Microservices log gathering, processing and storing

Siim-Toomas Marran
Univeristy of Tartu
J.Liivi 2
Tartu, Estonia
siimtoom@ut.ee

ABSTRACT

The aim of this work is to investigate and implement one of the possibilities of logging microservices in a centralized manner. Microservices are distributed pieces of software, which contain valuable information about the product and accessing the logs singly is a hideous task. Therefore the project experiments with popular Elastic's technology stack containing Logstash, Elasticsearch and Kibana.

Keywords

Microservices, Logs, Docker, Logstash, Elasticsearch, Kibana

1. INTRODUCTION

Microservices are in the core distributed systems, in more precise definition it is a software architecture style, where we put pieces of functionality into separate services. Often these services are in their own containers (e.g. Docker) and often they are in their own separated servers. Keeping those pieces of software as distributed, independent and finally stateless - instance of a service can be created, stopped, restarted, and destroyed at any time without impacting other services. For developers it is difficult to debug through the logs of multiple services to determine the origin of the problem. It becomes tedious work to log into multiple servers to access application logs manually. Therefore it is worthwhile to aggregate the logs into one central environment, where they could be examined. The project has one simplistic aggregate gateway application, which communicates with one microservice. The logs produced in the both parts shall be accumulated into third environment.

2. TECHNOLOGIES

2.1 Java technology stack

In the project, the microservices core technology is Java framework called Spring Boot[6], which is being used to create stand-alone, production-ready applications. It has distanced itself from old WAR era and lets you create applications, which already have embedded Tomcat, Jetty or Undertow server and therefore runs directly from JAR file. One of the key values is the creation of the application through the Spring Boot homepage with a selection among of multiple different compatible technologies. It creates according to the user's preferences Maven or Gradle configuration file. The technology is developer friendly, allowing automated Spring configuration and provides production-ready features as metrics, health checks and externalized configuration.

Front-end of the application has been covered with Thymeleaf[10], a modern server-side Java template engine. Helps out to develop modern-day HTML5 JVM web pages.

Hibernate[7] is an object-relational mapping framework for Java. It matches plain old Java classes to database's tables, meanwhile providing various data query and retrieval solutions.

Jersey[8] is technology to develop RESTful web services in Java. It consists of components to develop server side web services and client part to communicate with REST services, while provides frameworks to map classes into XML and JSON.

Log4j2[9] is a logging software with very powerful functionality. The second version comes with improved performance, better architecture for development and future compatibility to newer versions (separated API).

2.2 MariaDB

MariaDB[5] is a relational database management system, which has been forked from the MySQL after its acquisition by Oracle. The goal is to maintain high compatibility with MySQL. In this project it has been used to persist data in one of the microservices, while containerized with Docker.

2.3 Docker

Docker[1] is a lightweight software containerization platform, which is being used mainly for standardized software development - guarantees that the software is always running the same in every environment. The containers, in which the developer developed software runs, have a complete filesystem, which contain everything what is needed for running the code and that in the bare minimum fashion, no extra background processes like in virtual machines. More and more are companies moving towards the practice that they are using Docker instances in production because of its easiness to add new parallel instances for scalability.

In the project we are using Docker for simulating the distributedness of the entire product. Microservices, database,

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Logstash, Elasticsearch and Kibana are being containerized.

2.4 Logstash

Logstash[4] is a tool for collecting, processing and forward events and logs. It has wide variety of plugins to enhance and configure the pipeline. It supports the collection through multiple methods like raw sockets, packet communication, file tailing and several message bus clients. After the collection you can define the processing part, for example you can introduce filters to modify the data. After the processing you can define the forwarding. Usually Logstash is being used with another Elastic product called Elasticsearch, which stores the events and logs, but many other external programs and message bus implementations are allowed.

2.5 Elasticsearch

Elasticsearch[2] is a search engine for real time searching and analyzing data, it bases on Apache Lucene open-source information retrieval software library. Apache Lucene gives a possibility to do full text indexing and adds searching capability through the indexed data. Elasticsearch provides distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elastic provides Kibana software to inspect the stored events and logs in Elasticsearch.

While providing real-time advanced analytics, Elasticsearch has been architectured to be massively distributed and that by simply scaling horizontally with adding new nodes. Another good feature during the era of big data is that the clusters of Elasticsearch are resilient - maintaining high availability all time. There is going on a constant monitoring and if something changes (new node, failing) the cluster reorganizes data automatically. Some characteristic buzzwords: multitenancy, full-text search, document-oriented and schema-free.

