



UNIVERSITY OF PIRAEUS
DIGITAL SYSTEMS DEPARTMENT

MSc SECURITY IN DIGITAL SYSTEMS

STUDENT

VOULGARIS IOANNIS MTE 1803
PIRAEUS, GREECE, FEBRUARY 2020

**INFORMATION AND SECURITY EVENT
MANAGEMENT SYSTEM**

SUPERVISOR

PROF. KONSTANTINOS LABRINOUDAKIS





Table of contents

ABSTRACT.....	11
ACKNOWLEDGEMENTS.....	12
1. Security Operation Center and Log Files.....	13
1.1. General idea.....	13
1.2. Importance of Log Analysis.....	14
1.3. Efficient log analysis.....	15
1.4. Problems with log analysis.....	16
1.5. Security Operation Center (SOC).....	18
2. Elastic Stack.....	0
2.1. The present and future of monitoring with Elastic stack.....	0
2.2. Components.....	1
2.2.1. Elasticsearch.....	1
2.2.2. Kibana.....	20
2.2.3. Logstash.....	27
2.2.4. X-Pack.....	29
2.2.5. Beats.....	32
2.2.6. Common choices of architecture.....	33
2.4. Installation and Configuration Guides for Windows, Locally.....	37
2.4.1. Elasticsearch.....	38
2.4.2. Kibana.....	42
2.4.3. Logstash.....	49
2.4.4. Beats.....	53
2.4.5. X-Pack.....	57
2.5. Elastic Stack and GDPR.....	61
2.5.1. General.....	61
2.5.2. Preparation for Personal Data handling (1 st Set of processes).....	62
2.5.3. Protection of Personal Data (2 nd Set of processes).....	63
2.5.4. Privacy processes and maintenance of rights (3 rd Set of processes).....	63
2.5. Companies that use it.....	64
3. Document Management.....	0
3.1. Creating an index.....	0
3.2. Adding documents in an index.....	0
3.3. Document retrieval.....	2
3.4. Document supplanting.....	3



3.4.1. Original replacing and updating.....	3
3.4.2. Updates with scripts	5
3.4.3. Document upsert	6
3.5. Deleting.....	8
3.5.1. Documents	8
3.5.2. Indices	10
3.6. Processing multiple documents.....	10
3.7. Observing how the cluster works.....	13
4. Mapping.....	0
4.1. Preface.....	0
4.2. Dynamic.....	0
4.3. Meta-fields	3
4.4. Data types.....	4
4.4.1. Core data types.....	4
4.4.2. Complex data types	6
4.4.3. Specialized data types	7
5. Wazuh	0
5.1. General idea	0
5.2. Components	0
5.3. Required ports.....	1
5.4. Archiving data storage	2
5.5. Common use cases.....	2
5.5.1. Signature-based log analysis	3
5.5.2. Rootkit detection.....	4
5.5.3. File integrity monitoring	6
5.6. Rulesets.....	6
5.6.1. Introduction to Wazuh Rulesets.....	6
5.6.2. Decoders.....	7
5.6.3. Rules	4
5.6.4. Regex	8
6. Monitoring and Log Analysis Case Scenario.....	0
6.1. Installing tools on Linux	0
6.1.1. Elasticsearch and Kibana	0
6.1.2. Logstash	1
6.1.3. Filebeat.....	1
6.1.4. Wazuh (scenario)	3



6.1.5. Beats (scenario).....	16
6.4. Elastic SIEM	26
6.4.1. Security Information and Event Management.....	26
6.4.2. Introduction.....	26
6.4.3. Components	27
6.5. Elastalert	34
6.5.1. Installing and configuration	34
6.5.2. Elastalert rules.....	38
7. Concluding Remarks – Empirical Findings.....	0
8. Bibliography	0



Table of figures

Figure 1: Log data examples	13
Figure 2: Steps in traditional Log Analysis.....	14
Figure 3: Efficient Log Analysis steps	16
Figure 4: Tomcat Log file example	17
Figure 5: Apache Server Log file example	17
Figure 6: How decentralization works	18
Figure 7: Basic Elastic Stack structure.....	1
Figure 8: JSON Object for Elasticsearch	3
Figure 9: Index and document relation.....	3
Figure 10: How REST API works	3
Figure 11: DBMS and JSON format	4
Figure 12: Inverted index format storing.....	5
Figure 13: Concurrency Control Part 1.....	6
Figure 14: Concurrency Control Part 2.....	7
Figure 15: Cluster and nodes relation.....	8
Figure 16: Shard resilience Part 1	10
Figure 17: Shard resilience Part 2	11
Figure 18: Shard resilience Part 3	11
Figure 19: Synchronization of replica shards	13
Figure 20: Elasticsearch. How data searching works	14
Figure 21: Queries handling	15
Figure 22: Shard number calculation	16
Figure 23: Add and search document requests	16
Figure 24: Counting of shards	18
Figure 25: Indexing in Apache Lucene	19
Figure 26: Updating and re-indexing in Apache Lucene	20
Figure 27: Elastic Cloud prerequisites.....	22
Figure 28: Instances of Elastic Cloud (AWS).....	22
Figure 29: Instances of Elastic Cloud Google Cloud Platform	23
Figure 30: Region choosing	23
Figure 31: Elastic Stack version used	24
Figure 32: Final deployment info part 1.....	24
Figure 33: Final deployment info part 2.....	24
Figure 34: Final deployment info part 3.....	25
Figure 35: Elastic Cloud credentials	25
Figure 36: Elastic Cloud ID.....	25
Figure 37: Node receiving the request.....	26
Figure 38: Accessing Elasticsearch raw data.....	26
Figure 39: Changing the instance-node	27
Figure 40: Kibana dashboard in Elastic Cloud	27
Figure 41: Logstash place in ELK	29
Figure 42: Stages of Logstash.....	29
Figure 43: Common architecture part 1.....	33
Figure 44: Common architecture part 2.....	34
Figure 45: Common architecture part 3.....	35



Figure 46: Common architecture part 4.....	36
Figure 47: Common architecture part 5.....	37
Figure 48: Checking java version on Windows.....	38
Figure 49: Elasticsearch directory (Windows).....	39
Figure 50: Starting Elasticsearch (Windows).....	39
Figure 51: Accessing Elasticsearch via port.....	40
Figure 52: Configuring Elasticsearch on Windows part 1	41
Figure 53: Configuring Elasticsearch on Windows part 2	42
Figure 54: Kibana directory on Windows.....	43
Figure 55: Starting Kibana (Windows).....	43
Figure 56: Accessing Kibana via port.....	44
Figure 57: Configuring Kibana on Windows part 1	44
Figure 58: Configuring Kibana on Windows part 2	45
Figure 59: Configuring Kibana on Windows part 3	45
Figure 60: Posting data with curl on Elasticsearch.....	46
Figure 61: Index patterns in Kibana part 1.....	47
Figure 62: Index patterns in Kibana part 2.....	47
Figure 63: Console tool	48
Figure 64: Console syntax part 1.....	49
Figure 65: Console syntax part 2.....	49
Figure 66: Logstash directory on Windows.....	50
Figure 67: Testing if Logstash is up	50
Figure 68: Proof Logstash is up and running.....	50
Figure 69: Importing custom messages inside Logstash.....	51
Figure 70: Apache log example	51
Figure 71: Apache logs configuration file.....	52
Figure 72: Using apache.conf by Logstash	52
Figure 73: Proof that apache.conf has loaded	52
Figure 74: Results with Logstash index	53
Figure 75: Number of documents indexed through Logstash	53
Figure 76: Filebeat enabling Apache.....	53
Figure 77: installing Filebeat through PowerShell	54
Figure 78: Configuring apache.conf for Filebeat.....	54
Figure 79: Logstash configuration.....	55
Figure 80: Loading in Logstash the new apache.conf	55
Figure 81: Proof that Filebeat ingests data in Logstash	55
Figure 82: Filebeat index results	56
Figure 83: Metricbeat installation through PowerShell.....	56
Figure 84: Metricbeat index results	57
Figure 85: Visualize things with Kibana example	57
Figure 86: Installing X-Pack (Windows) part 1	58
Figure 87: Steps to install X-Pack	58
Figure 88: Installing X-Pack (Windows) part 2	58
Figure 89: Enabling X-Pack in Elasticsearch conf file	59
Figure 90: All passwords for the suite.....	59
Figure 91: Installing Kibana steps.....	60
Figure 92: Kibana configuration file	60
Figure 93: Kibana login screen	60



Figure 94: Logstash installation steps	61
Figure 95: How GDPR works with personal data according to Elastic	61
Figure 96: Which steps does Elastic cover	62
Figure 97: PUT request to show a specific index	0
Figure 98: POST request to add a new object.....	1
Figure 99: Proof that version changed.....	1
Figure 100: Adding a new object with ID example	2
Figure 101: GET request to retrieve an object.....	3
Figure 102: Updating an object. Single argument (Version proof)	4
Figure 103: Updating an object. Multiple argument (Version proof)	5
Figure 104: Update an object's value that already exists	6
Figure 105: Proof that the value changed.....	6
Figure 106: DELETE request and proof	7
Figure 107: Upsert example part 1 (POST request)	7
Figure 108: Upsert example part 2 (proof)	8
Figure 109: Adding new documents example.....	8
Figure 110: DELETE multiple documents example.....	9
Figure 111: DELETE multiple documents proof	9
Figure 112: DELETE request for an index	10
Figure 113: Proof that index was deleted	10
Figure 114: Adding values on multiple documents at once.....	11
Figure 115: Proof for multiple document processing	12
Figure 116: Update and delete documents on the same query	13
Figure 117: Cluster's health part 1	13
Figure 118: Cluster's health part 2	14
Figure 119: Node's health	14
Figure 120: Indices health part 1	15
Figure 121: Indices health part 2	15
Figure 122: Disk usage and shard allocation info part 1	15
Figure 123: Disk usage and shard allocation info part 2	16
Figure 124: Shard health and distribution part 1.....	16
Figure 125: Shard health and distribution part 2.....	16
Figure 126: Steps to dynamic mapping.....	1
Figure 127: Mapping API example for behind the scenes mapping	1
Figure 128: GET request for mapping information over an index part 1.....	2
Figure 129: GET request for mapping information over an index part 2.....	2
Figure 130: Core data types with examples.....	5
Figure 131: Complex data type structure example.....	6
Figure 132: Complex data type (geo_point) example.....	7
Figure 133: Specialized data types tool installation.....	8
Figure 134: Wazuh structure.....	1
Figure 135: Data storage/ archive.....	2
Figure 136: Signature-based log analysis example	3
Figure 137: Alert example for Signature-based log analysis.....	4
Figure 138: Rootkit detection example.....	5
Figure 139: FIM example.....	6
Figure 140: File used to update rulesets.....	7
Figure 141: Traditional decoder example	1



Figure 142: Dynamic decoding example	1
Figure 143: Decoder explained through a JSON output	2
Figure 144: Decoder syntax part 1	2
Figure 145: Decoder syntax part 2	3
Figure 146: Decoder syntax part 3	3
Figure 147: Decoder syntax part 4	4
Figure 148: Storing options for rules and decoders.....	5
Figure 149: Steps for adding a decoder file on Wazuh-Manager	5
Figure 150: Testing tool for logs	6
Figure 151: Log file to be decoded and tested.....	6
Figure 152: Decoding and testing result	6
Figure 153: Rules syntax part 1	7
Figure 154: Rules syntax part 2	7
Figure 155: Rules syntax part 3	8
Figure 156: Rules syntax part 4	8
Figure 157: Traditional Regex syntax	9
Figure 158: Wazuh Regex syntax part 1	10
Figure 159: Wazuh Regex syntax part 2	10
Figure 160: Wazuh Regex syntax part 3	10
Figure 161: Wazuh Regex syntax part 4	11
Figure 162: Wazuh Regex syntax part 5	11
Figure 163: Unzipping the Elasticsearch file on Linux.....	0
Figure 164: Starting Elasticsearch on Linux.....	0
Figure 165: Install CURL tool command.....	0
Figure 166: Showing with CURL tool the Elasticsearch cluster	1
Figure 167: Unzipping the Kibana file on Linux.....	1
Figure 168: Starting Kibana on Linux	1
Figure 169: Unzipping the Logstash file on Linux.....	1
Figure 170: Unzipping the Filebeat file on Linux.....	2
Figure 171: Starting Filebeat on Windows and proof part 1.....	2
Figure 172: Filebeat configuration on Linux part 1	2
Figure 173: Filebeat configuration on Linux part 2	3
Figure 174: Starting Filebeat on Windows and proof part 2.....	3
Figure 175: Install Wazuh-Manager with CURL tool	4
Figure 176: Starting Wazuh-Manager and proof	4
Figure 177: Install Wazuh API with CURL tool part 1	4
Figure 178: Install Wazuh API with CURL tool part 2	4
Figure 179: Proof that Wazuh API is up and running.....	5
Figure 180: All agents with their IDs	5
Figure 181: Importing authentication key and proof.....	6
Figure 182: Proof that the new agent is added and active	6
Figure 183: FIM example part 1	7
Figure 184: FIM example part 2	7
Figure 185: Rules and decoders directories.....	8
Figure 186: Custom decoder example 1	9
Figure 187: Custom rule example 1	9
Figure 188: Testing the new ruleset from example 1	9
Figure 189: Testing random logs part 1	10



Figure 190: Testing random logs part 2	10
Figure 191: Testing random logs part 3	11
Figure 192: Locating the rule that needs to be changed	11
Figure 193: Copying and pasting the needed file (Rules) and changes	12
Figure 194: Copying and pasting the needed file (Decoders)	13
Figure 195: Excluding a decoder	13
Figure 196: Syscheck debug logging enable.....	14
Figure 197: Creating test directories.....	14
Figure 198: Properties of the test files.....	15
Figure 199: Enabling syscheck FIM on Windows agent	15
Figure 200: Proof of monitoring part 1	15
Figure 201: Proof of monitoring part 2	16
Figure 202: Metricbeat directory on Windows agent with installation.....	17
Figure 203: Configuring Metricbeat on Windows agent part 1	17
Figure 204: Configuring Metricbeat on Windows agent part 2	18
Figure 205: Configuring Metricbeat on Windows agent part 3	18
Figure 206: Auditbeat directory on Windows agent with installation.....	19
Figure 207: Configuring Auditbeat on Windows agent part 1	20
Figure 208: Configuring Auditbeat on Windows agent part 2	20
Figure 209: Winlogbeat directory on Windows agent with installation	21
Figure 210: Configuring Winlogbeat on Windows agent	21
Figure 211: Winlogbeat runs as a service proof.....	22
Figure 212: Packetbeat directory on Windows agent with installation.....	22
Figure 213: Configuring Packetbeat on Windows agent.....	23
Figure 214: Packetbeat runs as a service proof	23
Figure 215: Metricbeat visualization part 1	23
Figure 216: Metricbeat visualization part 2	24
Figure 217: Auditbeat visualization part 1	24
Figure 218: Auditbeat visualization part 2	24
Figure 219: Winlogbeat visualization part 1	25
Figure 220: Winlogbeat visualization part 2	25
Figure 221: Packetbeat visualization part 1	25
Figure 222: Packetbeat visualization part 2	25
Figure 223: Elastic SIEM structure	27
Figure 224: Elastic SIEM components.....	28
Figure 225: Elastic SIEM Overview visualization	29
Figure 226: Elastic SIEM Hosts visualization	30
Figure 227: Creating timelines example part 1	31
Figure 228: Creating timelines example part 2	31
Figure 229: Elastic SIEM Networks visualization.....	32
Figure 230: Elastic SIEM Timelines visualization	32
Figure 231: Using a custom timeline part 1	33
Figure 232: Using a custom timeline part 2	33
Figure 233: Installing Elastalert.....	34
Figure 234: Creating Elastalert's indices	35
Figure 235: elastalert_status* visualization.....	35
Figure 236: elastalert_status_error* visualization	35
Figure 237: elastalert_status_silence* visualization.....	36



Figure 238: elasticsearch_status_status* visualization.....	36
Figure 239: Examining a document with elasticsearch_status_status* index	37
Figure 240: Elasticsearch configuration.....	38
Figure 241: Starting Elasticsearch.....	38
Figure 242: Elasticsearch rules directory.....	39
Figure 243: Running a rule independently.....	39
Figure 244: File that runs all rules in the background part 1	40
Figure 245: File that runs all rules in the background part 2	40
Figure 246: File that runs all rules in the background part 3	40
Figure 247: Elasticsearch example 1 configuration file	41
Figure 248: Group Policy Editor changes part 1.....	41
Figure 249: Group Policy Editor changes part 2.....	42
Figure 250: Group Policy Editor changes part 3.....	42
Figure 251: Group Policy Editor changes part 4.....	43
Figure 252: Group Policy Editor proof part 1	43
Figure 253: Group Policy Editor proof part 2	44
Figure 254: Group Policy Editor proof part 3	44
Figure 255: Group Policy Editor proof part 4	45
Figure 256: Proof that the rule works (Example 1) part 1	45
Figure 257: Proof that the rule works (Example 1) part 2	46
Figure 258: Elasticsearch example 2 configuration file	46
Figure 259: Proof that the rule works (Example 2) part 1	47
Figure 260: Proof that the rule works (Example 2) part 2	47
Figure 261: Elasticsearch example 3 configuration file	48
Figure 262: Proof that the rule works (Example 3) part 1	49
Figure 263: Proof that the rule works (Example 3) part 2	49
Figure 264: Elasticsearch example 4 configuration file	50
Figure 265: Proof that the rule works (Example 4) part 1	50
Figure 266: Proof that the rule works (Example 4) part 2	51
Figure 267: Elasticsearch example 5 configuration file	51
Figure 268: Proof that the rule works (Example 5).....	52



ABSTRACT

The cyber security field has evolved tremendously over the past decade. Cyber incidents and threats have been increasing rapidly both in figures as well as intricacy. As all fights, it is consisted by two or more participants; regularly an attacker and a defender. The defending is done by an organization while the attacker keeps changing the threat landscape by availing himself of the new exploits, weaknesses and possible security loopholes. These attackers usually have as an end goal to exfiltrate, alter or even delete valuable data.

This Thesis proposes, analyzes and evaluates some cyber security solutions. It is based on the Elastic Stack (ELK), an enterprise grade logging suite of tools which provides active threat hunting in a corporate environment. In the market it is mainly used as a search engine. Scenarios are presented with it being used alone as it is with Beats or combined with the Wazuh platform. The initial phases of this Thesis focus on what a SOC center, how important efficient log analysis is, how GDPR is affected by ELK and how Elasticsearch can be used as a search engine. Afterwards, some scenarios are presented followed by a detailed manual for these technologies. Lastly, from just this handful of scenarios and examples an opinion is presented about how these technologies can be effectual in an enterprise environment.

Keywords

Elastic Stack, BEATS, SOC, GDPR, search engine, threat hunting



ACKNOWLEDGEMENTS

I would like to thank my professor and advisor Mr. Konstantinos Lamprinoudakis for his continuous support and encouragement throughout my MSc Thesis. He instilled in me the spirit of pursuing real-world cyber security problems and guided me with perseverance during my studies. I am also thankful to him for providing opportunities to work on a wide spectrum of cyber security problems beyond my MSc Thesis.

I would like to thank Mr. Georgios Vassios and Mr. Christos Anagnostou for their guidance on the use of technical methods. Their insightful suggestions helped me immensely in carrying out this highly interdisciplinary work.

Finally, I would like to thank the members of the Hellenic Army Information Support Center for the many fruitful discussions and for a memorable military service and MSc experience.



1. Security Operation Center and Log Files

1.1. General idea

Millions of people at this moment produce logs, even without realising it. They are basically a stream of events regardless their usefulness. These logs are full of performance and usage indicators called Metrics. Two basic examples of log's data being emitted into readable format are presented below.

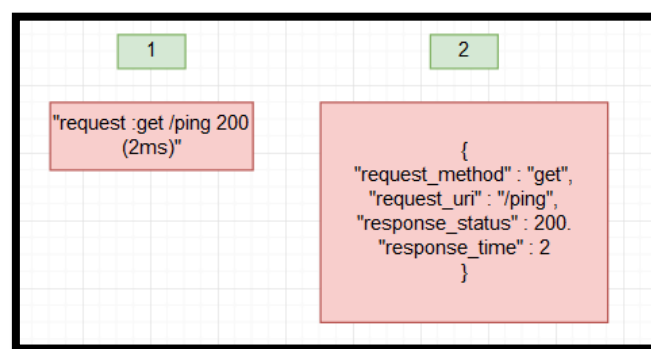


Figure 1: Log data examples

At the first example is the traditional way of reading logs by humans. It is clear that it is a request and its method is “get”. Furthermore, it is shown clearly that the route is “ping” the response code and how long it took in milliseconds. The bottom one on the other hand is structured in JSON format. It is debatable, if someone who is not familiar with this structure to read it. The important part though is that machines can read them without needing a resident regular expression user. It is of high significance to realise that parsing logs requires effort. Logs in each and every application are written out in their own manner so it is the first thing which the administrator should learn; reading and analysing the logs.



There are a number of universal truths in cybersecurity. One of those is that an expert will have a lot of log files. These may include log files from systems, switches or even routers. Simplified, every device that exists on a network has multiple log files associated with it. From these log files there is an amazing amount of intelligence and certainly a lot of history that the expert may have to go back and reference in those log files. One of the most common challenges a cybersecurity expert has is keeping it all straight by analysing them without pouring through countless pages of logs. This is a necessity since it's ludicrous for a human being to read through all of those logs. So, there are systems in place that help in analysing logs on his behalf. These systems are designed to take multiple, different log files and then consolidate them together, so that they are humanly readable. These log files as stated already countless times are incredibly useful if there happens to be a breach or if someone wants to find out what happened inside a specific time frame.

Log analysis can be either centralized or decentralized. Decentralized is something where the logs are generated on each and every web server. This has a result the administrator to log into each one of them to troubleshoot and drill down the issue. Of course, this is not an ideal scenario neither a good approach because it is time consuming therefore the solution of the issue is delayed. That's why it is recommended to store the logs at a central place for analysis.

1.2. Importance of Log Analysis

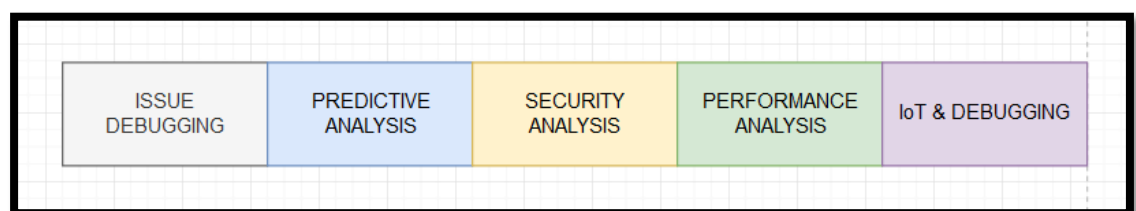


Figure 2: Steps in traditional Log Analysis

Whenever the user comes across an issue in the production scenarios, in order to debug the issue, he must read the logs. This is the importance of analysing the logs since it helps to debug the issue (Issue Debugging). Moreover, if the expert keeps on reading logs in a regular interval the occurrence of a certain issue or error can be reduced (Predictive Analysis). A case study that is interesting is suppose that the



expert reads across the logs and finds out that there is some error that can bring down the server or the web application (Predictive Analysis).

If there is any attack going on in the application, it will be captured in the logs. So, in the servers, logs that are called Access Logs exist. These logs are a huge part of a sysadmin's job since it is in his jurisdiction. By doing that a sysadmin can find with ease if a DDoS attack or a penetration attack was or is at the moment executed (Security Analysis).

The logs help experts to analyse the performance as well because an idea is given about how efficiently an application is operating. An expert can find for example, by going through the logs, what is actually provoking the lethargic performance or the degradation in terms of performance regards the application (Performance Analysis).

Furthermore, Internet of Things is something which is more relevant to the application part. This happens because the data is collected from a receiver or from a sensor and sent across to the cloud for analysis and routine check-ups. Subsequently, this data is stored as logs that are used for debugging purposes (IoT & Debugging).

1.3. Efficient log analysis

Efficient analysis of log information is often the most difficult aspect of log management, but it is also normally the most significant. Although analysing log information is sometimes perceived by administrators as boring and ineffective -small amount for some effort- getting strong log organization infrastructures is crucial. Furthermore, the automation of the analysis process, at best effort, will change the analysis procedure significantly by reducing time while producing more important results.

Though logging is not an entirely new idea, it has soon become the epitome of management and observation with the most used feature being remote monitoring. The most significant point of logging is that you learn about matters as and when they happen, even before users themselves begin realising it.

In real time it becomes a little more difficult. It is really challenging to be able to do any type of real time analysis on the huge amounts of log files that are streaming into these devices. There are tools out there however that can parse these logs and try to keep track of things at least in near real time. It is of high importance, for the security



expert to be able to be alerted and identified as quickly as possible should something odd be happening. Concluding, the real key when dealing with a vast amount of log files is just to find a way to automate the above-mentioned procedure.

Log analysis as stated earlier, includes the collection of log data, the cleaning of that data, the conversion into a structured form and analysing them to obtain results. Logs are always unstructured data so they must be collected in a place to extract the important information from them to do a successful conversion.

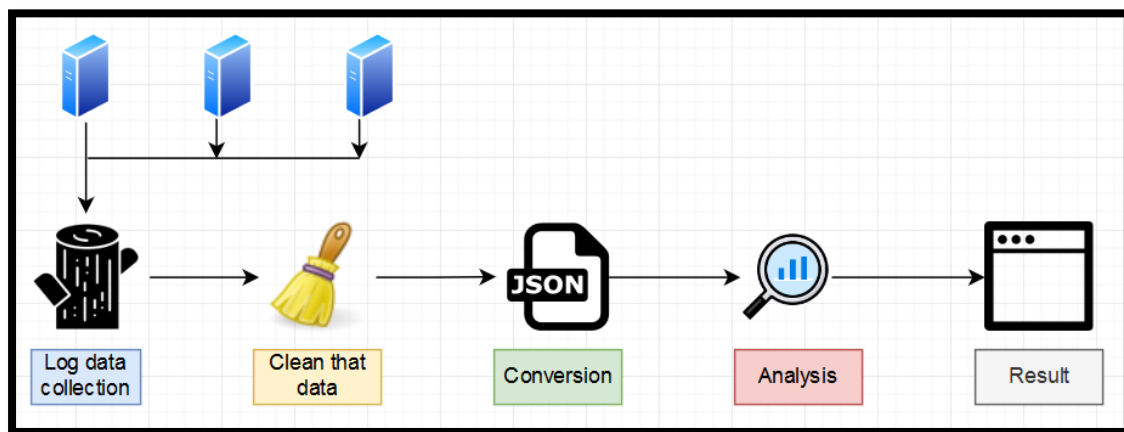


Figure 3: Efficient Log Analysis steps

This is the general flow that is followed for getting information that is relevant to the company's purpose. So rather than wasting time in reading from top to bottom it is not a common secret among administrators to drill down certain key-phrases and then analyse the data needed.

1.4. Problems with log analysis

The first challenge with log analysis is the lack of consistency in log and time format. Each application has its own way to produce logs. This can be easily seen between two different web applications; Apache server and Tomcat, since they use their own syntax. Below there is a demonstration for Tomcat and Apache server accordingly.



```
11-May-2016 12:08:02.318 INFO [http-nio-8084-exec-130] org.apache.catalina.core.ApplicationContext.log Destroying Spring FrameworkServlet 'dispatcher
11-May-2016 12:08:02.399 INFO [http-nio-8084-exec-130] org.apache.catalina.core.ApplicationContext.log Closing Spring root WebApplicationContext
11-May-2016 12:08:35.199 INFO [localhost-startStop-2] org.apache.catalina.core.ApplicationContext.log Destroying Spring FrameworkServlet 'dispatcher
11-May-2016 12:08:35.245 INFO [localhost-startStop-2] org.apache.catalina.core.ApplicationContext.log Closing Spring root WebApplicationContext
11-May-2016 12:04:13.366 INFO [localhost-startStop-1] org.apache.catalina.core.ApplicationContext.log No Spring WebApplicationInitializer types dete
11-May-2016 12:04:13.487 INFO [localhost-startStop-1] org.apache.catalina.core.ApplicationContext.log Initializing Spring root WebApplicationContext
11-May-2016 12:04:21.109 INFO [localhost-startStop-1] org.apache.catalina.core.ApplicationContext.log Initializing Spring FrameworkServlet 'dispatch
11-May-2016 12:05:36.422 INFO [http-nio-8084-exec-16] org.apache.catalina.core.ApplicationContext.log No Spring WebApplicationInitializer types dete
11-May-2016 12:05:36.501 INFO [http-nio-8084-exec-16] org.apache.catalina.core.ApplicationContext.log Initializing Spring root WebApplicationContext
```

Figure 4: Tomcat Log file example

```
30.11.2010 2:47:12 GET /wp-content/w3tc/min/dc2bc0/default.include.61659561.css HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/w3tc/min/dc2bc0/post.include.footer.4248003223.js HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/trentanove.gif HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/vertical_stripe.png HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/64/home.png HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/64/info.png HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/64/rss.png HTTP/1.1
30.11.2010 2:47:13 GET /wp-content/themes/restatement/images/64/attach.png HTTP/1.1
30.11.2010 2:47:14 GET /wp-content/themes/restatement/images/rarstnet.png HTTP/1.1
30.11.2010 2:47:14 GET /images/icons/windows.png HTTP/1.1
```

Figure 5: Apache Server Log file example

It is of high importance to understand the syntax that is used, to be able to read a log file of a particular application. These are some of the challenges which occur in a heterogeneous environment of multiple applications. This problem also happens when an application has a different configuration to describe time. In more detail, an application might have the UTC format while another might have the Eastern Time zone. Noteworthy is the fact that many log formats are unique and proprietary to a certain company's technology. As a result, many times the company to force the collecting organisation to use their technology since a decent log management system is of vital importance.

Moreover, every single server has its own log directories which are stored in a decentralized way. This is a discomfort for the expert since he has to log into each server to troubleshoot.

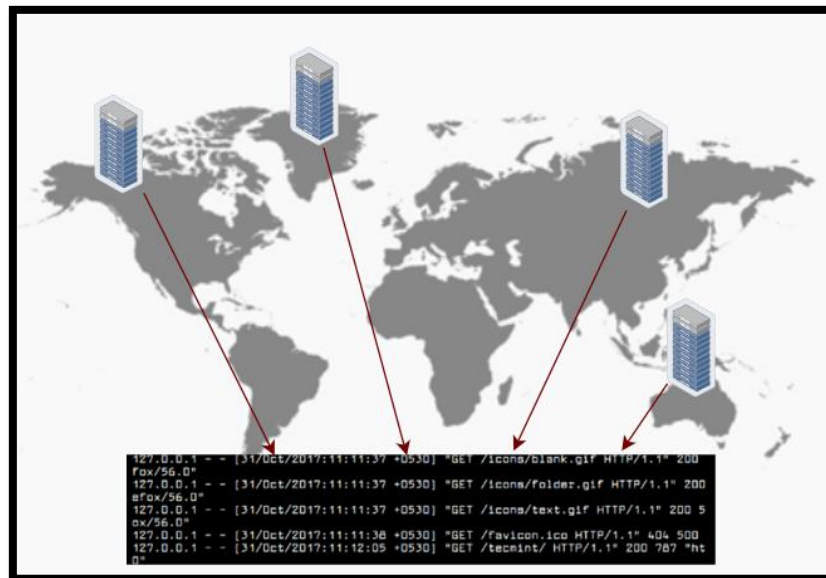


Figure 6: How decentralization works

Concluding, each user inside a team or company will not have access to those log directories to visualize the logs. This problem also includes the fact that common folk might not have technical expertise to understand the information and this can slow down the whole analysis procedure.

1.5. Security Operation Center (SOC)

It is essentially a facility that combines applications, databases, servers, rulesets and analyst tools put together to identify network and infrastructure threats. So basically, it is a centralized unit that deals with security issues on an organizational and technical level. Noteworthy, SOC is traditionally a physical facility which houses an information security team. SOC teams are made up of management, security analysts and sometime security engineers. It works with development and IT operations teams within the company.

The objective of a SOC is related mainly to the people, technologies and processes that provide situational awareness through the detection, containment and remediation of IT threats. A SOC will handle, on behalf of a company or institution any threatening IT incident, and will ensure it is properly investigated, analysed and reported. Lastly, it also monitors applications to identify and prevent a possible cyber-attack or event.



Building an efficient SOC requires organizing internal resources in a way that improves communication and increases efficiencies. Aside from the staff, which is fairly hard to find, it is imperative for the company to have the latest tools to keep up to date with threats.

A Security Operations Center is often referred to as Security Defence Center (SDC), Security Analytics Center (SAC) or even Network Security Operations Center (NSOC).

2. Elastic Stack

2.1. The present and future of monitoring with Elastic stack

Being proactive as opposed to reactive is the key component in the cyber security space. Since the importance of log analysis is analysed before, now the most commonly used -open-source- combination of projects will be analysed.

The Elastic Stack is the most popular log analytics method in the contemporary IT world. It gathers logs from all applications, services, networks, instruments, servers and more from the environment into a single, centralized place for process and investigation. It is utilized for analysis purposes (e.g. to troubleshoot issues, monitor services, and decrease the time it takes to resolve operating issues). Another purpose for this instrument is for security and auditing (e.g. to observe changes in groups - security wise- and changes in privileges). After getting alerts on these subjects, it is simple to pursue unauthorized users and suspicious actions. Elastic Stack is also commonly used for business for supervising users and their behaviour. It is important to mention that Elastic Stacks often are expensive but at the same time very efficient in giving users the power to quickly mine and chart logs.

Often used with its former name “ELK Stack”, is the acronym of three open source projects; Elasticsearch, Logstash and Kibana. Elasticsearch is the searching and analytics engine. Logstash is the server-side information processing line that ingests information from multiple sources simultaneously, transforms it, and afterwards transmits it to the “stash” like Elasticsearch. Kibana allows users to visualise information with charts and graphs at Elasticsearch. Despite each one of these three technologies being a separate project, they have been developed to work exceptionally and harmonically together. ELK Stack was re-branded as Elastic Stack since the Beats project was introduced, which made the acronym ELKB impossible to pronounce effortlessly. Beats are light agents that are established on hosts to gather various type of information for forwarding into the other components of the stack. Although it is an open source engine for searching and analysis, it is also based on the Apache Lucene web search engine, mainly referring to the Elasticsearch component.



Elastic Stack is actually really reliable and good even though it's an open source assemblage of projects. Multiple companies and organisations now rely on this technology to perform necessary tasks that will be later described thoroughly.

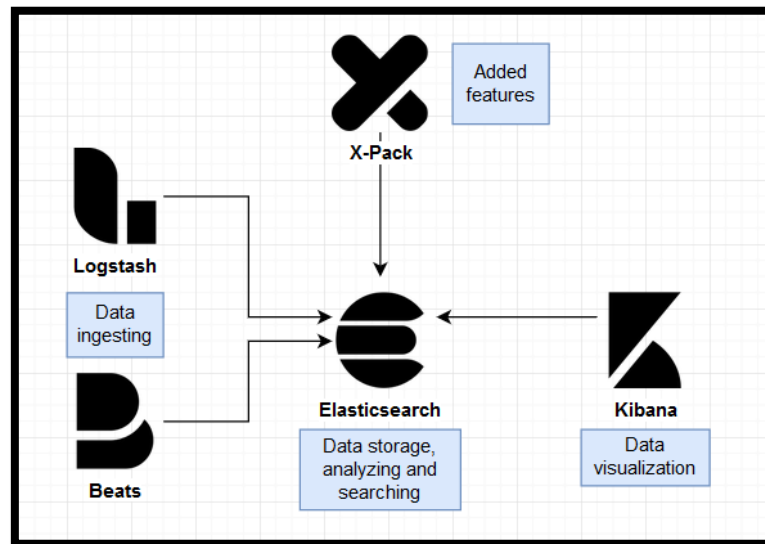


Figure 7: Basic Elastic Stack structure

2.2. Components

2.2.1. Elasticsearch

2.2.1.1. General idea

To begin with Elasticsearch, it is an open-source analytics and full-text search engine and the main building block of Elastic Stack. It is often used for enabling search functionality for applications. A great example could be that someone has an online store for which he wants customers to be able to explore the product selection that it offers. That could be categories, standalone items and so on. Complex search functionality is possible with Elasticsearch, like what someone can see on a search engine. This includes auto-completion, correcting typographic mistakes, highlighting matches and adjusting relevance etc.

By taking further the case study of the online store, suppose it has implemented a searching feature. Besides searching through product names and other full-text fields, it is highly advisable to take several aspects into consideration when sorting the end



results. If the products have ratings, it is probably a good idea to boost the relevance of a product that is best rated. Filtering and sorting options for possible clients is highly desirable such as price range, brand or even sex. To simplify things, Elasticsearch can do everything that is needed for developing a powerful search engine. Full-text searches though, are not the only thing Elasticsearch is capable of, as already stated before.

Queries can be made for structured data much like numbers and use Elasticsearch as an analytics platform. These queries can may well be for data and then the results contribute in making charts. It is of highly importance that Elasticsearch must not be confused with a business intelligence solution. Despite not being one, Elasticsearch can indeed provide the user a plethora of valuable information out of the data that is stored inside it.

Apropos of considerable amount of data analysing, Elasticsearch is really great. An example on what can be done, is to use machine learning to forecast sales based on historical data. This is an addition provided by X-Pack which will be described in detail later on this thesis. Another great example of usage could be a capacity management one. To be more precise if an internet service provider's support group integrates this technology to its infrastructure it will be simplified to manage personnel. This can happen with keeping track of how many phone calls are made and then forecasting and deciding with ease how much staff the company will need in the future.

In Elasticsearch, data is stored as documents, which is essentially just a unit of information. A document in Elasticsearch corresponds to a row in a relational database and can represent any entity imaginable. This document contains fields, which correspond to columns in a relational database. A document is essentially just a JSON object, so to add a car as a document, all that's needed is to send a JSON object describing a car to Elasticsearch, such as the example in the image below. More precisely, an object is added with a "brand", a "CC" and a "extra" property.

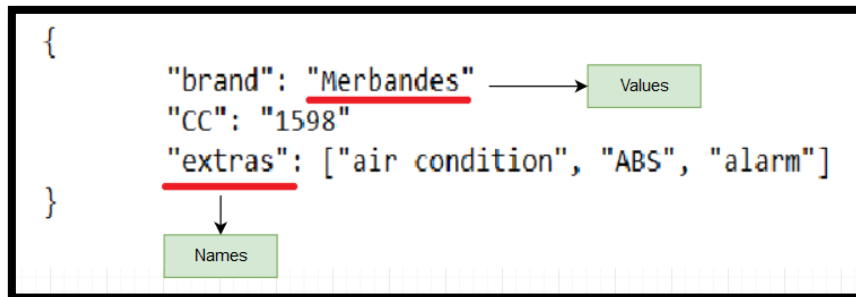


Figure 8: JSON Object for Elasticsearch

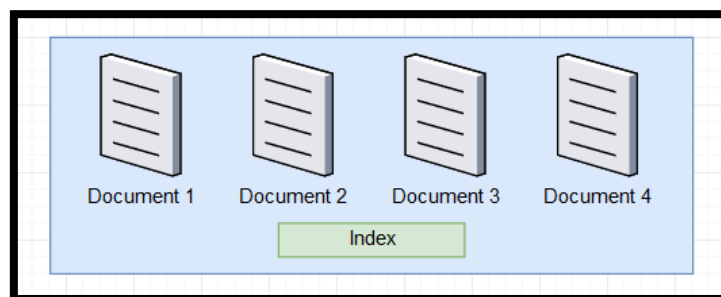


Figure 9: Index and document relation

Documents have IDs assigned to them either automatically by Elasticsearch, or by the user when they are added to an index. So, it only logical to uniquely identify documents by the index they are included in and their ID.

