University of Stuttgart

Text Technology Project Report

Instructor: Andre Blessing
Students: Brandon Sorensen, King DeLaney,
Haywood Shannon, Johannes Krämer

2019

1 Elasticsearch Project

Why did we choose Elasticsearch? "You know, for Search."

But seriously, Elasticsearch is a state of the art tool. It is used by research and industry extensively. This should be enough reason. Hearing the buzzword "Elasticsearch" certainly motivated us in this direction. But the real fascination came by watching introductory Youtube videos about Elasticsearch. Most people were so positively excited while talking about it that we basically were conviced about the power of Elasticsearch before even trying it out.

Our project approach was a heterogeneous one. Our goal was to explore, play around, learn about the full power of Elasticsearch without beeing bound to a specific downstream task. As a result we have used different datasets to illustrate different capabilities of the technology: the ufo dataset, the beer dataset and the political dataset.

2 The ELK Stack

The ELK Stack stands for Elasticsearch, Logstash, and Kibana. These three technologies are often referred to when talking about Elasticsearch on it's own. The reason is because together they form a pipeline where Elasticsearch as a database is the core technology.
From a dataflow point of view, Logstash is the first technology. It servers as a preprocessing tool which brings the data in the right shape. The second tool is Elasticsearch which stores and index the data. The third tool is Kibana, which provides a Graphical User Interface (GUI) to Elasticsearch's data and is reponsible for visualizing the data.

Figure 1: The Logstash configuration file which was used to preprocess the Ufo dataset.

The ELK Stack, is available for personal usage for free. But it can also be employed professionaly by either using it as Software as a Service (SaaS) in the Cloud, or by integrating it in the company server centers. With the last two approaches the above mentioned technologies are perfectly scalable to basically any size of data/storage, computing power.

2.1 Logstash

As already mentioned Logstash can be seen as a preprocessing tool. It prepares the data and sends the transformed data to the specified Elasticsearch instance. It is able to recieve data from all kinds of sources in all kinds of formats like from Databases, HTTP-Streams, CSV-files, JSON-files, Logfiles etc.

With the help of filters the data is parsed and transformed into a desired format. That way it is possible to achieve more structure, consistancy, readability etc.

In the next paragraph the configuration file which was used for the Ufo-sighting project will be explained (see Figure 1).

The Ufo-sightings are stored in a csv file. Hence the configuration file contains definitions specific for a csv file.

In line 0-5 the input is defined (the csv file). Once dynamically read from the standart input [1] (stdin), and once with a hardcoded path to the csv file.

From line 7-31 different filters are defined which deal with the content of the csv file:

The csv filter (line 8-12) specifies the structure of the original csv file. In this case the csv file is comma seperated (could be tab seperated aswell), has the following columns (see image) and the header is skipped because it is not part

---

[1] means the path to the file needs to be passed as an argument when running logstash

of the actual data.

The mutate filters (line 13-27) actually applied some changes to the data: In line 14 we a new field *location* is constructed out of the existing *latitude* and *longitude* fields. The reason for that is, that in order to represent geo-coordinates Elasticsearch requires geographical data to be stored as a tuple in one field instead of beeing stored in seperate fields.

On line 15 a couple of fields are removed, which where decided to be not relevant for the data analysis anymore. For example geographical information is now stored in the *location* field, hence the original *lalitude-* and *longitude-*fields are not needed anymore.

The *gsup* mutation (line 16-23) is a regular expression based string subsitution. The first column defines the source field, the second column defines the regular expression and the third field is the replacement string.

In this case the language codes are transformed into the original country names just for the sake of readability.

The last subsitution is a little hack to deal with a specificity of the Ufo-dataset. In the Ufo-dataset midnight was encoded as 24:00. But Elasticsearch's time format goes from 00:00:00-23:59:59. Although it is more common to represent midnight as 00:00, sometimes 24:00 is used to represent the end of the day and 00:00 the beginning of the day. Since the original midnight encoding was 24:00 (end of day) it was decided to remove one minute instead of adding one, such that the time still represents the end of a day instead of the beginning of the next day.

The date filter (line 28-30) deals with different date representations. Elasticsearch requires a consistent date-time representation to be automatically recognized as such. Therefore all possible date combinations[2] must be given to the date filter. The first entry is the column name of the date field followed by the different possible combinations of date representation.