2.6 Kibana

Kibana[3] is another product from Elastic's technology stack. It lies on top of Elasticsearch and therefore provides real-time summary and charting of streaming data. Kibana is a flexible analytics and visualization platform. Flexibility comes from the fact that it has been architected to work with Elasticsearch and Elastic has invested lots of energy to achieve seamless integration between two products. Accessing the events and logs from Elasticsearch in various forms, the developers could interpret large volumes of data in huge variety of sophisticated analytical charts, plots, histograms, maps etc.

It is easy to setup Kibana, with Docker it requires one line command in Linux environment. Kibana has create visual tools to interact with Elasticsearch REST APIs and therefore through Elasticsearch real-time properties' Kibana suits very well for production software's monitoring.

3. IMPLEMENTATION OF TEST ENVIRONMENT

3.1 Application design

The application has been implemented with a vision to keep all the elements separated in their respective Docker containers. In Figure 1 we can see the design of the application, where we have defined the boundary of entire product. There are two Docker instances, which are visible for the user. One contains of Spring Boot application, which aggregates all the microservices inside of the application boundary into a product. Second visible instance is for monitoring and for that we have Kibana. In this test environment we have one microservice behind the aggregator with two Docker instances, one with Spring Boot, which manages database entities of MariaDB Docker instance, the second instance. Spring Boot applications contain Log4j2 frameworks for logging and are configured to transport their respective logs to Logstash pipeline. Logstash processes the logs and passes them to Elasticsearch, which persists the logs. Kibana communicates with Elasticsearch API and provides the logs to the user to see.

3.1.1 Microservices idea and database design

In Figure 1 we can see one defined service boundary. That Spring Boot application is one microservice, which has fairly simple functionality - CRUD solution for managing memory game's questions. The microservice sits on top of MariaDB database, which is visible in Figure 2. It contains of questions and its respective author, difficulties and packages (e.g. geography, history etc).

3.1.2 Aggregate gateway application

Aggregate gateway application is an application, which aggregates inner services into one formidable product for outer world use. In Figure 1 we can see the Spring Boot application being on the outer communication boundary dash line. The application intermediates the communication between the user and the memory game's CRUD microservice. In Figure 3 there is a visible the user interface of the gateway application with some question examples. It lets to add new game questions and see the previous insertions.

3.2 Integration and logging

3.2.1 Transporting with Log4j2

Two Spring Boot applications produce logs with a wide variety: Spring framework, Jetty server access, Hibernate and the custom application logs. The logs are vital information for the people, who need to monitor the situation of microservices. To transfer the logs we are using Log4j2 framework, which we configure with a configuration file - *log4j2.xml*. In the configuration file we define loggers and appenders. Appenders' task is to deliver logging events to their destination. Destination could be a file, console, some pipeline like in our case - Logstash. Loggers are instances, which according to the configuration (defined rules about logging levels, packages etc) output and by using appenders they deliver the log events to the destination.

In Figure 4 we are seeing a syslog appender, which writes its output to a remote destination. In our example we are using UDP protocol for sending the log events over the network to respective IP and port. Also we see the logger data fields, which are being transported for the end consumer.

3.2.2 Processing with Logstash

In Figure 4 we can see Logstash's destination address. On this IP and port is a pipeline, which processes the log events and passes them to Elasticsearch. In Figure 5 we can see two messages from two different IP's, which are our Spring Boot applications. The pipeline has three big attributes defined