Moreover, to query documents in Elasticsearch, a REST API is used. A RESTful API as shown in the dictionary is just a way of designing HTTP APIs. The queries that are sent to Elasticsearch are also in JSON format.

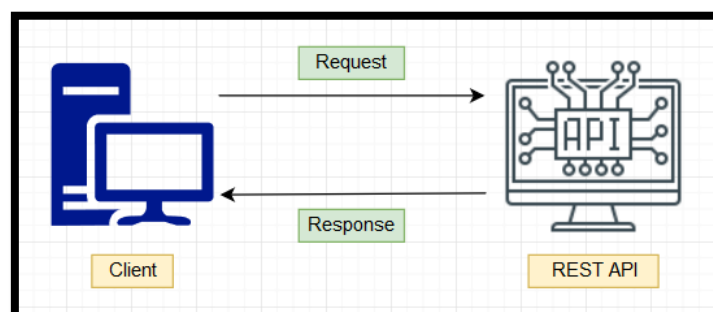


Figure 10: How REST API works



Elasticsearch is written in Java and is built on top of Apache Lucene. It has gained a lot of popularity, due to its relative ease of use and the fact that it is exceptionally good at scaling. While it is easy to get started with simple things, it is still a very complex technology if an organisation wants to make use of its full potential. Since it is distributed by design, it scales very well in terms of increasing data volumes and query throughput. So even if the search must be executed through countless documents, searches are still going to be almost instantaneous.

It is of high significance, that Elasticsearch is a near real time search platform. This means that there is a slight latency -normally one second more or less- from the time a document is indexed until the time it becomes searchable. In other words, Elasticsearch is sort of a distributed database with lots of extra capabilities.

2.2.1.2. DBMS and Elasticsearch

In a common Database Management System, data is presented in tables as shown in the figure below. It is clear that these tables consist of columns and rows while on the other hand, as stated earlier in Elasticsearch, the JSON format is used. So, it is clear that one row in a DBMS is equal to one document in the Elasticsearch.

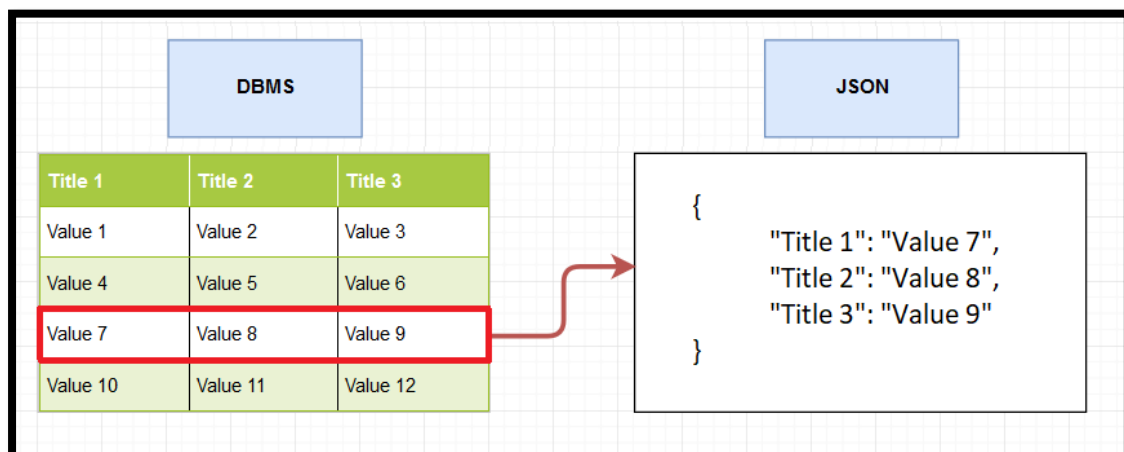


Figure 11: DBMS and JSON format

To continue, an index in Elasticsearch is similar to a database in a Relational Database Management System. More precisely, an index is a collection of documents that have somewhat similar characteristics. A RDBMS consists of a number of tables, which hold a number of records. Similarly, in A RDBMS, Tables are used which are



referred as Types in Elasticsearch. Distinguishing between indices and types is often very difficult for beginners, and understanding when to use each, is often a challenge. A Type is a logical category/ partition of the index whose semantics is completely up to the expert. It is important that one or more Types can be defined within an index. Types although being important they got somewhat removed from Elasticsearch gradually.

So, it works as shown in the figure below. Any given document, in any format, hosts data that are afterwards ingested by Elasticsearch. On this application all the individual words are scripted and stored in an inverted index format. It is very helpful that way for when data is fetched through a quick search or even for analytical purposes.

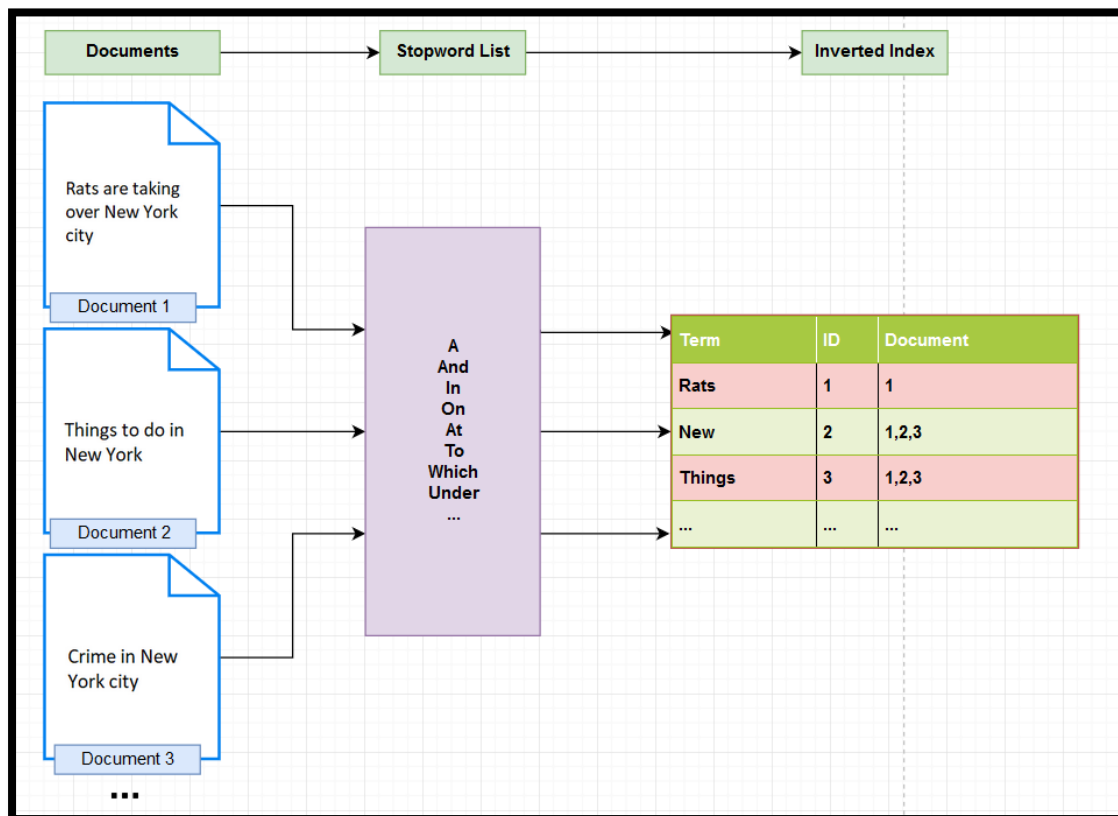


Figure 12: Inverted index format storing

2.2.1.3. Concurrency issues

When the topic of discussion is distributed systems, sometimes really weird problems with concurrency pop up. The question is, what really happens when two different



clients are trying to do the same thing at the exact same time and who actually wins. Elasticsearch deals with these issues.

The use case as already stated is about two clients that are running a web application on a big distributed site and they are maintaining page counts for given documents that are viewed. These documents will be called pages on this website. So, two people are viewing the same page at an exact moment through two different web servers. Basically, at this point two clients exist for Elasticsearch, retrieving a view count for a page from Elasticsearch. Since they are both asking at the same time at this point, they are just trying to get the current count for that page. Let's say that it comes back with a number ten. So, ten people look to this page so far. Now both of these clients want to increment that at the same time. They will eventually go ahead and increment the view count for that page and figure out that a new update for this document must be written, for a view count of eleven. Subsequently, they will both in turn write a new document update with a new view count of eleven, but it should have been twelve.

It is obvious that there is a brief window of time between retrieving the current view count for the page and writing the new view count of the page during which things went wrong due to this concurrency issue. This is a very real problem if a lot of people are hitting a specific website or the Elasticsearch service at the same time.

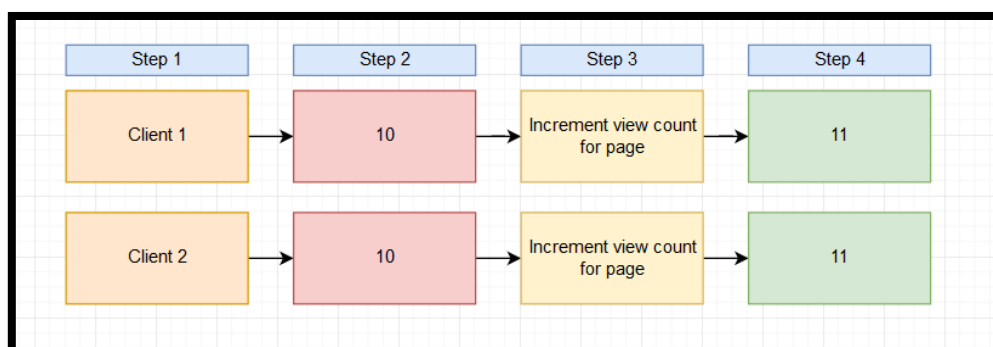


Figure 13: Concurrency Control Part 1

The solution is called Optimistic Concurrency Control. It is important to understand that whenever a request for Elasticsearch is given, it returns back the version number for that document. So, for this use case, the view count of ten is associated explicitly with a given version number of that document. Just for sake of this example the



version will be called “_version: 7”. Eventually, when an update occurs the version must be also specified to see if two or more people try to update the same version at the same time. This has a result only for one of them to be successful. Elasticsearch could then change the version to “_version: 8” and announce that the second client is basing this update on the wrong information. At that point the client tries to reacquire the current view count for that page.

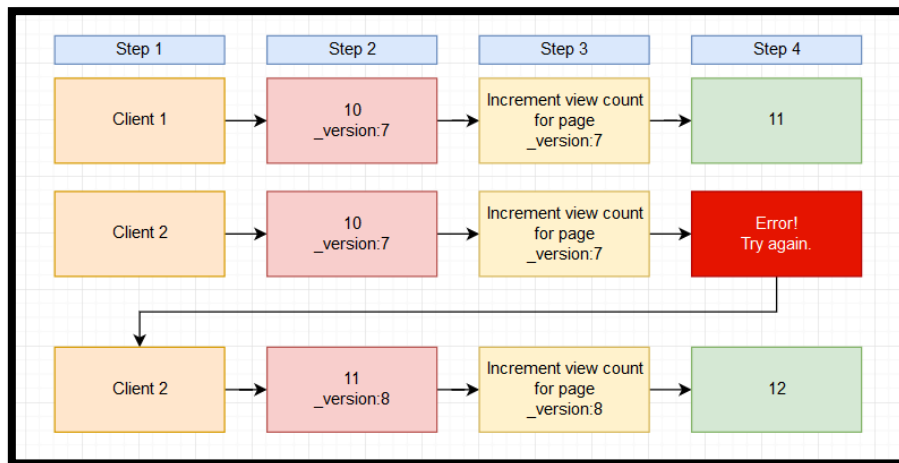


Figure 14: Concurrency Control Part 2

To avoid doing this by hand there is another parameter called `retry_on_conflicts` (use `retry_on_conflicts = N` to automatically retry) that will allow you to automatically retry if this happens so that’s kind of a nifty feature.

2.2.1.4. Scaling

2.2.1.4.1. Architecture

2.2.1.4.1.1. Nodes and clusters

By cluster of computers, it is meant, a collection of one or more nodes that together hold all the data and provides federated indexing and search capabilities across all nodes. Therefore, the collection of nodes contains the entire data set for the cluster. A node on the other hand, is a single server that is part of the cluster that stores the data and participates in the cluster’s indexing and search capabilities. This means that a



node will participate in a given search query by searching the data that it stores. Usually a master node exists which is responsible for lightweight cluster-wide actions such as creating or deleting an index, tracking which nodes are part of the cluster, and deciding which shards to allocate to which nodes. So basically, it is the cluster coordinator and the component that is responsible for updating the state of the cluster. Each node can be assigned as being the master node by default. It is important for cluster viability to have a stable master node. It is perfectly valid to have a cluster with only one node.

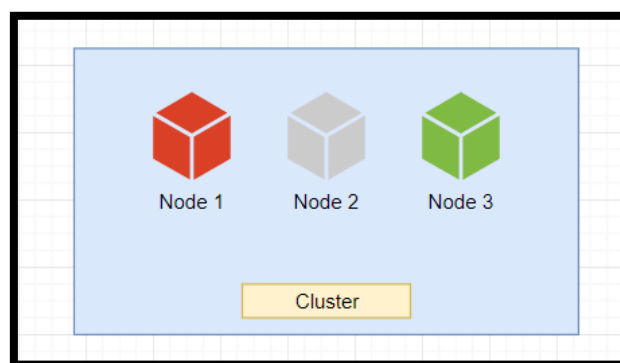


Figure 15: Cluster and nodes relation

Apart from that, it is worth mentioning that every node within the cluster can handle HTTP requests for clients that want to send a request to the cluster. This is done by using the HTTP REST API that the cluster exposes. A given node then receives this request and will be responsible for coordinating the rest of the work. Also, a given node within the cluster knows about every node in the cluster and is able to forward requests to a given node by using a transport layer, while the HTTP layer is exclusively used for communicating with external clients, such as an application, for instance.

Both clusters and nodes are identified by unique names. For clusters, the default name is `elasticsearch` in all lowercase letters and the default name for nodes is a Universally Unique Identifier also referred to as a UUID. This can be changed according to the expert's needs. It is worth knowing that the names of nodes are important because that is the way to identify which physical or virtual machines correspond to which Elasticsearch nodes. By default, nodes join a cluster named `elasticsearch`, but the nodes can be configured to act differently. If no cluster already exists with that name,



it will be formed. Therefore, in a production environment it is a good idea to change the default name, to make sure that no nodes accidentally join a production cluster. Lastly as with documents and their corresponding IDs, indices are identified by names which must be in all lowercased letters. These names are then used when searching for documents, in which case the index must be specified to search through for matching documents. The same applies for adding, removing and updating documents.

2.2.1.4.1.2. Shards

One of the most common questions that occur when discussing about Elasticsearch is how well it scales itself out to run on an entire cluster of computers. The main trick is that an index is split into shards. Every shard is a self-contained instance of Lucene and of itself. This means that it is just a logical partition of the index. So, the idea is that if a cluster of machines exist, these shards can be spread across them. If more capacity is needed though, simply more computers can be introduced to the cluster and more shards added to the entire index for a more efficient spread of the load. Every time a given server in the cluster is used, once it figures out in what document the user is interested in, it hashes the document to a particular shard ID. As a result, a mathematical function will be used to find expeditiously which shard owns the given document and then redirect the user to the appropriate shard on the cluster. So simply, index is distributed among many different shards and a shard can live on different machines within a cluster.

In order for Elasticsearch to maintain resiliency to failure, shards are divided into two categories: primary and replica ones. A primary shard and its replicas are referred to as a replication group. A big problem that occurs when a cluster of computers exist, is that those computers can fail sometimes, and something must be done in order to prevent or countermeasure it. An example is present in the figure below where an index is presented that has two primary shards and two replicas. So, in this example three nodes will be used, where each node is an installation of Elasticsearch. Usually one node is installed per physical server in a cluster. The design is such that if any given node in the cluster goes down, the end user will not even realise its absence.



Primary shards, in this example, are basically the primary copies of the index data and that's where write requests are going to be routed to initially. Then that data will be replicated to the replica shards which can also handle read requests whenever it is desired by the user. Elasticsearch figures everything automatically in behalf of the user, so if the user wants two primaries with two replicas, in three given nodes, the end figure will be as described.

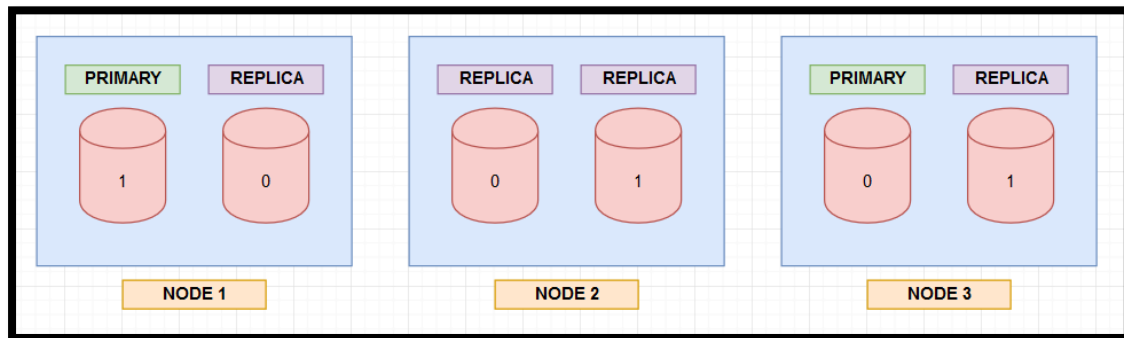


Figure 16: Shard resilience Part 1

The case scenario is that Node 1 suddenly fails for some unspecified reason (the power supply burned out, disk failure etc). So, in this case primary shard 1 and replica shard 0 will be lost. This is a minor problem since a replica shard 1 exists both in node 2 and 3. Elasticsearch would figure that out and it would elect one of the replica nodes on 2 or 3 to be the new primary. Since the shards are there, they can keep on accepting new data and servicing read requests because now there are only down to one primary and one replica. This will happen until the lost node is restored. Similarly, if node 3 collapses, Elasticsearch would pick a new primary shard 0 from node 1 or 2.

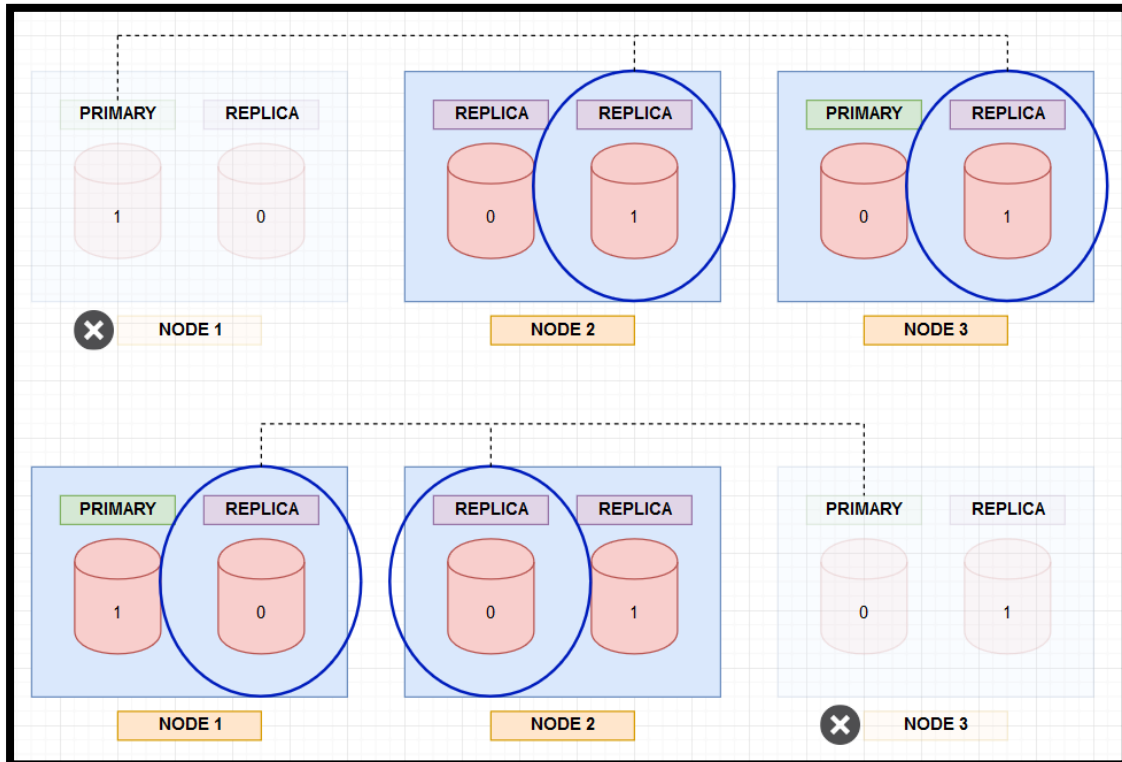


Figure 17: Shard resilience Part 2

By using a scheme like the above mentioned, a fault—tolerant system is created. In fact, it can even support the system even if it loses multiple nodes at the same time. According to this example, two nodes can collapse at the same time with them being node 2 and node 1. Since in node 3 a primary shard 0 exists all there is to be done is for Elasticsearch to elect the replica shard 1 into primary shard 1. For this sort of resiliency, it is recommended to use an odd number of nodes.

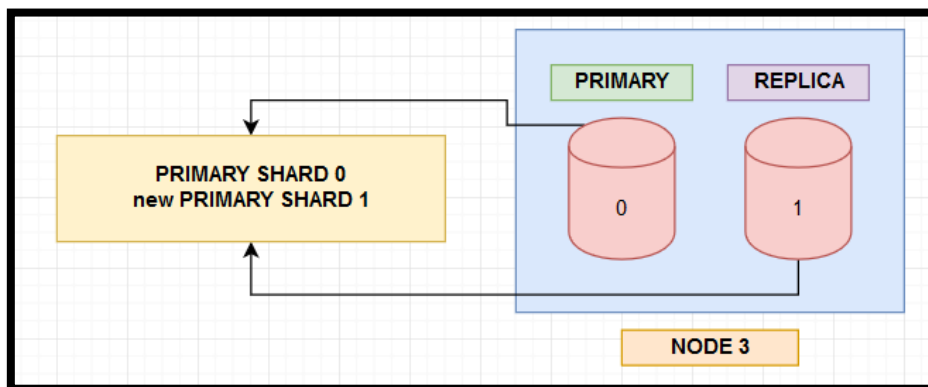


Figure 18: Shard resilience Part 3



To conclude the idea is that the system will round-robin requests as an application among all the different nodes in the cluster so that the initial traffic load will be spread out evenly. It is only logical for this procedure to be efficient that a replica node should never be stored on the same node as its primary shard.

2.2.1.4.2. Replica shards synchronization

It is clear enough that replicas need to be kept in sync, because otherwise problems would popup. These problems occur mainly when deleting or updating a document since it duplicated across nodes. This has a result for queries to become unpredictable, mainly because the result would depend on which particular replica shard is read from. To deal with this Elasticsearch uses a model named primary backup for its data replication. This means that the primary shard in a replication group acts as the entry point for indexing operations. More specifically, all operations that affect the index- such as adding, updating or removing documents- are sent to the primary shard. The primary shard is then responsible for validating the operations and ensuring that everything is good. This involves checking if the request is structurally invalid, such as trying to add a number to an object field or something equivalent. When the operation has been accepted by the primary shard, the operation will be performed locally on the primary shard itself. Afterwards, at the time of completion, the operation will be forwarded to each of the replica shards in the replication group. If the shard has multiple replicas, the operation will be performed in parallel on each of the replicas. Eventually, if the operation is completed successfully on every replica and responded to the primary shard, the primary shard will respond to the client that the operation was a success.

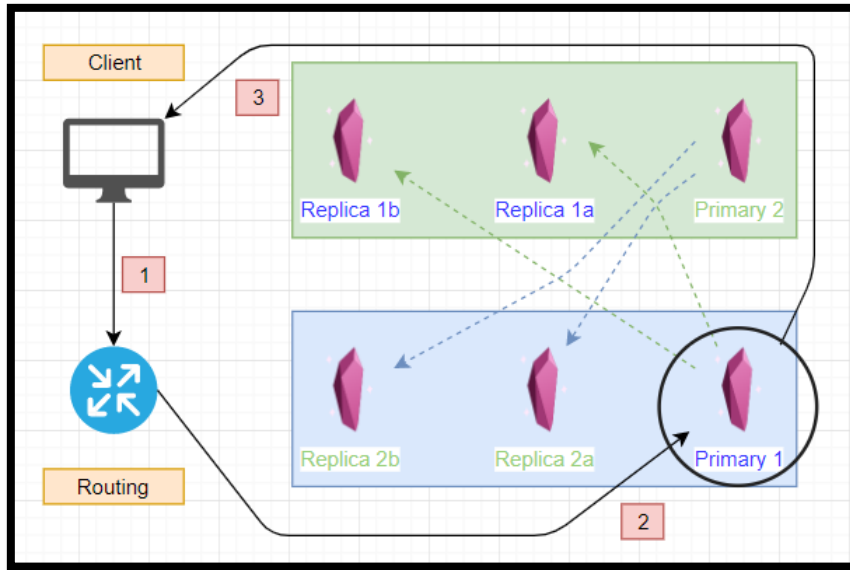


Figure 19: Synchronization of replica shards

A simple case study would be a client/ server that communicates with the cluster. The cluster is consisted of two nodes, which are parted off one primary shard and the replicas of the other node's primary shard. Firstly, a document from the index is to be deleted. At this point, Elasticsearch needs to find the correct replication group, and thereby also the primary shard. With correct routing the client finds the corresponding replication group and its primary shard. The operation is then routed to the primary shard where it is validated and subsequently executed. Once the operation completes on the primary shard itself, it is sent to the replica shards in the replication group. In this case that means that the delete operation is sent to Replica 1a and Replica 1b. When on both of these replicas the operation completes, the primary shard acknowledges that the request was successful to the client. If there were no replicas the operation would have just been executed directly on the shard.

2.2.1.5. Data searching

Below a figure is presented, with what a developer would typically see when running search queries towards an Elasticsearch cluster. The client in reality can be often a server. That client communicates with the Elasticsearch cluster by sending search queries over HTTP. Afterwards, the cluster handles the procedure based on the index and query that is specified in the HTTP request. When the results are ready, the cluster responds with the results, and the client can then use the results for multiple purposes.

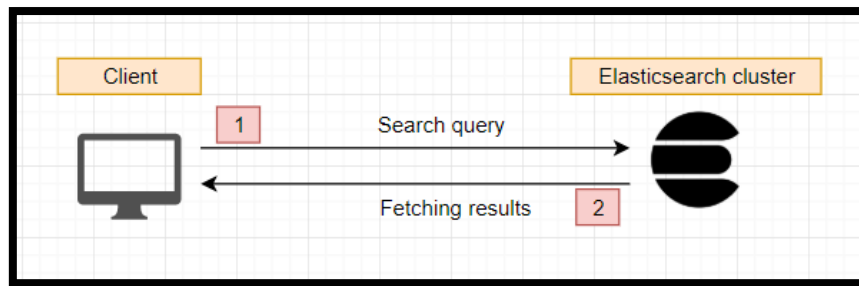


Figure 20: Elasticsearch. How data searching works

For deeper inspection a case study is used as well. As shown below, a cluster consisting of three nodes exists. It contains one index distributed across three shards; Shard 1, Shard 2, Shard 3. Each shard has two replicas, so each replication group consists of a primary shard and two replicas. The first thing that the client does is to send a search query to the cluster which ends up on the Node 2 containing Shard 2. This node is the so-called “coordinating node”, meaning that this node is responsible for sending queries to other nodes, assembling the results and responding to the client. So basically, coordinating the query, hence the name “coordinating node”. By default, every node may act as the coordinating node and may receive HTTP requests as aforementioned. Since the coordinating node itself contains a shard which should be searched, the node will perform the query itself. This may not be the case in other scenarios, but since there is only one index, this will always be the case. Afterwards, the coordinating node broadcasts the request to every other shard in the index, being either a primary shard or a replica shard. In this example the primary shards, Shard 1 and Shard 3 receive the requests. When the other shards respond with each of their results, the coordinating node merges them together and sorts them, and lastly returns the results to the client.

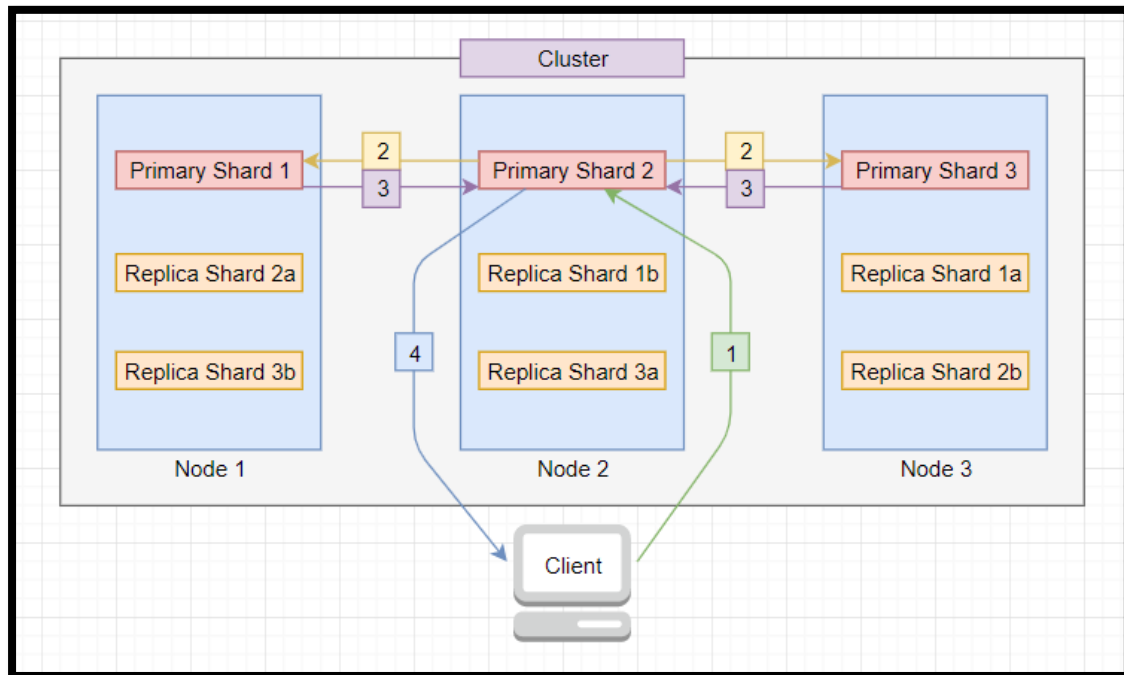


Figure 21: Queries handling

It is noteworthy that the approach is different if a single document is retrieved by its ID. When doing this, the request is routed to the appropriate shard instead of being broadcasted to all of the index's shards.

2.2.1.6. Document distribution and routing

Elasticsearch is capable of knowing on which shard to store a new document, and how to find it when retrieving it by ID. Documents should be distributed evenly between nodes by default, so that one shard will not contain way more documents than another. So, determining which shard a given document should be stored in or has been stored in, is called routing. To make Elasticsearch more user friendly and easier to handle, routing is handled automatically by default, and most users will not even need to manually deal with it. The way it works by default, is that Elasticsearch uses a simple formula for determining the appropriate shard. By default, the "routing" value will be equal to a given document's ID. This value is then passed through a hashing function, which generates a number that can be used for the division.



$$\text{shard} = \text{hash}(\text{routing}) \% \text{total_number_of_primary_shards}$$

Figure 22: Shard number calculation

The remainder of dividing the generated number with the number of primary shards in the index, will give the shard number. This is how Elasticsearch determines the location of specific documents. When executing search queries (not looking a document with its ID), the process is fairly different, as the query is then broadcaster to all shards. This default behaviour ensures that documents are distributed evenly across shards. Since this is just a default configuration, it can be changed at will. To be more specific, when retrieving, deleting, or updating documents, a custom routing value can be specified to change the way documents are distributed.

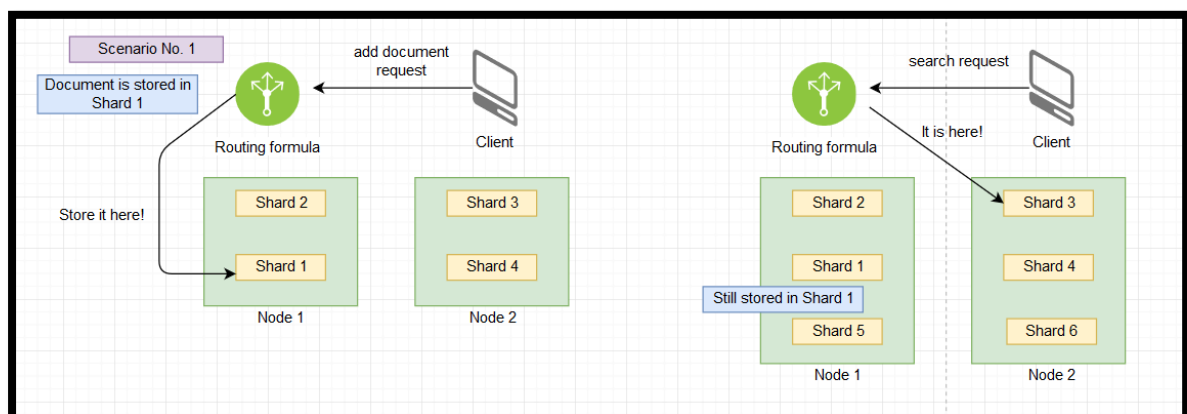


Figure 23: Add and search document requests

An example of this could be if a document existed for each car, in which case the shard could be determined by the country that they are manufactured at. In that case, a potential problem could be if the majority of the cars were manufactured in the same country, because then the documents would not be evenly spread out across the primary shards.



The number of shards for an index cannot be changed once an index has been created. Taking the routing formula into consideration, then the answer is pretty obvious as to why this is the case. If the number of shards changed, then the result of running the routing formula would change for documents. So, if a document has been stored on Shard 1 when there are four shards in total, because that is what the outcome of the routing formula was at the given time. Also, hypothetically the number of shards is changed later into a total number of six. If someone tries to lookup the document by ID, the result of the routing formula might be different. Now the formula might route to Shard 3, even though the document is actually stored on Shard 1. This means that the document would never be found and that would cause some major issues. Concluding that is why the number of shards cannot be changed once an index has been created. The only solution is to create a new index and move the documents to it. The same problem could happen if custom routing is introduced within an existing index that contains documents that have been routed using the default routing formula.

2.2.1.7. Read and write capacities

A clear use case would be to index a new document into Elasticsearch, that is going to be a write request. When this is done, whatever node the system talks to, will respond ‘‘ok, here is the location of the primary shard for this document you are trying to index. So, I will redirect you to where that primary shard is located’’. Afterwards the data index will be written into the primary shard and then that will automatically get replicated to any replicas for that shard.

When read occurs on the other hand, the process is not only quicker but simpler as well. It is routed to the primary shard or to any replica of that shard so that the load of reads can be spread out more efficiently. This means also that the more replicas there are, the read capacity is increased for the entire cluster. It is only the write capacity that is going to be bottlenecked by the number of primary shards.

One big issue with this architecture is that the number of primary shards is unchangeable. This is defined when the index is created up front and the syntax of that is presented in the below figure.



```
PUT /unipi_index
{
  "settings":
  {
    "number_of_shards": 3,
    "number_of_replicas": 1
  }
}
```

Figure 24: Counting of shards

It is as can be seen a put request, 3 primary shards and 1 replica. It is not clearly visible, but the cluster here ends up with 6 shards. This happens because 3 primary shards will be used and 1 replica for each will be created. So, if the request consisted of 2 replicas, the total number of shards would add up to 9, and so on.

This is not as bad as it sounds because a lot of applications of Elasticsearch are very read heavy. More specifically, this can be easily seen when a search index is powered on a big website like an online encyclopedia. There, the number of new read requests will outnumber the indexes for new documents. So, it's not as rough as it sounds since a lot of applications oftentimes can get more replicas inside the cluster's infrastructure later. As already stated, this can add more read capacity, but in order to add more write capacity, a re-index of the data into a new index must happen. Although these are a viable option, it is far more desirable to plan and make sure that enough primary shards exist to handle any growth that might reasonably appear soon.

2.2.1.8. Apache Lucene

Lucene is a powerful Java search library which is built for creating indexes that are easily searchable. An Elasticsearch shard is essentially a Lucene index. This Apache framework is the full-text search library, that Elasticsearch is built on. Elasticsearch is basically making Lucene's functionality available in a distributed setting.

Within a Lucene index there are segments. These segments are simply, mini indexes and within them there are certain data structures, like an inverted index, stored fields, document values etc. When working with search, the key data structure to understand is the inverted index.



The inverted index consists of two parts; the sorted dictionary and a posting list. The dictionary contains the index terms and for every term its posting list is the documents that contain the term.

In the figure above a basic example of indexing is shown but the principle is the same for all kinds of searches. First the user operates on the dictionary to find candidate terms and then operate on the postings.

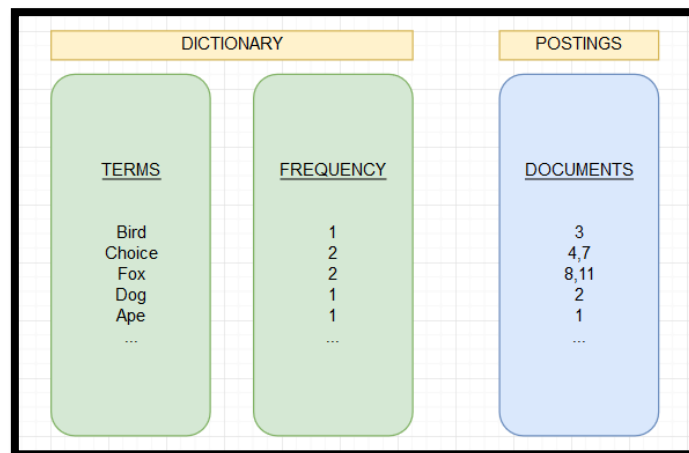


Figure 25: Indexing in Apache Lucene

It is noteworthy that segments are immutable. This means that they cannot be completely deleted if a document is deleted. More specifically whenever a document is deleted there is a bitmap that marks the document as deleted and Lucene will filter it out for every subsequent search. The segment itself doesn't change though. So, when update for example is essentially a delete followed by a reindex this should be kept in mind.

Lucene on the other hand, uses all the tricks in the book to compress things. Segments can be created in one of two ways. First as new documents are indexed Elasticsearch will buffer these documents, and then every refresh interval, which defaults to every second, it will write a new segment and the documents will become available for search. This means that over time lots of segments will stack up. To solve this problem, every now and then Elasticsearch will merge them together and during this process deleted documents are finally, completely removed. That's why adding documents sometimes can cause the index to be smaller.

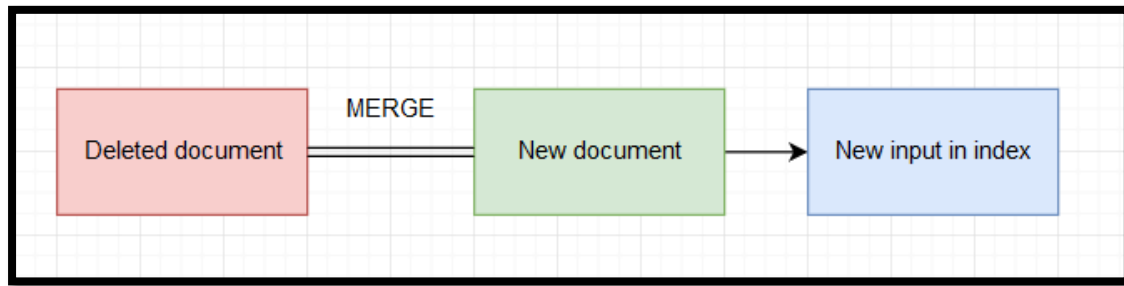


Figure 26: Updating and re-indexing in Apache Lucene

Many of the aforementioned are equally presented at Elasticsearch. This was expected though, since Elasticsearch was built on top of Lucene.