From line 32-42 the output is defined. This means where the transformed data is sent to: In line 33 debugging information is sent to the standart output (stdout) - e.g. the console. From line 34-41 the Elasticsearch target instance is defined together with the index the data should be stored at. In this case (line 36) Elasticsearch runs locally (localhost) and the index (line 37) is called *ufoindex*. As mentioned earlier Elasticsearch can be used as a service in the cloud. In line 35 a Elasticsearch target instance can be seen which was hosted on Amazons AWS Cloud-system[3].

2.2   Elasticsearch

As already mentioned Elasticsearch is a database and a search engine. The following query and search examples were made with the beer dataset.

---

[2]A day represented with one digit. A day represented as two digits. A month represented as one digit. A month represented as two digits.

[3]A free trial was used in order to have all the material for the presentation ready in one place. The trial (and hence the url) is not active anymore.

```
GET beers/_search
{
  "query":{
    "match_all": {
    }
  },
  "size":0,
  "aggs": {
    "Methods": {
      "terms": {
        "field": "Style.keyword",
        "size": 100
        }
    }
  }
}
GET beers/_search
```

Figure 2: Example for the Elasticsearch Query Language

2.2.1 Query Language

Elastic search queries use a structure similar to JSON (see Figure 2). All features of the query are nested within within the "query" field, and sub-fields such as "match", "bool", and "wildcard" are there to match the needs of the user. Within the query, filters, modifications, and precision adjustments can also be implemented. Attributes of the query such as "size" are beneath and not nested directly in the original query. The size field will control the number of results of the query or "buckets" during aggregation. Additionally, actions such as aggregations are also outside and below the query. Additional aggregations, can also be added to the end or within other aggregations.

2.2.2 Mapping

Data being stored in Elasticsearch is a stringified JSON format. Mappings (see Figure 3) are needed to provide the structure to values that are to be extracted and evaluated in methods other than those of plain text. JSON Mappings allow for basic types beyond strings such as integers, dates, Boolean values, and range data. There are also more complex types, like arrays and dictionaries (or other JSON structured data), and additionally, there more specific data types such as Geo-point and IP data types. For values that are integers or a more complex data type and would like to be use for searches, aggregations within a range, plotted over a geographical map a mapping of the index must be used to create

4

```
PUT beers
{
  "mappings": {
    "recipe": {
      "properties":{
        "ABV": {"type":"integer"},
        "BoilTime": {"type":"integer"},
        "OG": {"type":"integer"},
        "FG": {"type":"integer"},
        "Color": {"type":"integer"},
        "IBU": {"type":"integer"},
        "Size(L)":{"type":"integer"}
      }
    }
  }
}
```

Figure 3: Example for a Mapping

the index. In this mapping, the index "beers" has seven fields where the values will be type "integer" the remaining as well as any additional fields will be recognized as type "string".

2.2.3   Basic Search

The basic search allows to match a value to specific fields or just searched over the whole of each document in the index. In the queries (see Figure 4), there is a basic search for all items in the index and a query for the string value "saaz" from the specific field "recipe" . The second search also has limit to which fields of the document within the index will be presented within the query results. By selecting "source", we specified the fields "recipe", "ABV", and "Style" to be present in the query results.

2.2.4   Aggregations

Elasticsearch allows for quick aggregations from a variety of different means. Aggregations can made on unique field values or ranges as well as a filter value that can be applied to a given aggregation. Limits can also be set with the "size" field within the query to limit or expand the number of query and/or aggregation results. In this query (see Figure 5), we have utilized the unique values of within the field "Style" to first aggregate the result documents of a

Figure 4: Example for a basic search usecase

Figure 5: Example for aggregations

```
GET beers/_search
{ "query":{
    "query_string": {
      "default_field": "recipe",
      "query": "cascade AND centennial^3"
    }
  }
}
GET beers/_search
{ "_source":["Name","Style", "recipe"],
  "query":{
    "query_string": {
      "default_field": "recipe",
      "query": "Mittelfr*h"
    }
  }
}
```

```
24    },
25    {
26      "_index" : "beers",
27      "_type" : "recipe",
28      "_id" : "2200",
29      "_score" : 1.0,
30      "_source" : {
31        "recipe" : "Method: All Grain, Style: Oktoberfes
              /Märzen, ABV 5.69%, IBU 22.22, SRM 10.91,
              Fermentables: (Munich - Light 10L, Vienna,
              Pilsner, CaraMunich) Hops: (Hallertau
              Mittelfruh)",
32        "Style" : "Oktoberfest/Märzen",
33        "Name" : "Oktobourbon"
34      }
35    },
36    {
```