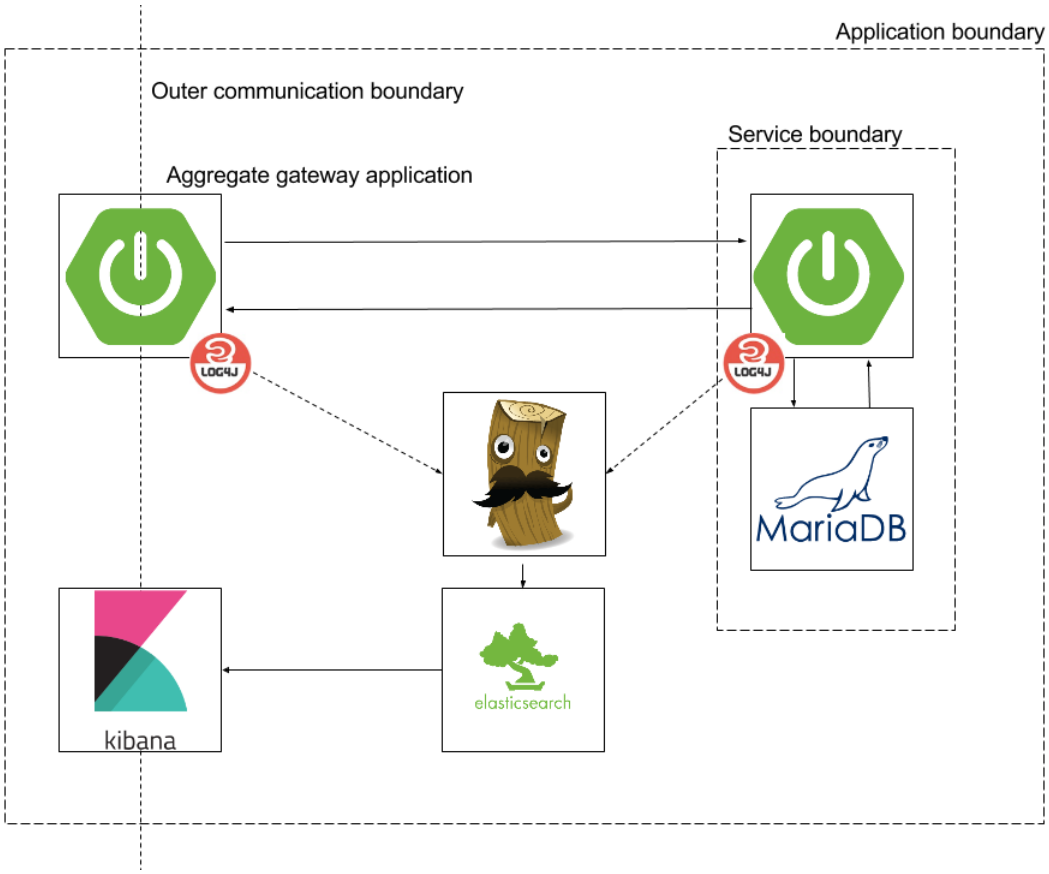


Figure 1: Application design

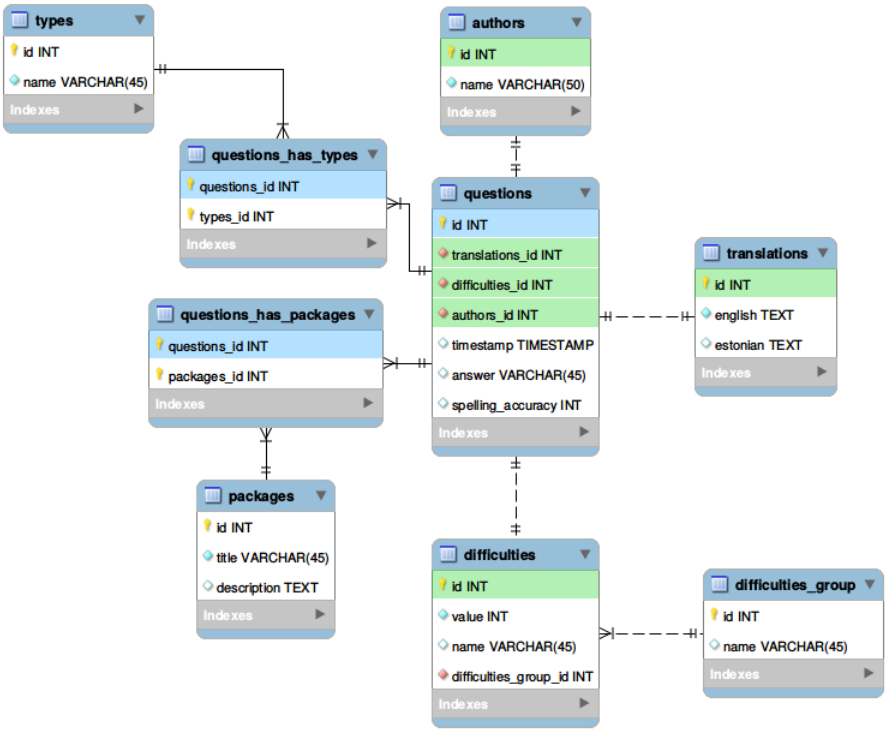
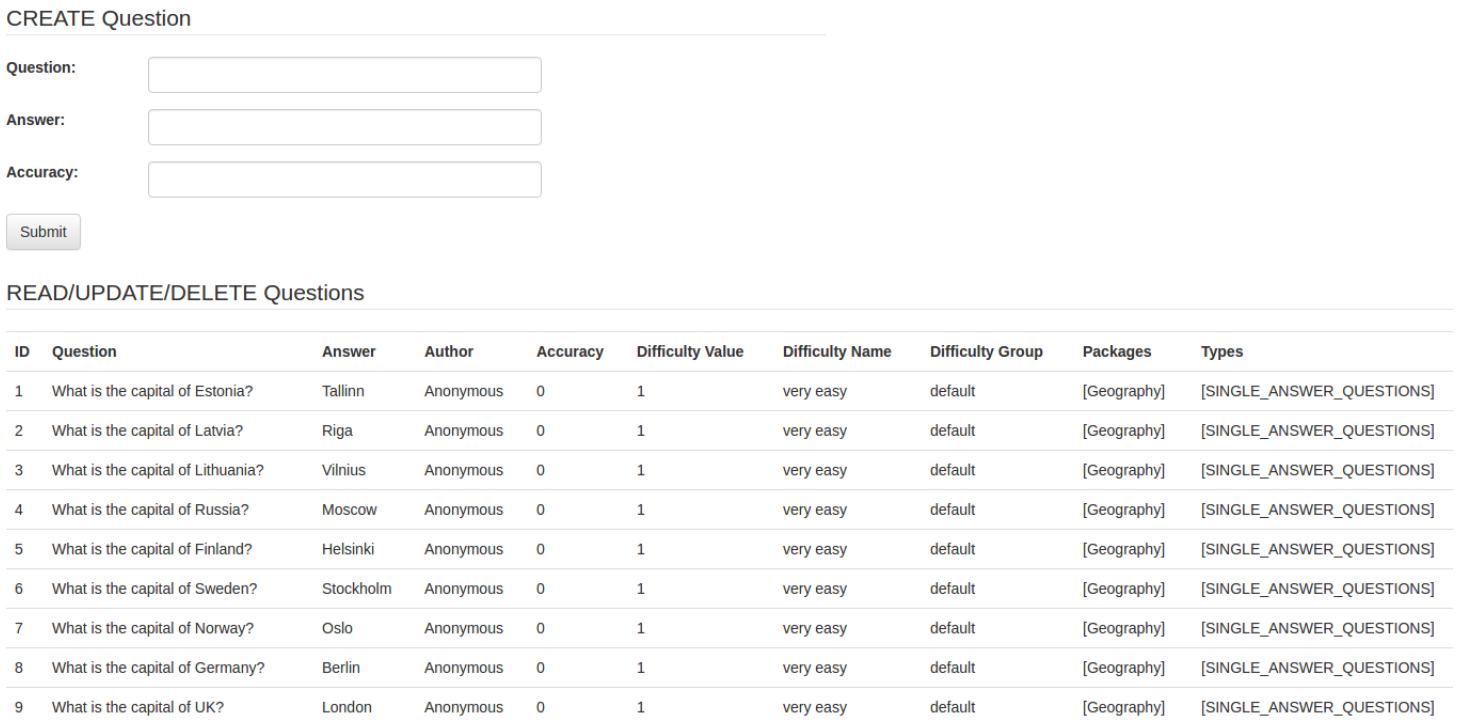


Figure 2: Microservice's database diagram



ID	Question	Answer	Author	Accuracy	Difficulty Value	Difficulty Name	Difficulty Group	Packages	Types
1	What is the capital of Estonia?	Tallinn	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
2	What is the capital of Latvia?	Riga	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
3	What is the capital of Lithuania?	Vilnius	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
4	What is the capital of Russia?	Moscow	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
5	What is the capital of Finland?	Helsinki	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
6	What is the capital of Sweden?	Stockholm	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
7	What is the capital of Norway?	Oslo	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
8	What is the capital of Germany?	Berlin	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]
9	What is the capital of UK?	London	Anonymous	0	1	very easy	default	[Geography]	[SINGLE_ANSWER_QUESTIONS]

Figure 3: The gateway application's user interface

```
<Syslog name="RFC5424" format="RFC5424" host="172.17.0.5" port="5
protocol="UDP" appName="questionnaire" includeMDC="true"
facility="SYSLOG" enterpriseNumber="18060" newLine="true"
messageId="log4j2" mdcId="mdc" id="App"
connectTimeoutMillis="1000" reconnectionDelayMillis="5000">
    <LoggerFields>
        <KeyValuePair key="thread" value="%t"/>
        <KeyValuePair key="priority" value="%p"/>
        <KeyValuePair key="category" value="%c"/>
        <KeyValuePair key="exception" value="%ex"/>
        <KeyValuePair key="message" value="%m"/>
    </LoggerFields>
</Syslog>
```