2.2.2. Kibana

2.2.2.1. General idea

Kibana is a really good way for querying and visualizing information at Elasticsearch. It is a JavaScript-based web application. The Kibana tool is a platform mainly used for analytics and visualization for the convenience of the user. It is called as the Graphical User Interface (GUI), named by the IT community. To be more precise, Kibana lets you visualize data from Elasticsearch and analyse it to make sense of it. Someone can say that Kibana is an Elasticsearch dashboard where you can create visualizations such as pie charts, line charts and many others.

For example, a company can plot its website visitors onto a map and show traffic in real time. You can aggregate website traffic by browser and find out which browsers are important based on its particular audience. Kibana is also where change detection and forecasting are configured. It provides an interface to manage certain parts of Elasticsearch, such as authentication and authorization. Generally speaking, Kibana can be seen as a web interface to the data that is stored within Elasticsearch. It uses the data from Elasticsearch and basically just sends queries using the same REST API. It just provides an interface for building those queries and lets the user configure how to display the results. This can be a huge time saver because there isn't a necessity for the user to implement all these by himself.



Dashboards can be built where the user can place a number of metrics and visualizations or even be used for system administrators to monitor the performance of servers, such as CPU and memory usage. Another example that is worth mentioning is about developers since they can monitor the number of application errors and API response times. It is fairly obvious at this point that user is likely to store a lot of different kinds of data in Elasticsearch, apart from the data that can be searched or presented at the end. It is worth mentioning that user might not even need to use Elasticsearch for implementing search functionality at all.

Searching in Kibana is performed by selecting the search box at the top of the page. Afterwards, by entering a string of data that the user wants to view in the logs (this can be an IP address for instance) the results appear below. Kibana offers the ability to search down to a high level of granularity. More specifically, the user has the option to go down to the millisecond if he needed to.

Moreover, if the user has errors in his syntax, he will be helped by Kibana in ways to refine his query. Documentation is also pointed regarding the syntax which is also very convenient. An error is presented if the search bar has gone red or if the fifteen-minute is exceeded (if not changed).

2.2.2.2. Elastic Cloud

An easier way to use the Elastic Stack suite, without installing its components, is by using a technology called Elastic Cloud. It is a hosted and managed solution for those who want to easily deploy an Elasticsearch cluster and avoid having to manage the underlying infrastructure.

It is free for fourteen days, which is sufficient for the purposes of this Thesis, to understand the basic principles of Elastic Stack. That way as mentioned earlier, installing and configuring Elasticsearch and Kibana is avoided.



Figure 27: Elastic Cloud prerequisites

While setting up the cloud the first thing to do is to select the size of the cluster. This is not possible though since the trial is locked at the following specs. The reason that the slider is locked and that it is not possible to choose a smaller cluster, is that Elastic wants to provide you with high performance for the duration of the trial, just to give the user a memorable and good experience. Of course, someone can put his credit card information and scale it up and down as he pleases.

The following thing to do is to choose a cloud provider. That is because Elastic Cloud runs on top of a cloud provider, being either Amazon Web Services (AWS) or the Google Cloud Platform. It is worth mentioning that AWS has already a service named Elasticsearch Service. The fundamental difference between this and the aforementioned option is that Elastic Cloud provides more capabilities; access to X-Pack. It is also noteworthy that Logstash is not a part of the Elastic Cloud, but it is expected to be added soon.

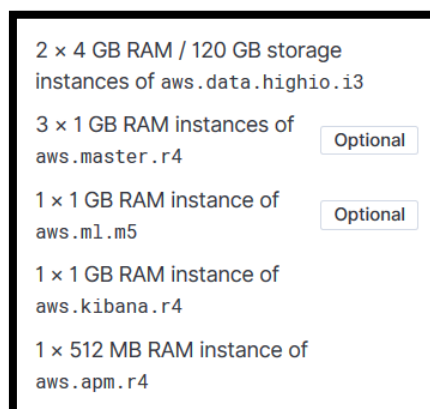


Figure 28: Instances of Elastic Cloud (AWS)



On the other hand, if the Google Cloud Platform is used, then the advantages are even greater, because there is currently no other managed Elasticsearch service available. So, in order to run the Elastic Stack on Google's cloud platform, the user would have to set everything up himself or use a prebuild image. Subsequently though, the user is responsible for managing the cluster and keeping the underlying servers running smoothly. The cloud provider of choice for this Thesis is Amazon though.

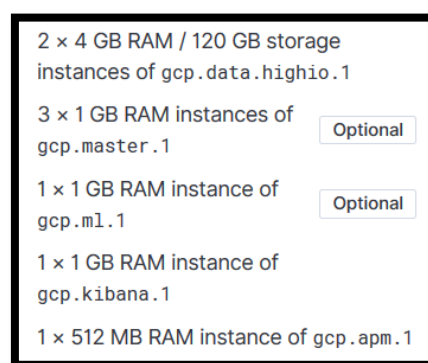


Figure 29: Instances of Elastic Cloud Google Cloud Platform

Afterwards a region should be chosen which is basically where the cluster will be hosted geographically, and the version of the Elastic Stack. The area of choice is EU Frankfurt while the version of the Elastic Stack is 7.3.0. Unless, there are existing applications that are written against a specific version of Elasticsearch, the latest version is recommended.

It is important to understand, that if the user wants to use the cluster for something irrelevant to developing and testing, the region that is chosen should be close to the cluster that it is connected to. This will ensure the lowest possible latency.

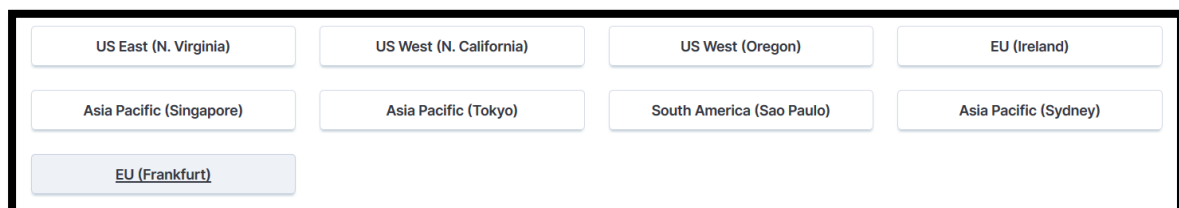


Figure 30: Region choosing

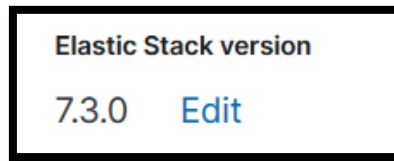


Figure 31: Elastic Stack version used

Finally, the name chosen for this case is JOHN and the final deployment information is presented in the figures below.

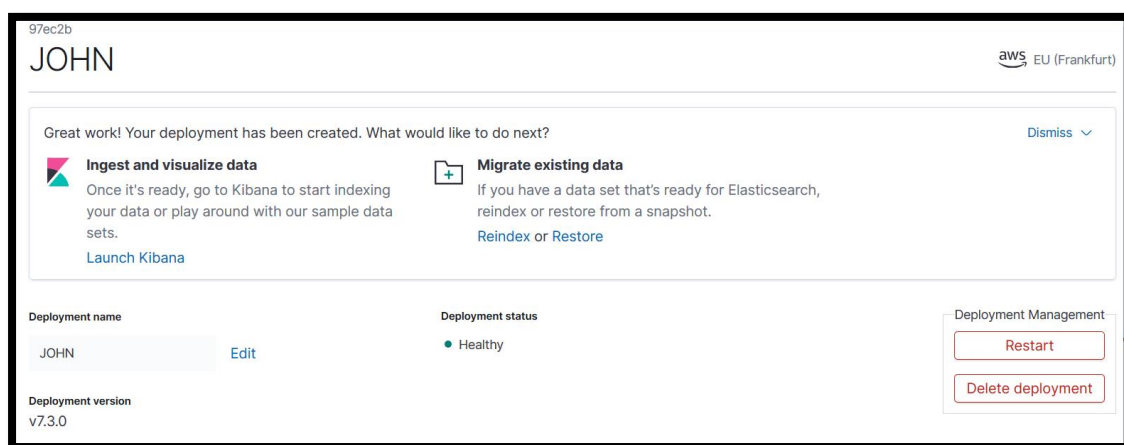


Figure 32: Final deployment info part 1



Figure 33: Final deployment info part 2

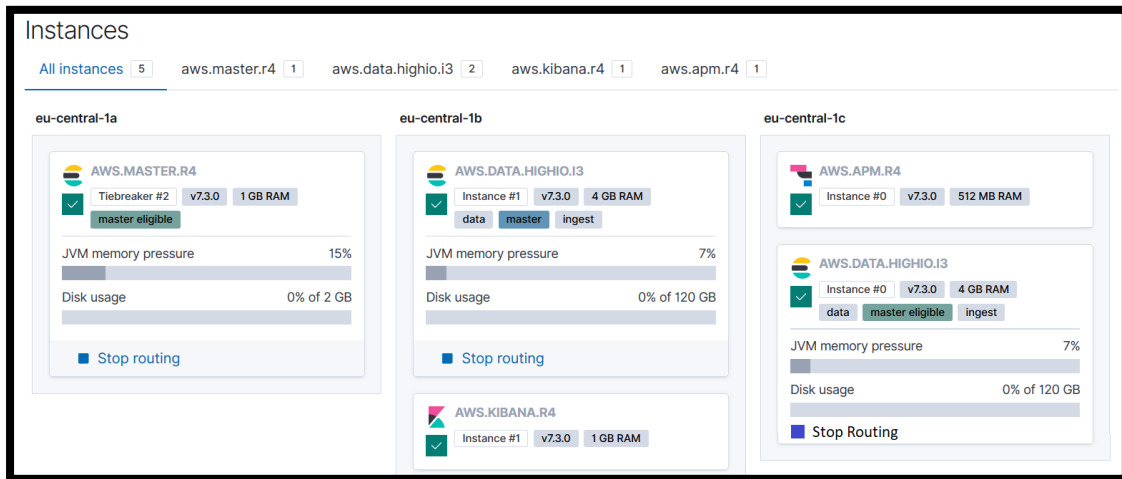


Figure 34: Final deployment info part 3

After a short moment, a dialog appears with the username and password of the cluster. It is crucial to either save it or download it because it is not possible to find it anywhere inside the suite. This happens because it is secured by a login by default, which is a feature of X-Pack. A Cloud ID is also provided, which is needed if data is sent to the cluster from Beats or Logstash.

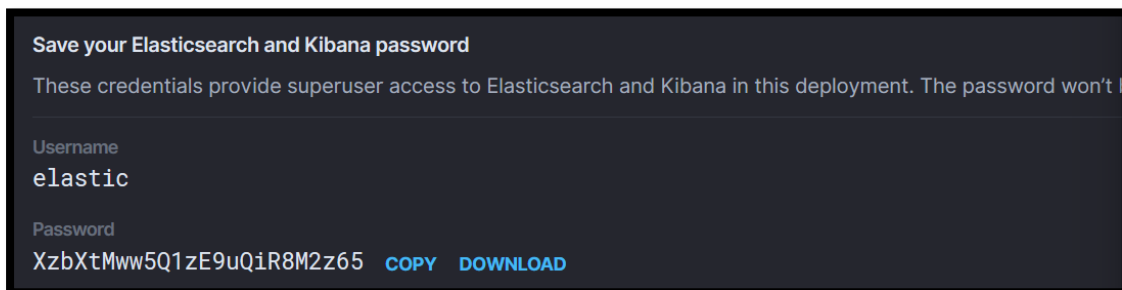


Figure 35: Elastic Cloud credentials

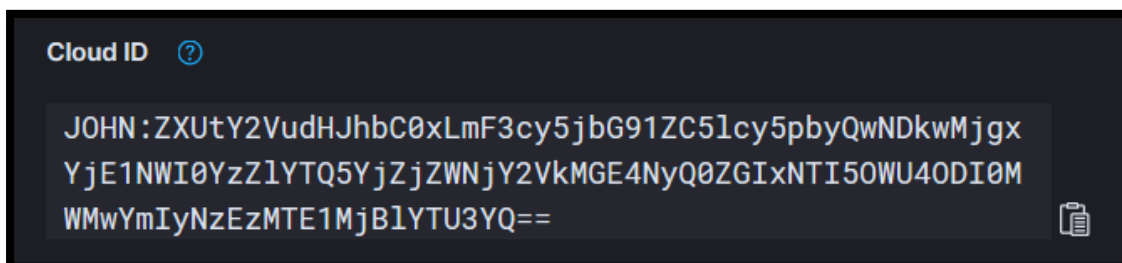


Figure 36: Elastic Cloud ID



A good example about how this service works would be to send a request to the cluster, just to make sure that it can be accessed. To do so, curl HTTP tool will be used in a Terminal window. Since the cluster is secured with X-Pack, authentication is needed to be able to communicate with it. For this to happen, basic HTTP authentication is needed, so it is essential to add a -u argument to curl, followed by the username, a colon, the password and the Elasticsearch endpoint URL.

```
C:\Users\Giannis>curl -u elastic:XzbXtMww5Q1zE9uQiR8M2z65 https://0490281b155b4c6ea49b6ceccced0a87.eu-central-1.aws.cl
oud.es.io:9243
{
  "name" : "instance-000000000", Node that received the request
  "cluster_name" : "0490281b155b4c6ea49b6ceccced0a87",
  "cluster_uuid" : "UfVqPY3ZTG06KRf3mN61NA",
  "version" : {
    "number" : "7.3.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "de777fa",
    "build_date" : "2019-07-24T18:30:11.767338Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 37: Node receiving the request

It is fairly clear from the JSON response, that it contains various details about the cluster, meaning that the user can communicate with the cluster. Another way to access the raw data of Elasticsearch, is to launch it straight through the Elastic Cloud and then type the credentials that were given.

```
{
  "name" : "instance-0000000001",
  "cluster_name" : "0490281b155b4c6ea49b6ceccced0a87",
  "cluster_uuid" : "UfVqPY3ZTG06KRf3mN61NA",
  "version" : {
    "number" : "7.3.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "de777fa",
    "build_date" : "2019-07-24T18:30:11.767338Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 38: Accessing Elasticsearch raw data



From the previous figure, it is visible which node received the request. If a couple more requests are sent, the other node can be reached with a request as well -even though it is on the other data center. In this case, a data center is also referred to as an Availability Zone, in the context of Amazon Web Services.

```
C:\Users\Giannis>curl -u elastic:XzbXtMww5Q1zE9uQiR8M2z65 https://0490281b155b4c6ea49b6cecccd0a87.eu-central-1.aws.c
oud.es.io:9243
{
  "name": "instance-000000001", Node changed
  "cluster_name": "0490281b155b4c6ea49b6cecccd0a87",
  "cluster_uuid": "UfVqPY3ZTG06KRf3mN61NA",
  "version": {
    "number": "7.3.0",
    "build_flavor": "default",
    "build_type": "tar",
    "build_hash": "de777fa",
    "build_date": "2019-07-24T18:30:11.767338Z",
    "build_snapshot": false,
    "lucene_version": "8.1.0",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}
```

Figure 39: Changing the instance-node

The same credentials are also used for launching Kibana which is done by simply pressing the launch option in the Elastic Cloud platform. Since the cluster is empty, an option pop ups for the user to decide if he wants random results to show on the Kibana dashboard, which is what was chosen for the purpose of this Thesis.

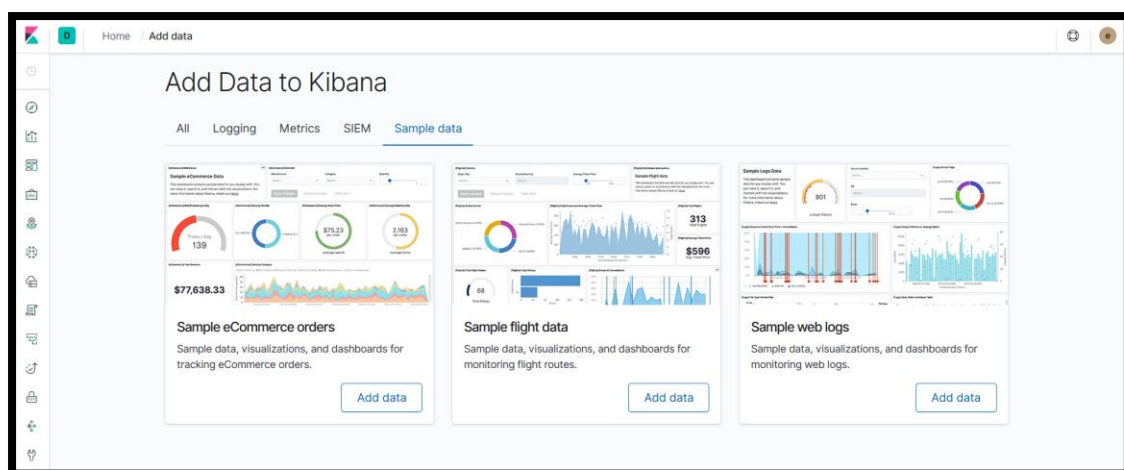


Figure 40: Kibana dashboard in Elastic Cloud

2.2.3. Logstash



2.2.3.1. General idea

Traditionally, Logstash has been used to process logs from applications and send them to Elasticsearch, hence the name. That's still a popular use case, but Logstash has evolved into a more general-purpose tool, meaning that it is a data processing pipeline. The data that Logstash receives, will be handled as events, which can be anything from log file entries to ecommerce orders and chat messages. Afterwards, these events are processed by Logstash and shipped off to one or more destinations. A couple of examples could be Elasticsearch, a Kafka queue, an e-mail message, or to an HTTP endpoint.

A Logstash pipeline consists of three parts/ steps; inputs, filters, and outputs. Each stage can make use of a so-called plugin, which can be a file, for instance, meaning that Logstash will read events from a given file. Another case would be that someone sends events to Logstash over HTTP, or to look rows from a relational database.

While input plugins are how Logstash receives events, on the other side, filter plugins are all about how Logstash should process them. Many formats can be parsed with these plugins such as CSV, XML or even JSON. To be more precise, data enrichment can be performed such as looking up an IP address and resolving its geographical location or even look up data in a relational database.

Finally, an output plugin is where the processed events are sent to. Formally, those places are called stashes. So, in a nutshell, Logstash receives events from one or more inputs, processes them, and sends them to one or more stashes. A user can have multiple pipelines running within the same Logstash instance if he wants to. This can be helped since Logstash is horizontally scalable. It is worth mentioning, that a Logstash pipeline is defined in a proprietary markup format that is like JSON. However, it's not only a markup language, as it is possible to add conditional statements and make a pipeline dynamic.

While Elasticsearch is the database component, the Logstash is the streaming component which pushes the logs into Elasticsearch and to other database technologies, as presented below.

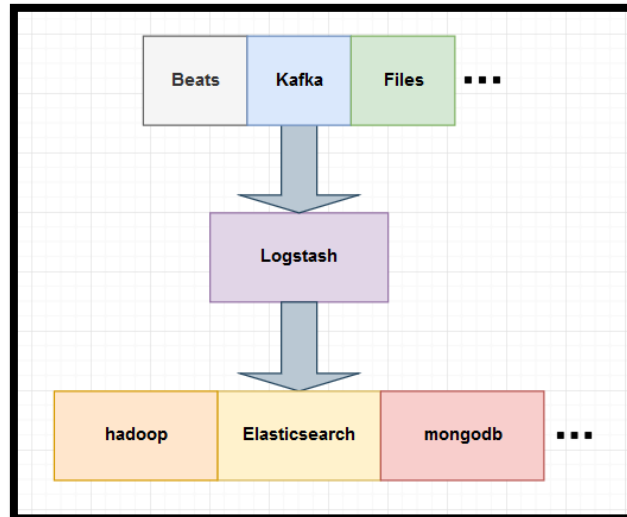


Figure 41: Logstash place in ELK

2.2.3.2. Main stages

Logstash consists of three stages that are shown also to the figure below. The first stage is getting data into the pipeline from the data source - whatever the data source is – and it goes into a thing named “inputs”. So, basically is the first stage where the system is getting the input of data. Then there is a filtering process where that data is filtered out to data that are desired and to data that must be thrashed. Lastly, inside the Logstash pipeline, there is the output. It is the thing that makes its way into the data destination – whatever that may be. In the Elastic Stack the data destination is of course Elasticsearch.

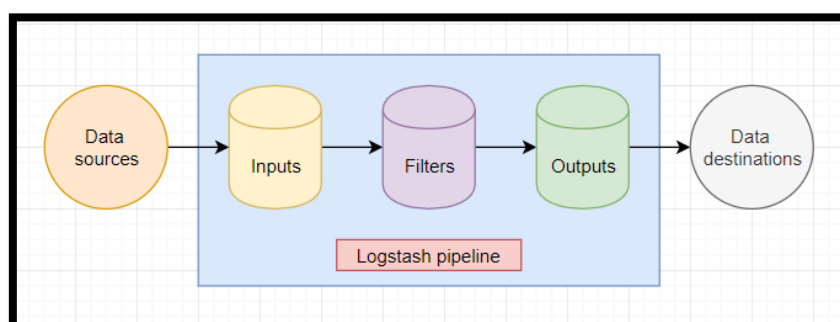


Figure 42: Stages of Logstash

2.2.4. X-Pack



X-Pack is a pack of features that adds additional functionality to Elasticsearch and Kibana. It adds functionality in various feature areas and below the most important ones will be presented.

Firstly, there is security. X-Pack adds both authentication and authorization to both Kibana and Elasticsearch. Regarding authentication, Kibana can integrate with LDAP, Active Directory and other technologies to provide it. Users and roles can be also added and then configured exactly to what a given user or role can access. This is very useful, since different people might need different privileges. A very characteristic example is one of a marketing department or management team. They cannot do any changes to data as expected but they should have read-only access to certain data of interest to them. A great tool that X-Pack uses is Shield, that features authentication and authorization options.

Next up, X-Pack enables the monitoring of performance of the Elastic Stack. Specifically, the user can see the CPU and memory usage, disk space and many other useful metrics, which enables the user to stay on top of the performance and easily detect any problems.

What's more, is that users can even set up alerting and be notified if something out of the water happens. It is important to determine the relation between alerting and monitoring, since alerting is not specific to the monitoring part of the Elastic Stack. This happens because alerting can be set to alert anything the user wants. For example, the user might want to be alerted if CPU or memory usage of the company's web servers go through the roof, or if there is a spike in application errors. Many times, it is used to stay on top of suspicious user behaviour, such as if a given user has signed in from three different countries within the past hour. The alert part comes afterwards with the notification by e-mail, Slack or other means.

Last but not least reporting is another crucial part of X-Pack. With reporting, the user can export the Kibana visualisations and dashboards to PDF files. Reports can be generated on demand or be scheduled so that the user can receive them directly to his e-mail inbox. A user might want daily or weekly reports of a company's key performance indicators or any useful information to engineers or system administrators. Furthermore, reports can also be triggered by specifying conditions, kind of as with alerting, so the user defines rules for when the reports should be



generated and delivered. It is important to mention that for a more professional look reports can be generated with the usage of a personal logo or perhaps for sharing reports with customers. Apart from exporting visualizations and data as PDF files, the user can potentially export data as CSV files, which could be useful for importing data into a spreadsheet, for instance.

X-Pack is also what enables Kibana to do the machine learning. So basically, the functionality is provided by X-Pack, and the interface is provided by Kibana. First with machine learning anomaly detection is possible, such as detecting unusual changes in data based on what the neural network believes is normal. This feature can be easily tied together with alerting. This is not a common secret since, a specific machine of user's choice will watch the number of daily visits to a website of the user's jurisdiction and if there is a significant drop -or increase for that matter- this will be identified. The other thing that this feature is capable of is to forecast future values. This is especially useful for capacity planning, such as figuring out how many visitors will visit the aforementioned website in the foreseeable future. Features like this could be helpful for spinning up additional servers if not using auto scaling, or to have more support agents available. An example could be that a online store with diving gear gets a lot more traffic in summer months.

Moreover, there is a feature in X-Pack named SQL. It is basically a statement about the query language used in relational databases. In Elasticsearch, documents are queried with a proprietary language called Query DSL. This is at heart a JSON object defining the query. This language is malleable since the user can execute numerous things, but it might be a bit rambling at times. For developers who come from a background of working with relational databases, it would be more straightforward to just work with SQL. This is possible since SQL queries can be sent to Elasticsearch over HTTP, or alternatively use the provided JDBC driver. Afterwards what Elasticsearch does, is to translate the SQL query into the Query DSL format behind the scenes, so internally the query is handled the same way after that translation.

Lastly and most importantly since it has the most direct appeal to the user X-Pack features graph. Graph is all about relationships between data. An example could be that when someone is viewing a product on an ecommerce website, it is desirable to show related products on that page as well. Or perhaps suggest the next song in a



music streaming application, based on what the listener desires. For example, if the user likes an exact folk song, there is a good chance that he also likes another song of the same genre. To make this work though it is decisive to fathom the difference between popular and relevant. This can be easily with a simple poll out on the street is performed with ten different people whether they are using a specific, popular search engine. Most of them will answer yes but that doesn't mean necessarily that they have anything else in common. That's just because the platform which was the universe of discourse is popular amongst all kinds of people that use the internet. On the other hand, though, if you ask ten people if they use a precise site about cyber security, the ones that say yes indeed have something in common, because this website is related to an IT field. So basically, the uncommonly common is desired because it says something about relevance and not reputation. The point is that purely looking at the relationships in data without looking at relevance can be seriously misleading that's why graph uses the relevance capabilities of Elasticsearch when determining what is akin and what's not. Graph exposes an API that can be used to integrate this into applications. Apart from the previously mentioned examples, another case study could be to send out product recommendations in a news feed based on a person's purchase history. This tool also supports a plugin for Kibana where data can be visualised as an interactive graph. Of course, this can be very useful when the user knows what he is looking for but far more helpful when he doesn't. The UI lets the user to drill down, navigate and explore the relations in data that weren't obvious.

2.2.5. Beats

Beats as already stated is a collection of so-called data shippers. They are lightweight agents with a single purpose that are installed on servers, which then send data to Logstash or Elasticsearch. There are multiple beats that collect different kinds of data and serve different purposes. One of the most commonly used beats is Filebeat, which is used for amassing log files and sending the log entries off to either Logstash or Elasticsearch. Filebeat ships with modules for common log files, such as nginx, the Apache web server or MySQL. This is very helpful for collecting log files such as access logs or error logs.



Another beat that is highly worth mentioning is Metricbeat, which collects system-level with, not mandatorily though, service metrics. It is mainly used for collecting CPU and memory usage for the analogous operating system, and any services running on the system as well. Metricbeat also ships with modules for popular services such as nginx or MySQL, so you it is possible to monitor how they perform. There is a generous number of beats currently available but Filebeat and Metricbeat are the most frequently used.

2.2.6. Common choices of architecture

At this point a couple of architectures about how Elasticsearch and Elastic Stack in general are being used. The first use case would be an ecommerce application running on a web server. The data is stored within a database, such as the product categories and the products themselves. So, when a product page is requested, the web application looks up the product within the database, renders the page, and sends it back to the visitor's browser. For the search functionalities the system has been using only the database, so it is only wise to enhance it since that's not what databases are good at. The best tool for the searching part is Elasticsearch.

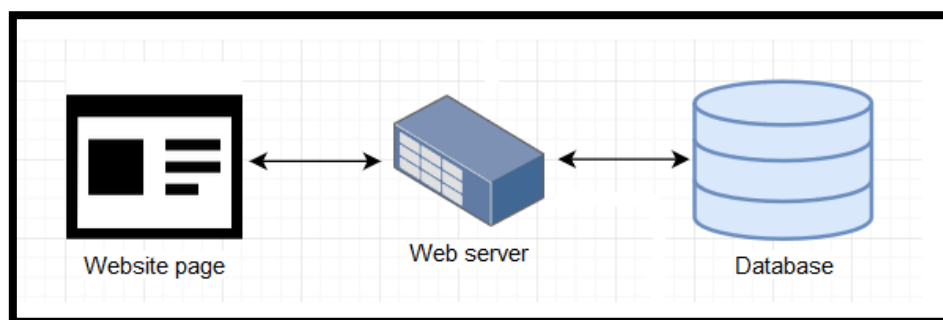


Figure 43: Common architecture part 1

To integrate it into the current architecture, the application should communicate directly with Elasticsearch. Subsequently, when someone enters a search query on the website, a request is sent to the web application, which then sends a search query to Elasticsearch. When the application receives a response, it can process it and send the results back to the browser. The real challenge though is how to get data into Elasticsearch in the first place, and how does it stay updated. To be more specific,



whenever a new product is added or updated, the product should be also added or updated in Elasticsearch too, apart from within the database.

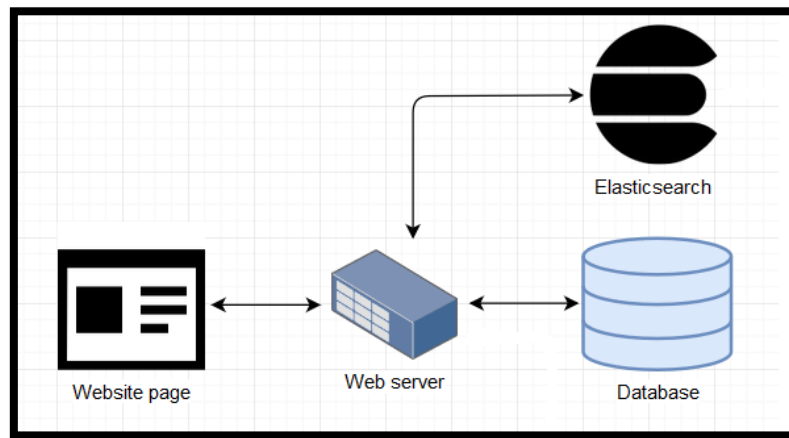


Figure 44: Common architecture part 2

Essentially some of the data will be duplicated. That might sound like a poor approach but The aforementioned, is an excellent way for launching a website with Elasticsearch from the beginning but is not really conceivable when it is added to an existing application that already has data. When data that is to be added is relatively small, a script could be written that does it all in one go. On the other hand, if the data is more than that, the user would need to paginate-or scroll through- them. There are some open source projects that can help with this, but not all of them are actively maintained, and usually it is a fairly simple task to write a script that does this. Then, from the moment the data has been imported, the application will keep it up to date whenever something is altered. This is probably the simplest usage of Elasticsearch that someone will come across.

Afterwards a way for visualization of the number of orders per week and the revenue to a dashboard, is needed. A company can build its own interface, but it is highly advisable to use Kibana since it can save the company a lot of money. As already stated, multiple times Kibana is just a web interface that interacts with Elasticsearch, so there is no need to add any data to it. It runs somewhere on a machine of the users choosing. To keep things simple, it is assumed that there is a dedicated machine- or a virtual machine- to run Kibana on. So, all that is needed to be done, is to run Kibana on a machine and configure it to connect to Elasticsearch.

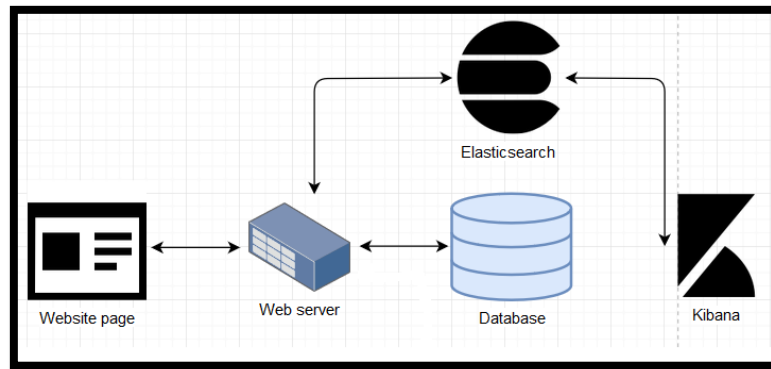


Figure 45: Common architecture part 3

As time passes though, the business continues to grow, and the traffic on the website constantly rises. The server is facing multiple difficulties when trying to keep up so it is of high significance to make sure that it can handle future growth. A part of this is to handle spikes in traffic and knowing when it is time to add another web server. To deal with this, a Beat is needed and the most suitable for this kind of work is Metricbeat. As already explained on the Beat section Metricbeat is used to monitor system level metrics such as CPU and memory usage. Metricbeat can be also instructed to monitor the web server process. Since beats supports sending data to both Logstash and Elasticsearch, that is how data end up in Elasticsearch. What will be done is, have Metricbeat send data to an Elasticsearch ingest node. Now with Metricbeat it is possible to monitor how a server is performing and set up alerts within Kibana if the CPU or memory usage reaches some threshold. When that process is completed an additional web server can be added.

Apart from monitoring system level information on the web server, two more things need to be monitored; the access and error logs from a web server. Although Google Analytics are used on the website, the access log can provide some useful information, such as how long it took the web server to process each request. This enables the ability to monitor the response times and aggregate the requests per endpoint. By doing this, it is fairly easy to identify if some bad code is deployed, causing the response time to spike for a certain endpoint. Not only can this cause a bad user experience, but it can also put additional stress on the system, so it is desirable to detect if that happens. Since the business has grown so has the number of features that need to be implemented. The development team is bigger now, and as a



result more code can be produced. Of course, it is reasonable that this situation also increases the risk of introducing bugs, so it is pleasant to stay on top of that apart from the testing that needs to be done before deploying new code. The web server's error log can help with that, and optionally any application log that might exist at the time. For this job a Beat called Filebeat is used since it is perfect for this job. All there is to do, is to start up Filebeat and the logs will be processed and stored within Elasticsearch and will be ready for analysis within Kibana. Filebeat has built-in modules for this, which take care of the heavy lifting in terms of parsing the logs and handling multi-line events (stack traces, etc.). So, for simple use case scenarios no configuration is needed for this Beat to work.

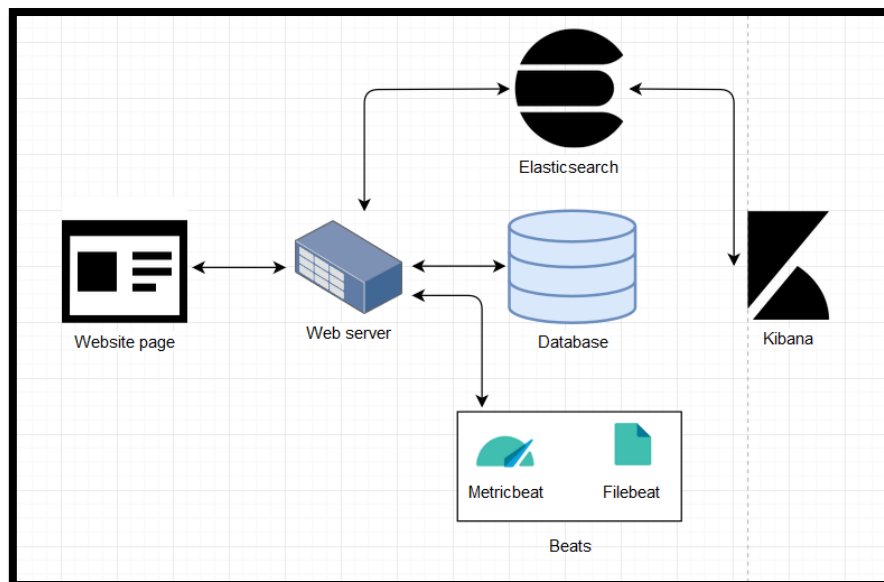


Figure 46: Common architecture part 4

Since the traffic and workload has risen in an agile way, two more web servers have been added to handle them. More kinds of data are stored now within Elasticsearch, including events. A characteristic use case study could be when a product is added to the cart. At that point an analysis within Kibana must be performed. So far, only Elasticsearch's ingest nodes have been used for simple data transformation. Although, now it is time to do more advanced processing, like data enrichment. This can be done within the web application, but it has a few disadvantages.

First of all, the business logic code will be cluttered with event processing, and this might increase response time unless it is done asynchronously. Nevertheless, it leads



to more code, which is not directly related to actually adding a product to the cart, for instance. Secondly, event processing will happen in multiple places. It is not that big of a deal that it happens on more than one web server, because they should all be running the same version of the code. Nonetheless, a back-office server might exist where administrative tasks are performed, such as managing products. Whatever the case might be, it can easily escalate to a point where events are processed in many different places, making things harder to maintain and manage. Ultimately, it would be better to centralize the event processing, and have it all done in one place. This can be accomplished with Logstash, which also offers more flexibility in terms of processing events than Elasticsearch's ingest nodes do.

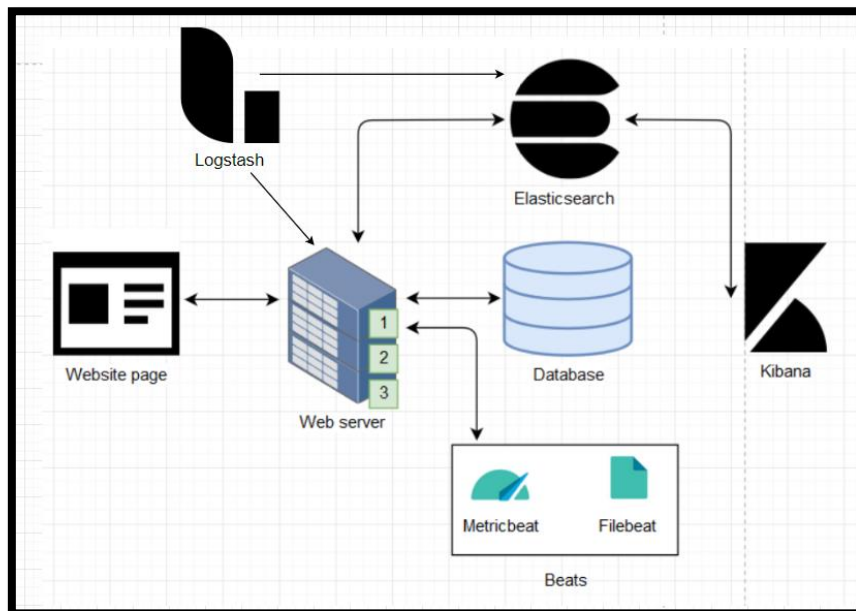


Figure 47: Common architecture part 5

That way events will be sent over HTTP from the webservers to Logstash. Logstash will then process the events however it is instructed to and send them off to Elasticsearch. This has a result, to keep event processing out of the web application. Furthermore, all that there is to be done is to send the initial data to Logstash, and there the rest of the processing will take place.

2.4. Installation and Configuration Guides for Windows, Locally



2.4.1. Elasticsearch

2.4.1.1. Installation

Another approach -which is the most common one- to running Elasticsearch, is to install it on a machine, either virtual or real. Where the Docker approach runs Elasticsearch in an isolated environment, this approach will run Elasticsearch on an operating system like any other application that is or can be installed. In order to mention how the installation procedure is performed, the term ‘‘Installing Elasticsearch’’ must be defined. By saying ‘‘Installing Elasticsearch’’, that is probably not entirely accurate. Elasticsearch will not be installed as an application, because it consists of files with the .jar (Java archives) file format at the end. Jar files are basically filing that aggregate Java class files, metadata and resources. So, they are the zip files for Java projects.

Elasticsearch itself is packaged as a .jar file, along with its dependencies, such as Apache Lucene. With these .jar files, running Elasticsearch is as simple as running a convenient script that is distributed together with the .jar files. To simplify the procedure, files will be downloaded, which are needed to run Elasticsearch.