Figure 6: Example for weighted and wildcard queries

```
GET beers/_search
{"_source":["recipe"],
  "query":{
    "query_string": {
      "default_field": "recipe",
      "query": "\"Hops: Other:\"-1"
    }
  }
}
GET beers/_search
{ "_source":["Name", "Style"],
  "query": {
    "match": {
      "Style": {
        "query": "German Pale Ale",
        "minimum_should_match": "70%"
      }
    }
  }
}
```

```
15      "_index" : "beers",
16      "_type" : "recipe",
17      "_id" : "221",
18      "_score" : 3.2074432,
19      "_source" : {
20        "Style" : "American Pale Ale",
21        "Name" : "PALE ALE 2.0"
22      }
23    },
24    {
25      "_index" : "beers",
26      "_type" : "recipe",
27      "_id" : "236",
28      "_score" : 3.2074432,
29      "_source" : {
30        "Style" : "American Pale Ale",
31        "Name" : "Four Pines Pale Ale"
```

Figure 7: Example for proximity and precision

string-query for the value "belgian" . The number of styles was limited to ten style "buckets", and within those buckets, we were once again able to aggregate them according the range value of "ABV". We have only three ranges here, but the limit is at least one, with an indetermined upper-bound limit.

2.2.5 Weighted and Wildcard Queries

Elasticsearch allows for weighting or boosting the significance of a query term. In the a query, the "^" symbol is used to begin the boosting of a specific term. This boosting can occur over many values and each will adjust the weighted ranking of the result documents. In this instance, we made the value of "centennial" three times more significant in the ranked results than the value "cascade". Wildcard queries are also easily performed in Elasticsearch. The "*" character is used for wildcard queries. In the query (see Figure 6), we used a wildcard for the value "Mittelfrüh". Terms with characters that are less common, in this case the "u" with an umlaut, might return results with correct and incorrect spellings of the term.

7

			cream-ale																
2	2	Southern Tier Pumking clone	/homebrew/recipe/view/16367/southern-tier-pumking-clone	Holiday/Winter Special Spiced Beer	85	20.82	1.083	1.021	8.16	60.65	15.64	24.61	60	1.07	70	N/A	Specific Gravity	All Grain	N/A	N/A
3	3	Zombie Dust Clone - EXTRACT	/homebrew/recipe/view/5920/zombie-dust-clone-extract	American IPA	7	18.93	1.063	1.018	5.91	59.25	8.98	22.71	60	N/A	70	N/A	Specific Gravity	extract	N/A	N/A

Figure 8: An excerpt of the beer dataset

2.2.6 Proximity and Precision

A feature of Elasticsearch text search is proximity searches. The use of boundary "\" and quotes can define the values and a number outside will define the proximity space between the two terms. In the query (see Figure 7), it queries for "Hops:" and "Other:" separated by no more than one additional term between. Additionally, precision within the searches can be improved by targeting a minimum match. Using the field "minimum-should-match", a string percentage value is given by which Elasticsearch tries to match to as a minimum. The percentage may be decreased to the nearest evenly distributed percentage based on the length of the query. In our query "German Pale Ale", our likely distribution is that each word constitutes 33.33% of the query. If we match to "70", it will try to match at least 66.67%. Within the results, "Pale Ale" is most likely to occur and returns many results with style "American Pale Ale".

2.2.7 Beer Dataset

The Brewer's Friend Beer recipe dataset (see Figure 8) that was selected started from Kaggle.com. It contained several fields already, including several aspects of a beers color, alcohol content, style, etc. The dataset was, however, lacking in areas involving methodology and ingredients. To make a more comprehensive and interesting dateset, more data needed to be gathered. For each entry, there was a link the recipe page where more information about the recipe could be ascertained. Extracting the data for each recipe would be the most effective way to compile more data into this data set.