Figure 4: The syslog appender

```
{
        "message" => "<47>1 2016-08-10T08:57:56.307Z 526fc5268a47 c
mdc@18060 category=\"com.zeeble.questionnaire.rest.QuestionnaireSe
message=\"Populating DTOs\" priority=\"DEBUG\" thread=\"http-nio-8
g DTOs ",
       "@version" => "1",
    "@timestamp" => "2016-08-10T08:57:56.307Z",
          "type" => "log4j2",
          "host" => "172.17.0.3"
}
{
        "message" => "<47>1 2016-08-10T08:57:56.423Z a7db99149093 c
060 category=\"com.zeeble.gateway.rest.clients.QuestionnaireClient
ge=\"return values: 9\" priority=\"DEBUG\" thread=\"http-nio-8090-
: 9 ",
       "@version" => "1",
    "@timestamp" => "2016-08-10T08:57:56.423Z",
          "type" => "log4j2",
          "host" => "172.17.0.4"
}
```

Figure 5: The Logstash pipeline logs

in the configuration file: input, filter and output. Input and output are straightforward in terms of what they are for - defining incoming ports and types for the different log event types and the destination, for our case we defined Elasticsearch's address. In the filtering you can mutate different logs according to their origin, there are many plugins and possibilities. In our example we are just substituted the tabs and newline elements out of the delivered log message.

3.2.3 *Saving with Elasticsearch*

In Figure 6 we see the information of Elasticsearch. Logstash's pipeline forwards all the log events to the cluster, which currently consists of one node, and indexes and stores the deliverables. In our project's scope is not to change or optimize Elasticsearch's configuration parameters.