Firstly Java 8 should be installed. To do so, it is crucial for the user to know which version is installed or if it even installed at all. If Java isn’t installed at all, an error will appear saying that the syntax of the command is incorrect when trying to start Elasticsearch. If Java Development Kit (JDK) is needed, then the user must head to Oracle’s webpage and download the according files. It is noteworthy that JDK is not required while the Java Runtime Environment (JRE) is obligatory. In order to check the version of Java the user will have to use the command prompt as shown in the below figure.

```
C:\Users\Giannis>java -version
java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

Figure 48: Checking java version on Windows



Afterwards the user should head to elastic.co, where he will be able to download Elasticsearch. It is obvious that the .zip file must be extracted somewhere inside the disk. The main file consists of directories that are organised according to their content; lib, config and bin. The lib directory contains the .jar files that were mentioned earlier. To be more precise it includes Elasticsearch itself and Apache Lucene, along with a couple of other dependencies, such as log4j. Moreover, a config directory exists which contains the configuration files for Elasticsearch, log4j and the Java Virtual Machine. Lastly, a directory called bin exists containing binary files for both Linux and Windows. Since the operating system that is used for the purposes of this Thesis is Windows, the file named “elasticsearch.bat” will be used.

To start up an Elasticsearch cluster, the only thing that needs to be done is to run this file by typing the aforementioned file’s name into the command prompt. To do so though, the user should navigate to the extracted archive.

```
Directory of C:\Elastic Stack\elasticsearch-7.3.0
19/08/2019  11:45 πμ    <DIR>      .
19/08/2019  11:45 πμ    <DIR>      ..
19/08/2019  11:45 πμ    <DIR>      bin
19/08/2019  11:45 πμ    <DIR>      config
19/08/2019  11:45 πμ    <DIR>      jdk
19/08/2019  11:45 πμ    <DIR>      lib
24/07/2019  06:27 μμ      13.675 LICENSE.txt
24/07/2019  06:32 μμ    <DIR>      logs
19/08/2019  11:45 πμ    <DIR>      modules
24/07/2019  06:32 μμ      502.598 NOTICE.txt
24/07/2019  06:32 μμ    <DIR>      plugins
24/07/2019  06:26 μμ      8.478 README.textile
          3 File(s)      524.751 bytes
          9 Dir(s)   54.263.746.560 bytes free
```

Figure 49: Elasticsearch directory (Windows)

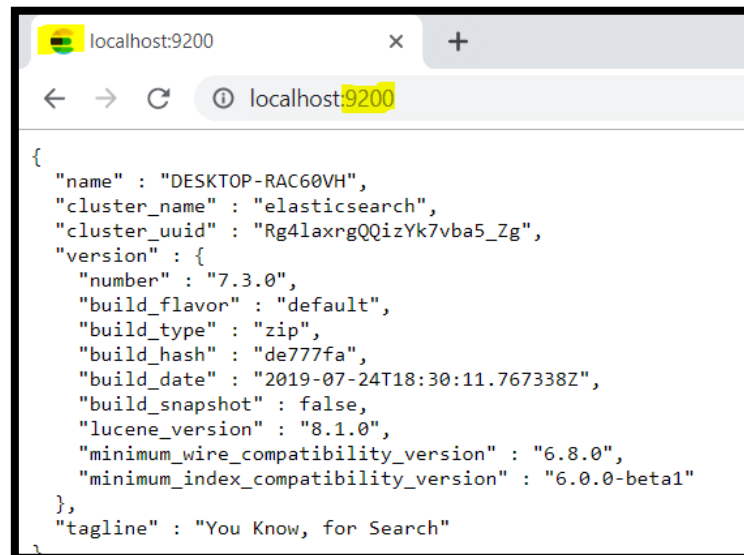
```
C:\Elastic Stack\elasticsearch-7.3.0>bin\elasticsearch.bat
```

Figure 50: Starting Elasticsearch (Windows)

With the above command, Elasticsearch is spinning up a cluster. Once it is ready the way to stop the node is to press CTRL + C simultaneously. To ensure that it works correctly, an HTTP request is issued to it. This can be performed by contacting port



9200 on localhost, which is the default endpoint for the Elasticsearch cluster. For this, any HTTP client will do, and the endpoint remains the same. For keeping things simple this is performed via the browser where the <http://localhost:9200> address is used.



```
{
  "name" : "DESKTOP-RAC60VH",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "Rg4laxrgQQizYk7vba5_Zg",
  "version" : {
    "number" : "7.3.0",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "de777fa",
    "build_date" : "2019-07-24T18:30:11.767338Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 51: Accessing Elasticsearch via port

As shown in the figure above, the JSON information is returned meaning that the cluster is indeed up and running. It is important to remember that a cluster named ‘elasticsearch’ is formed by default, which is specified within the JSON presented above.

2.4.1.2. Configuration

In order to configure how Elasticsearch operates, the user should open the ‘elasticsearch.yml’ file located within the ‘config’ directory. Elasticsearch has really good default values, so if it is used just for developing, often no change is needed anywhere. Regardless of the common usage, some basic options will be presented that a user may need or want to change depending on his use case.

First, there is an option for changing the name of the cluster that is formed. By default, the cluster will be named ‘elasticsearch’ as shown in the HTTP request but it can be set to anything the user wants by removing the comment from the line. A generally good idea would be to use a custom name with an ‘_dev’ postfix or



something equivalent. For production clusters though, the “_production” postfix is usually added. Likewise, the name of the node that will be started up can be specified. The installation that was setup for this Thesis is only for one node. If more than one node were to start up, the same procedure described in the previous installation paragraph must be done to another machine/s.

```
#
# ----- Cluster -----
#
# Use a descriptive name for your cluster:
#
#cluster.name: my-application Cluster name parameter
#
# ----- Node -----
#
# Use a descriptive name for the node:
#
#node.name: node-1 Node name parameter
#
# Add custom attributes to the node:
#
#node.attr.rack: r1
#
```

Figure 52: Configuring Elasticsearch on Windows part 1

The reason for this approach, is that full control of the configuration is granted at a per-node level, meaning that each node may be configured differently. This also means that the name of a particular node in the cluster can be configured. It is generally a good idea to change the default names because it can be pretty confusing to look at in the middle of the night if something goes out of hand. Looking at some random identifier, it’s not immediately apparent which physical machine that particular node is running on. Also, these identifiers change when restarting nodes, so this could quickly get frustrating if there are log files that mention node names that no longer exist.

In the network section of the configuration file, it is worth mentioning that the user can specify the network host and the HTTP port to which nodes listen to requests. This is left commented out when only developing.

Afterwards, there is a discovery section with an option specifying the unicast hosts. To be more precise, this option allows the user to specify a list of nodes that the node to be configured should try to contact. When starting up a node, it either needs to contact a node in an existing cluster or form a cluster on its own. The network



addresses that are specified within this option are the ones the node will try to contact when joining a cluster. This doesn't mean that all of a cluster's nodes must be specified here, because that would quickly become hard to maintain when adding more nodes. In fact, it just needs to contact a single node to retrieve the current cluster state. Having retrieved the cluster state from a node, it then contacts the master node and joins the cluster. Therefore, at least one of the nodes that are specified is crucial to be available when starting up the node. Typically, a couple of nodes would be fine. Accordingly, if only a single node is needed/ preferred this should be left commented out and everything will work as planned. For the changes to take effect the node must be restarted.

```
# ----- Network -----  
#  
# Set the bind address to a specific IP (IPv4 or IPv6):  
#  
#network.host: 192.168.0.1      Port and host change  
#  
# Set a custom port for HTTP:  
#  
#http.port: 9200  
#  
# For more information, consult the network module documentation.  
#  
# ----- Discovery -----  
#  
# Pass an initial list of hosts to perform discovery when this node is started:  
# The default list of hosts is ["127.0.0.1", "[::1]"]  
#  
#discovery.seed_hosts: ["host1", "host2"]      Contact-nodes  
#  
# Bootstrap the cluster using an initial set of master-eligible nodes:  
#  
#cluster.initial_master_nodes: ["node-1", "node-2"]  
#  
# For more information, consult the discovery and cluster formation module documentation.  
#
```

Figure 53: Configuring Elasticsearch on Windows part 2

2.4.2. Kibana

2.4.2.1. Installation

Immediately after the Elasticsearch setup, Kibana must also be installed and configured to enable the graphical interface. If the user is going with the Docker approach there is no need to install Kibana, as it is already available on localhost at port 5601.

The steps for installing Kibana, are very similar to what happened earlier with the Elasticsearch installation. First, the user must navigate to elastic.co and download the



product. Afterwards unzipping and moving the files in the same directory with Elasticsearch is to be done.

To open the service, the user should navigate through the correct directory via the command prompt and use a specific command called “bin\kibana” since it is much like Elasticsearch, inside the bin directory. This is going to start up a web server on localhost at port 5601 and try to contact an Elasticsearch cluster on localhost at port 9200.

```
Directory of C:\Elastic Stack\kibana-7.3.0-windows-x86_64
19/08/2019  01:47  µµ    <DIR>      .
19/08/2019  01:47  µµ    <DIR>      ..
24/07/2019  06:59  µµ    <DIR>      .nodegit_binaries
24/07/2019  06:59  µµ    <DIR>      bin
24/07/2019  06:58  µµ    <DIR>      built_assets
24/07/2019  06:58  µµ    <DIR>      config
19/08/2019  02:14  µµ    <DIR>      data
24/07/2019  06:58  µµ          13.675 LICENSE.txt
24/07/2019  06:58  µµ    <DIR>      node
24/07/2019  06:58  µµ    <DIR>      node_modules
24/07/2019  06:58  µµ          1.385.050 NOTICE.txt
24/07/2019  06:58  µµ    <DIR>      optimize
24/07/2019  06:58  µµ          738 package.json
24/07/2019  06:58  µµ    <DIR>      plugins
24/07/2019  06:58  µµ          4.048 README.txt
24/07/2019  06:58  µµ    <DIR>      src
24/07/2019  06:58  µµ    <DIR>      webpackShims
24/07/2019  06:58  µµ    <DIR>      x-pack
         4 File(s)          1.403.511 bytes
        14 Dir(s)      52.386.570.240 bytes free
```

Figure 54: Kibana directory on Windows

```
C:\Elastic Stack\kibana-7.3.0-windows-x86_64>bin\kibana
```

Figure 55: Starting Kibana (Windows)

Once Kibana has started up, the user can simply press CTRL + C simultaneously to stop it. Finally, after the status changes from yellow to green, the user should be able to load Kibana in the browser on localhost at port 5601, by using the according URL; <http://localhost:5601>.

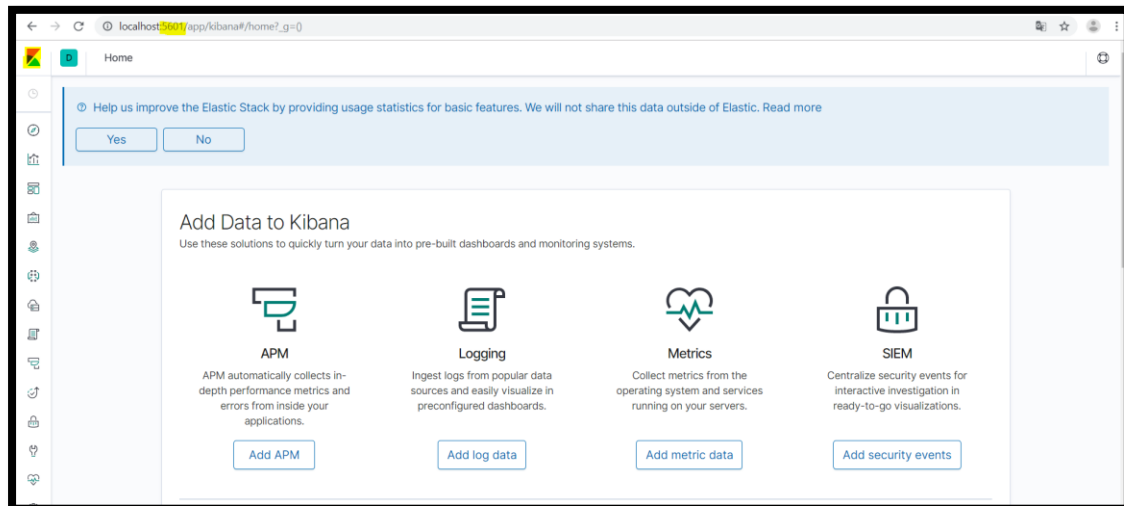


Figure 56: Accessing Kibana via port

2.4.2.2. Configuration

2.4.2.2.1. Basic Configuration

Immediately after the installation, some common configuration options that the user might need to change will be analysed. Kibana's configuration file is stored within a file named "kibana.yml" stored inside the "config" directory. Taking a look at this file, the user will find the most common options along with very helpful comments.

The first two options are the port and host of the Kibana server, which defaults to 5601 and localhost respectively. If Kibana is set up on a server, then this must be changed depending on the setup that is used. Also the server name can be changed, mainly for display purposes while a more important option is the Elasticsearch URL. This URL helps Kibana to communicate with an Elasticsearch cluster, since it is just a web interface. By default, it looks for a cluster at port 9200 on localhost, but this may not be where the cluster is located.

```
# Kibana is served by a back end server. This setting specifies the port to use.
#server.port: 5601 Default port

# Specifies the address to which the Kibana server will bind. IP addresses and host names are both valid values.
# The default is 'localhost', which usually means remote machines will not be able to connect.
# To allow connections from remote users, set this parameter to a non-loopback address.
#server.host: "localhost" Default server
```

Figure 57: Configuring Kibana on Windows part 1



```
# The Kibana server's name. This is used for display purposes.
#server.name: "your-hostname" Server name to be changed

# The URLs of the Elasticsearch instances to use for all your queries.
#elasticsearch.hosts: ["http://localhost:9200"] Elasticsearch URL
```

Figure 58: Configuring Kibana on Windows part 2

Next up, there is an option named Kibana index, which allows the user to change Kibana's index name. It is worth mentioning that Kibana actually creates an index in the Elasticsearch cluster for storing various data. Typically, the user would leave the index name at the default value, but it is important to understand that Kibana creates an index.

```
# Kibana uses an index in Elasticsearch to store saved searches, visualizations and
# dashboards. Kibana creates a new index if the index doesn't already exist.
#kibana.index: ".kibana"
```

Figure 59: Configuring Kibana on Windows part 3

There is a plethora of options within this configuration file, while some of them are very specific. Just to mention a couple of examples, options for enabling SSL on the Kibana server, timeouts, logging etc. can be found within this file. If the user is just developing on his local machine, chances are that he doesn't need to change any of the aforementioned. If he does though it is crucial to restart Kibana for the changes to take effect.

2.4.2.2.2. *Index patterning prerequisites*

Since a recent update, Kibana now requires data to be present within a cluster before being able to configure an index pattern. Therefore, some data must be imported into the cluster before someone can work with Kibana.

Using a JSON generator, some fake data were created. More analytically, the data consists of around three hundred fifty documents representing food and drink products. These have been randomly generated but should be a good database for



searching queries. Each product has a name, description, number of items in stock, number of items sold, an ‘is active’ Boolean, the created date, and an array of tags. In theory a user could copy the contents of the file into Kibana’s console, but this obviously doesn’t scale well. Instead of that the documents will be added through the command line. This will be performed with the help of curl again, but any HTTP tool would be also viable. Firstly, the user must navigate to where the JSON file is stored. Then he can easily send a POST request to the Bulk API, which will be explained later on. Also, it is crucial to use the ‘-H’ parameter in the command to set the content-type header and then -XPOST being the request method, or the HTTP verb. Finally, where the user wants to send the request to is added -on this example to localhost at port 9200. The ‘pretty’ query parameter is used to make everything nicely presentable.

```
C:\Elastic Stack>curl -H "Content-Type: application/json" -XPOST "http://localhost:9200/product/default/_bulk?pretty" --data-binary "@unipi.json"
```

Figure 60: Posting data with curl on Elasticsearch

2.4.2.3. Introduction to hands-on Kibana

2.4.2.3.1. *Creating an index pattern*

The first thing the user will see when headed to Kibana, is a page where he is prompted to set a so-called index pattern. There he can insert a pattern for which indices he wants to include within the Kibana interface. Index patterns allow the user to bucket disparate data sources together so their shared fields may be queried in Kibana.

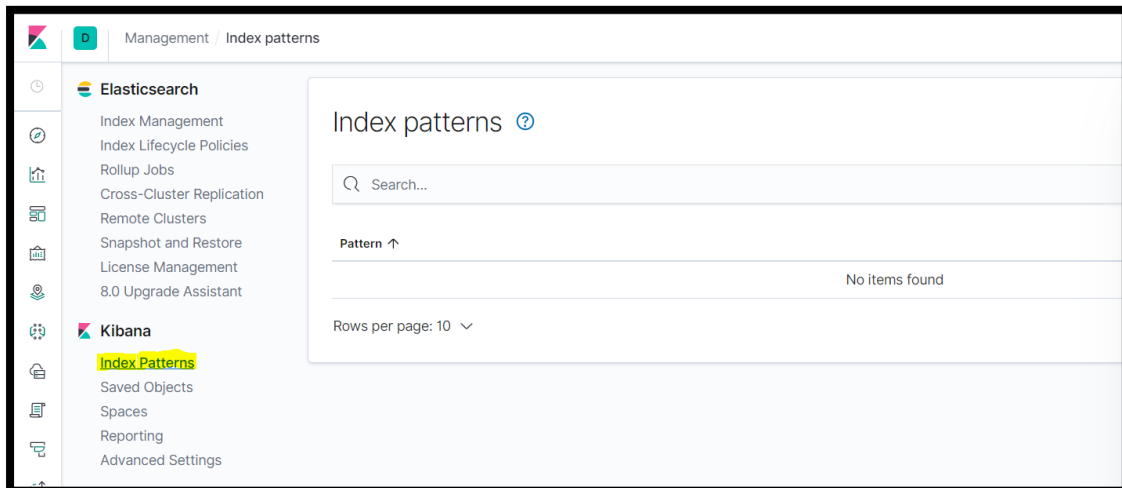


Figure 61: Index patterns in Kibana part 1

For the purposes of this Thesis the pattern that will be used will be the asterisk to include everything. Also, no time filter will be used since it is not needed.

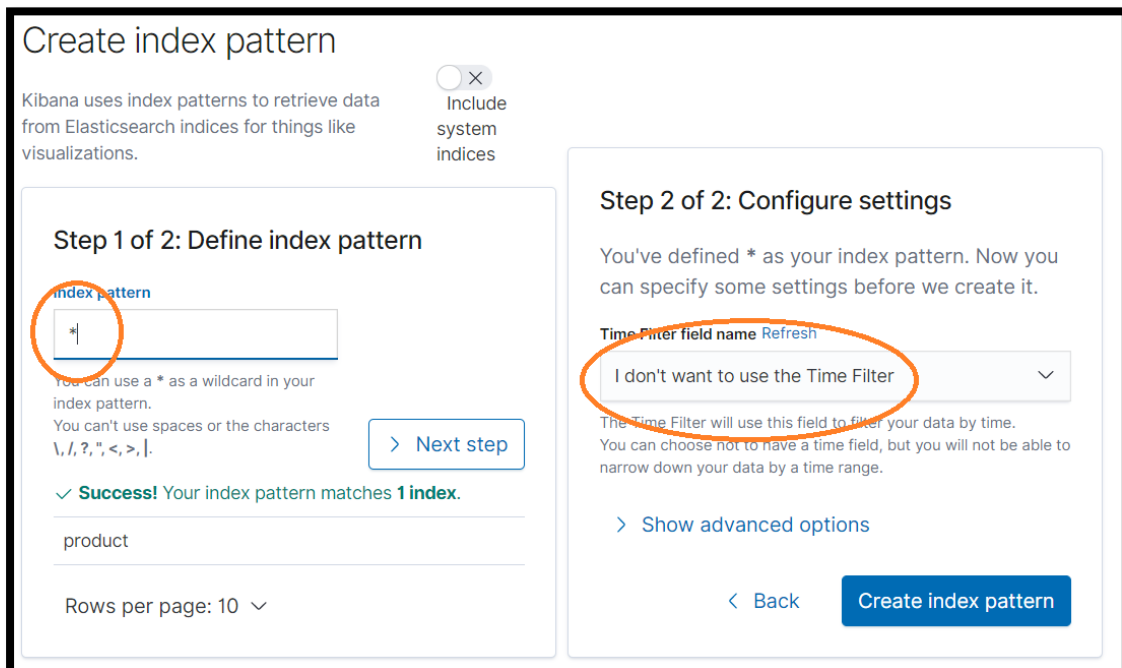


Figure 62: Index patterns in Kibana part 2

The real reason this configuration takes place is for finding a convenient way of sending queries to the Elasticsearch cluster. The tool is named “Console” and can be found within the “Dev Tools” tab in the left-hand menu.



2.4.2.3.2. Console tool and formatting

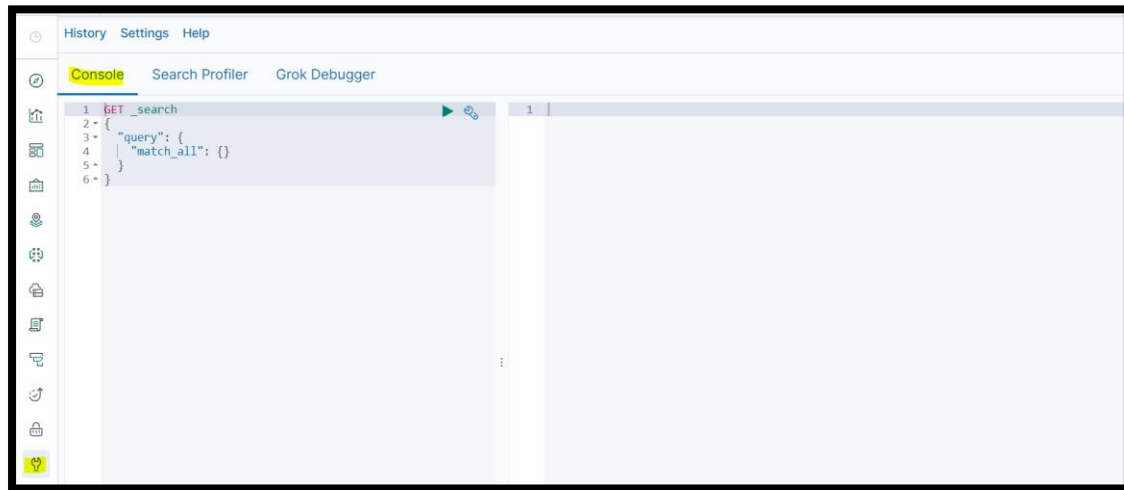


Figure 63: Console tool

Console is a tool that lets the user to enter his queries without having to deal with HTTP headers, formatting responses, etc. On top of that, it also provides syntax highlighting and code completion. Therefore, it is a much more convenient place for the user to write his queries than in a terminal, for instance.

Since Elasticsearch clusters expose an HTTP REST API, there is nothing special happening under the hood. All the communication with the cluster is still done through the same API that someone can also access from the command line or any other HTTP client for that matter. The way the Console works is that when the user enters queries on the left-hand side, the results appear to the right. An example query is already offered which should be deleted since it is useless for this use case.

The way the user communicates with Elasticsearch clusters, is by sending HTTP requests as already mentioned. The request is defined with a combination of the HTTP verb, request URI and request body. Since a REST API is used, the HTTP verb is important, i.e. GET, POST, PUT or DELETE. When using Kibana's dev tools, the user simply enters the HTTP verb. This is of course different than using curl or similar tools. Following the HTTP verb and a space is a request URI. The first part is the index name, type and the API that is desired. Subsequently that is the basic format in which the user can specify endpoints in Kibana's console.

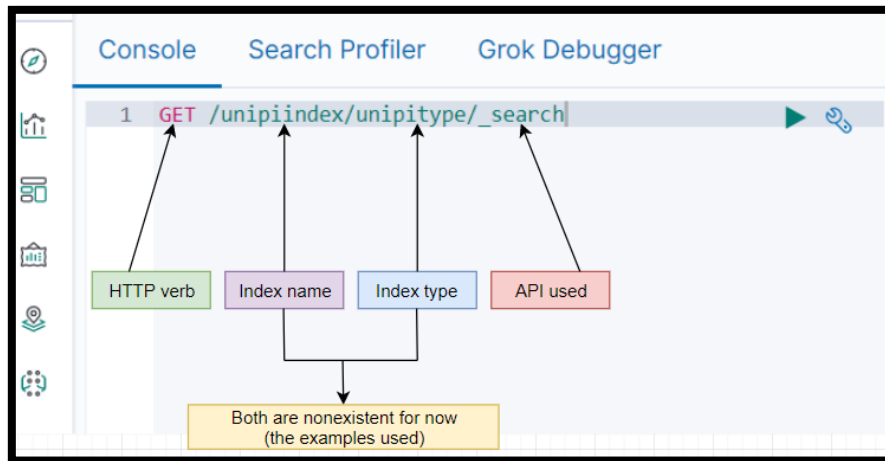


Figure 64: Console syntax part 1

First, there is the REST verb, followed by a space. Then an optional forward slash, the index name, a slash, the index type, another slash and the desired API. Within Kibana, the hostname of the Elasticsearch cluster should not be specified as that is prepended to request URI by Kibana automatically. If the user uses curl or another similar tool, then he would need to enter <http://localhost:9200> or whatever the hostname and port of the cluster is, following the appropriate index, type and API.



Figure 65: Console syntax part 2

2.4.3. Logstash

2.4.3.1. Installation

It must be recalled that Logstash is a very powerful open source data processing pipeline engine that is used to ingest data from a multitude of sources simultaneously. The sources of information that Logstash is fed with doesn't matter since it can come



from anywhere. This data can be ingested not only into Elasticsearch but also to other no sequel engines. So basically, all shapes, sizes and sources of data can be ingested.

To install this particular tool, much like Kibana and Elasticsearch, the tool is downloaded but is extracted directly to the “C:” folder since it is bugged and cannot be on the same folder with them. Afterwards with the help of the terminal the user can examine if Logstash can run with the following command.

```
Directory of C:\Elastic Stack\logstash-7.3.1
23/08/2019  10:16 πμ <DIR>      .
23/08/2019  10:16 πμ <DIR>      ..
23/08/2019  10:16 πμ <DIR>      bin
23/08/2019  10:16 πμ <DIR>      config
19/08/2019  09:53 μμ          2.276 CONTRIBUTORS
19/08/2019  09:53 μμ <DIR>      data
19/08/2019  09:53 μμ          4.144 Gemfile
19/08/2019  09:53 μμ          23.109 Gemfile.lock
23/08/2019  10:16 πμ <DIR>      lib
19/08/2019  09:53 μμ          13.675 LICENSE.txt
23/08/2019  10:16 πμ <DIR>      logstash-core
23/08/2019  10:16 πμ <DIR>      logstash-core-plugin-api
23/08/2019  10:16 πμ <DIR>      modules
19/08/2019  09:53 μμ          808.305 NOTICE.TXT
23/08/2019  10:16 πμ <DIR>      tools
23/08/2019  10:16 πμ <DIR>      vendor
23/08/2019  10:16 πμ <DIR>      x-pack
          5 File(s)          851.509 bytes
          12 Dir(s)      53.383.299.072 bytes free
```

Figure 66: Logstash directory on Windows

```
C:\logstash-7.3.1>bin\logstash -e "input { stdin { } } output { stdout { } }"
```

Figure 67: Testing if Logstash is up

```
[2019-08-23T10:27:32,693][INFO ][logstash.agent ] Successfully started Logstash API endpoint {:port=>9600}
```

Figure 68: Proof Logstash is up and running

According to the confirmation message Logstash could successfully boot up at the 9600 port. So basically, Logstash is ready to accept inputs and give outputs. A simple test would be to enter some messages at the console and see that indeed Logstash filters them and presents them afterwards.



```

I am John and I am doing my MSc Thesis in UNIPI!
C:/logstash-7.3.1/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesom
: warning: constant ::Fixnum is deprecated
{
  "@version" => "1",
  "@timestamp" => 2019-08-23T07:32:25.067Z,
  "message" => "I am John and I am doing my MSc Thesis in UNIPI!\r",
  "host" => "DESKTOP-RAC60VH"
}

```

Figure 69: Importing custom messages inside Logstash

2.4.3.2. Indexing Apache application logs

Before even beginning to test indexing, a file was created in the same folder as Logstash directory, called “data” that includes a folder called “logs”. This is where all the log files that are needed will be stored. Afterwards an example was downloaded from GitHub that includes typical log files from an Apache dump.

```

17 83.149.9.216 - - [28/May/2014:16:13:47 -0500] "GET /presentations/logstash-monitorama-2013/images/elasticsearch.png HTTP/1.1" 200 54662 "http://semicomple
18 83.149.9.216 - - [28/May/2014:16:13:47 -0500] "GET /presentations/logstash-monitorama-2013/images/github-contributions.png HTTP/1.1" 200 34245 "http://se
19 83.149.9.216 - - [28/May/2014:16:13:47 -0500] "GET /presentations/logstash-monitorama-2013/css/print/paper.css HTTP/1.1" 200 4254 "http://semicomplete.c
20 83.149.9.216 - - [28/May/2014:16:13:47 -0500] "GET /presentations/logstash-monitorama-2013/images/1983-delorean-dmc-12-pic-38289.jpeg HTTP/1.1" 200 22054
21 83.149.9.216 - - [28/May/2014:16:13:46 -0500] "GET /presentations/logstash-monitorama-2013/images/simple-inputs-filters-outputs.jpg HTTP/1.1" 200 1168622
22 83.149.9.216 - - [28/May/2014:16:13:46 -0500] "GET /presentations/logstash-monitorama-2013/images/tiered-outputs-to-inputs.jpg HTTP/1.1" 200 1079983 "ht
23 83.149.9.216 - - [28/May/2014:16:13:53 -0500] "GET /favicon.ico HTTP/1.1" 200 3638 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.3
24 24.236.252.67 - - [28/May/2014:16:14:10 -0500] "GET /favicon.ico HTTP/1.1" 200 3638 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:26.0) Gecko/20100101
25 93.114.45.13 - - [28/May/2014:16:14:32 -0500] "GET /articles/dynamic-dns-with-dhcp/ HTTP/1.1" 200 18948 "http://www.google.co.uk/?asact=next&asares=355
26 93.114.45.13 - - [28/May/2014:16:14:32 -0500] "GET /reset.css HTTP/1.1" 200 1015 "http://www.semicomplete.com/articles/dynamic-dns-with-dhcp/" "Mozilla/4
27 93.114.45.13 - - [28/May/2014:16:14:33 -0500] "GET /style2.css HTTP/1.1" 200 4877 "http://www.semicomplete.com/articles/dynamic-dns-with-dhcp/" "Mozilla
28 93.114.45.13 - - [28/May/2014:16:14:33 -0500] "GET /favicon.ico HTTP/1.1" 200 3638 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:25.0) Gecko/20100101 Firefox/4
29 93.114.45.13 - - [28/May/2014:16:14:33 -0500] "GET /images/jordan-80.png HTTP/1.1" 200 6146 "http://www.semicomplete.com/articles/dynamic-dns-with-dhcp/"
30 93.114.45.13 - - [28/May/2014:16:14:33 -0500] "GET /images/web/2009/banner.png HTTP/1.1" 200 52315 "http://www.semicomplete.com/style2.css" "Mozilla/5.0

```

Figure 70: Apache log example

Inside the logs folder a configuration for apache file is needed so a new file called “apache.conf” will be created. It is going to be basically the Logstash configuration file that is used to parse the logs file. This file consists of configuration lines for each stage of Logstash pipeline with the according configuration for each one’s corresponding plugins.

In input the file plugin exists that is used for the path of the log file to be imported. Next, in filters multiple plugins are used. Firstly, a plugin named “grok” which is a regular expression. Afterwards the “mutate” plugin converts data types and the “date” plugin as the name states formats dates. Likewise, the “geoIP” plugin is able to locate and separate the IP from a log file and the “useragent” defines if the user used a phone, table, PC etc. While on output, a standard output is used which is for printing dots to the screen for every line that gets piped into Logstash.



```
input
{
  file {
    path => "C:\data\logs\logs"
    type => "logs"
    start_position => "beginning"
  }
}

output
{
  stdout {
    codec => dots
  }
  elasticsearch {
  }
}

filter
{
  grok{
    match => {
      "message" => "%{COMBINEDAPACHELOG}"
    }
  }
  mutate{
    convert => { "bytes" => "integer" }
  }
  date {
    match => [ "timestamp", "dd/MMM/YYYY:HH:mm:ss Z" ]
    locale => en
    remove_field => "timestamp"
  }
  geoip {
    source => "clientip"
  }
  useragent {
    source => "agent"
    target => "useragent"
  }
}
```

Figure 71: Apache logs configuration file

It is worth mentioning that Logstash can deduce for example from an IP the location of the user, meaning that it is not just a tool for parsing things, but it can seriously enhance the information that is given from logs and indexes.

To load the log file in Logstash all that there is to be done, is to specify the configuration file and where is that located.

```
C:\logstash-7.3.1>bin\logstash -f C:\data\apache.conf
```

Figure 72: Using apache.conf by Logstash

```
[2019-08-23T13:46:43,749][INFO ][logstash.agent ] Successfully started Logstash API endpoint {:port=>9600}
.....
Dot printing since the stdout was dots at config file
.....
```

Figure 73: Proof that apache.conf has loaded

Each dot represents a particular event or ingestion that Logstash went through to index that log entry.

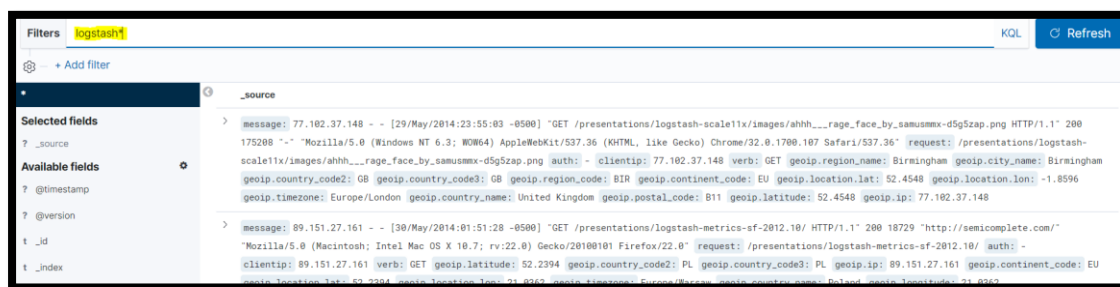


Figure 74: Results with Logstash index

It is worth mentioning that for the user to see how many documents are indexed through Logstash, he simply has to visit http://localhost:9200/logstash-*/_count.

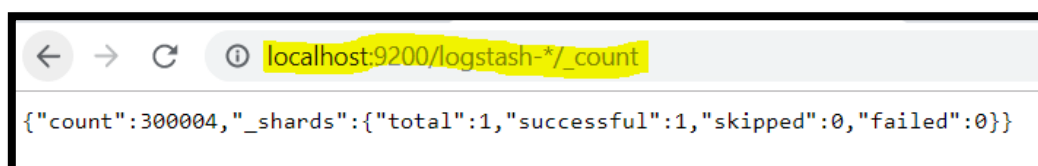


Figure 75: Number of documents indexed through Logstash

2.4.4. Beats

2.4.4.1. Filebeat

2.4.4.1.1. Installation

At this point Filebeat will be installed to ingest logs into Elasticsearch. The exact method that was used to do so, even though is described inside the Kibana platform at the “Add log data” section, it doesn’t work due to many complications. So, after checking that the configuration file is ok, it is manually used and opened through PowerShell, with Administrator privileges.

```
C:\Program Files\Filebeat>filebeat.exe modules enable apache
Enabled apache
```

Figure 76: Filebeat enabling Apache



```
PS C:\Program Files\Filebeat> .\filebeat.exe -c filebeat.yml -e -d "2019-08-26T12:14:38.513+0300" INFO instance/beat.go:606 Home path: C:\Program Files\Filebeat Data path: [C:\Program Files\Filebeat\data] Logs p
```

Figure 77: installing Filebeat through PowerShell

2.4.4.1.2. Sending files to Logstash

The most common use of Filebeat is to implement it to work with Logstash. To do so a guide was used at the Elastic website in the doc section (Filebeat Reference). It is important to mention that Filebeat uses the "metadata" field to send metadata to Logstash. More specifically it defines what the indexed name is going to be.

The changes are performed inside the aforementioned configuration file for apache. Firstly, the input is no longer a file in the system but a Beat that uses the port 5044, which is the default port for Filebeat. This is actually going to be listening for Filebeat to be sending data to this port on this server. Afterwards, some information about elasticsearch is filled at the output part.

```
input
{
  beats {
    port => 5044
  }
}

output
{
  stdout {
    codec => dots
  }
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
  }
}
```

Figure 78: Configuring apache.conf for Filebeat



Thereupon, some changes were performed in the configuration file of Filebeat called “filebeat.yml”. Firstly, input should be enabled, and the path must be specified. Subsequently output should be changed from Elasticsearch to Logstash.

```
type: log

# Change to true to enable this input configuration.
enabled: true Changed to true from false

# Paths that should be crawled and fetched. Glob based paths.
paths:
  - C:/data/logs/* Path for logs
  #- c:\programdata\elasticsearch\logs\*

#----- Elasticsearch output -----
# output.elasticsearch: Commented out
# Array of hosts to connect to.
# hosts: ["localhost:9200"]

# Optional protocol and basic auth credentials.
#protocol: "https"
#username: "elastic"
#password: "changeme"

#----- Logstash output -----
output.logstash: Uncommented
# The Logstash hosts
hosts: ["localhost:5044"]
```

Figure 79: Logstash configuration

```
C:\logstash-7.3.1>bin\logstash -f C:\data\apache.conf
```

Figure 80: Loading in Logstash the new apache.conf

```
[2019-08-26T13:50:31,699][INFO ][logstash.inputs.beats] Beats inputs
5044"} New input is from beats
[2019-08-26T13:50:31,715][INFO ][logstash.javapipeline] Pipeline sta
[2019-08-26T13:50:31,870][INFO ][logstash.agent] Pipelines ru
:non_running_pipelines=>[]}
[2019-08-26T13:50:31,894][INFO ][org.logstash.beats.Server] Starting ser
[2019-08-26T13:50:32,858][INFO ][logstash.agent] Successfully
confirmation that it runs
```

Figure 81: Proof that Filebeat ingests data in Logstash

Lastly, after restarting all services of the Elastic Stack a confirmation that the log sent was successful, can be seen by searching “Filebeat*” inside Kibana. It must be



recalled that in order for the logs to load, a change was performed inside the apache log file.

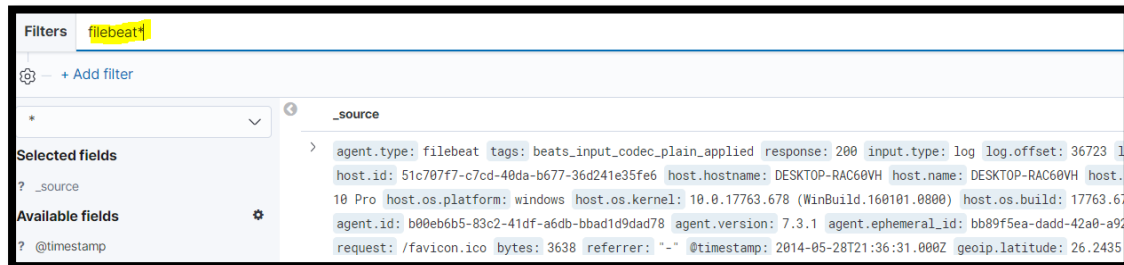


Figure 82: Filebeat index results

2.4.4.2. Metricbeat

2.4.4.2.1. Installation

Much like Filebeat the user must head to the elastic website and download the installation file. After the extraction inside the same file as Filebeat, the main directory of the tool is renamed to “Metricbeat”. Later on, many complications show up that don’t allow the installation to be performed correctly, so as a result it is booted and used manually.