2.2.8 Crawler

For this particular website, the recipe data was in several tables throughout the page. The corresponding data was also located in meta tags within the Html. This spider was constructed via Scrapy (see Figure 9). It sifted through the list of urls created from the Brewer's friend website and the recipe extension from the dataset. It extracted the attribute data from the meta tags. This data contained some of the data found in the original data set as well as various types of ingredients in up to four different categories and notes that the author felt necessary. All data was extracted to CSV files in smaller batches as the process

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3  import pickle
4  from beers.items import BeersItem
5  with open('/home/nadezh/Documents/Text Technology/project Beer/url_gen.pkl', 'rb') as f:
6      urls = pickle.load(f)
7
8  class RecipeSpiderSpider(scrapy.Spider):
9      name = 'recipe_spider'
0
1      def start_requests(self):
2          allowed_domains = ['https://www.brewersfriend.com']
3          start_urls = [allowed_domains[0]+i for i in urls]
4          headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:48.0) Gecko/20100101 Firefox/48.0
5          for url in start_urls:
6              item=BeersItem()
7              item['url'] = url.replace('http://www.brewersfriend.com','')
8              yield scrapy.Request(url=url, meta=item, headers=headers, callback=self.parse)
9      def parse(self, response):
0          item=response.meta
1          try:
2              item['recipe'] = response.xpath('//meta[@property="og:description"]/@content').extract()
3          except:
4              item['recipe'] ='' 
5          yield item
```

Figure 9: Scrapy Spider Code

```
1  recipes[9]['recipe']
```

'Method: All Grain, Style: Imperial IPA, ABV 8.88%, IBU 98.09, SRM 8.55, Fermentables: (Pale 2-Row, Caramel / Cryst
al 20L, Flaked Wheat, Rolled Oats) Hops: (Magnum, Centennial, Citra, Zythos) Other: (Whirlfloc Tab, Pureed Frozen M
ango, Pureed Habanero Pepper, Organic Mango Juice, Sliced Habanero Pepper) Notes: Add the pureed mango and habanero
to the boil with about 5 minutes left. When transferring to the primary do NOT strain the wort. One week in primary
Add mango juice, sliced habanero and dry hops when transferring to the secondary. Two Weeks in secondary OG 1.065 F
G 1.009 ABV 7.35% OG 1.080 - lots of sediment FG 1.012 ABV 8.9% Batch - San Diego Super Yeast - Very little pepper
flavor or heat OG 1.070 FG 1.010 ABV 7.9% OG 1.084 FG 1.012 ABV 9.5%'

Figure 10: Example of self-extracted beer data

of scraping 73,000 recipes took over 70 hours.

2.2.9   Extraction

As stated, some of the data that was extracted was identical to that which came
in the original dataset. The additional data (see Figure 10) here often contained
the author Username, fermentable ingredients, Steeping grains, Hops selection,
additional ingredients, and notes from the author. This format was suitable
for Elasticsearch full text queries, but it was limited in its use for further data
analysis outside of aggregation counts of terms that occurred in the data.

2.2.10   Data Processing

Beyond extraction, there was a significant amount of cleaning and preprocess-
ing that need to be done to make it more usable for analysis (see Figure 11).
From the sing string that was extracted, there were several aspects that should
have been extracted, such as ingredients and notes. There were four different

9

