic_archiver`.

6 Kibana

6.1 Basics and installation

The third component of the ELK stack is Kibana, which constitutes a highly configurable visualization platform for Elasticsearch data. It allows to import persistent logs and depict the results in a understandable and intuitive manner, offering lots of illustration possibilities like charts, diagrams and tables. Thus, getting a rough overview of a server cluster and its state only demands interpreting a user-friendly dashboard instead of having immerge into Elasticsearch internals [26].

Already having installed Logstash and Elasticsearch, setting up Kibana is business as usual and can be done as shown in Listing 25 [23].

```
$ echo "deb http://packages.elastic.co/kibana/4.4 \
  /debian stable main" | sudo tee −a /etc/apt/ \
  sources.list
$ sudo apt−get update && sudo apt−get install \
  kibana
```

Listing 25: Installing Kibana [23]

After startup, Kibana asks for at least a single default index pattern before it takes up employment. This index pattern may correspond to an existing index, but can also be a regular expression matching several of them. Kibana then uses this information to query all the data sets stored under the appropriate index or indices. For example, defining *logstash-** as an index pattern prompts the import of every entry whose index starts with the this prefix [23].

Since establishing searching patterns concerning our facility-based indices is almost everything we did for a start, a detailed discussion about Kibana's visualization capabilities shall be skipped at this point. Of course, Kibana offers a terrific number of possibilities to prepare and illustrate information, which is essential in order to get value out of the vast amount of logs produced by servers and webapps. Simultaneously, this is also a matter of individual preferences and requirements in the first place, and we are also still working on a dashboard that works best for us.

6.2 The "Export Everything" exception

When we launched Kibana for the first time, there was an interesting bug we ran into. After having defined our first index pattern, everything that has been displayed by the browser was a blank page, decorated by the Kibana banner. The browser console reported an *internal server error* along with the HTTP status code 500, so there must have been a problem with the Elasticsearch server. Checking its log files, the error log actually revealed an exception as the cause of the discovered misbehavior. Summing up, the error message told us something like:

Result window is too large [..] must be less than or equal to: [10000] but was [2147483647].

Indeed, we were able to retrace a request sent by Kibana, including a *max_result_window* parameter with a value of 2147483647 (which by the way equals Java's *Integer.MAX_VALUE* or $2^{31} - 1$) assigned.

At the point we faced this issue, we could not find a community-proved workaround to resolve the problem. However, because we accidentially installed Kibana 4.3 instead of the latest version 4.4, the error vanished after an update to the latest version. In contrast to the previous version, Kibana 4.4 employed a considerable lower value (i.e. 10000) for the *max_result_window*, which worked perfectly.

It was a few weeks later when an official bug report appeared in the issue section of the Kibana Github repository (issue #5524, *Kibana 4.3 "Export Everything" exception*) [41], having suggested that the bug seemed to be related to Kibana rather than Elasticsearch. The discussion in the ticket points out that the value for the result window has been reduced with Elasticsearch 2.1.0. Further investigation on the impact of the result window as well as the reason for its adaption guided us to another discussion in the Elasticsearch forum [12] that explains the background in detail. According to the latest contribution, reducing the result window as much as possible is a common practice to avoid the waste of resources due to deep paging, which results in discarding the bigger part of the search results since the majority of the data sets is merely needed to encounter what has to be displayed on a requested page. The higher the number of the requested page, the larger the amount of data that is thrown away. For more information on this topic, please consider [12], where an exemplary calculation can be found.

Obviously, the Kibana developers forgot about adjusting the result window request parameter at first. However, simply blaming Kibana for this kind of error is to easy. So far, we could not find an answer to the question, why Elasticsearch disregared the implementation of a fallback in order to guarantee downward compatibility.

7 Conclusion

For us, building our logging infrastructure from scratch using ELK turned out as success. We especially learned to appreciate the advantages of a homogeneous tool stack, since joining individual components on to a functioning system could not have been easier. Moreover, ELK benefits from its large community, which made it grow to a wide ecosystem consisting of additional tools and plugins. Subsequently, it is no big deal to tie new as well as legacy applications to ELK because of its support for diverse logging frameworks and message formats. In our case, getting our *Media-Cockpit* staging systems to work with ELK demanded a minimum of extra configuration, since all we needed was another log4j Appender.

Another advantage lies in the possibility to prepare logs in a way that there is no need for a user to be a software or systems engineer in order to understand what going on inside a system.

Unfortunately, the *Elastic* business model does not provide that their complete set of tools comes for free. Nevertheless, the Elasticsearch REST API particularly represents a great starting point for putting your own ideas into practice, as we proved ourselves with our *ElasicArchiver* implementation.