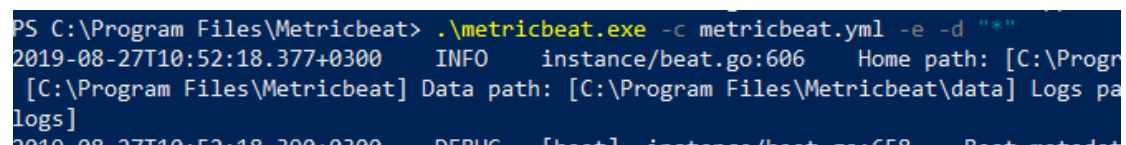


Figure 83: Metricbeat installation through PowerShell

For the confirmation that the device sends through Metricbeat log files is done inside the Kibana platform.

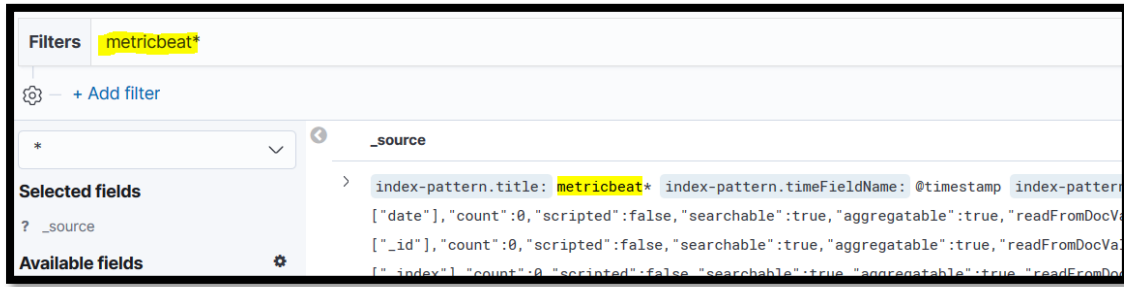


Figure 84: Metricbeat index results

2.4.4.2.2. Kibana visualization

Firstly, a new index must be created with the name “metricbeat*”. Afterwards a new visualization is created through Kibana that represents data in a pie chart. The data that is presented, originates from the field “system.network.out.packets.”.

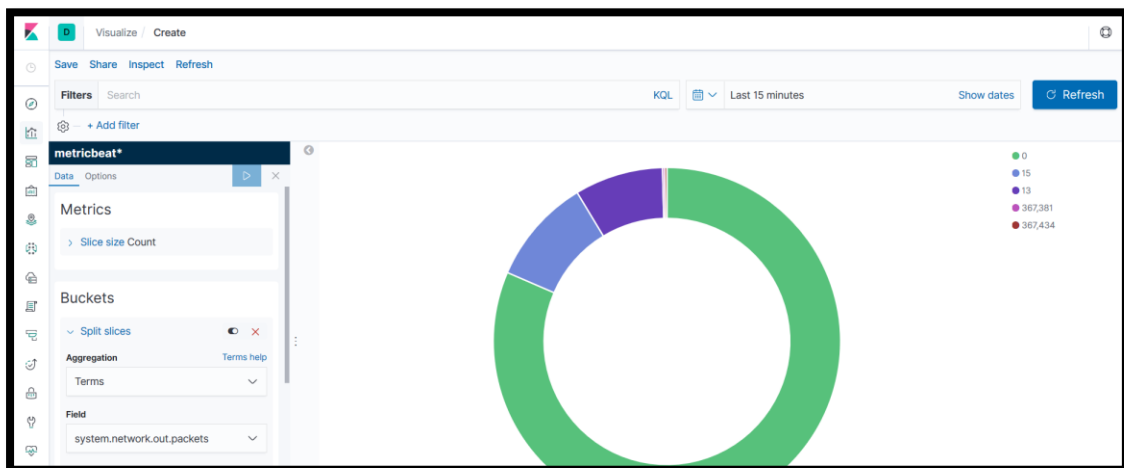


Figure 85: Visualize things with Kibana example

2.4.5. X-Pack

X-Pack as opposed to the previously mentioned tools and platforms is installed both in Elasticsearch and Kibana. This happens because it adds features that can work standalone for either of them. Firstly, it must be installed into Elasticsearch with the following command. This is already done since newer distributions have it included already.



```
C:\Elastic Stack\elasticsearch-7.3.0\bin>elasticsearch-plugin install x-pack  
ERROR: this distribution of Elasticsearch contains X-Pack by default
```

Figure 86: Installing X-Pack (Windows) part 1

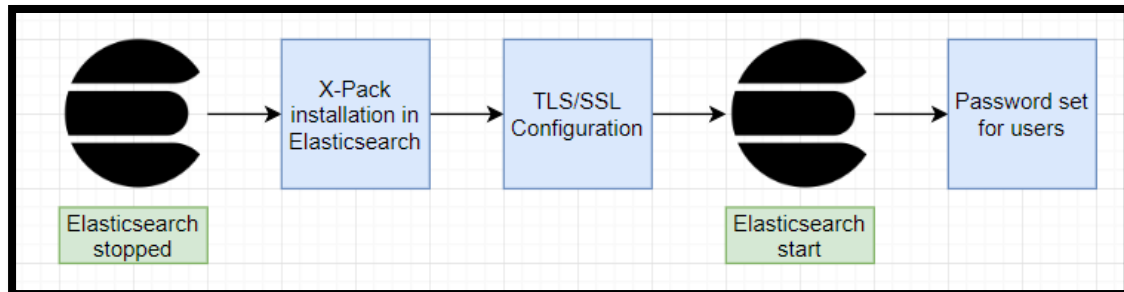


Figure 87: Steps to install X-Pack

Afterwards, the TLS/ SSL protocol must be configured to run in the cluster settings. It is crucial that the user configures this for every single node of the cluster. This requirement applies mainly to clusters with more than one node and to clusters with a single node that listens on an external interface. It is noteworthy that single-node clusters that use a loopback interface do not have this requirement.

To begin with a certificate must be created for secure communication for the node that exists in the cluster. When it asks the user for a file name, the user will have to put it in the configuration directory, since that is where Elasticsearch will look for it.

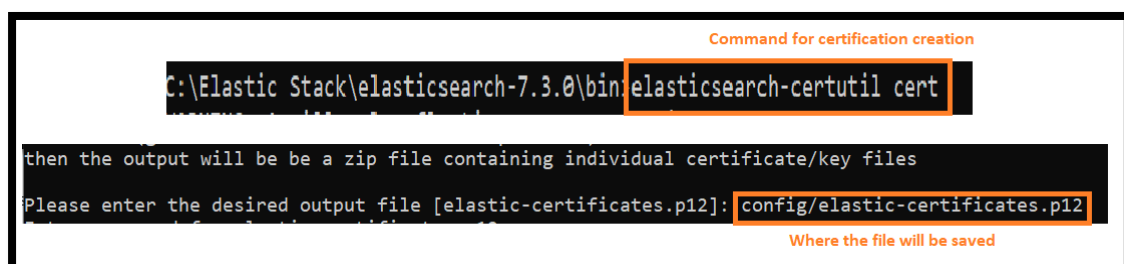


Figure 88: Installing X-Pack (Windows) part 2



Once that is done, some commands that are needed to enable security and configure TLS must be pasted inside the Elasticsearch's configuration file, called "elasticsearch.yml".

```
xpack.security.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.keystore.path: elastic-certificates.p12
xpack.security.transport.ssl.truststore.path: elastic-certificates.p12
```

Figure 89: Enabling X-Pack in Elasticsearch conf file

Now it is crucial that Elasticsearch starts again for the user to setup the password for the cluster. To do so the Elasticsearch command for auto setting passwords is used. This will generate random passwords for the various internal stack users.

```
C:\Elastic Stack\elasticsearch-7.3.0\bin>elasticsearch-setup-passwords auto
Initiating the setup of passwords for reserved users elastic,apm_system,kiba
The passwords will be randomly generated and printed to the console.
Please confirm that you would like to continue [y/N]y

Changed password for user apm_system
PASSWORD apm_system = [REDACTED]

Changed password for user kibana
PASSWORD kibana = [REDACTED]

Changed password for user logstash_system
PASSWORD logstash_system = [REDACTED]

Changed password for user beats_system
PASSWORD beats_system = [REDACTED]

Changed password for user remote_monitoring_user
PASSWORD remote_monitoring_user = [REDACTED]

Changed password for user elastic
PASSWORD elastic = [REDACTED]
```

Figure 90: All passwords for the suite

Now that Elasticsearch is configured, user should also set up Kibana. To do so, from the passwords terminal the user must grab the kibana password and paste it inside the Kibana configuration file called "kibana.yml". Afterwards Kibana should also boot up.

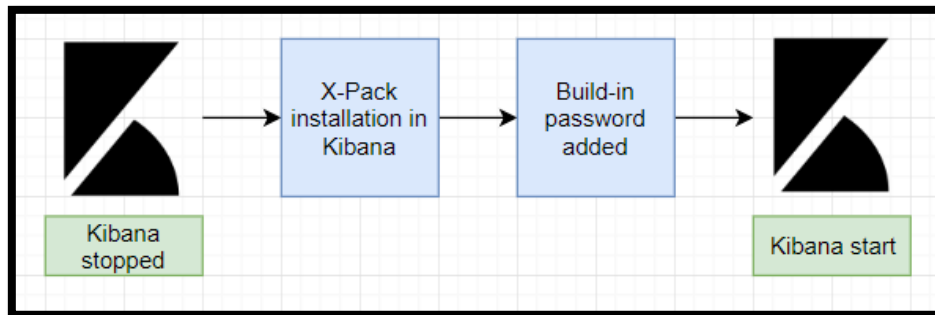


Figure 91: Installing Kibana steps

```
# If your Elasticsearch is protected with basic authentication,  
# the username and password that the Kibana server uses to perform  
# index at startup. Your Kibana users still need to authenticate  
# if proxied through the Kibana server.  
elasticsearch.username: "kibana"  
elasticsearch.password: "5yoNFhP1lgUDuaCCfLuN"
```

Figure 92: Kibana configuration file

After all the successful configuration, the user logs in the Kibana platform with the credentials of super user “elastic”. There new roles and users can be configured.

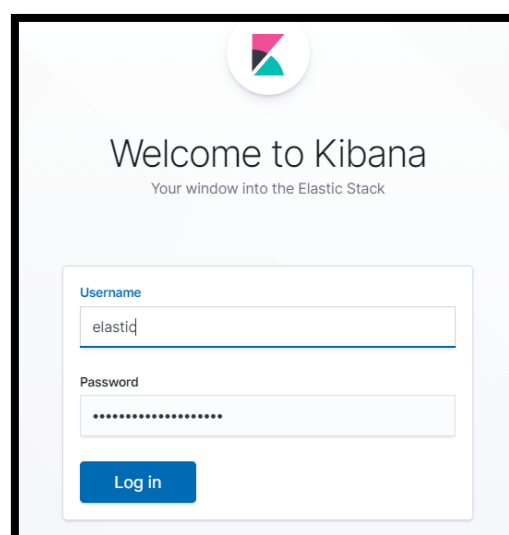


Figure 93: Kibana login screen

It is important to note that the same exact procedure follows for Logstash.

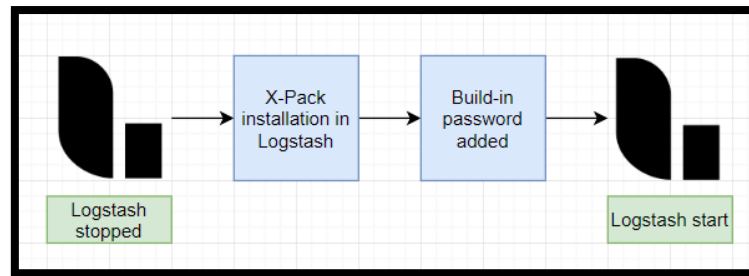


Figure 94: Logstash installation steps

2.5. Elastic Stack and GDPR

2.5.1. General

The General Data Protection Regulation or GDPR is a regulation in European Union's law on data protection and privacy for all individual citizens of the EU and the European Economic Area (EEA). It also addresses the transfer of personal data outside these areas. It aims basically to give control to individuals over their personal data and to simplify the regulatory environment for international business by unifying the regulation within the EU. A simplified diagram on GDPR handling of personal data is presented below (by personal data it is meant any information which are related to an identified or identifiable natural person).

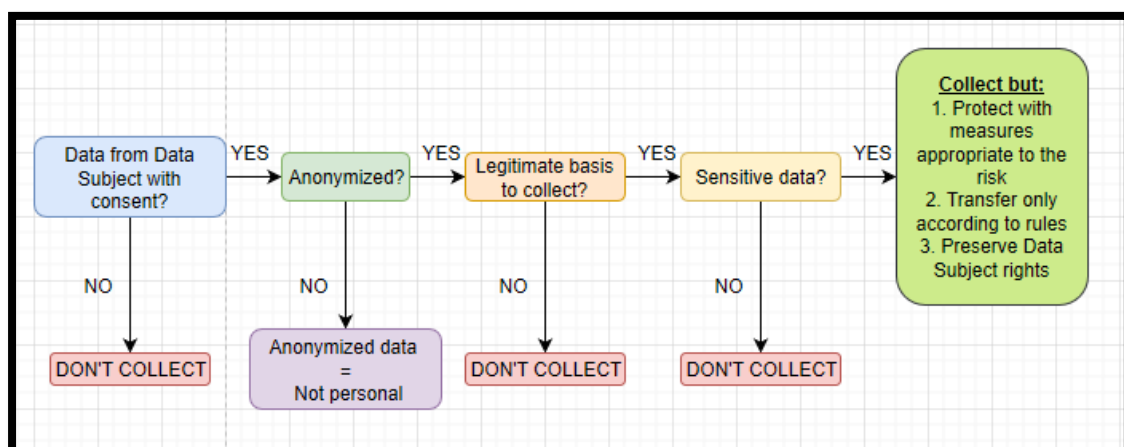


Figure 95: How GDPR works with personal data according to Elastic



As can be seen in the next figure, the GDPR handling of personal data, is performed as a three-stage process. Firstly, there is the preparation of an organization for controlling or processing personal data. Afterwards, a guide on how to protect any such personal data. Lastly, how are privacy processes done in order to preserve the rights of the Data subject over their data. The design below, has the implementation steps that can be performed with the Elastic Stack.

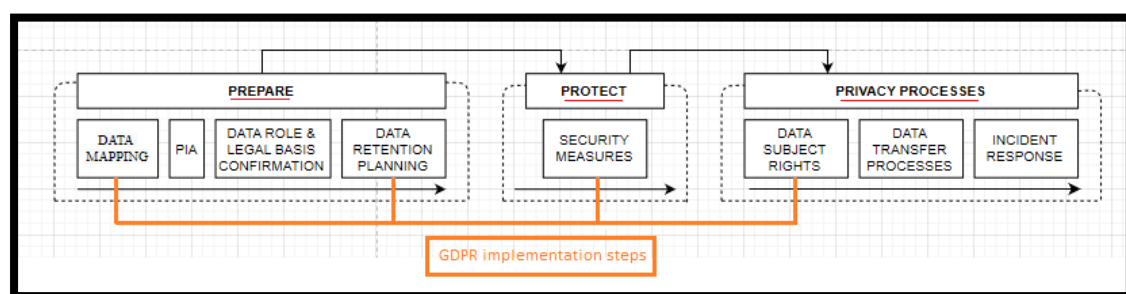


Figure 96: Which steps does Elastic cover

2.5.2. Preparation for Personal Data handling (1st Set of processes)

When initiating a GDPR compliance process, a certain preparation must be performed for it to be successful. Firstly, as the diagram implies is the Data Flow Mapping process. Through this process, all Data Flow processes are identified and documented within the organization that controls or processes personal data. If this documentation is incomplete or irrelevant due to the organization's inability to identify them, the GDPR initiative may be ineffective. It is crucial, to take into consideration the place where personal data are stored since it is highly correlated with the whole process of indexing information. This process is a technique that is advised to be used whenever personal data are stored, since Elasticsearch, by indexing, can execute a full-text search in a matter of seconds.

Furthermore, in the preparation stage, it is decided by the organization how long the personal data must be stored. Since GDPR specifies limited retention the organization must delete any personal data not needed or when the data subject withdraws consent.



This is easily manageable with indices. This is performed by using time-based indices that will be deleted after a certain expiration date.

2.5.3. Protection of Personal Data (2nd Set of processes)

After the preparation phase, it is only consequent for the protection phase to take place. By protection phase, it is meant every possible security measure adaptation. The first feature of Elastic Stack that helps with compliancy is correlated with privacy by design and privacy by default terms. The stack can put the organization on the track from the start by limiting access, maintaining accuracy and ensuring that data is secure. Only data that is relevant to the project will exist inside a cluster, which makes the minimization of personal data possible.

Additionally, in order to prevent unauthorized access to the cluster there must be a way to authenticate users. Elastic Stack features a way to validate a user for who he claims to be. This can also work along with other technologies as an Active Directory or even a PKI. It must be stated that only by authenticating users, GDPR compliance is not enough, that's why techniques like whitelisting and blacklisting IPs can also be used. Furthermore, assigning access privileges to specific roles and assigning these roles to users is also considered a necessity.

Moreover, when the personal data is stored inside the Elastic Stack, its security features can create and maintain audit trails by auditing security events. These events can help the organization to observe who accesses, or tries to without any success, the cluster. Insights can be collected by looking at attempted attacks or breaches. On the other hand, if Elastic Stack is not the primary place where data are stored, it is used as a centralized logging platform which is basically a security analytics solution.

Either if the stack works as a place that stores data or not, it can be an essential component for monitoring and threat detection. It is not a common secret that deployments follow basic principles of monitoring data store health and monitoring log continuity within the environment. Through features of the stack, administrators can keep track on the health of the cluster while on the same time be alerted for interruptions or failures.

2.5.4. Privacy processes and maintenance of rights (3rd Set of processes)



When a data subject uses their right to erasure or even withdraws their consent, one of the most difficult tasks for the organization is to find that data. Elasticsearch is a tool that can identify sharply multiple personal data for the organization to delete them. This can be performed in queries, reports or even applications. This can be performed by features such as Delete API or Update API.

2.5. Companies that use it

This suite is used by big time companies like the following:

- **Wikipedia**

Elasticsearch is used for full-text search

- **Stack Overflow**

It relies on Elasticsearch as a means to support full-text search capabilities, thus providing source related question or/and answers

- **Netflix**

It heavily relies on Elastic Stack for monitoring and analysing customer service-related operations and security related logs

- **LinkedIn**

It uses the Elastic Stack along with kafka to monitor performance and security in real time

- **GitHub**

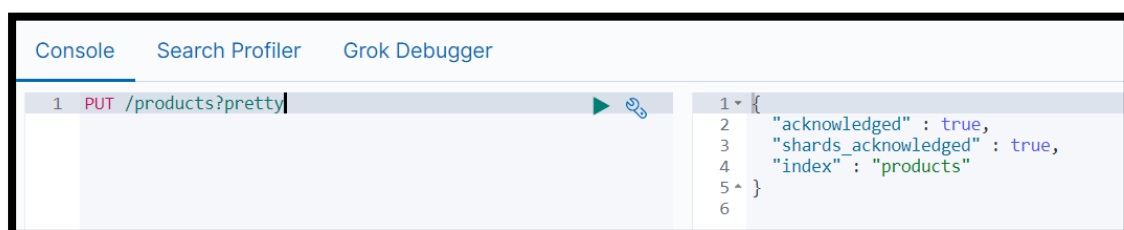
Elasticsearch is used to query billion lines of code everyday

3. Document Management

3.1. Creating an index

Firstly, as an example an index will be created. This will activate the procedure of adding documents to it because without them or indices an Elasticsearch cluster is not much. Since the JSON file that was used earlier contains products, the index that will be added will be named “products”.

The PUT HTTP verb will be used and the path will simply be a slash followed by the name of the index. To make the result of the query easy to read, a query parameter named “pretty” can be added which will format the JSON for humans. Since the query parameter is a flag, there is no need to specify a value, so it can simply be done as shown in the figure below. Furthermore, since the Console tool is used, there is not an actual need to do this, because it will format everything nicely automatically, but if another tool is used for sending requests it is a necessity. After pressing the play button, it can be seen at the results that the index was created, which will be used to store several products in the form of documents. If the default number of shards and replicas remains at the default values, there is nothing more to add about creating indices.



```
Console Search Profiler Grok Debugger
1 PUT /products?pretty
1 {
2   "acknowledged" : true,
3   "shards_acknowledged" : true,
4   "index" : "products"
5 }
6
```

Figure 97: PUT request to show a specific index

3.2. Adding documents in an index

To add documents inside an index, a POST request is sent to a URI consisting of the index name, followed by a type. It is noteworthy, that since types got removed from



Elasticsearch, a type name will be specified as “default”. Since the index name is “products”, that makes the request URI “/product/default”. Although this is the HTTP verb and the endpoint, the document that is to be added is missing. To specify that document, a JSON object is added on a new line, which can be of any structure that the user desires. This arbitrary JSON object will be the document, so any JSON is valid here. In this case, a car will be added as a product. So, a “model” field is added, which will be the name of the car model, followed by an object called “extras”. This object will include the engine’s cc, the wheel horsepower and the number of cylinders; “cc”, “WHP”, “cylinders”. To execute the query, much like creating an index the green triangle is pressed and the results are as follows.

```
1 POST /products/default
2 {
3   "model": "B180",
4   "extras": {
5     "cc": "two thousand",
6     "wHP": "two hundred",
7     "cylinders": "four"
8   }
9 }
```

Figure 98: POST request to add a new object

```
1 #! Deprecation: [types removal] Specifying types in document index requests is deprecated, use the typeless endpoints
  instead (/{index}/_doc/{id}, /{index}/_doc, or /{index}/_create/{id}).
2 {
3   "_index": "products",
4   "_type": "default",
5   "_id": "j-qprmwB5bBHU7vdzlk0",
6   "_version": 1,
7   "result": "created",
8   "_shards": {
9     "total": 2,
10    "successful": 1,
11    "failed": 0
12  },
13   "_seq_no": 1,
14   "_primary_term": 1
15 }
16 }
```

Figure 99: Proof that version changed

The “_id” field is an identifier that Elasticsearch automatically generated for the document, because an ID for the document was not specified in the time of creation.



To do so the HTTP verb should change to PUT and after the type specification the ID should be assigned.

```
1 POST /products/default
2 {
3   "model": "B180",
4   "extras": {
5     "cc": "two thousand",
6     "WHP": "two hundred",
7     "cylinders": "four"
8   }
9 }
10
11 PUT /products/default/1
12 {
13   "model": "B200",
14   "extras": {
15     "cc": "three thousand",
16     "WHP": "three hundred",
17     "cylinders": "six"
18   }
19 }
```

```
1  #! Deprecation: [types re
2  instead ({index}/_doc/
3  {
4    "_index" : "products",
5    "_type" : "default",
6    "_id" : "1",
7    "_version" : 1,
8    "result" : "created",
9    "_shards" : {
10     "total" : 2,
11     "successful" : 1,
12     "failed" : 0
13   },
14   "_seq_no" : 2,
15   "_primary_term" : 1
16 }
```

Figure 100: Adding a new object with ID example

It is crucial to understand that sending a request to index a document for an index and type that don't exist, isn't an obstacle since Elasticsearch will create them for and on behalf of the user behind the scenes. The purpose of this, is to make Elasticsearch as easy to use and user friendly as possible. Of course, this can be disabled in case the user wants to stay in control of which indices are created.

3.3. Document retrieval

To perform a document retrieval, it is crucial to know and actually use a unique ID. More specifically this can be performed by using the GET HTTP verb. The request path will be the index name, the type and the ID of the document.



```
1 GET /products/default/1  
1 #! Deprecation: [types removal] Sp  
  endpoint instead.  
2 {  
3   "_index" : "products",  
4   "_type" : "default",  
5   "_id" : "1",  
6   "_version" : 1,  
7   "_seq_no" : 2,  
8   "_primary_term" : 1,  
9   "found" : true,  
10  "source" : {  
11    "model" : "B200",  
12    "extras" : {  
13      "cc" : "three thousand",  
14      "whp" : "three hundred",  
15      "cylinders" : "six"  
16    }  
17  }  
18 }  
19 }
```

Figure 101: GET request to retrieve an object

It is clear from the results that Elasticsearch, when storing documents, adds some meta fields, which are all prefixed by an underscore. These include the index, type, ID and version of the document. The document itself can be found within the “_source” meta field which is usually the only thing needed when searching for it.

3.4. Document supplanting

3.4.1. Original replacing and updating

To perform a document replacement a case example will be presented. A product was added earlier that the user forgot to put a price tag on it. Since the user knows the ID of that document, it can easily get replaced. It is fairly easy because the query is exactly the same as for adding a new document with a given ID. Therefore, only a new field must be added.



```
1 PUT /products/default/1
2 {
3   "model": "B200"
4   "price": 30000,
5   "extras": {
6     "cc": "three thousand",
7     "WHP": "three hundred",
8     "cylinders": "six"
9   }
10 }
11
12 GET /products/default/1

1 #! Deprecation: [types removal] s
2 endpoint instead.
3 {
4   "_index": "products",
5   "_type": "default",
6   "id": "1",
7   "_version": 2,
8   "_seq_no": 3,
9   "_primary_term": 1,
10  "found": true,
11  "source": {
12    "model": "B200"
13    "price": 30000,
14    "extras": {
15      "cc": "three thousand",
16      "WHP": "three hundred",
17      "cylinders": "six"
18    }
19  }
20 }
```

Figure 102: Updating an object. Single argument (Version proof)

Within the results, at the “_source” field, it is visible that the “price” field has been added. Also, it should be noted how the “_version” meta field now has a value of two instead of one, indicating how many times the document has been changed.

If the user on the other hand, wants to update a value without replacing the entire document another procedure takes place. To do that, a POST request will be sent to the Update API, which is available by using the “_update” endpoint on a specific document. The Update API expects a JSON object and within this object a “doc” property is added, which itself should be an object. In the “doc” object, the user can specify the fields that he wants to change. In this case scenario, a key named “WHP” will be added in order to update the value of wheel horsepower of the car. It should be emphasized that with this approach new fields can be added as well. A “lbs” key to the object will be added for describing the weight of the car.



```
1 POST /products/default/1/_update
2 {
3   "doc": {"extras": {"whp": "four hundred"}, "lbs": 9000}
4 }
5
6 GET /products/default/1

1 #! Deprecation: [types removal]
2 endpoint instead.
3 {
4   "_index" : "products",
5   "_type" : "default",
6   "_id" : "1",
7   "_version" : 4,
8   "_seq_no" : 5,
9   "_primary_term" : 2,
10  "found" : true,
11  "source" : {
12    "model" : "B200",
13    "price" : 30000,
14    "extras" : {
15      "cc" : "three thousand",
16      "whp" : "four hundred",
17      "cylinders" : "six"
18    }
19  }
20  "lbs" : 9000,
```

Figure 103: Updating an object. Multiple argument (Version proof)

It is notable that even though the whole document wasn't replaced, that the “_version” value still changed to another value.

As already mentioned, documents in Elasticsearch are immutable, meaning that they cannot be changed once they have been indexed. Even though it looks as if the document has been partially updated, this is actually not the case internally. What actually happens, is that the Update API retrieves, changes and re-indexes the document. This means that it will actually retrieve the document, change it according to the user's specification and then replace the existing document. It just does this in a convenient way so that the user doesn't have to deal with multiple requests himself. All of this also happens within a shard so that there is not network overhead of multiple requests as there would be if this was done manually.

3.4.2. Updates with scripts

Apart from specifying the new values for fields or new fields directly within an update query, it is also possible to use scripts. A good example would be to reduce the weight of the car by a hundred lbs. More thoroughly, instead of first retrieving the document to find the current weight of a car and then updating the document with the previous weight minus a hundred, Elasticsearch can do this in a single query. This is accomplished by using something called scripting. It allows the user to do dynamic things within queries.



Much like before, the same HTTP verb and request URI will be used but the “doc” property will be replaced with a “script” one. The document object can be accessed on a variable named “ctx”. This variable contains the fields that were presented in query results, such as the “_id” meta field and the “_source” field.

```
1 POST /products/default/1/_update
2 {
3   "script": "ctx._source.lbs -= 100"
4 }
5
6 GET /products/default/1
```

Subtracted amount

```
1 #! Deprecation: [types remc
  /_update/{id} instead.
2 {
3   "_index" : "products",
4   "_type" : "default",
5   "_id" : "1",
6   "_version" : "6",
7   "result" : "updated",
8   "_shards" : {
9     "total" : 2,
10    "successful" : 1,
11    "failed" : 0
12  },
13   "_seq_no" : 7,
14   "_primary_term" : 2
15 }
16
```

Version changed again

Figure 104: Update an object’s value that already exists

```
10
11
12
13
14 GET /products/default/1
```

Confirmation that it is down by 100 lbs

```
9 "found" : true,
10 "source" : {
11   "model" : "B200",
12   "price" : 30000,
13   "extras" : {
14     "cc" : "three thousand",
15     "WHP" : "four hundred",
16     "cylinders" : "six"
17   },
18   "lbs" : 8900,
19   "WHP" : "four hundred"
20 }
21 }
22
```

Figure 105: Proof that the value changed

3.4.3. Document upsert

Firstly, the document that was created in earlier stages will be deleted for the purposes of showing how upsert works. The deletion and confirmation are presented below.



```
1 DELETE /products/default/1 Delete request
2
3 GET /products/default/1
4
5
6
7
8
```

```
1 #! Deprecation: [types remo
  endpoint instead.
2 {
3   "_index" : "products",
4   "_type" : "default",
5   "id" : "1",
6   "found" : false
7 }
8 Wasn't found since it
  doesn't exist
```

Figure 106: DELETE request and proof

Another common thing to do, is to update a document if it exists and add it otherwise. This procedure is referred to as an “upsert”. To perform this the Update API is used with a script as earlier but this time an “upsert” key will be also specified.

```
1 POST /products/default/1/_update
2 {
3   "script": "ctx._source.lbs -= 100",
4   "upsert":
5   {
6     "lbs": 1000
7   }
8 }
9
10
```

```
1 #! Deprecation: [types re
  /_update/{id} instead.
2 {
3   "_index" : "products",
4   "_type" : "default",
5   "id" : "1",
6   "version" : 1,
7   "result" : "created",
8   "shards" : {
9     "total" : 2,
10    "successful" : 1,
11    "failed" : 0
12  },
13   "_seq_no" : 10,
14   "_primary_term" : 2
15 }
16
```

Version reseted to 1

Created successfully

Figure 107: Upsert example part 1 (POST request)

The aforementioned query in the figure states that if the document already exists, the script will run and increase the weight of the car by a hundred. On the other hand, if the document does not exist, then the object for the “upset” key is added to the document. In this case, this means that an object with a “lbs” key and a value of a thousand will be added.



```
1 POST /products/default/1/_update
2 {
3   "script": "ctx._source.lbs -= 100",
4   "upsert":
5   {
6     "lbs": 1000
7   }
8 }
9
10 GET /products/default/1
```

```
1 #! Deprecation: [types re
  endpoint instead.
2 {
3   "_index": "products",
4   "_type": "default",
5   "_id": "1",
6   "_version": 1,
7   "_seq_no": 10,
8   "_primary_term": 2,
9   "found": true,
10  "source": {
11    "lbs": 1000
12  }
13 }
14
```

Search request and confirmation

Figure 108: Upsert example part 2 (proof)

3.5. Deleting

3.5.1. Documents

Deleting documents can be extremely easy. It can be done by using the “DELETE” HTTP verb and specifying the index, type and ID of the document that is to be deleted. Although updating multiple documents at once is not possible, deleting multiple ones in one query is. For the purposes of this example three new documents have been added.

```
1 POST /products/default/2
2 {
3   "name": "prius",
4   "category": "sedan"
5 }
6
7 POST /products/default/3
8 {
9   "name": "passat",
10  "category": "sedan"
11 }
12
13 POST /products/default/4
14 {
15   "name": "GLA180",
16   "category": "crossover"
17 }
18
```

Figure 109: Adding new documents example



Afterwards it is desired that all the products that with a category of “sedan” will be deleted. With the documents currently in the index, this will match two documents: 2nd and 3rd. To do so an API named “Delete by Query” is used, which is available at the index level. So, a POST request is issued with the stated API. Afterwards a JSON object must be added with a “query” property, which will contain the type of query. So, this object will contain the conditions for the documents that are to be deleted. The type of the query will be a “match” query, so a key of that name will be added, and an object as a value. This object should then contain the field name to match as a key and the value that the field should contain in order for a document to match, as the value. In this case “category” and “sedan”, respectively.

```
1 POST /products/_delete_by_query
2 {
3   "query":
4   {
5     "match":
6     {
7       "category": "sedan"
8     }
9   }
10 }
```

```
1 {
2   "took" : 45,
3   "timed_out" : false,
4   "total" : 2,
5   "deleted" : 2,
6   "batches" : 1,
7   "version_conflicts" : 0,
8   "noops" : 0,
9   "retries" : {
10    "bulk" : 0,
11    "search" : 0
12  },
13  "throttled_millis" : 0,
14  "requests_per_second" : -1.0,
15  "throttled_until_millis" : 0,
16  "failures" : [ ]
17 }
18
```

Figure 110: DELETE multiple documents example

```
1 GET /products/default/2
2
3
4
5
6
7
8
9
10
```

```
1 #! Deprecation: [types r
2 endpoint instead.
3 {
4   "_index" : "products",
5   "_type" : "default",
6   "id" : "2",
7   "found" : false
8 }
```

```
1 GET /products/default/3
2
3
4
5
6
7
8
9
10
```

```
1 #! Deprecation: [types r
2 endpoint instead.
3 {
4   "_index" : "products",
5   "_type" : "default",
6   "id" : "3",
7   "found" : false
8 }
```

Figure 111: DELETE multiple documents proof



As shown above it is confirmed that not only two documents were deleted, but that it is not possible to search for them either. It should also be emphasized that if the user wants to delete all the documents from the index, deleting the index itself is far more efficient.

3.5.2. Indices

Deleting an index is as simple as using the “DELETE” HTTP verb and specifying the name of the index with a slash in front of it. So, the user can just write DELETE, an optional forward slash, and the name of the index which is “products”. After running the query, it is confirmed that the index has been indeed deleted.

```
1 DELETE /products
2
3
4
```

```
1 {
2   "acknowledged" : true
3 }
4
```

Figure 112: DELETE request for an index

```
1 DELETE /products
2
3
4
5
6
7
8
9 GET /products
```

```
1 {
2   "error" : {
3     "root_cause" : [
4       {
5         "type" : "index not found exception",
6         "reason" : "no such index [products]",
7         "resource.type" : "index_or_alias",
8         "resource.id" : "products",
9         "index_uuid" : "_na_",
10        "index" : "products"
11      }
12    ],
13    "type" : "index_not_found_exception",
14    "reason" : "no such index [products]",
15    "resource.type" : "index_or_alias",
16    "resource.id" : "products",
17    "index_uuid" : "_na_",
18    "index" : "products"
19  },
20   "status" : 404
21 }
```

Error because the index wasn't found since it was deleted

Figure 113: Proof that index was deleted

3.6. Processing multiple documents

To import test data, as already done into the “product” index, processing of multiple documents will be used, in order to have multiple specimens. This means that many



documents will be added in a single request. If the user wants to import hundreds or even thousands of documents, it would be much better to run a single query instead of a thousand because there is some overhead in sending a request. This type of processing is done by using a special format for the request using what is called the Bulk API. Bulk as its name states about quantities in bulk. This API is not limited to adding documents, because it can also be used to update or delete documents. Either way, what must be done, is to send a POST request.

```
1 POST /products/default/_bulk
2 {"index": {"_id": "10" }}
3 {"model": "yaris"}
4 {"index": {"_id": "11" }}
5 {"model": "galardo"}
6 {"index": {"_id": "12" }}
7 {"model": "twingo"}
8 {"index": {"_id": "13" }}
9 {"model": "carrera"}
```

Figure 114: Adding values on multiple documents at once

It is recalled that even though the index was earlier deleted, with the request above it was automatically created. Since different kinds of operations can be performed within the same request, the type of operation that is to be performed must be specified. Firstly, a line is added specifying the type of operation, which will be to index a document in this case. So, the operation will be "index" to add a document to the index, and for that key a JSON object is needed. Within this object, an "_id" key is added containing the ID of the document. On the next line the values of the document itself will be described.



```
3  "took" : 222,
4  "errors" : false,
5  "items" : [
6  {
7  "index" : {
8  "_index" : "products",
9  "_type" : "default",
10 "id" : "10",
11 "_version" : 1,
12 "result" : "created",
13 "shards" : {
14 "total" : 2,
15 "successful" : 1,
16 "failed" : 0
17 },
18 "_seq_no" : 0,
19 "_primary_term" : 1,
20 "status" : 201
21 },
22 },
23 {
24 "index" : {
25 "_index" : "products",
26 "_type" : "default",
27 "id" : "11",
28 "_version" : 1,
29 "result" : "created",
30 "shards" : {
31 "total" : 2,
32 "successful" : 1,
33 "failed" : 0
34 },
35 "_seq_no" : 1,
36 "_primary_term" : 1,
37 "status" : 201
38 },
39 },
40 },
41 {
42 "index" : {
43 "_index" : "products",
44 "_type" : "default",
45 "id" : "12",
46 "_version" : 1,
47 "result" : "created",
48 "shards" : {
49 "total" : 2,
50 "successful" : 1,
51 "failed" : 0
52 },
53 "_seq_no" : 2,
54 "_primary_term" : 1,
55 "status" : 201
56 },
57 },
58 {
59 "index" : {
60 "_index" : "products",
61 "_type" : "default",
62 "id" : "13",
63 "_version" : 1,
64 "result" : "created",
65 "shards" : {
66 "total" : 2,
67 "successful" : 1,
68 "failed" : 0
69 },
70 "_seq_no" : 3,
71 "_primary_term" : 1,
72 "status" : 201
73 },
74 }
```

Figure 115: Proof for multiple document processing

So, this request added four unique documents with different IDs; 10, 11, 12, 13. If a user wants to do multiple things at the same query such as deleting and updating different documents, it is possible with this API. In this case, document with the "11" ID will be updated while the document "13" will be completely deleted.



```
0 POST /products/default/ bulk
1 {"update": {"_id": "11"}}
2 {"doc": {"model": "CHR"}}
3 {"delete": {"_id": "13"}}
4
5
6
7
8
9
10
11
12 "result": "updated",
13 "shards": {
14   "total": 2,
15   "successful": 1,
16   "failed": 0
17 },
18 "_seq_no": 4,
19 "_primary_term": 1,
20 "status": 200
21 }
22 },
23 {
24   "delete": {
25     "_index": "products",
26     "_type": "default",
27     "_id": "13",
28     "_version": 2,
29     "result": "deleted",
30     "shards": {
31       "total": 2,
32       "successful": 1,
```

Figure 116: Update and delete documents on the same query

3.7. Observing how the cluster works

For the sake of looking at how the cluster actually looks like “cat API” will be used. This API is basically what provides the user with human readable information about clusters and nodes. It is very convenient to use in alliance with a terminal. It is significant that there is also a “cluster API”, but it returns illegible and verbose JSON objects with a lot of details.

The API is entered directly after the GET HTTP verb, because it is no operating at a specific index and type. The “v” query parameter is used to add a header to make the results more readable.

```
1 GET /_cat/health?v
```

Figure 117: Cluster's health part 1



```
epoch      timestamp cluster      status node.total node.data shards
1566385363 11:02:43  elasticsearch yellow          1         1     4

pri relo init unassign pending_tasks max_task_wait_time active_shards_percent
4      0      0          2              0              -                    66.7%
```

Figure 118: Cluster's health part 2

The results show the health of the cluster. More specifically present that there is a single cluster containing one node, four primary shards and two replicas (unassigned). The replicas are unassigned since there is only a single node within the cluster. It is worth recalling that, replica shards are never assigned to the same nodes as the primary shards. So, since there is no other node inside the cluster, this means that there is nowhere to store the shards. This is also why the status of the cluster is “yellow” (it can be either green for fully functional cluster, yellow for all cluster’s data availability but not all replicas are allocated, red for no availability in some or all of the data).