3.2.4 *Visualizing with Kibana*

Kibana connects with Elasticsearch extremely easy, needs only the address of the search engine and the rest is done by the product seamlessly. In Figure 7 we are seeing all the log events in live, plus a default graph showing the volume of incoming data. These logs can be also filtered with Kibana by using some regex patterns. To get some specific and more information complex views you should use some regex patterns to filter out the events with which you are interested in. In Figure 8 you can see the possibilities what to do with your log events. If your log events contain some geographical data then you could use map visualization, for example.

```
{
  "name" : "War",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.3.3",
    "build_hash" : "218bdf10790eef486ff2c41a3df5cfa32dadcfde",
    "build_timestamp" : "2016-05-17T15:40:04Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```


Figure 6: The Elasticsearch's version and the indicator that the system is up

In our scope we used a line graph and produced a counter to see how huge is the volume, what is going through our applications. In Figure 9 we can see one window from the dashboard. This window has been made for a testing purpose and shows the very primitive level what Kibana can do. The problem, which is out of the scope, is that there needs to be done excessive analysis what kind of logs should be stored and passed along the expensive network and in what form as well, because later on creating the dashboard views gets dreadful.

4. RESULTS

We finish with a test environment, which has been dockerized into separate Docker instances, as we can see in Figure 1. The entire application has been put into the same situation like the distributed systems in goal to imitate the challenges, which rise and trouble the developers in such situations. Docker has been proved to be fairly easy tool for containerizing your own custom applications, but the biggest bonus for me is the community, which has created many official images from various software from which we could run third party services against what we need to develop our own, like in this project with MariaDB database, Logstash, Elasticsearch and Kibana. There were needed only one or two rows of Docker commands and some configuration files and you were rewarded with 4 separate container instances with their respective IP addresses. By using Docker Compose you could combine them all into one common network and link with each other with an ease.

Secondly, we ended with a minimum viable product to show how to create monitorable Java stack software meanwhile maintaining a close look on it with Elastic's stack software.

Thirdly, observing and using successfully one of the popular choices in current software development to monitor the application's performance. Elastic has put create effort to create easy to use off the shelf solution. The integration between the pieces is seemingless, which for the users is a good thing, but usually in customization cases tends to be bit tricky. Kibana with Logstash and Elasticsearch gives immense visibility what is going on over the multiple applications.

5. CONCLUSION

Elastic's technology stack provides very easy out of the box monitoring for multiple applications, microservices etc. There are lot of things, which need to be observed. The bigger issue comes with the log events, how to prepare them semantically in the application for Logstash and Kibana. To prepare easy log patterns is vital because it makes for Logstash easier to filter them and in Kibana we can separate different user environments and applications from each other with simple regex patterns and therefore produce quality and high knowledge graphs. Eventually getting the maximum out of the monitoring software with some smart preplanning.

Essentially we need to research out the possible logs, which are out there. We can not deliver all the logs into Elasticsearch, because it causes with bigger systems quite huge communication over the lines and shackling the performance (CPU and network bandwidth). Among of different log events like Tomcat server's access logs to some application's trace logs there needs to be done excessive investigation what information is valuable. It is kind of obvious to monitor exceptions and error level events.

6. REFERENCES

[1] Docker. Docker - build, ship, and run any app, anywhere. https://www.docker.com/, 2016.

[2] Elastic. Elasticsearch. https://www.elastic.co/products/elasticsearch, 2016.

[3] Elastic. Kibana. https://www.elastic.co/products/kibana, 2016.

[4] Elastic. Logstash. https://www.elastic.co/products/logstash, 2016.

[5] M. Foundation. Mariadb.org - ensuring continuity and open collaboration. https://mariadb.org/, 2016.

[6] S. Foundation. Spring boot. http://projects.spring.io/spring-boot/, 2016.

[7] Hibernate. Hibernate. everything data. http://hibernate.org/, 2016.

[8] Jersey. Restful web services in java. https://jersey.java.net/, 2016.

[9] Log4j2. Apache log4j2. http://logging.apache.org/log4j/2.x/, 2016.

[10] Thymeleaf. Thymeleaf. http://www.thymeleaf.org/, 2016.

Figure 7: All log events visible in the main screen with count graph over time

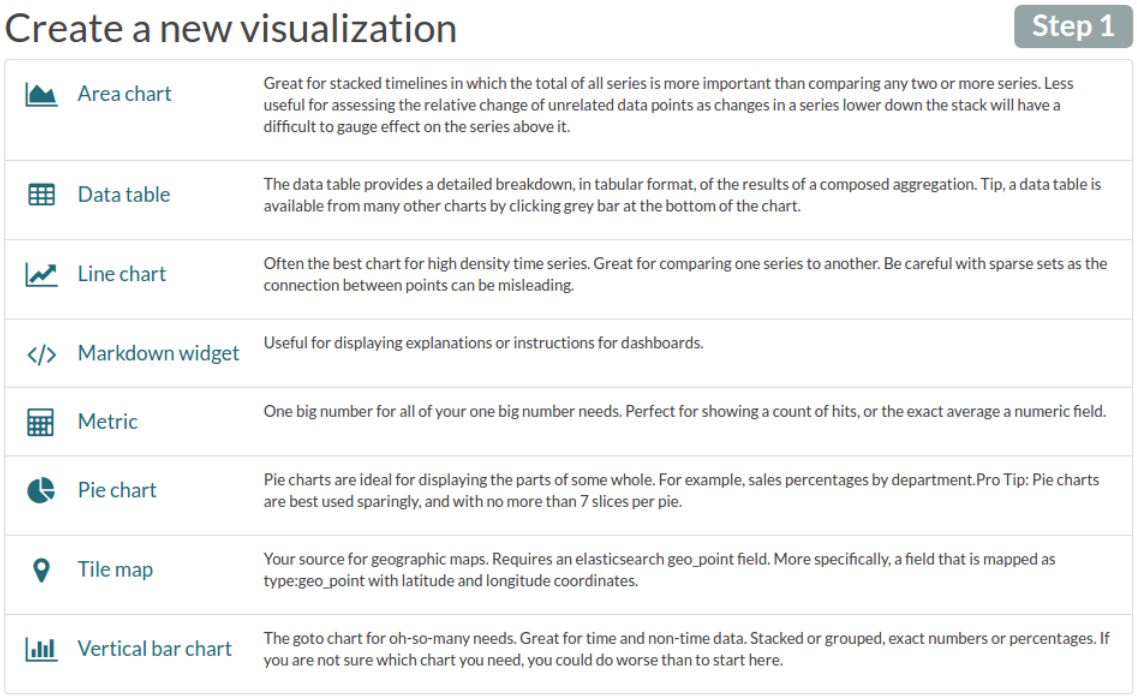


Figure 8: The visualziation possibilities with Kibana

Figure 9: The custom dashboard created for showing one of the counting features with a graph