8 Further thoughts

8.1 How to supervise a logging infrastructure?

A nontrivial question that emerged during our work on a monitoring system was: How can we ensure that not just our applications, but also our logging systems are highly available and reliable? On closer inspection, a sample web application consisting of a web server, an application server as well as a database server is not so different from an ELK stack, since both are distributed systems which may fail due to arbitrary errors. By means of ELK, we can reduce a webapp's down time and therefore minimize the associated business risks. However, what happens if the monitoring system itself falls down? At first sight, this might be not as worse as application failures. However, consider that as long as the monitoring does not work, your systems are on a blind flight. If anything goes wrong before the logging infrastructure has recovered, this might have major implications. The more complex a distributed system, the more reliable we need our monitoring systems to be.

How to get a way from this dilemma? As for Elasticsearch, *Marvel* [32] constitutes a possible approach. It establishes agents on every Elasticsearch instance as well as Kibana and uses them to aggregate information about the nodes' health state, which is then displayed on the Marvel dashboard which integrates into Kibana. Though, Marvel is also part of the Elastic license bundle and thus not always affordable for every small or medium-sized company. Moreover, Marvel becomes useless in case Logstash or Kibana fail. So, what to do?

Elastic encounters these challenges with a mixture of sophisticated protection measures and redundancy. Logstash for example builds upon a ingenious thread model to protect itself from overload and achieve fault tolerance. Besides, just like Elasticsearch, Logstash can easily scale out to multiple instances, which are able to balance the load across several Elasticsearch nodes on their part [30].

On the whole, ELK is engineered to be as reliable and fault tolerant as possible. As employing another monitoring infrastructure to supervise an existing monitoring system does not really make sense due to the problem's recursive nature, replication and fault tolerance might be the best approach at this juncture. However, especially replication always requires the availability of additional resources and capacity. Therefore, moving an ELK stack to a cloud provider may be a worthwhile alternative especially for smaller organizations. The Elastic company itself offers the possibility to host Elasticsearch and Kibana on their server infrastructure, but also

Amazon Web Services or DigitalOcean might be a reasonable alternative. This way, a great part of the risk to experience a total failure can at least be delegated and subsequently underlies a provider's responsibility.

This paper can neither provide a final answer the question introduced at the beginning, nor give any hints how to come up to a logging infrastructure that is 100 percent reliable and available. However, it will be interesting to observe how things will evolve from status quo.

8.2 Monitoring as a feature

Another interesting thought centers on regarding monitoring systems not only as an additional burden that is needed to keep servers and applications into operation, but also as a part of a company's business model. On the one hand, monitoring can be inlcuded into an organization's marketing strategy, implicating a high level of confidentiality in respect of the products and services it offers. On the other hand, especially Kibana has the potential to play an important role as far as the communication with customers is concerned. Because supervising a system with Kibana can also be done by non-experts, an interested customer could always check the health state of the services he or she purchased. Thereby, an application could be more than a black box for a customer, which would always be involved and would be able to argue on the basis of metrics instead of just calling and complaining that something does not work as expected. As a consequence, e.g. a vendors SLAs could be transparently verified, creating confidence between the two parties. Both sides might profit from such an approach.

References

[1] Mitchell Anicas. *How To Install Elasticsearch, Logstash, and Kibana (ELK Stack) on Ubuntu 14.04*. DigitalOcean. Mar. 10, 2015. URL: `https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14-04` (visited on 02/29/2016).

[2] Apache Software Foundation. *Apache Lucene*. Apache LuceneTM 5.5.0 Documentation. 2016.

[3] Apache Software Foundation. *Log4j – Log4j 2 Guide - Apache Log4j 2*. 2015. URL: `http://logging.apache.org/log4j/2.x/` (visited on 03/28/2016).

[4] Apache Software Foundation. *Log4j2 Appenders*. 2015. URL: `https://logging.apache.org/log4j/2.x/manual/appenders.html#SocketAppender` (visited on 03/01/2016).

[5] Elastic. *Backing Up Your Cluster*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/guide/current/backing-up-your-cluster.html` (visited on 03/29/2016).

[6] Elastic. *Basic Concepts*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html` (visited on 03/09/2016).

[7] Elastic. *Create an Index*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/_create_an_index.html` (visited on 03/10/2016).

[8] Elastic. *Curator Examples*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/client/curator/current/examples.html` (visited on 03/29/2016).

[9] Elastic. *Curator Installation*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/client/curator/current/installation.html` (visited on 03/29/2016).

[10] Elastic. *Delete Index*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-delete-index.html` (visited on 03/10/2016).

[11] Elastic. *Elastic - Home*. 2016. URL: `https://www.elastic.co/` (visited on 03/01/2016).

[12] Elastic. *Elastic: Index max_result_window*. Discuss Elasticsearch, Logstash and Kibana | Elastic. 2016. URL: `http://discuss.elastic.co/t/index-max-result-window/38388` (visited on 03/20/2016).