Although it is clear that only a single node exists it is worth looking some details about it. This procedure is again performed with the “cat API”.

```
1 GET /_cat/nodes?v

ip      heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
127.0.0.1      10          80 25          dim      *      DESKTOP-RAC60VH
```

Figure 119: Node's health

With this someone can observe that this node is also a master node, which is not surprising since it is the only node in the cluster. The “node.role” column specifies that the node may be elected as master, that it is a data node and that it is an ingest node.



The next level of hierarchy inside a cluster is indices. To explore them again the same API is used. The following query provides the user with some useful information such as index health, primary shards, replica shards, number of documents and storage space used.

```
1 GET /_cat/indices?v
```

Figure 120: Indices health part 1

	health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
1	green	open	.kibana_task_manager	sKXJxgw_QE-RTZZJ8hprNA	1	0	2	0	53.7kb	53.7kb
2	yellow	open	product	2UZ-s2-SQK-9_nas7SSyMw	1	1	1000	0	385.7kb	385.7kb
3	green	open	.kibana_1	VxLmk9iwQYmYY1zEM_9cvg	1	0	5	1	44.5kb	44.5kb
4	yellow	open	products	aeNFumoLS5GgXCvf0VpwTg	1	1	3	1	3.9kb	3.9kb
5										
6										

Figure 121: Indices health part 2

It should be stated that two more APIs are useful for observing how shards operate. To be more precise a user can see how the shards have been allocated inside the cluster and more. Subsequently from the figure, it can be seen that all four shards are allocated to the only node that exists in the cluster, while the replica ones wait for another node to be added in order for them to be assigned. The query also provides information on disk usage.

```
1 GET /_cat/allocation?v
```

Figure 122: Disk usage and shard allocation info part 1



1	shards	disk.indices	disk.used	disk.avail	disk.total	disk.percent	host	ip	node
2	4	487.9kb	70.8gb	47.7gb	118.6gb	59	127.0.0.1	127.0.0.1	DESKTOP-RAC60VH
3	2								UNASSIGNED
4									

Figure 123: Disk usage and shard allocation info part 2

Lastly, this query lists all the shards within a cluster. It is visible which index each shard belongs to, whether it is a replica or a primary, the state of the shard and how many documents it contains. The documents should be evenly distributed across the shards, which is thanks to the default routing that was aforementioned in this Thesis. This is not visible since no action was taken for the index “product” so all are stored inside one primary shard.

```
1 GET /_cat/shards?v
```

Figure 124: Shard health and distribution part 1

1	index	shard	prirep	state	docs	store	ip	node
2	product	0	p	STARTED	1000	385.7kb	127.0.0.1	DESKTOP-RAC60VH
3	product	0	r	UNASSIGNED				
4	.kibana_1	0	p	STARTED	5	44.5kb	127.0.0.1	DESKTOP-RAC60VH
5	products	0	p	STARTED	3	3.9kb	127.0.0.1	DESKTOP-RAC60VH
6	products	0	r	UNASSIGNED				
7	.kibana_task_manager	0	p	STARTED	2	53.7kb	127.0.0.1	DESKTOP-RAC60VH
8								

Figure 125: Shard health and distribution part 2

4. Mapping

4.1. Preface

In Elasticsearch, mappings are used to define how documents and their fields should be stored and indexed. The point of doing this, is to store and index data in a way that is appropriate for how the user wants to search data. A couple of examples of what mappings can be used for, could be to define which fields should be treated as full text fields, which fields contain numbers, dates or even geographical locations. Date formats and analysers for full text fields can also be specified. Basically, mappings in Elasticsearch are the equivalent of a schema definition for a table in a relation database -such as MySQL. Referring to a previous statement in this Thesis, Elasticsearch and relational databases are very different technologies, so this analogy is not too accurate, but perhaps it helps to give an overall idea of what mapping is all about. It should be pointed out, that for simple examples, user should not consider to actively deal with mappings. On the other hand, if greater control over how Elasticsearch handles data is needed, then defining mappings is crucial.

4.2. Dynamic

Mapping can be defined explicitly, which is referred to as explicit mapping. That is when the user instructs Elasticsearch what to do with the data when new documents are added. There is a convenient alternative to explicitly defining the data types for all fields; called dynamic mapping. Dynamic mapping means that no mapping is defined explicitly, or at least not for some fields. When new documents are added, Elasticsearch will automatically add mapping for any fields, that do not have any mappings already.

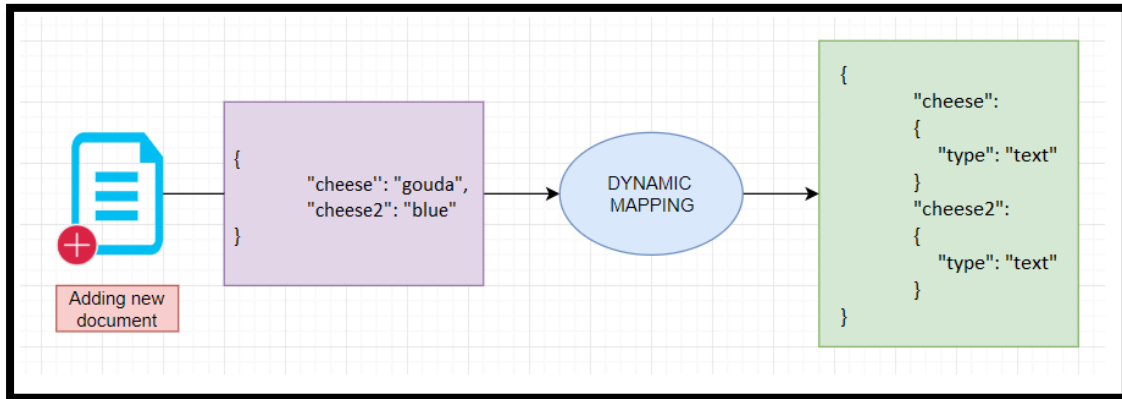


Figure 126: Steps to dynamic mapping

This is done by inspecting the types of values for a document's fields. For example, if a new field called "cheese" is added consisting a string, then Elasticsearch will automatically detect this field as containing characters and use the "text" data type.

It is possible to define rules and defaults for dynamic mapping, but it is something that users are not advised to do. To see which mappings have been added by Elasticsearch behind the scenes Mapping API is used in the Console tool. Type is not used at the request since it runs an error after Elasticsearch version 7.0.

```
{<br>  <br>  \"_id\":2}<br>  Part Shells - Savory\", \"price\":99, \"in_stock\":10, \"sold\":430, \"tags\": [], \"description\": \"Pellentesque at nulla. Suspendisse potenti. Cras in purus eu magna v<br>  {<br>  <br>  \"_id\":3}<br>  Kirsch - Schloss\", \"price\":25, \"in_stock\":24, \"sold\":215, \"tags\": [], \"description\": \"In eleifend quam a odio.\", \"is_active\": true, \"created\": \"2000/11/17\"<br>  {<br>  <br>  \"_id\":4}<br>  Coffee - Colombian Portioned\", \"price\":37, \"in_stock\":37, \"sold\":477, \"tags\": [\"Coffee\"], \"description\": \"Lorem ipsum dolor sit amet, consectetur adipiscing<br>  {<br>  <br>  \"_id\":5}<br>  Venison - Liver\", \"price\":87, \"in_stock\":24, \"sold\":261, \"tags\": [], \"description\": \"Etiam pretium iaculis justo. In hac habitasse platea dictumst. Etiam fauc<br>  {<br>  <br>  \"_id\":6}<br>  Hestea - Iced Tea\", \"price\":121, \"in_stock\":23, \"sold\":431, \"tags\": [], \"description\": \"Donec ut mauris eget massa tempor convallis. Nulla neque libero, conva
```

Figure 127: Mapping API example for behind the scenes mapping

The first field that can be seen is "created" where dates are supplied. It is obvious that Elasticsearch, correctly detected this and used the "date" data type. When adding new string fields, Elasticsearch performs date detection and checks if the field's contents match any of the dynamic date formats that have been defined. By default, that would be a year, month and day separated by slashes and an optional timestamp. If there is a match, the matching date format will be added to the mapping for the field. It should be underlined that date detection can be configured to be turned off.



The “description” field has been mapped as a text field and the “in_stock” field as the “long” data type. This happens because it has no way of knowing how large numbers/ integers are intended to be stored. It is clear that “is_active” field has been mapped to a Boolean field because it contains Boolean values; true or false.

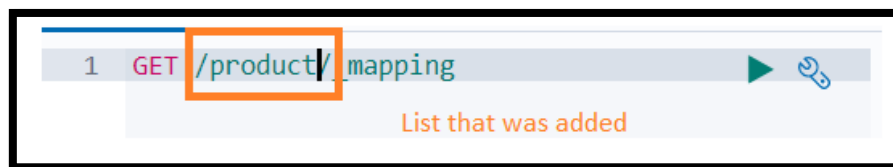


Figure 128: GET request for mapping information over an index part 1



Figure 129: GET request for mapping information over an index part 2

For the most part, the rules of how Elasticsearch dynamically maps fields, are quite simple. One thing that is a bit special though, is how text fields are treated. For example, the "description" field is mapped as the type “text”, when at the same time it has a “fields” property containing a field named “keyword” with a type of “keyword”. What this means is that the field has two mappings, meaning that a field can have multiple mappings. By default, each text field is mapped using both the “text” type and the “keyword” type. The difference between the two, is that the “text” type is used for full-text searches, and the “keyword” type for exact matches,



aggregations and such. To facilitate both use cases, Elasticsearch automatically maps each text field in both ways.

4.3. Meta-fields

Every document that is stored within an Elasticsearch cluster, has some meta-data associated with them, apart from the data fields that were specified earlier in this Thesis, when indexing documents. These fields are called meta fields. There are ten of them and are not equally important. Often times, they will not be needed for direct use, but it is a valuable feature to exist.

First there is a meta field named “_index”, which contains the name of the index to which a document belongs. This field is added to documents automatically and is used by Elasticsearch internally. Next, the “_id” meta field unsurprisingly stores the ID of documents and can be queried within certain queries. Usually this field will not be queried directly, but it is used when looking up documents based on their IDs. The “_source” meta field contains the original JSON object that was passed to Elasticsearch when indexing the document. The field is not indexed, and therefore it cannot be searched, but it can be retrieved. Furthermore the “_field_names” meta field contains the names of every field that contains a non-null value. It should be recalled that routing helps with shard manipulation by Elasticsearch. For this property “_routing” meta field is used. More specifically, if custom routing is used to route documents to shards based on a specified value, then this value is stored within that meta field. Unless custom routing is used, this meta field is of no importance. Elasticsearch as mentioned earlier in this Thesis, uses versioning of documents internally with a meta field named “_version”. If a document is retrieved by ID, this meta field will be part of the result. The value is simply an integer which starts at one and is incremented every time a change occurs on the document. Lastly, the “_meta” field can store custom data that is not touched in any way by Elasticsearch. It is therefore a place, where whatever application specific data that the user might have, can be store.



4.4. Data types

Each field within Elasticsearch is of a given data type. Data types can be divided into four categories: core data types, complex data types, geo data types and specialized data types.

4.4.1. Core data types

Firstly, core data types will be examined. To begin with there is a data type called “text”, which is used for full-text values. Examples of such values would be product descriptions, blog posts and so on. Due to the nature of full-text fields they are rarely used for sorting and aggregating documents. That is something that is typically accomplished with “keyword” fields. Such fields also contain text, but not “full text”. This is because “keyword” fields are not analysed. It basically means that values are stored exactly as defined at the time of adding documents to an index, whereas “text” fields are stored in a way that is optimal for performing full-text searches. A couple of examples of “keyword” fields would be a “model” or a “category” field. So “text” fields are used for text that the user wants to search, whereas “keyword” fields are you used for values that the user wants to filter or use for aggregations. The above mentioned was a slight simplification, but accurate.

Next, “numeric” data types exist, which is a group of data types that are used for numeric values. This can include integer, float, long, half_float, scaled_float and many more that are used in most programming languages. Two of them are important and are worth mentioning; “byte” data type and “scaled_float” data type. First, the “byte” data type is used for integer values between -128 and 127, so lower numbers even than “short” data type. Secondly, the “scaled_float” data type is basically a floating point, but it is stored as a “long” internally. That is possible because the data type is accompanied by a so-called scaling factor. What this does, is that at index time, the floating point is multiplied by this scaling factor and rounded off to the closest long value. For example, a floating point of 4.56 and a scaling factor of 10, would be stored as forty-six internally. All search queries will behave as if the document had a value of 4.6, even though that is not the value that is stored internally.



The idea is that floating point types are more efficient to store as integers, because it saves disk space, as integers are easier to compress. By doing so, accuracy is sacrificed but that may be a good compromise if no ultra-precise numbers are needed.

Afterwards, “date” data type exists, which as its name implies, is used for storing dates and can supply dates in three unique ways; by specifying the date as a string, an integer representing the number of seconds since the epoch or as a long representing the milliseconds since the epoch. The date format that should be used for a field when supplying a string value, can be configured. If no format is configured, a default format is used, which can either be a string which optionally contains time, or the number of milliseconds since the epoch. Internally, dates are stored as a long value representing the number of milliseconds since the epoch.

Moving on, there is the “Boolean” data type, which is used for storing true or false values. Binary data can be stored with the “binary” data type, which expects a Base64 encoded string. It must be stressed that the value is not stored by default, meaning that the user cannot search the field and he also cannot retrieve it independently of the “_source” meta field.

Finally, the last core data type is named “range” which is a bit special. It is used for range values, such as date ranges or integer intervals like five to one hundred. A lower and an upper boundary is defined when indexing a document, and this can be done by using the keywords “gt”, “gte”, “lt”, “lte”; greater than, greater than or equal to, less than, less than or equal to. There is a query named “range” which utilizes this data type, which is mainly used in the searching part of Elasticsearch.

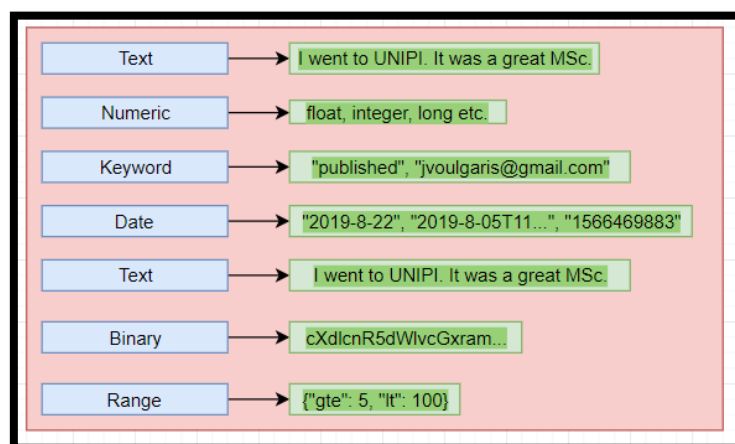


Figure 130: Core data types with examples



4.4.2. Complex data types

The next category of data types is referred to as “complex”. These data types as their name states are more complex data types than the core ones. The first data type is the “object” one. It is mainly used for storing objects. Since objects, are plainly JSON objects there is nothing special about them. They consist of fields that are of a given data type, and they may contain nested objects as well. When an object is indexed, the system is supplied with a JSON object, but Elasticsearch flattens the object for storage. This means that internally, the object simply consists of key-value pairs, and any nested objects are handled by adding dots to the key names to preserve the hierarchy of the objects.

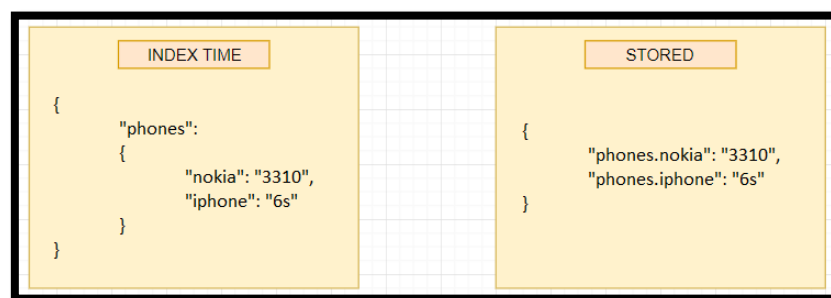


Figure 131: Complex data type structure example

Next up, there is an “array” data type. Any field in Elasticsearch may contain zero or more values by default, such as an array of strings. That is possible without having the user to explicitly declare this, and it will work even if the data type is defined as “integer”, for instance. It is important that all values must be of the same data type. Arrays are flattened out within Elasticsearch. For example, [1, [2, 3]] is flattened to [1, 2, 3]. It should be emphasized that even though an array of objects can be stored, the user cannot perform queries independently for one object alone, since he is obligated to do it for the whole array. That is how Elasticsearch and by extension Lucene works, since they have no concept of inner objects.

Lastly, the last category of data types in this section is the “geo” data types. As the name suggests, these data types are used for handling geographical data, such as



latitude and longitude pairs. A data type that handles exactly that, is the “geo_point” type. It allows the user to specify topographical pairs in four different formats. The first option is an object with “lat” and “lon” keys. The next option is as a string with latitude and longitude separated by a comma. The third option is a so-called geohash, and the fourth way is as an array with latitude and longitude on this specific order. The point is that this data type is used when a specific geographical point exists that can be expressed as a latitude and longitude pair.

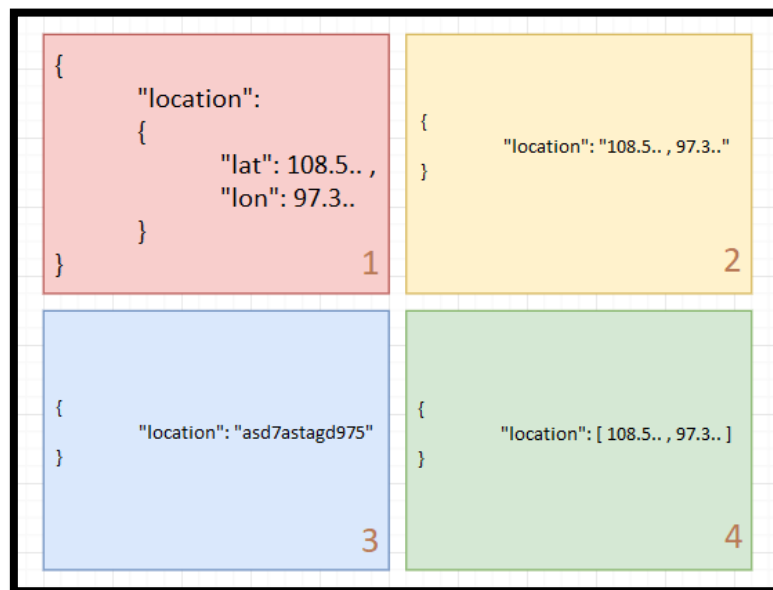


Figure 132: Complex data type (geo_point) example

4.4.3. Specialized data types

The last category of data types is referred to as “specialized”. These data types as their name states have a very specific purpose, such as storing IP addresses, attachments etc.

The data type “IP” stores IP addresses as either IPv4 or IPv6 addresses. These queries are used according to the CIDR notation. The next data type is about enabling auto-completion and search suggestions. That is done by using something called suggesters. Using this data type enables very efficient lookups, because auto-completion/ search as you type, needs to be sharp. Elasticsearch therefore uses data structures that are slow to build, but enable very fast lookups, and stores this in



memory. All things considered the last data type of this category is named as ‘attachment’ data type. This data type is used for indexing documents that contain text, and to make this text searchable. For example, a PDF document exists that must be available for search by the users. Doing this requires a plugin named “Ingest Attachment Processor”. This plugin uses a library named ‘Apache Tika’ for performing text recognition.

```
sudo bin/elasticsearch-plugin install ingest-attachment
```

Figure 133: Specialized data types tool installation

5. Wazuh

5.1. General idea

Wazuh is a security detection, visibility and compliance open source project. Although its purpose wasn't it, it was integrated with Elastic Stack, evolving into a more comprehensive solution. It is used mainly for monitoring: applications, user behaviour, file integrity and cloud. Also, important is that it can be used as a Detection tool since it offers, diagnostics and information about intrusion attempts, vulnerabilities and malware activity. It is noteworthy that the Response feature that it offers for prevention and forensics is a crucial part of a suite of this calibre.

5.2. Components

The main components of the Wazuh architecture include Wazuh Agents and Wazuh Servers. Wazuh agents run on Windows, Linux and Mac operating systems. They are used to collect different types of system and application data that are forwarded to the Wazuh Server. This information is sent through an encrypted and authenticated channel. Furthermore, the agents can be used to monitor physical servers, virtual machines and cloud instances like Azure or Google cloud.

On the other hand, the Wazuh server, is the component that is in charge of analysing the data received from the agents. Subsequently, it triggers alerts when an event matches a rule (e.g. file changed). The server usually runs on a stand-alone physical machine, virtual machine or cloud instance. As shown in the figure below Wazuh is integrated and works fluently with Elastic Stack (single node example). The main components that Wazuh needs from Elastic Stack to work are Elasticsearch, Kibana and Filebeat.

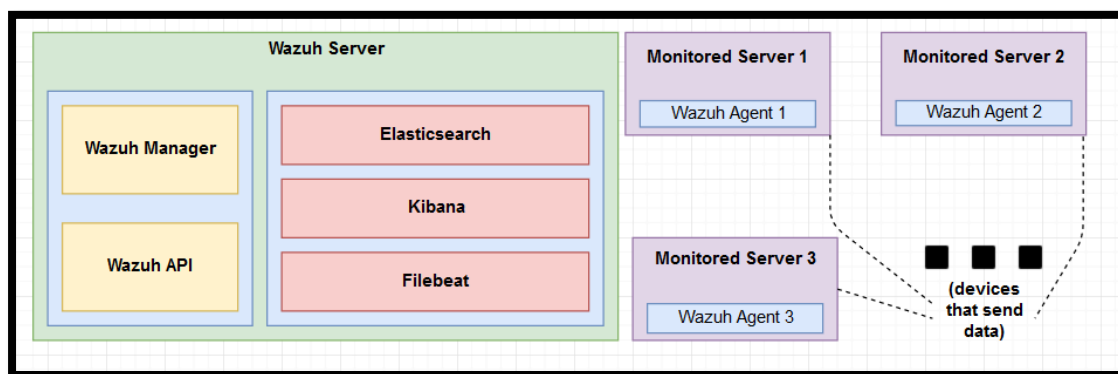


Figure 134: Wazuh structure

5.3. Required ports

In order for Wazuh and Elastic Stack to operate, several ports must be available and opened. The ports needed for Elastic Stack were mentioned earlier in this Thesis but will be summarized in a table as well with the ones used by Wazuh. These ports are used for the different components of the system to communicate freely and efficiently with each other.

WAZUH			
Component	Port	Protocol	Purpose
Wazuh Manager	1514	TCP	Collected events are sent from agents through this port (for TCP)
	1514	UDP	Collected events are sent from agents through this port (for UDP)
	1515	TCP	Service for agent registration
	1516	TCP	Used for Wazuh cluster communications
	514	TCP	Collected events from syslog are sent through this port (for TCP)



	514	UDP	Collected events from syslog are sent through this port (for UDP)
Wazuh API	55000	TCP	Used for incoming HTTP requests
ELASTIC STACK			
Component	Port	Protocol	Purpose
Elasticsearch	9200	TCP	Used by the Elasticsearch RESTful API
	9300-9400	TCP	Used for the cluster communications
Kibana	5601	TCP	Used for the Kibana web interface

5.4. Archiving data storage

In addition to being sent to Elasticsearch, both alerts and non-alert events are stored in files on the server side. These files are written in JSON format or in plain text as logs. They are daily compressed and signed with the MD5 and SHA1 cryptographic algorithms and the path of the directory is presented below.



Figure 135: Data storage/ archive

5.5. Common use cases



5.5.1. Signature-based log analysis

Since Wazuh is used to automatically aggregate and analyse log data, there are many cases where evidence of a probable attack can be found inside the logs. The agent running on a client is responsible for reading log messages and sending them to the server where the analysis takes place. It is of high importance to know, that when no agent is deployed, the server can receive data from network devices or applications through syslog. Wazuh afterwards will use decoders to identify the source application of the log message and then analyse the data with a particular ruleset in mind. Rules can be found in the following directory while at the same time some examples are shown.

```
<rule id="5716" level="5">
  <if_sid>5700</if_sid>
  <match>^Failed|^error: PAM: Authentication</match>
  <description>SSHD authentication failed.</description>
  <group>authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,</group>
</rule>
```

Figure 136: Signature-based log analysis example

Rules as can be seen include a “match” field, which is used to define the pattern that the rule should look for. The resulting alert priority is specified with the “level” argument. To be more precise about how this operates, when an event is collected by an agent, the manager will generate an alert. It is a prerequisite that the alert level must be higher than zero for this to be successful. An example of how an alert will look like is presented below.



```
{
  "agent": {
    "id": "1041",
    "ip": "10.0.0.125",
    "name": "vpc-agent-centos-public"
  },
  "decoder": {
    "name": "sshd",
    "parent": "sshd"
  },
  "dstuser": "root",
  "full_log": "Mar 5 18:26:34 vpc-agent-centos-public sshd[9549]: Failed
  "location": "/var/log/secure",
  "manager": {
    "name": "vpc-ossec-manager"
  },
}
```

Figure 137: Alert example for Signature-based log analysis

Once the alerts are generated, they are sent to Elasticsearch where they are enriched with geolocation information and visualized inside Kibana.

5.5.2. Rootkit detection

Each monitored system that hosts a Wazuh Agent, is scanned both at a kernel and user level to detect rootkits that might sneaked inside. Rootkits can hide other processes, files or network connections like themselves. It is a type of malware that replaces or changes existing operating system components for the adversary's favour. Wazuh uses the Rootcheck component to detect such anomalies or suspicious activities.

ACTION	DETECTION MECHANISM	SYSTEM CALLS	BINARY
Detection of hidden processes	Compare output of system binaries and system calls	Setsid() Getpgid() Kill()	ps
Detection of hidden files	Compare output of system binaries and system calls	Stat() Opendir() Readdir()	ls



Detection of hidden ports	Compare output of system binaries and system calls	Bind()	netstat
Detection of known rootkits	Finding in a database about malicious files	Stat() Fopen() Opendir()	
	Inspecting files content by using signatures	Fopen()	
	Ownership anomalies and file permission anomalies detection	Stat()	

An example of an alert that is generated when a hidden process is found can be seen in the figure below.

```
{
  "agent": {
    "id": "1030",
    "ip": "10.0.0.59",
    "name": "diamorphine-POC" Rootkit name
  },
  "decoder": {
    "name": "rootcheck"
  },
  "full_log": "Process '562' hidden from /proc. Possible kernel level rootkit.",
  "location": "rootcheck",
  "manager": {
    "name": "vpc-ossec-manager"
  },
  "rule": {
    "description": "Host-based anomaly detection event (rootcheck).",
    "firedtimes": 4,
    "groups": [
      "ossec",
      "rootcheck"
    ],
    "id": "510",
    "level": 7
  }
}
```

Figure 138: Rootkit detection example



5.5.3. File integrity monitoring

Wazuh offers a component called FIM that detects and alerts when operating system and application files are altered. This is often used to detect modification and unauthorized access to data that is sensitive. It is highly notable that if the servers are in scope with PCI DSS, the requirement 11.5 mention that a file integrity monitoring solution must be installed. An alert that is generated when a file is altered is presented below with its according metadata; SHA1 and MD5 checksums, the file sizes before and after the modification, privileges, ownership and who-data information.

```
{
  "timestamp": "2018-07-10T14:05:28.452-0800",
  "rule": {
    "level": 7,
    "description": "Integrity checksum changed.",
    "id": "550",
    "firedtimes": 10,
    "mail": false,
    "groups": [
      "ossec",
      "syscheck"
    ],
    "pci_dss": [
      "11.5"
    ],
    "size_before": "421",
    "size_after": "433",
    "perm_after": "100644",
    "uid_after": "0",
    "gid_after": "0",
    "md5_before": "4b8ee210c257bc59f2b1d4fa0cbbc3da",
    "md5_after": "acb2289fba96e77cee0a2c3889b49643",
    "sha1_before": "d3452e66d5cfd3bcb5fc79fbcf583e8dec736",
    "sha1_after": "b87a0e558ca67073573861b26e3265fa0ab35c",
    "sha256_before": "6504e867b41a6d1b87e225cfafae3779a3",
    "sha256_after": "bfa1c0ec3ebfaac71378cb62101135577521"
  }
}
```

Figure 139: FIM example

5.6. Rulesets

5.6.1. Introduction to Wazuh Rulesets

The Ruleset, which is include in the Wazuh Manager installation by default, is a set of XML files called Decoders and Rules. The Decoders are used by the “Analysis Daemon” to extract the required fields and values from the incoming events or log messages. Then, the Rules are used to generate an alert based on the fields extracted by the corresponding Decoder. The alerts will follow the data flow and they will be finally displayed on the Kibana web.



There is a Wazuh-Ruleset repository in GitHub where someone can find all of the Decoders and Rules that are currently available in the corresponding version of Wazuh.

Furthermore, it is highly notable that there is a file called “update_ruleset” in the following directory that is used to update the current ruleset on the system in case of a possible out-of-date.

`/var/ossec/bin/update_ruleset`

Figure 140: File used to update rulesets

5.6.2. Decoders

5.6.2.1. Traditional decoding

Every time an event pops up, in order to extract information, there is an important step for detection and processing of threats. More specifically, Wazuh uses decoders that extract the most relevant fields of an event and identify event types. This has a result in enhancing event’s information to help further in indexing and analysis. OSSEC traditionally provides thirteen fields for storing extracted information while only eight of them can be extracted at the same time. These predefined fields are:

- user
- srcip
- dstip
- srcport
- dstport
- protocol
- Action
- id
- url
- data
- extra_data
- status
- system_name

An example about how a typical decoder works is shown in the following screenshot.



```
<decoder name="web-access-log">
  <type>web-log</type>
  <prematch>^\d+.\d+.\d+.\d+ - </prematch>
  <regex>^\(d+.\d+.\d+.\d+\) - \S+ [\S+ -\d+] </regex>
  <regex>"\w+ (\S+) HTTP\S+ (\d+) </regex>
  <order>srcip,url,id</order>
</decoder>
```

0-9 numbers

Anything but whitespace

Regular expression

Figure 141: Traditional decoder example

5.6.2.2. Dynamic decoding

It is often necessary to extract more than eight relevant fields from an event. Also, the actual data items extracted have no relationship to the limited list of predefined field names. Since there is no way to operate within these constraints, Wazuh has extended OSSEC to allow the decoding of an unlimited number of fields with field names that clearly relate to what is being extracted.

Wazuh transforms any field name included in the <order> tag into a JSON field.

```
Decoder
<decoder name="auditd-config_change">
  <parent>auditd</parent>
  <regex offset="after_regex">^audit=(\S+) ses=(\S+) op="(\S+)"</regex>
  <order>audit.auid,audit.session,audit.op</order>
</decoder>

Extracted information from an alert
** Alert 1486483073.60589: - audit,audit_configuration,
2017 Feb 07 15:57:53 wazuh-example->/var/log/audit/audit.log
Rule: 80705 (level 3) -> 'Auditd: Configuration changed'
type=CONFIG_CHANGE/msg=audit(1486483072.194:20): auid=0 ses=6 op="add rule" key="audit-wazuh-a" list=4 res=1
audit.type: CONFIG_CHANGE
audit.id: 20
audit.auid: 0
audit.session: 6
audit.op: add rule
audit.key: audit
audit.list: 4
audit.res: 1
```

Figure 142: Dynamic decoding example



```
JSON Output
},
"full_log": "type=CONFIG_CHANGE msg=audit(1486483072.19
"audit": {
  "type": "CONFIG_CHANGE",
  "id": "20",
  "aud": "0",
  "session": "6",
  "op": "add rule",
  "key": "audit",
  "list": "4",
  "res": "1"
},
"decoder": {
  "parent": "auditd"
  "name": "auditd"
},
},
{
  "rule": {
    "level": 3,
    "description": "Auditd: Config
    "id": 80705,
    "firedtimes": 2,
    "groups": [
      "audit",
      "audit_configuration"
    ]
  }
},
},
```

Figure 143: Decoder explained through a JSON output

5.6.2.3. Decoder syntax

The decoders extract the information from the received events. When an event is received, the decoders separate the information in blocks to prepare them for their subsequent analysis. According to Wazuh's documentation, the attributes listed below define a decoder and its link with another parent decoder.

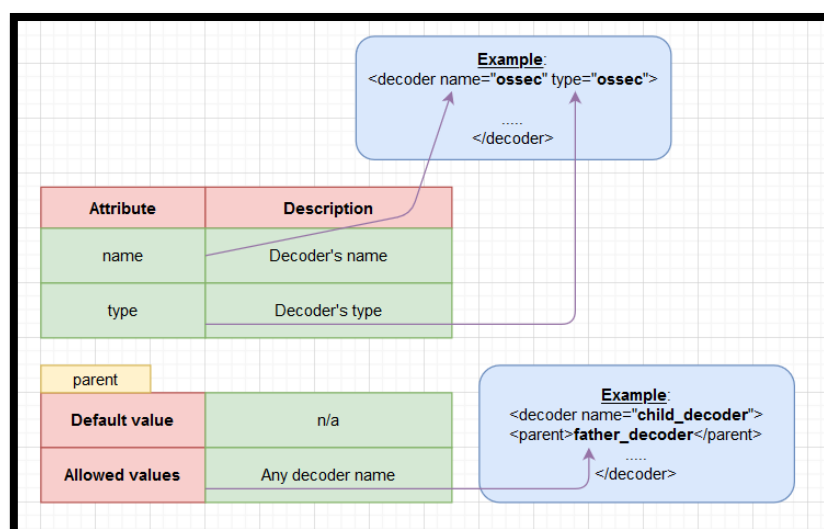


Figure 144: Decoder syntax part 1



After, there are attributes called regexes or Regular expressions. They are sequences of characters that define a pattern. Decoders use them to find words or other patterns into the rules.

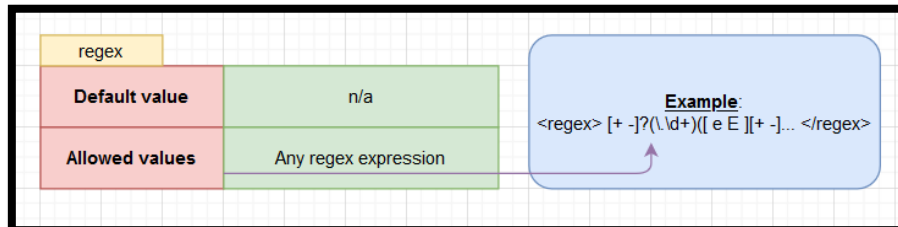


Figure 145: Decoder syntax part 2

Another attribute of a decoder is the fts. It is used to designate a decoder as one in which the first time it matches the administrator would like to be alerted.

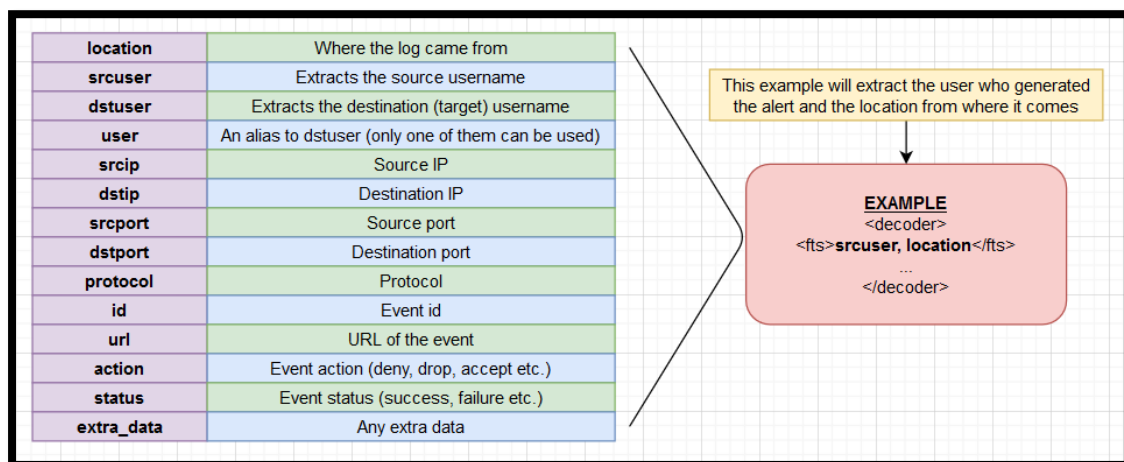


Figure 146: Decoder syntax part 3

Finally, the last argument that can be placed inside a decoder is var. It defines a variable that may be used in any place of the same file. An example can be seen in the screenshot below.



Attribute	Value
Name	Name for the variable

```
EXAMPLE
<var name="header">myprog</var>
<var name="offset">after_parent</var>
<var name="type">syscall</var>

<decoder name="syscall">
  <prematch>^$header</prematch>
  </decoder>

<decoder name="syscall-child">
  <parent>syscall</parent>
  <prematch offset="$offset">^: $type </prematch>
  <regex offset="after_prematch">(\S+)</regex>
  <order>syscall</order>
  </decoder>
```

Figure 147: Decoder syntax part 4

5.6.3. Rules

5.6.3.1. General idea

Regarding the Rules creation process, there is a custom Rules and Decoders creation guide in the Wazuh site. It is used when there is a specific log message or event with a format that is not ready to be managed by the current Ruleset. In those cases, firstly a custom Decoder must be created to extract the necessary fields and their values. Afterwards, the corresponding custom Rule must be created to generate an alert according to the requirements. The place to store all these Rules and Decoders is as follows.

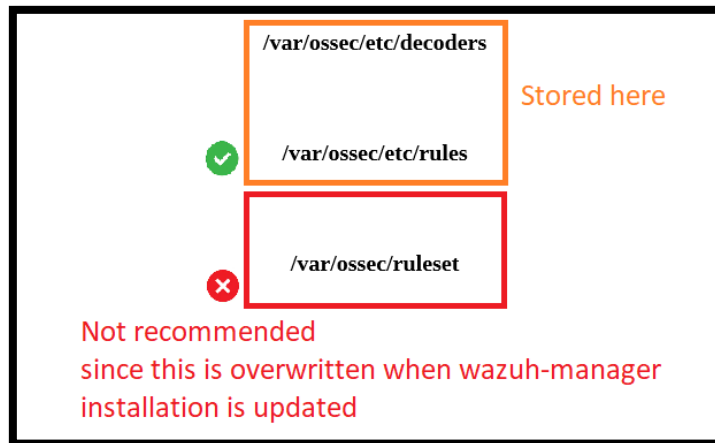


Figure 148: Storing options for rules and decoders

These XML files can be created and edited directly from the CLI with any text editor and then can be uploaded to the Wazuh-Manager Server. It is worth mentioning that this procedure can also be performed inside the Kibana Wazuh application.