```
]: {'Author': ' Stikks 2-2-12, Method: All Grain, Style: Cream Ale, ABV 5.48%, IBU 19.44, SRM 4.83',
   'Fermentables': ['pale 2-row',
   'white wheat',
   'pale 6-row',
   'flaked corn',
   'caramel',
   'crystal 20l',
   'carapils',
   'flaked barley',
   'honey'],
   'Hops': ['cascade', 'saaz'],
   'Other': ['pure vanilla extract',
   'yeast nutrient',
   'whirlfloc',
   'vanilla beans - in 2oz vodka'],
   'Notes': ' 1600 ml. starter at 1.04 Ferment at 58 to 60 degrees with Wyeast 2565,62 to 64 degrees with WLP 029 or
   65 to 67 degrees with US-05 attenuation better than expected finished at 1.011 about 5.3 ABV carbonate 3 weeks at 6
   8-70 degrees cold condition 1 week in fridge frothy head,nice lacing tasty summer beer,easy drinking. A great beer
   for the female in your life. I`ve made this brew 7 times now,and it is a mainstay for my woman.I`ve used US-05 WLP
   029 and WYEAST 2565 all with great results.Used Williamette in place of saaz and prefer it.Last batch came out at 6
   .0 ABV with a modification to my fly sparge setup.Grain bill holds at 11 lbs. Last batch was with WLP 029,I think W
   yeast 2565 was the best of the 3 yeast types that I have used. . Made batch #8 with US-05 on 4-19-13 used 4 Madagas
   car vanilla beans soaked in vodka for a week and added to secondary It added a bit more vanilla flavor.Nice additio
   n for added flavor.Will make this a regular step in future batches,recipe has not changed but 2565 rules the brew.
   I made batch #9 on 9-22-13.Used original recipe with wyeast 2565, by far the best. Bottled on 11-1-2013.The only tw
   eak was adding 12 oz clover honey at flameout.The Vanilla beans in secondary are the bomb. Recipe will not change f
   rom here on out. This batch started at 1.057 and finished at 1.01 - 5.51 ABV.It is exactly where I Want it. SWMBO a
   nd my friends love it.I like it alot too but I am more of a stout porter guy.I hope you like it and have good luck
   with it if you try it. -enjoy- 12-8-2013 Vanilla flavor has mellowed nicely,lacing last`s till the end of the glass
   .Very pleased with this batch. Batch #10 1-23-2015 recipe has not changed will be bottling in about a week or so. T
   his brew is our house brew all year,it`s one for the ladies.Will be ready for St. Paddy`s day. batch #10 is long go
   ne.Finally retired and will be moving to Florida in 2 weeks.I will update next batch when I brew in in the heat for
   the first time,Goodbye Chicago.I hope you enjoy the recipe.-Will be back to brewing soon. -Cheers-'}
```

Figure 11:   Example of a beer datapoint

categories of ingredients to extract and in addition, much of the data contained typos and short hand for different ingredients and descriptions in the notes. Much further refinements would also be necessary to make the data more presentable.

2.3   Kibana

In the previous Section the speed, flexibility and powerfull search capability were shown. But technical power alone is rarely enough. In order to work with data one has to have a good intuition about it. And since the human brain is better in interpreting images than a matrix of values a common approach for better understanding data is to visualize it.

Kibana is a simple to use and still powerfull visualization tool which it build on top of Elasticsearch. It comes with a lot of inbuild visualizations: Histograms, Pie Charts, Lines, Heat Maps, Geographical Maps etc. These visualizations can be organized in dashboards. The dashboards are interactive, which means if one clicks on a value in a visualization (e.g. the biggest part in a Pie Chart) this value is used as a filter for all visualizations which are part of the same dashboard. This functionality makes explorative descriptive statistics really convinient, because different aspects of the data can be looked at by playing and clicking around in the dashboard.