[13] Elastic. *elastic/curator*. GitHub. 2016. URL: `https://github.com/elastic/curator` (visited on 03/29/2016).

[14] Elastic. *Elasticsearch*. 2016. URL: `https://www.elastic.co/products/elasticsearch` (visited on 03/01/2016).

[15] Elastic. *elasticsearch*. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html` (visited on 03/08/2016).

[16] Elastic. *Elasticsearch Reference - Getting Started*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html` (visited on 03/09/2016).

[17] Elastic. *Elasticsearch Watcher: Introduction*. 2016. URL: `https://www.elastic.co/guide/en/watcher/current/introduction.html` (visited on 03/11/2016).

[18] Elastic. *Elatsicsearch Watcher - Getting Started*. 2016. URL: `https://www.elastic.co/guide/en/watcher/current/getting-started.html` (visited on 03/11/2016).

[19] Elastic. *Event Dependent Configuration*. 2016. URL: `https://www.elastic.co/guide/en/logstash/2.2/event-dependent-configuration.html` (visited on 03/29/2016).

[20] Elastic. *Exploring Your Cluster*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/_exploring_your_cluster.html` (visited on 03/10/2016).

[21] Elastic. *Filter plugins*. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/filter-plugins.html` (visited on 03/04/2016).

[22] Elastic. *gelf*. Dec. 10, 2015. URL: `https://www.elastic.co/guide/en/logstash/current/plugins-inputs-gelf.html` (visited on 03/04/2016).

[23] Elastic. *Getting Kibana Up and Running*. 2016. URL: `https://www.elastic.co/guide/en/kibana/current/setup.html` (visited on 03/29/2016).

[24] Elastic. *grok*. grok. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html` (visited on 03/04/2016).

[25] Elastic. *Index and Query a Document*. 2016. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/current/_index_and_query_a_document.html` (visited on 03/10/2016).

[26] Elastic. *Introduction*. 2016. URL: `https://www.elastic.co/guide/en/kibana/current/introduction.html` (visited on 03/18/2016).

[27] Elastic. *Kibana*. 2016. URL: `https://www.elastic.co/products/kibana` (visited on 03/01/2016).

[28] Elastic. *log4j*. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/plugins-inputs-log4j.html` (visited on 03/03/2016).

[29] Elastic. *Logstash*. 2016. URL: `https://www.elastic.co/products/logstash` (visited on 03/01/2016).

[30] Elastic. *Logstash Processing Pipeline*. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/pipeline.html` (visited on 03/20/2016).

[31] Elastic. *Managing Your License*. 2016. URL: `https://www.elastic.co/guide/en/watcher/current/license-management.html` (visited on 03/29/2016).

[32] Elastic. *Marvel Documentation [2.2]*. 2016. URL: `https://www.elastic.co/guide/en/marvel/current/index.html` (visited on 03/20/2016).

[33] Elastic. *Package Repositories*. 2016. URL: `https://www.elastic.co/guide/en/logstash/current/package-repositories.html` (visited on 02/29/2016).

[34] Elastic. *Query DSL*. 2016. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html (visited on 03/10/2016).

[35] Elastic. *Setting Up an Advanced Logstash Pipeline*. 2016. URL: https://www.elastic.co/guide/en/logstash/current/advanced-pipeline.html (visited on 02/29/2016).

[36] Elastic. *stdout*. 2016. URL: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-stdout.html#plugins-outputs-stdout (visited on 03/09/2016).

[37] Elastic. *Watch Log Data for Errors*. 2016. URL: https://www.elastic.co/guide/en/watcher/current/watch-log-data.html (visited on 03/11/2016).

[38] Graylog. *GELF*. Graylog2/documentation. Feb. 15, 2016. URL: https://github.com/Graylog2/documentation (visited on 03/04/2016).

[39] Graylog. *Graylog2/log4j2-gelf*. GitHub. Mar. 4, 2016. URL: https://github.com/Graylog2/log4j2-gelf (visited on 03/04/2016).

[40] Kenneth Reitz. *Requests: HTTP for Humans — Requests 2.9.1 documentation*. 2016. URL: http://docs.python-requests.org/en/master/ (visited on 03/29/2016).

[41] *Kibana 4.3 "Export Everything" exception · Issue #5524 · elastic/kibana*. GitHub. Dec. 9, 2015. URL: https://github.com/elastic/kibana/issues/5524 (visited on 03/20/2016).

[42] kkos. *oniguruma*. GitHub. 2016. URL: https://github.com/kkos/oniguruma (visited on 03/29/2016).

[43] Patrick Kleindienst. *PaddySmalls/elastic_archiver*. GitHub. 2016. URL: https://github.com/PaddySmalls/elastic_archiver (visited on 03/29/2016).