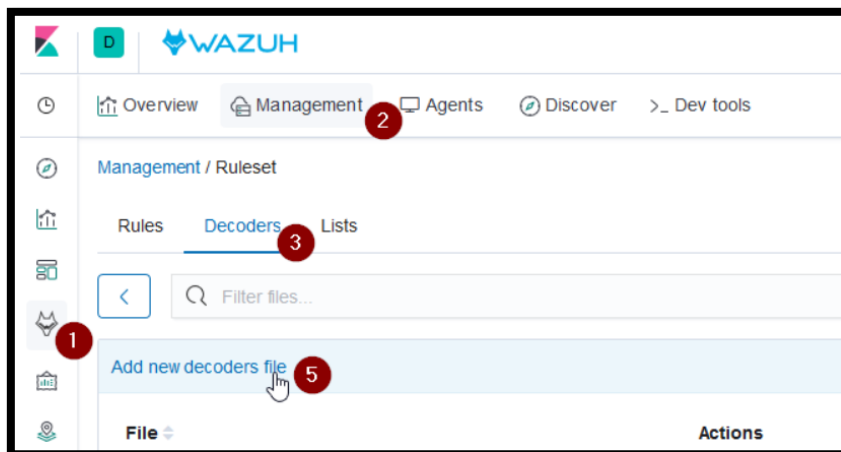


Figure 149: Steps for adding a decoder file on Wazuh-Manager

Finally, every change requires to restart the Wazuh-Manager service to update the file changes. The logs that the expert wants to parse can be tested with the following.



```
/var/ossec/bin/ossec-logtest
```

Figure 150: Testing tool for logs

Basically, it tests how an event is decoded and if an alert is actually generated. To do so an example will be as follows.

```
Log file to be decoded and tested
Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: Accepted publickey
for root from 73.189.131.56 port 57516
```

Figure 151: Log file to be decoded and tested

```
$ /var/ossec/bin/ossec-logtest
Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: Accepted publickey fo
**Phase 1: Completed pre-decoding. The log file changed into
full event: 'Mar  8 22:39:13 ip-10-0-0-10 sshd[2742]: A
hostname: 'ip-10-0-0-10'
program_name: 'sshd'
log: 'Accepted publickey for root from 73.189.131.56 po
**Phase 2: Completed decoding. Matching with a decoder
decoder: 'sshd'
dstuser: 'root'
srcip: '73.189.131.56'
**Phase 3: Completed filtering (rules). Matching with a rule
Rule id: '5715'
Level: '3'
Description: 'sshd: authentication success.'
**Alert to be generated.
```

Figure 152: Decoding and testing result

5.6.3.2. Rule syntax

In this section, XML labels used to configure a rule will be introduced. The first label used inside a rule is called `<rule>` and it includes basic information about the rule.

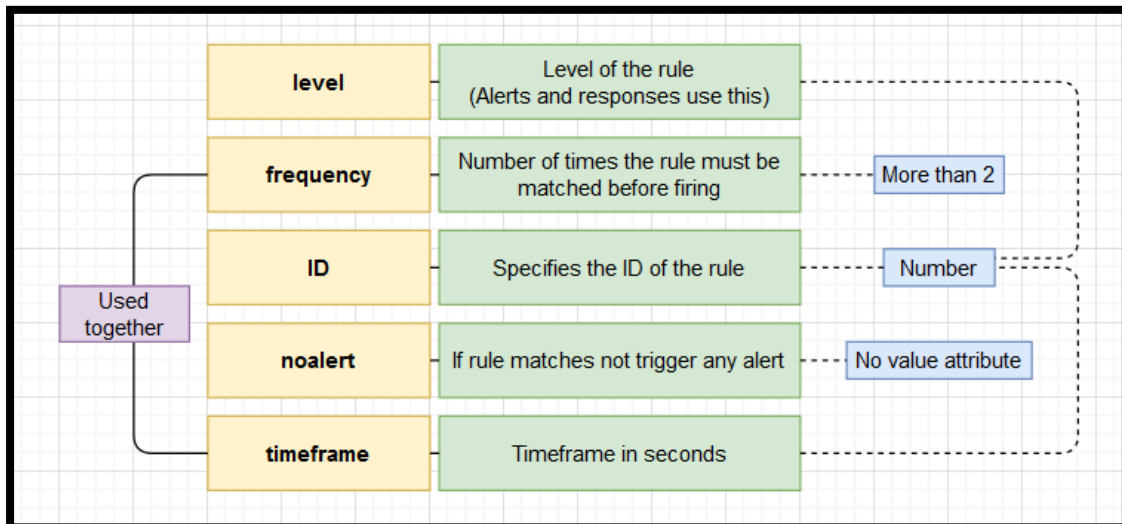


Figure 153: Rules syntax part 1

Another important label is `<match>`. It is basically an if condition, where the rule is activated when the condition is true. The same way `<if_sid>` and `<regex>` works, since the first checks if the id is matched and the second tests if there is any match with a regex.

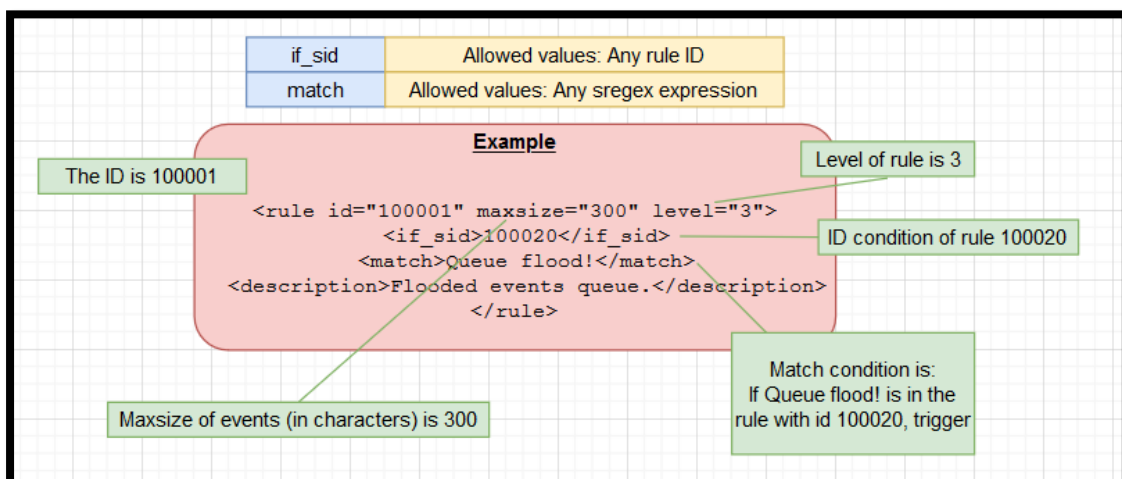


Figure 154: Rules syntax part 2

Another use case would be to match something related to IP addresses. With `<srcip>`, any IP address can be compared to an IP decoded as `srcip`. Furthermore with `<dstip>`, any IP address can be compared to an IP decoded as `dstip`.



```
<rule id="100225" level="0">  
  <if_sid>40101</if_sid>  
  <srcip>127.0.0.1</srcip>  
  <description>Ignore this</description>  
</rule>
```

If rule 100225 is triggered by localhost, don't produce any alert

Figure 155: Rules syntax part 3

Subsequently as shown above, there is a field called <description>, which as its name states, it gives a human readable description to the rule in order to provide context to each alert regarding the nature of the events matched by it. It is important to note that it is a necessary field.

To conclude there are some arguments that specify that a decoded attribute of the rule must be the same. Therefore, they all start with the term “same_”.

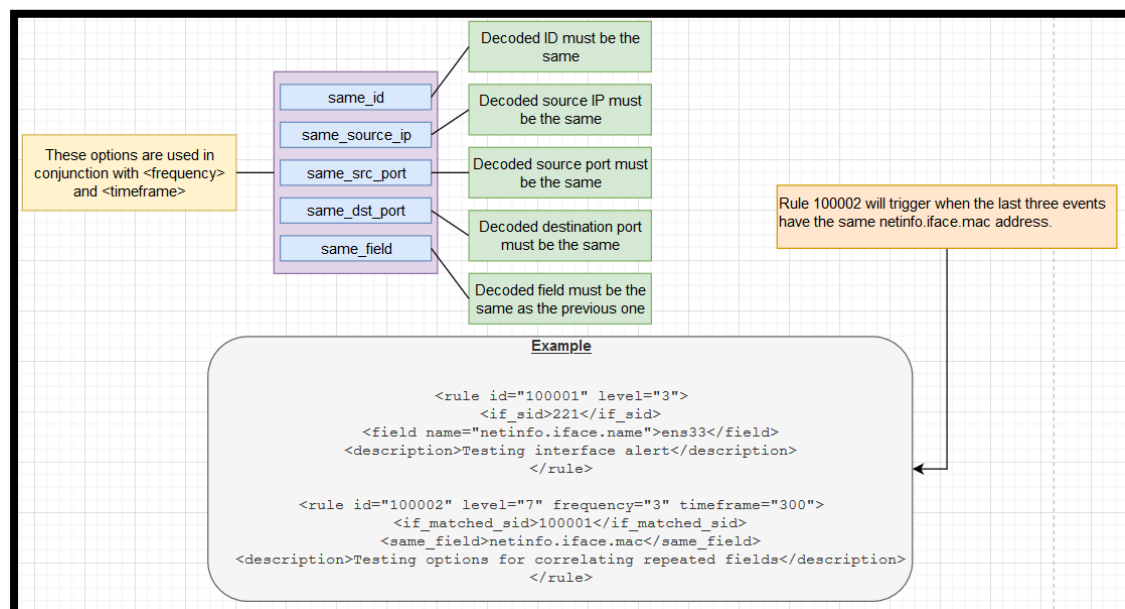


Figure 156: Rules syntax part 4

5.6.4. Regex



5.6.4.1. General idea

A regular expression or regex is a sequence of characters that define a search pattern. Usually such patterns are used by string searching algorithms for “find” or “find and replace” operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory. Regex is often referred to as rational expression or regexp. There are two types of regular expressions in Wazuh: regex (OS_Regex) and sregex (OS_Match). A great example of a regex is displayed below.

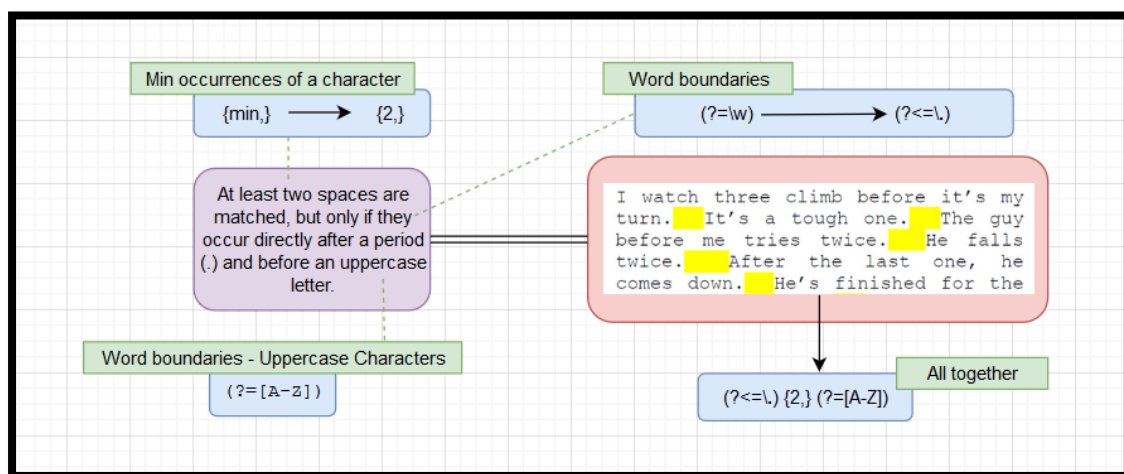


Figure 157: Traditional Regex syntax

5.6.4.2. Regex syntax

The types of regular expressions have both different syntax and meaning. OS_Regex or regex is a fast and simple library for regular expressions in C. It is designed to be simple while still supporting the most common regular expressions.



SUPPORTED EXPRESSIONS	
EXPRESSIONS	VALID CHARACTERS
\w	a-z, A-Z, 0-9, '-', '_', '@'
\d	0-9 character
\s	Spaces or " "
\t	Tabs
\p	()*+,-,.;:<=>?[]!{}£%^\&
\W	Anything not w
\D	Anything not d
\S	Anything not s
\.	Anything

MODIFIERS	
EXPRESSIONS	ACTIONS
+	To match one or more times
*	To match zero or more times

Figure 158: Wazuh Regex syntax part 1

SPECIAL CHARACTERS	
EXPRESSIONS	ACTIONS
^	To specify the beginning of the text
\$	To specify the end of the text
	To create a logical or between multiple patterns

CHARACTERS ESCAPING					
\$	()	\		<
\\$	\(\)	\\	\	\<

Figure 159: Wazuh Regex syntax part 2

It is noteworthy that there are some limitations.

- The “*” and “+” modifiers can only be applied to backslash expressions, not bare characters

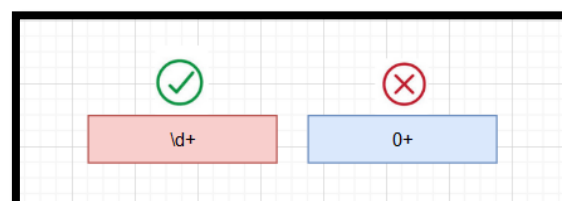


Figure 160: Wazuh Regex syntax part 3



- Alternation in a group cannot be performed, e.g. “(fruits | vegies)” is not allowed
- Complex backtracking is not supported. For instance, “\p*\d*\s*\w*:” does not match a single colon, because “\p*” consumes the colon
- There is a difference between “.” and “\.”

.	Matches a literal dot
\.	Matches any character

Figure 161: Wazuh Regex syntax part 4

- There is no syntax to match a literal caret (“^”), asterisk (“*”) or plus (“+”)

On the other hand, OS_Match or sregex is faster but only supports simple string matching and the following special characters.

EXPRESSIONS	ACTIONS
^	To specify the beginning of the text
\$	To specify the end of the text
	To create a logic: or, between multiple patterns
!	To negate the expression

Figure 162: Wazuh Regex syntax part 5

6. Monitoring and Log Analysis Case Scenario

6.1. Installing tools on Linux

6.1.1. Elasticsearch and Kibana

As with windows a combination of files and tools must be downloaded either from the elastic website or from the web. Firstly, since Linux are used for this example a ‘.tar’ file must be downloaded and extracted from elastic’s website.

```
john@john-virtual-machine:~/Downloads$ tar -zxf elasticsearch-7.3.1-linux-x86_64.tar.gz
```

Figure 163: Unzipping the Elasticsearch file on Linux

And the tool is booted as follows.

```
root@john-virtual-machine:~/usr/share/elasticsearch/bin# ./elasticsearch
```

Figure 164: Starting Elasticsearch on Linux

Afterwards to make sure that the procedure was successful and that the tool indeed runs properly the curl tool must be downloaded in order to see if the request for fetching Elasticsearch queries is working.

```
john@john-virtual-machine:~$ sudo apt install curl
```

Figure 165: Install CURL tool command



```
john@john-virtual-machine:~$ curl http://localhost:9200
{
  "name" : "john-virtual-machine",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "DjQNWszbQf-RcO-4XbRGSw",
  "version" : {
    "number" : "7.3.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "4749ba6",
    "build_date" : "2019-08-19T20:19:25.651794Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 166: Showing with CURL tool the Elasticsearch cluster

It is not a common secret that this procedure must be followed for Kibana as well but in order to evaluate the successfulness of the procedure, user must head to the localhost address.

```
john@john-virtual-machine:~/Downloads$ tar -zxf kibana-7.3.1-linux-x86_64.tar.gz
```

Figure 167: Unzipping the Kibana file on Linux

```
root@john-virtual-machine:/usr/share/kibana/bin# ./kibana --allow-root
```

Figure 168: Starting Kibana on Linux

6.1.2. Logstash

Much like all the other tools it is downloaded, extracted and configured through the terminal.

```
john@john-virtual-machine:~/Downloads$ tar -zxf logstash-7.3.1.tar.gz
```

Figure 169: Unzipping the Logstash file on Linux

6.1.3. Filebeat



It is noteworthy that Filebeat doesn't need Logstash to operate. On the other hand, Logstash will be downloaded for the purpose of this scenario. Firstly, at the elastic's website the package is downloaded. Afterwards, much like in the aforementioned tools it is extracted and then booted as follows.

```
john@john-virtual-machine:~/Downloads$ tar -zxf filebeat-7.3.1-linux-x86_64.tar.gz
```

Figure 170: Unzipping the Filebeat file on Linux

```
john@john-virtual-machine:~/Downloads/filebeat-7.3.1-linux-x86_64$ sudo chown root filebeat.yml
john@john-virtual-machine:~/Downloads/filebeat-7.3.1-linux-x86_64$ sudo ./filebeat -e
```

Figure 171: Starting Filebeat on Windows and proof part 1

It goes without mentioning that first the “.yml” file for Filebeat must be altered in order for it to work as planned. The configuration can be seen in the figure below.

```
#===== Filebeat inputs =====
filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/log/*.log
    #- c:\programdata\elasticsearch\logs\*
```

Figure 172: Filebeat configuration on Linux part 1



```
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]

setup.kibana:
  # Kibana Host
  # Scheme and port can be left out.
  # In case you specify and additional
  # IPv6 addresses should always be
  host: "localhost:5601"
```

Figure 173: Filebeat configuration on Linux part 2

Confirmation that the installation of Filebeat was successful is presented in the figure below.

```
john@john-virtual-machine:~$ sudo systemctl daemon-reload
john@john-virtual-machine:~$ sudo systemctl start filebeat.service
john@john-virtual-machine:~$ sudo systemctl daemon-reload
john@john-virtual-machine:~$ sudo systemctl enable filebeat.service
Synchronizing state of filebeat.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable filebeat
john@john-virtual-machine:~$ sudo systemctl stop filebeat.service
john@john-virtual-machine:~$ sudo systemctl start filebeat.service
john@john-virtual-machine:~$
```

Figure 174: Starting Filebeat on Windows and proof part 2

6.1.4. Wazuh (scenario)

6.1.4.1. Manager

Wazuh is separated to two parts on the server side. The first component is called Wazuh Manager. To install Wazuh manager the following commands are used and then at the second figure a confirmation that it indeed runs correctly is shown.



```
# curl -Ls https://github.com/wazuh/wazuh/archive/v3.9.5.tar.gz | tar zx  
  
# cd wazuh-*  
# ./install.sh
```

Figure 175: Install Wazuh-Manager with CURL tool

```
Σεπ 03 13:45:42 john-virtual-machine env[61963]: Started ossec-analysisd  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: 2019/09/03 13:45:42 oss  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: 2019/09/03 13:45:42 oss  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: Started ossec-syscheckd  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: Started ossec-remoted..  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: Started ossec-logcollec  
Σεπ 03 13:45:42 john-virtual-machine env[61963]: Started ossec-monitord.  
  
john@john-virtual-machine:~/Desktop/wazuh-3.9.5$ systemctl status wazuh-  
manager
```

Figure 176: Starting Wazuh-Manager and proof

6.1.4.2. API

The second component is called Wazuh API and the installation is performed in the following matter. Likewise, a confirmation is performed by using another system type command.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -  
apt-get install -y nodejs  
npm config set user 0
```

Figure 177: Install Wazuh API with CURL tool part 1

```
curl -s -o install_api.sh https://raw.githubusercontent.com/wazuh/wazuh-api/v3.9.5/install_api.sh && bash ./install_api.sh download
```

Figure 178: Install Wazuh API with CURL tool part 2



```
john@john-virtual-machine:~/Desktop/wazuh-3.9.5$ systemctl status wazuh-api
● wazuh-api.service - Wazuh API daemon
   Loaded: loaded (/etc/systemd/system/wazuh-api.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2019-09-03 13:57:04 EEST; 12s ago
     Docs: https://documentation.wazuh.com/current/user-manual/api/index
```

Figure 179: Proof that Wazuh API is up and running

6.1.4.3. Agent on Windows machine

The Wazuh Agent is always installed on the device that the users are using and that needs to be monitored. So, it was connected on a Windows physical machine in order to send log information not only to Elasticsearch but also to the Wazuh Manager.

Analytically, the installation starts with the user's computer. To do so the agent program is installed, but without entering the registration key. The key must be generated through the machine where Wazuh manager is installed.

```
root@elkwazuh:/var/ossec/bin# ./manage_agents

*****
* Wazuh v3.11.1 Agent manager.          *
* The following options are available: *
*****

(A)dd an agent (A).
(E)xtract key for an agent (E).
(L)ist already added agents (L).
(R)emove an agent (R).
(Q)uit.
Choose your action: A,E,L,R or Q: E

Available agents:
ID: 002, Name: [REDACTED], IP: [REDACTED]
ID: 003, Name: [REDACTED], IP: [REDACTED]
ID: 004, Name: [REDACTED], IP: [REDACTED]
ID: 005, Name: [REDACTED], IP: [REDACTED]
ID: 006, Name: DESKTOP-8M9AG4T, IP: [REDACTED]
Provide the ID of the agent to extract the key (or '\q' to quit): 006

Agent key information for '006' is:
[REDACTED]
```

Figure 180: All agents with their IDs

Afterwards the generated key will be pasted to the agent so that it can be registered into the system.

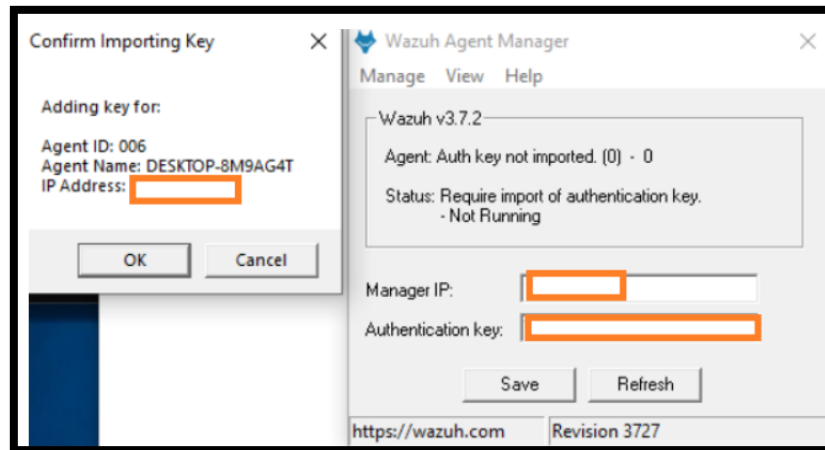


Figure 181: Importing authentication key and proof

Inside the Wazuh option in Kibana, it is confirmed that the registered agent is connected to the central system.

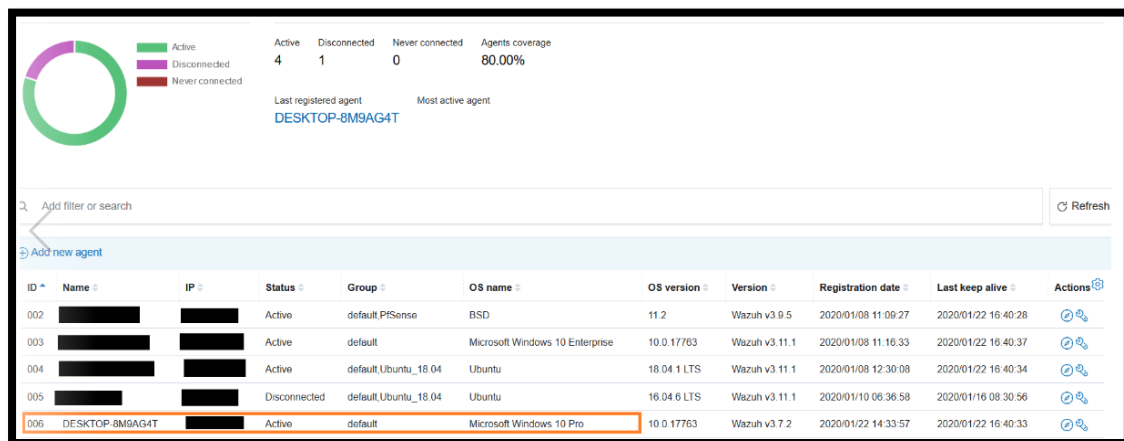


Figure 182: Proof that the new agent is added and active

An example for file integrity monitoring for this device will be done at this point. To do so, a proper query and index are used. Then by opening the registered log file, it is clear if a file has been altered or updated by comparing the checksums.

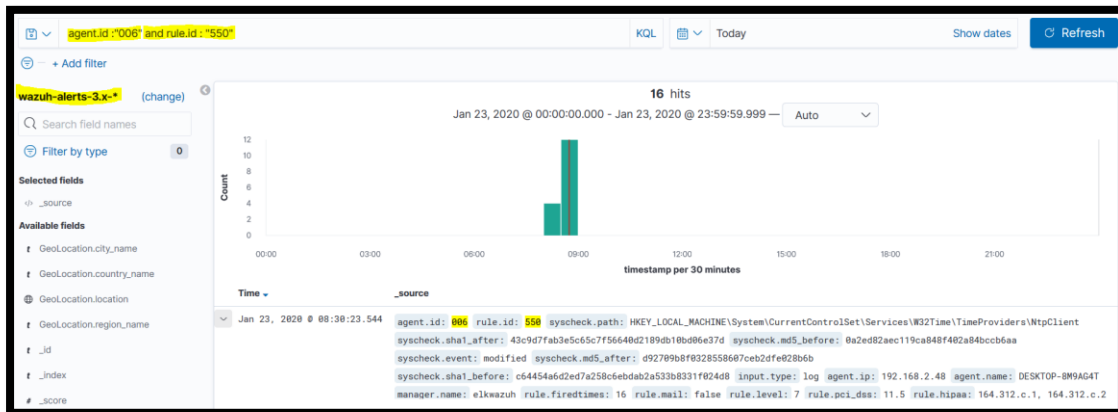


Figure 183: FIM example part 1

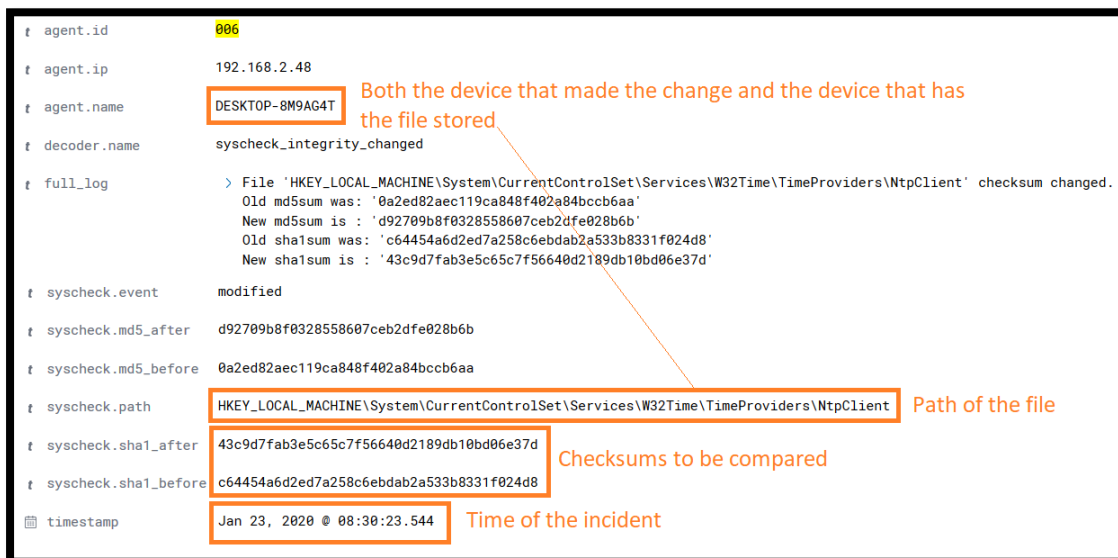


Figure 184: FIM example part 2

6.1.4.4. Custom Rules and Decoders

It is possible to modify the default rules and decoders from the Wazuh Ruleset and also to add new ones in order to increase Wazuh's detection capabilities. It is noteworthy that with this procedure, Wazuh might not be as efficient since the user decides what to filter but will be more robust since it is highly customisable.

To implement small changes "local_decoder.xml" and "local_rules.xml" files will be used. On the other hand, for larger scale changes or additions to the stock rules and



decoders, it is recommended for a new decoder/ rule file creation. These files can be found on the following directories.



Figure 185: Rules and decoders directories

It is important to state that some rules must be followed in order for Wazuh to run smoothly and with ease. Firstly, inside the “/rules” and “/decoders” directory it is already stated that only “local_decoders” and “local_rules” “.xml” files must be altered since any changes to another file may modify the behaviour of entire chains of rules. Secondly, the ID of the custom rule must exceed the number one hundred thousand since the rest of the IDs are reserved. Interfering and modifying those IDs equals with tampering the distributed rules files themselves. To be more precise any update of Wazuh will overwrite all the hard work that the expert has put into it. The third rule is for the user to maintain order in the rules. As the rule’s parser is loading the rules at start-up, it validates the existence of references inside the rules and groups. Therefore, if a reference of a rule is out of place (not loaded rule) the parser will collapse and fail its purpose.

To begin with, information must be decoded, so a new decoder is added to the aforementioned file.



```
<decoder name="example">
  <program_name>^example</program_name>
</decoder>

<decoder name="example">
  <parent>example</parent>
  <regex>User '(\w+)' logged from '(\d+.\d+.\d+.\d+)'</regex>
  <order>user, srcip</order>
</decoder>
```

Figure 186: Custom decoder example 1

Same applies for the rule that is needed.

```
<rule id="100010" level="0">
  <program_name>example</program_name>
  <description>User logged</description>
</rule>

</group> Must be inside a group
```

Figure 187: Custom rule example 1

Now that the new decoder and new rule are ready, they must be tested. This can be performed as already stated with the help of the “ossec-logtest” tool.

```
Dec 25 20:45:02 MyHost example[12345]: User [redacted] logged from [redacted]

**Phase 1: Completed pre-decoding.
  full event: 'Dec 25 20:45:02 MyHost example[12345]: User '[redacted]' logged from '[redacted]'
1.100'
  timestamp: 'Dec 25 20:45:02'
  hostname: 'MyHost'
  program name: 'example'
  log: 'User '[redacted]' logged from '[redacted]'
```

The decoder that was added earlier

```
**Phase 2: Completed decoding.
  decoder: 'example'
  dstuser: '[redacted]'
  srcip: '[redacted]'
```

The described rule from before

```
**Phase 3: Completed filtering (rules).
  Rule id: '100010'
  Level: '0'
  Description: 'User logged'
```

Figure 188: Testing the new ruleset from example 1



Some examples of log testing can be seen below.

```
root@elkwazuh:/var/ossec/bin# ./ossec-logtest
2020/01/27 06:29:10 ossec-testrule: INFO: Started (pid: 13767).
ossec-testrule: Type one log per line.

Dec 25 20:45:02 MyHost example[12345]: User 'admin' logged from '192.168.1.100'

**Phase 1: Completed pre-decoding.
  full event: 'Dec 25 20:45:02 MyHost example[12345]: User 'admin' logged from '192.168.1.100''
  timestamp: 'Dec 25 20:45:02'
  hostname: 'MyHost'
  program_name: 'example'
  log: 'User 'admin' logged from '192.168.1.100''

**Phase 2: Completed decoding.
  No decoder matched.
```

Figure 189: Testing random logs part 1

```
May 4 19:12:08 server custom-app: Failed login from '4.5.6.7' as testuser Log

**Phase 1: Completed pre-decoding.
  full event: 'May 4 19:12:08 server custom-app: Failed login from '4.5.6.7' as testuser'
  timestamp: '(null)'
  hostname: 'elkwazuh'
  program_name: '(null)'
  log: 'May 4 19:12:08 server custom-app: Failed login from '4.5.6.7' as testuser'

**Phase 2: Completed decoding.
  No decoder matched.

*Phase 3: Completed filtering (rules).
  Rule id: '2501'
  Level: '5'
  Description: 'syslog: User authentication failure.'
*Alert to be generated.
```

Figure 190: Testing random logs part 2



```
May 4 19:12:07 server custom-app: No error detected during startup!

**Phase 1: Completed pre-decoding.
  full event: 'May 4 19:12:07 server custom-app: No error detected during startup!'
  timestamp: '(null)'
  hostname: 'elkwazuh'
  program_name: '(null)'
  log: 'May 4 19:12:07 server custom-app: No error detected during startup!'

**Phase 2: Completed decoding.
  No decoder matched.

**Phase 3: Completed filtering (rules).
  Rule id: '1002'
  Level: '2'
  Description: 'Unknown problem somewhere in the system.'
**Alert to be generated.
```

Figure 191: Testing random logs part 3

6.1.4.5. Changing existing files

6.1.4.5.1. Rules

In order to preserve the changes, the changes are performed inside the “local_rules.xml” file. The example that will be explained refers to an SSH rule with an ID value of 5710.

The first step is to find the corresponding file to the rule that is to be changed. This file is inside the “/rules” directory and is named as follows.

```
root@elkwazuh:/var# cd ossec/ruleset/rules
root@elkwazuh:/var/ossec/ruleset/rules# ls
0010-rules_config.xml          0325-opensmtpd_rules.xml
0015-ossec_rules.xml          0330-sysmon_rules.xml
0016-wazuh_rules.xml          0335-unbound_rules.xml
0020-syslog_rules.xml         0340-puppet_rules.xml
0025-sendmail_rules.xml       0345-netscaler_rules.xml
0030-postfix_rules.xml        0350-amazon_rules.xml
0035-spamd_rules.xml          0360-serv-u_rules.xml
0040-imapd_rules.xml          0365-auditd_rules.xml
0045-mailscanner_rules.xml    0375-usb_rules.xml
0050-ms-exchange_rules.xml    0380-redis_rules.xml
0055-courier_rules.xml        0385-oscap_rules.xml
0060-firewall_rules.xml       0390-fortigate_rules.xml
0065-pix_rules.xml            0395-hp_rules.xml
0070-netscreenfw_rules.xml    0400-openvpn_rules.xml
0075-cisco-ios_rules.xml      0405-rsa-auth-manager_rules.xml
0080-sonicwall_rules.xml      0410-imperva_rules.xml
0085-pam_rules.xml            0415-sophos_rules.xml
0090-telnetd_rules.xml        0420-freeipa_rules.xml
0095-sshd_rules.xml           0425-cisco-estreamer_rules.xml
```

Figure 192: Locating the rule that needs to be changed



The next thing to do is inspect the “.xml” file with the “nano” command. Then any rule that is to be changed must be copied in order to be pasted inside the “local_rules.xml” file.

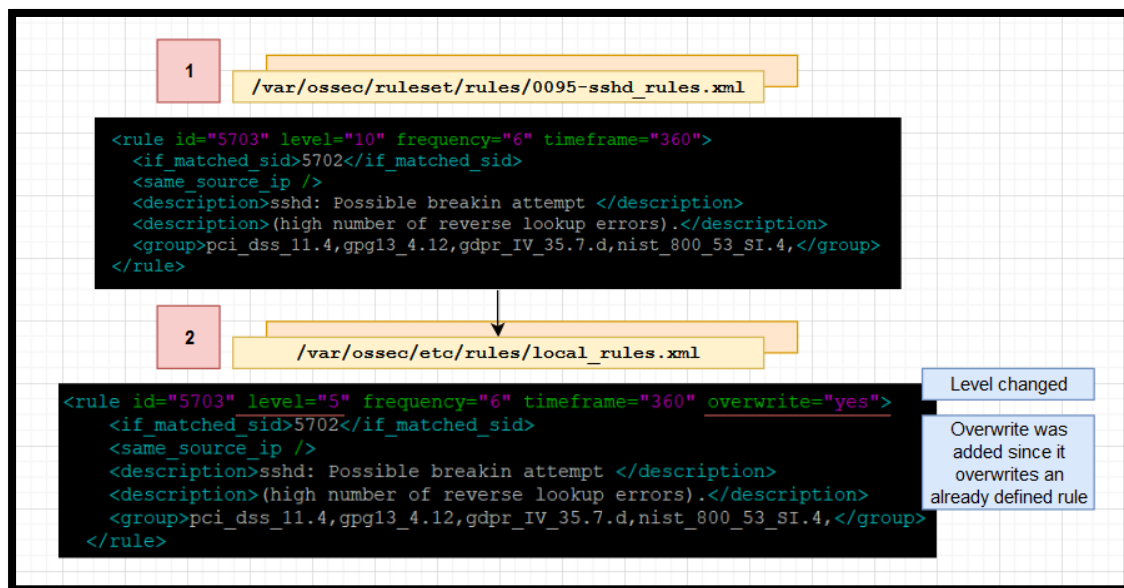


Figure 193: Copying and pasting the needed file (Rules) and changes

6.1.4.5.2. Decoders

Unlike how rules are changed, decoders cannot be overwritten in that way. This happens because there is no facility for this operation. However, it is still possible to do so.

Firstly, it must be issued that the decoder that is hosted by “0310-ssh_decoders.xml” is the one to be altered, for the purpose of this scenario. In order for this to happen it must be copied and pasted from the “/var/ossec/ruleset/decoders” directory to “/var/ossec/etc/decoders” directory. This is done so that changes will stay on the system even if an update is performed.



```
File to be copied
root@elkwazuh:/var/ossec/ruleset/decoders# cp 0310-ssh_decoders.xml /var/ossec/e
tc/decoders/

Proof
root@elkwazuh:/var/ossec/etc/decoders# ls
0310-ssh_decoders.xml local_decoder.xml local_decoder.xml.save
```

Figure 194: Copying and pasting the needed file (Decoders)

The next step is to exclude the original decoder file from Wazuh’s loading list. More specifically, this can be done by adding the `<decoder_exclude>` parameter inside the “ossec.conf” file.

```
root@elkwazuh:/var/ossec/etc# nano ossec.conf
<ruleset>
  <!-- Default ruleset -->
  <decoder_dir>ruleset/decoders</decoder_dir>
  <rule_dir>ruleset/rules</rule_dir>
  <rule_exclude>0215-policy_rules.xml</rule_exclude>
  <list>etc/lists/audit-keys</list>
  <list>etc/lists/amazon/aws-eventnames</list>
  <list>etc/lists/security-eventchannel</list>

  <!-- User-defined ruleset -->
  <decoder_dir>etc/decoders</decoder_dir>
  <rule_dir>etc/rules</rule_dir>

  <decoder_exclude>ruleset/decoders/0310-ssh_decoders.xml</decoder_exclude>
</ruleset>
```

Figure 195: Excluding a decoder

To conclude, inside the “/var/ossec/etc/decoders/0310-ssh_decoders.xml” file changes must be performed in order for this procedure to be successful.

6.1.4.6. Automating File Integrity Monitoring



File integrity monitoring and registry change monitoring can be performed by Wazuh's syscheck system. To do so, some configuration must be done on a windows-agent machine to monitor specific directories for changes. Afterwards it is only logical to perform some alterations in that place and observe the generated alerts.

Firstly, syscheck debug logging must be turned on. This will be done on the agent's machine, by creating a file with the name "local_internal_options.conf" inside the "C:\Program Files (x86)\ossec-agent\".