Additionally Kibana provides a lot more functionalities like Machine Learning based anomaly detection, customizable visualizations which are not in the scope of this report.

But one additional functionality has to be mentioned at this point. It is the possibility to create online scripts, which process data on the fly. In Figure 13 the *JSON Input* field is an example for such a script. This script turns seconds into minutes by dividing the current value by 60. That way the original data does not
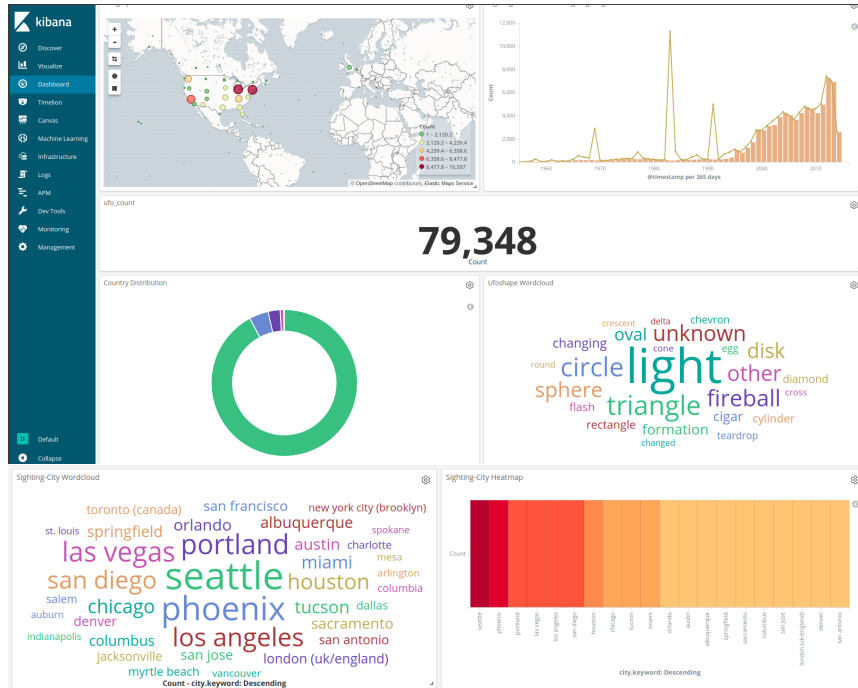
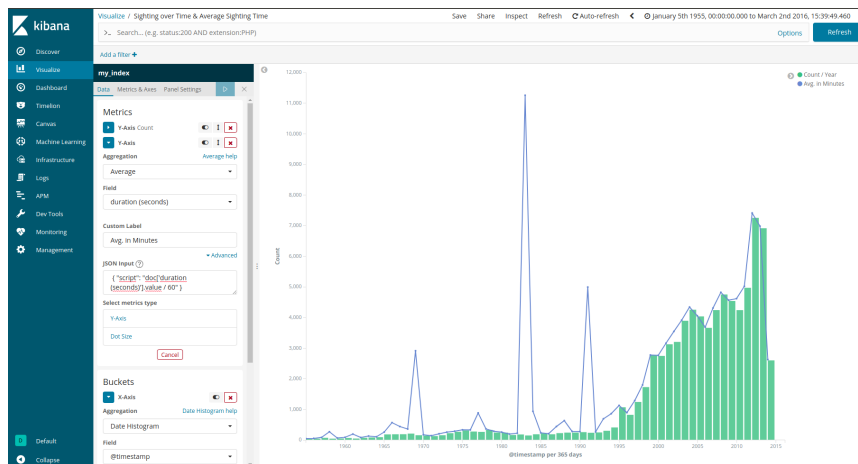Figure 12: The dashboard for the Ufo-dataset

Figure 13: This is an visualization in the editive mode.

need to be touched and the calculation is done only for the visualization. The drawback of this scripting functionality is, that the script needs to be run every time the visualization is loaded. For huge datasets this can lead to long waiting times, especially when the scripting functionality is used in many visualizations.

In Figure 12 different visualizations are shown for the ufo dataset, which are organized into a dashboard:
In the upper left, a geographical visualization is shown. It marks all the points where ufos were seen. Bigger circles are collections of datapoints. This can be seen by zooming in, then the bigger circle is devided into multiple smaller circles.
In the upper right a histogram and a line is shown. The histogram describes how many ufo sightings were recorded per year. The line is the average time (in minutes) per year that ufo's were seen[4]. Since the line visualization contains an outlier around 1964, this would be a typical scenario to apply a filter by clicking on the outlier and inspecting the results of the other visualizations. That way it could be investigated if something interesting happened in 1964 or if this was an error during data collection.
The next visualization below is just a simple count of data points (size of the dataset). In a live application this would be more interesting, because one could inspect how the dataset grows.
Below that on the left there is a Pie Chart which shows the distribution over Countries: Green = America, Blue = Canada, Violett = Great Britain, Purple = Australia, Red = Germany (not visible anymore on the PDF).
On the right of the Pie Chart there is a frequency based wordcloud, showing the different ufo shapes which people described during the ufo sighting. It has to be mentioned that it is only possible to use a wordloud properly on (pseudo) categorical values - only one word per field. There is not preprocessing step (like tokenization) involved.
Below on the left is again a wordcloud which shows roughly the distribution over cities.
To the right of the city wordcloud, the information about cities is encoded as a heatmap (just for the sake of showing exploring different visualizations).

As a quick summary and interpretation of the visualization the following can be recorded: most of the ufo sightings happened in the US; in 1964 there might have been an ufo boom; lights, circles, triangles, fireballs are popular descriptions of the ufo-look; seattle, phoenix, portland, las vegas are the most popular cities were people claim to have seen ufos.

One of us should also show how it looks like when a filter is applied.

---

[4]The line is not on the same scale as the histogram. In order to see the actual minutes-values one needs to hover with the mouse to the point of interest.

3 Conclusion

Fast, Powerfull bla bla bla...

4 Personal Reflection

Here we can put our personal opinions and say what we liked most.