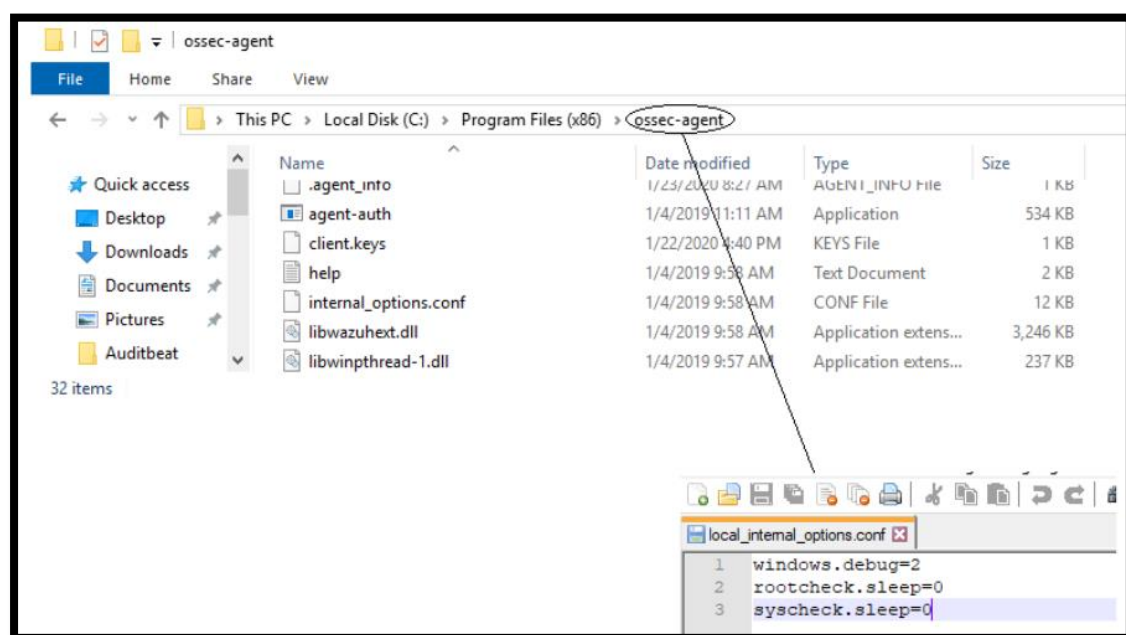


Figure 196: Syscheck debug logging enable

Afterwards two directories will be created with the help of cmd.

```
C:\>mkdir c:\TEST1  
C:\>mkdir c:\TEST2
```

Figure 197: Creating test directories



TEST1	Periodic scan 1. Start of Wazuh agent 2. Restart of Wazuh agent 3. Every 300 seconds afterwards	Changes will not include details of actual file content changes
TEST2	Real time monitoring	Changes will trigger an alert that includes the details of the actual changed text

Figure 198: Properties of the test files

Now it is time to enable syscheck FIM on the windows-agent on par with the aforementioned table. This is performed by going to the “view config” option inside the agent. There, the “<syscheck>” field must be replaced as it can be seen on the following screenshot.

```
<!-- File integrity monitoring -->  
<syscheck>  
  <disabled>no</disabled>  
  <scan_on_start>yes</scan_on_start>  
  <frequency>300</frequency>  
  <directories check_all="yes" realtime="yes" report_changes="yes">c:/TEST2</directories>  
  <directories check_all="yes">c:/TEST1</directories>  
</syscheck>
```

Figure 199: Enabling syscheck FIM on Windows agent

Now after restarting the agent, it is clear inside the “view logs” option that the new entries refer to the new syscheck monitoring of the two new directories.

```
ossec-agent: INFO: Monitoring directory: 'c:\test2'  
ossec-agent: INFO: Monitoring directory: 'c:\test1'
```

Figure 200: Proof of monitoring part 1

Lastly in order to check if it works, a new file will be created inside “TEST2” directory in order to see any changes inside the Kibana app.



Rule ID	Description	Level	Count
554	File added to the system.	5	2
553	File deleted.	7	1

Security events Integrity monitoring Inventory data

test2 test1

manager.name: agent.id: + Add filter

Figure 201: Proof of monitoring part 2

6.1.5. Beats (scenario)

Beats are used instead of Wazuh. They perform the same workload but are heavier for a device to handle. They are also heavily customisable from the user making them extremely robust. Beats are often preferred over Wazuh since Wazuh needs lots of modification when handling logs from different sources while Beats recognise them with ease.

6.1.5.1. Installing Metricbeat

Much like Wazuh, beats must be installed and configured locally, on a client's device. As far as Metricbeat goes, the installation on a Windows machine was straightforward. After downloading the file, it was moved in a specific directory and then the install ".ps1" was executed.



```
PS C:\Program Files> cd Metricbeat
PS C:\Program Files\Metricbeat> ls

Directory: C:\Program Files\Metricbeat

Mode                LastWriteTime         Length Name
----                -
d-----          11/26/2019   2:09 AM             kibana
d-----          11/26/2019   2:09 AM             module
d-----          11/26/2019   2:13 AM             modules.d
-a----          11/26/2019   2:13 AM              41 .build_hash.txt
-a----          11/26/2019   2:09 AM          584358 fields.yml
-a----          11/26/2019   2:13 AM           864 install-service-metricbeat.ps1
-a----          11/26/2019   1:26 AM          13675 LICENSE.txt
-a----          11/26/2019   2:12 AM         97596928 metricbeat.exe
-a----          11/26/2019   2:13 AM          72164 metricbeat.reference.yml
-a----          12/4/2019    10:49 AM           6180 metricbeat.yml
-a----          11/26/2019   1:27 AM         302830 NOTICE.txt
-a----          11/26/2019   2:13 AM           808 README.md
-a----          11/26/2019   2:13 AM           254 uninstall-service-metricbeat.ps1

PS C:\Program Files\Metricbeat> .\install-service-metricbeat.ps1
```

Figure 202: Metricbeat directory on Windows agent with installation

For the configuration process, inside the “modules.d” directory all modules can be found. These modules can be enabled accordingly in order to be used.

File Name	Last Write Time	File Type	Size
kibana-xpack.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
kubernetes.yml.disabled	11/26/2019 2:13 AM	DISABLED File	2 KB
kvm.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
logstash.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
logstash-xpack.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
memcached.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
mongodb.yml.disabled	11/26/2019 2:13 AM	DISABLED File	2 KB
mssql.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
munin.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
mysql.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
nats.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
nginx.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
oracle.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
php_fpm.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
postgresql.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
prometheus.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
rabbitmq.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
redis.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
statsd.yml.disabled	11/26/2019 2:13 AM	DISABLED File	1 KB
system.yml	12/4/2019 10:55 AM	YML File	1 KB

Figure 203: Configuring Metricbeat on Windows agent part 1

Also, it is important to notice that inside this directory a file called “system.yml” exists that is used for deciding the information that will be sent from that device.



```
- module: system
  period: 10s
  metricsets:
    - cpu
    #- load
    - memory
    - network
    - process
    - process_summary
    - socket_summary
    #- entropy
    - core
    #- diskio
    #- socket
```

Figure 204: Configuring Metricbeat on Windows agent part 2

Finally, in the “metricbeat.yml” the Elasticsearch host IP is filled.

```
#----- Elasticsearch output -----
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["192.168.1.1:9200"]

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  #username: "elastic"
  #password: "changeme"
```

Figure 205: Configuring Metricbeat on Windows agent part 3

6.1.5.2. Installing Auditbeat

Much like Metricbeat, Auditbeat follows the same procedure as far as installation goes.



```
PS C:\Program Files> cd .\Auditbeat\  
PS C:\Program Files\Auditbeat> .\install-service-auditbeat.ps1  
  
_GENUS          : 2  
_CLASS         : __PARAMETERS  
_SUPERCLASS    :  
_DYNASTY       : __PARAMETERS  
_RELPATH       :  
_PROPERTY_COUNT : 1  
_DERIVATION    : {}  
_SERVER        :  
_NAMESPACE     :  
_PATH          :  
ReturnValue    : 5  
PSComputerName :  
  
_GENUS          : 2  
_CLASS         : __PARAMETERS  
_SUPERCLASS    :  
_DYNASTY       : __PARAMETERS  
_RELPATH       :  
_PROPERTY_COUNT : 1  
_DERIVATION    : {}  
_SERVER        :  
_NAMESPACE     :  
_PATH          :  
ReturnValue    : 0  
PSComputerName :  
  
Status        : Stopped  
Name          : auditbeat  
DisplayName   : auditbeat
```

Figure 206: Auditbeat directory on Windows agent with installation

To continue though, the configuration is slightly different. Likewise, there modules as well here inside the "auditbeat.yml" file. Firstly, a module called "file_integrity". Some paths are included in this module, some of which are added by the user. Whenever something is altered in one of those directories a log will appear in the Elasticsearch. More specifically this occurs whenever a file is deleted, created or updated.

Following, a module called "system" which includes some data sets that supplies the system with valuable information about the host, the user, the stopped and started processes etc. One example, regarding the host, is how long he has been running while monitoring user login and logouts. In this module, any transactions related to software installation on the system, are also contained. It is highly noticeable, that the value called "state.period" determines how often the system will send the system data to the Elasticsearch. By default, every 12 hours these details are sent to the Elasticsearch. It is important to state that hours are represented with an "h" while minutes are represented with an "m".



```
##### Modules configuration #####
auditbeat.modules:
- module: file_integrity
  paths:
  - C:/windows
  - C:/windows/system32
  - C:/Program Files
  - C:/Program Files (x86)
- module: system
  datasets:
  - host # General host information, e.g. uptime, IPs
  - process # Started and stopped processes

# How often datasets send state updates with the
# current state of the system (e.g. all currently
# running processes, all open sockets).
state.period: 12h
```

Figure 207: Configuring Auditbeat on Windows agent part 1

Finally, in the same file, Elasticsearch host IP is filled.

```
----- Elasticsearch output -----
output.elasticsearch:
# Array of hosts to connect to.
hosts: ["192.168.1.1:9200"]

# Optional protocol and basic auth credentials.
#protocol: "https"
#username: "elastic"
#password: "changeme"
```

Figure 208: Configuring Auditbeat on Windows agent part 2

6.1.5.3. Installing Winlogbeat

The last beat used in this scenario is called “Winlogbeat”. The installation process is the same as before.



```
PS C:\Program Files> cd Winlogbeat
PS C:\Program Files\Winlogbeat> ls

Directory: C:\Program Files\Winlogbeat

Mode                LastWriteTime         Length Name
----                -
d-----          11/26/2019   2:14 AM             kibana
d-----          11/26/2019   2:14 AM             module
-a-----          11/26/2019   2:16 AM              41 .build_hash.txt
-a-----          11/26/2019   2:14 AM          111182 fields.yml
-a-----          11/26/2019   2:16 AM           864 install-service-winlogbeat.ps1
-a-----          11/26/2019   1:26 AM          13675 LICENSE.txt
-a-----          11/26/2019   1:27 AM          302830 NOTICE.txt
-a-----          11/26/2019   2:16 AM           825 README.md
-a-----          11/26/2019   2:16 AM           254 uninstall-service-winlogbeat.ps1
-a-----          11/26/2019   2:15 AM          50642432 winlogbeat.exe
-a-----          11/26/2019   2:14 AM          47291 winlogbeat.reference.yml
-a-----          12/4/2019   11:21 AM           6762 winlogbeat.yml

PS C:\Program Files\Winlogbeat> .\install-service-winlogbeat.ps1
```

Figure 209: Winlogbeat directory on Windows agent with installation

Then on the configuration, again inside the “winlogbeat.yml” the Elasticsearch host IP is filled.

```
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["192.168.1.1:9200"]

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  #username: "elastic"
  #password: "changeme"
```

Figure 210: Configuring Winlogbeat on Windows agent

To start all three services, the “Services” tab is opened to windows and by manually right clicking on them the user starts them. When booting the beat start-up is automatic. This beat is basically a tool for centralizing Windows event logs.

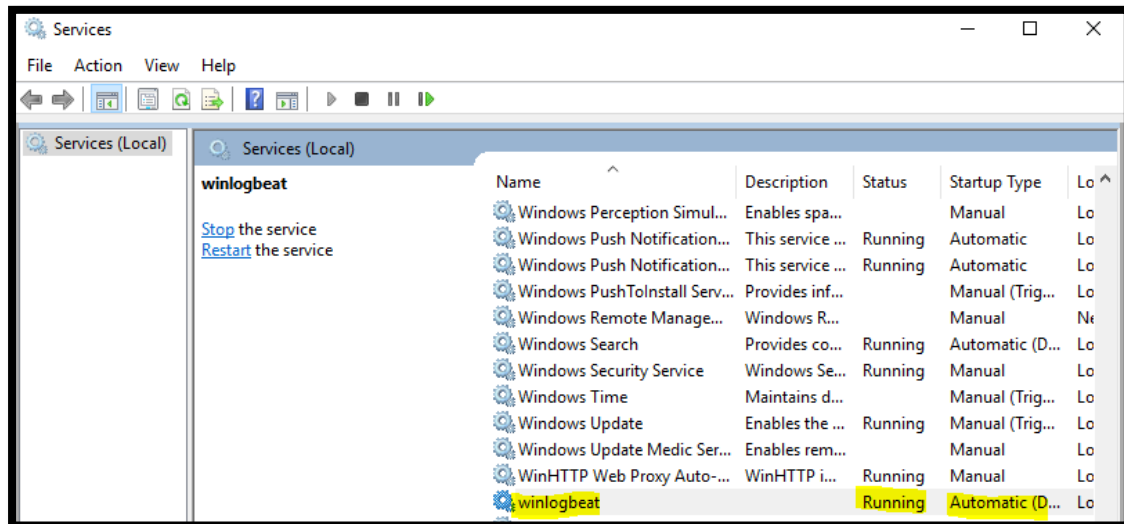


Figure 211: Winlogbeat runs as a service proof

6.1.5.4. Installing Packetbeat

The first step in installing Packetbeat includes the installation of a packet sniffing library, such as Npcap, which implements the libpcap libraries. Afterwards, the windows zip file must be downloaded for Packetbeat like all the aforementioned beats. Its contents will be placed in a folder called “Packetbeat” as shown below.

```
PS C:\Program Files> cd .\Packetbeat\  
PS C:\Program Files\Packetbeat> ls  
  
Directory: C:\Program Files\Packetbeat  
  
Mode                LastWriteTime         Length Name  
----                -  
d-----           12/16/2019 11:40 PM             kibana  
-a----           12/16/2019 11:45 PM              41 .build_hash.txt  
-a----           12/16/2019 11:40 PM          165150 fields.yml  
-a----           12/16/2019 11:45 PM           864 install-service-packetbeat.ps1  
-a----           12/16/2019 11:15 PM          13675 LICENSE.txt  
-a----           12/16/2019 11:16 PM          302830 NOTICE.txt  
-a----           12/16/2019 11:42 PM        53080576 packetbeat.exe  
-a----           12/16/2019 11:45 PM          65650 packetbeat.reference.yml  
-a----           12/16/2019 11:45 PM           8769 packetbeat.yml  
-a----           12/16/2019 11:45 PM           832 README.md  
-a----           12/16/2019 11:45 PM           254 uninstall-service-packetbeat.ps1  
  
PS C:\Program Files\Packetbeat> .\install-service-packetbeat.ps1
```

Figure 212: Packetbeat directory on Windows agent with installation



Finally, in the configuration file called “packetbeat.yml”, the host IP of Elasticsearch is filled.

```
----- Elasticsearch output -----
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["192.168.1.10:9200"]

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  #username: "elastic"
  #password: "changeme"
```

Figure 213: Configuring Packetbeat on Windows agent

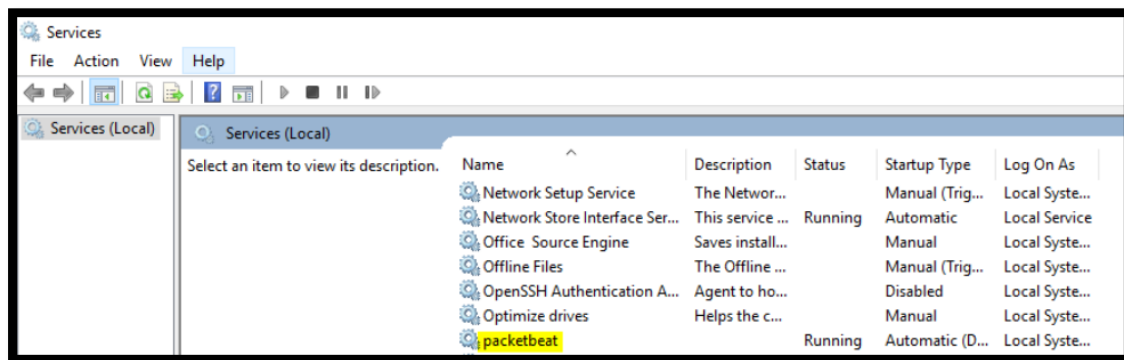


Figure 214: Packetbeat runs as a service proof

6.1.5.5. Visualization

METRICBEAT VISUALIZATION



Figure 215: Metricbeat visualization part 1

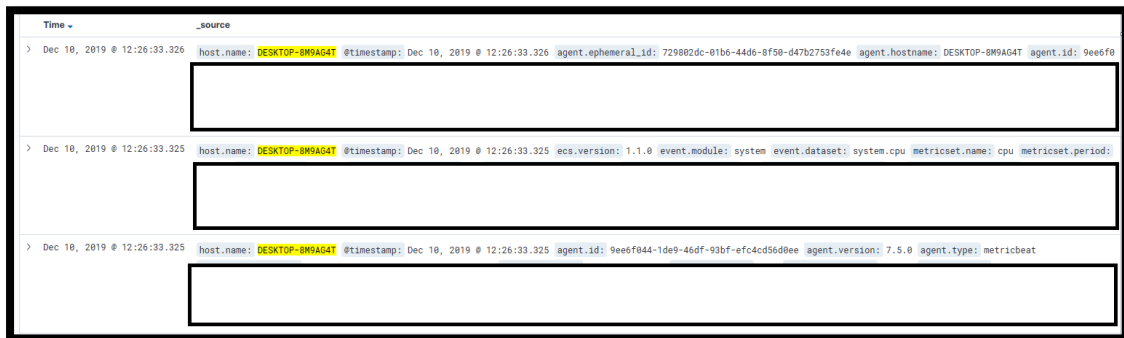


Figure 216: Metricbeat visualization part 2

AUDITBEAT VISUALIZATION

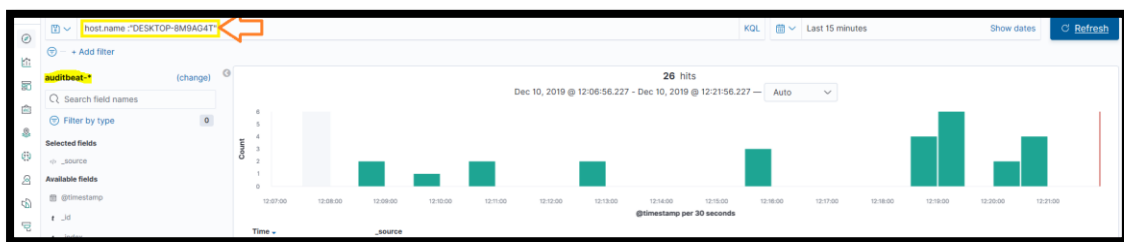


Figure 217: Auditbeat visualization part 1

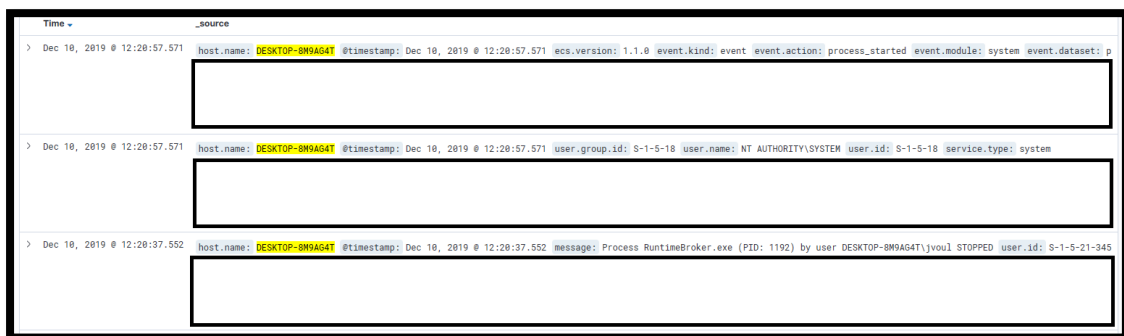


Figure 218: Auditbeat visualization part 2

WINLOGBEAT VISUALIZATION

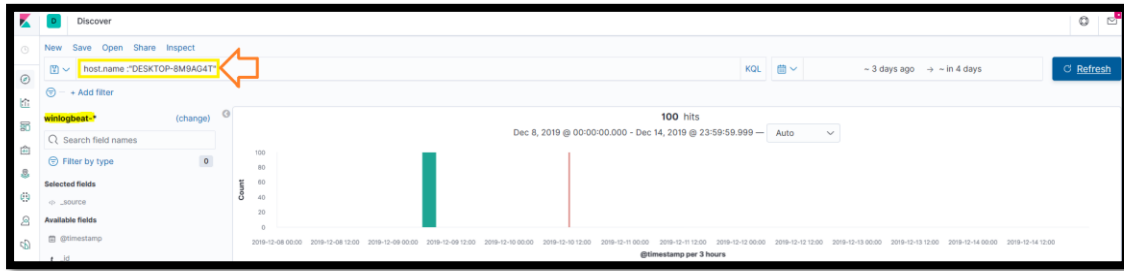


Figure 219: Winlogbeat visualization part 1

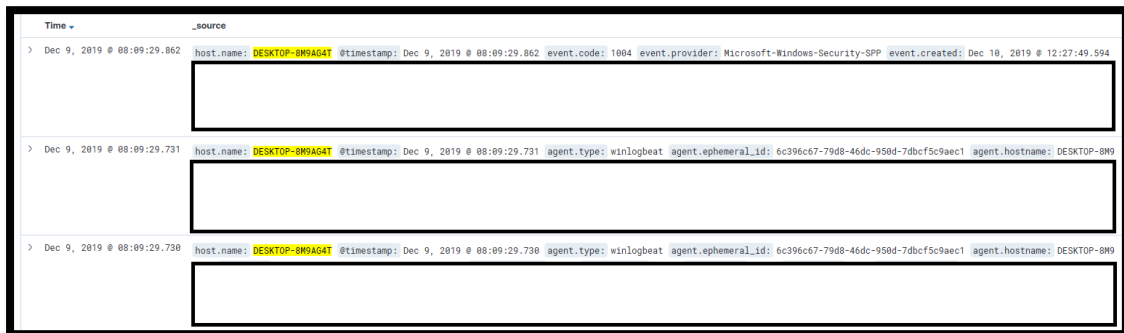


Figure 220: Winlogbeat visualization part 2

PACKETBEAT VISUALIZATION

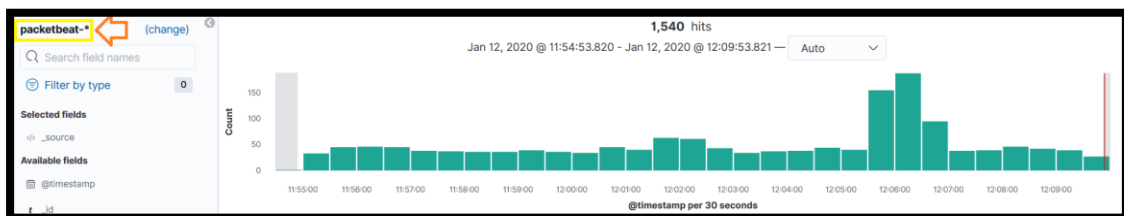


Figure 221: Packetbeat visualization part 1

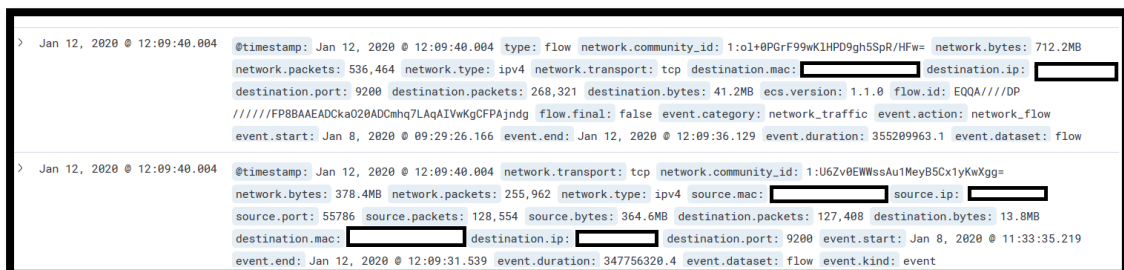


Figure 222: Packetbeat visualization part 2



6.4. Elastic SIEM

6.4.1. Security Information and Event Management

In the field of IT security, a SIEM is basically the combination of Security Information Management (SIM) and Security Event Management (SEM). These acronyms have sometimes been used interchangeably, but generally refer to the different primary focus of products. Firstly, a SIM is a system that stores (for long periods of time), analyses and reports log data. Afterwards, a SEM is a system that monitors (in real time), monitors and notifies for any security related event.

As already mentioned, a SIEM combines these two and provides real time analysis of security alerts generated by network hardware and applications.

6.4.2. Introduction

Elastic SIEM, which is offered for free as part of the default distribution, offers security practitioners features such as new data integrations, intuitive ways to triage events, network-related security event analysis, interactive and composable timeline event viewers for collaborative threat hunting. Basically, it provides a solution for the users to achieve security analysis. It is very important to understand that it is a product that integrates all technologies like docker containers, cloud, firewalls etc. into a single ecosystem. It enables analysis of two important types of data which is host-related and network-related. The most important advantage of Elastic SIEM is that it is built to be collaborative. This means that it can be highly interactive- for the user to just move things around and play with the data.

In order to introduce network data into Elasticsearch it is crucial to use some products and technologies like Packetbeat or Filebeat. Through Packetbeat, DNS information and flows can be transferred to Elasticsearch, and through Filebeat IDS, IPS or even Firewall modules information.

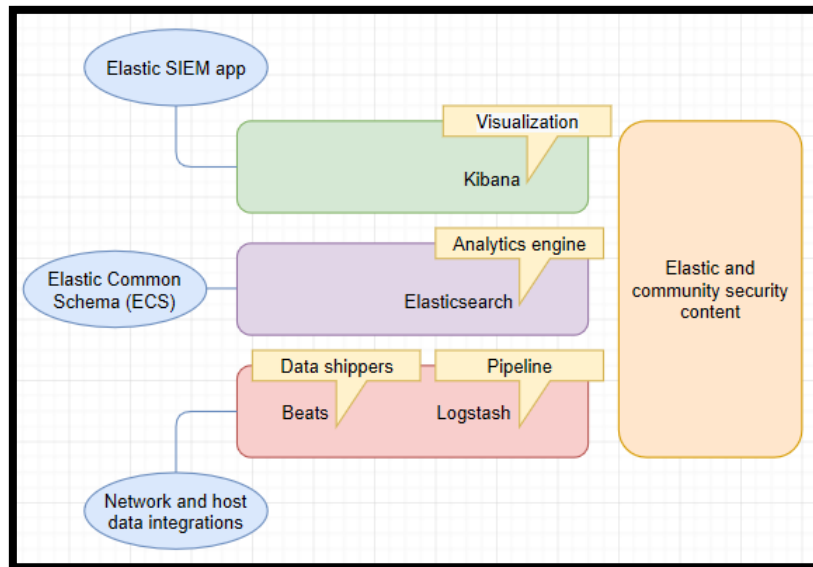


Figure 223: Elastic SIEM structure

The figure above is in the same context of SIEM since it has ECS included which is a very important part of the solution itself. A huge priority also, is the interaction with the elastic community to make sure that this product is heading in the right direction. So, there are many types of data shippers (Beats), as mentioned before multiple times, that are used for ingesting host and network security events data into the SIEM app.

Furthermore, when multiple data from different sources is ingested, it must be ensured that there is correlation among those data sources. This is performed by using the Elastic Common Schema; a tool that basically normalizes data if there is a common set of fields. It also provides some best practices as far the field and mapping defining goes.

Finally, ECS must be explained a little more thoroughly. It is an open source specification, developed with support from the Elastic user community. ECS defines a common set of fields to be used when storing event data in Elasticsearch, such as logs and metrics. The goal of ECS is to enable and encourage users of Elasticsearch to normalize their event data, so that they can better analyze, visualize, and correlate the data represent in their events.

6.4.3. Components



The SIEM has different components although it is a simple application as it is. Four different options are offered. Firstly, there is an option called ‘‘Overview’’, which gives information about the type of data that are to be processed. The second option is called ‘‘Hosts’’ which gives basically information about the data coming from the host. Afterwards, another option is called ‘‘Network’’, which is the same with the ‘‘Hosts’’ option but targets anything related to network activities. The next option, called ‘‘Timelines’’, is probably the defining part between Elastic SIEM and every other SIEM out there. The user, through this option, can generate and provide data with queries. It is important to state that the last two options are either not needed for this case scenario or in a paid package.

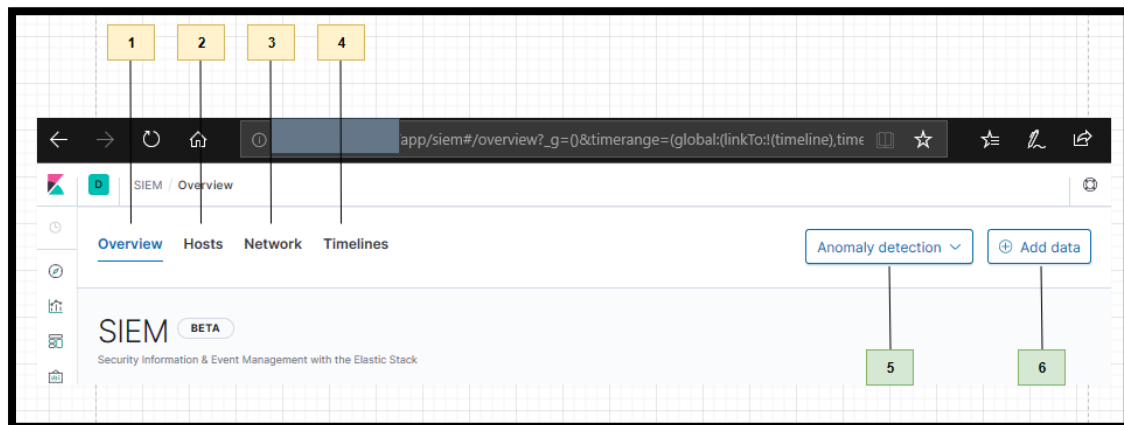


Figure 224: Elastic SIEM components

6.4.3.1. Overview

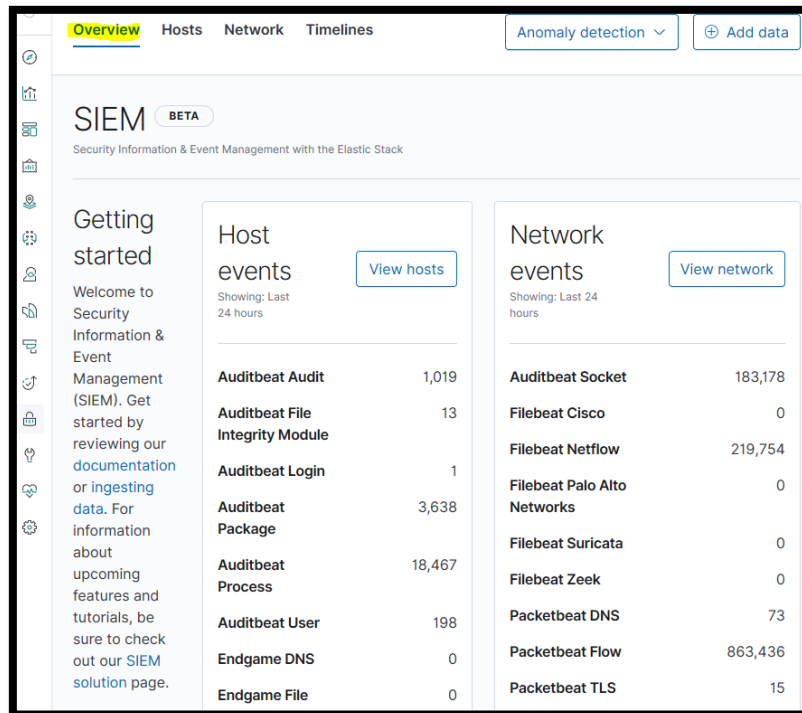


Figure 225: Elastic SIEM Overview visualization

In the Overview tab, a bird's-eye overview of the data volume in the past 24 hours is presented. This data is collected through a variety of different sources. The volume of the data is highly correlated with the number of machines connected to the SIEM through the beats.

6.4.3.2. Hosts

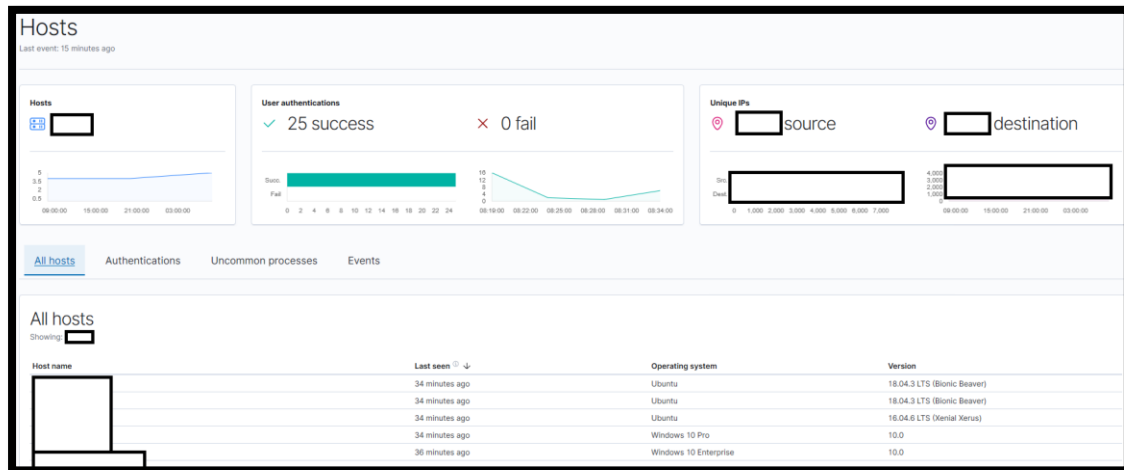


Figure 226: Elastic SIEM Hosts visualization

The Hosts tab basically presents information for all or a subset of the hosts inside this cluster. If there is a search query to find within this security environment and call out any specific behaviour that needs to be looked at (as the uncommon processes and events).

The first feature of the Elastic SIEM application is a very user friendly yet powerful query building capability. It is highly notable, that a security professional must always search along his investigation journey. This means that he has to navigate within the collected security events without any compromises on his query performance. So, Elastic SIEM wants to free the user from canned reports and dashboards without taking a tremendous amount of time to compose the query.

As the user looks at different places within the same application, he can drag and drop a UI component into the timeline UI widget. This will open a window behind the scenes with info sent by a query to the backend Elasticsearch cluster. It is important to state that this particular query will be matched with security events which will be later on presented on the window.

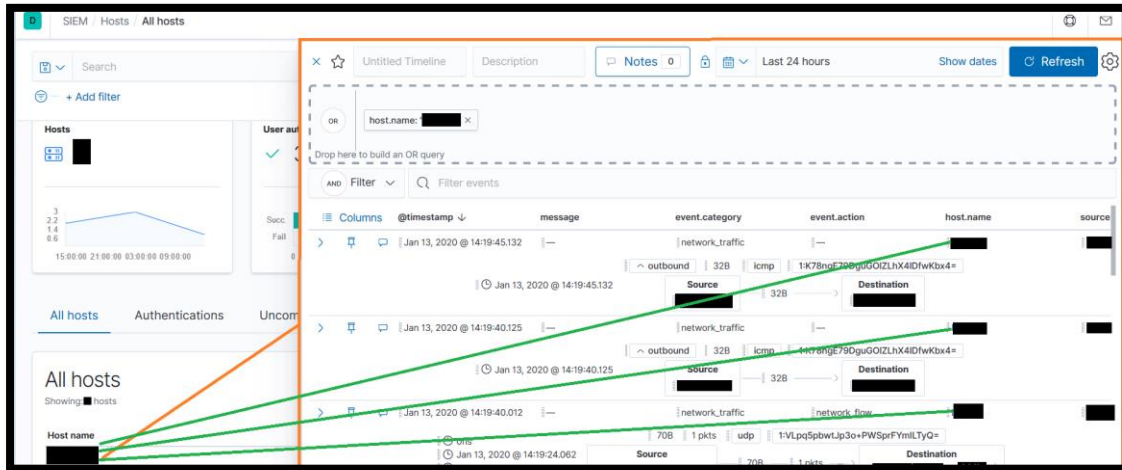


Figure 227: Creating timelines example part 1

These events as seen are a lot. So, in order to narrow down the results, the query can be enhanced with another drag and drop option.

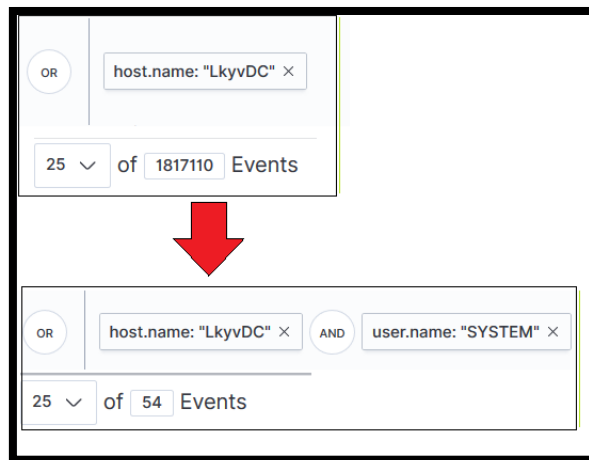


Figure 228: Creating timelines example part 2

6.4.3.3. Network

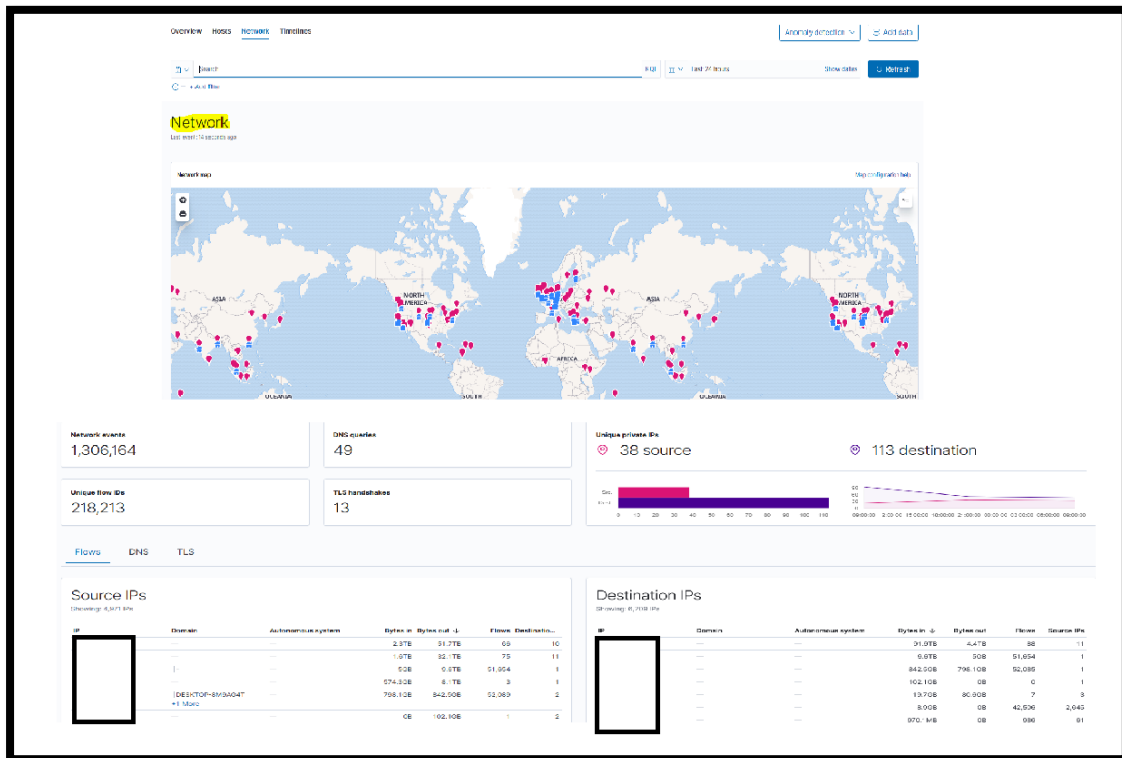


Figure 229: Elastic SIEM Networks visualization

Similarly, for the Network tab, the same application presents all kinds of information related to the network entities in the security environment. Much like Hosts specific entities or behaviours can be checked out, such as the top talkers and top DNS domains.

6.4.3.4. Timelines

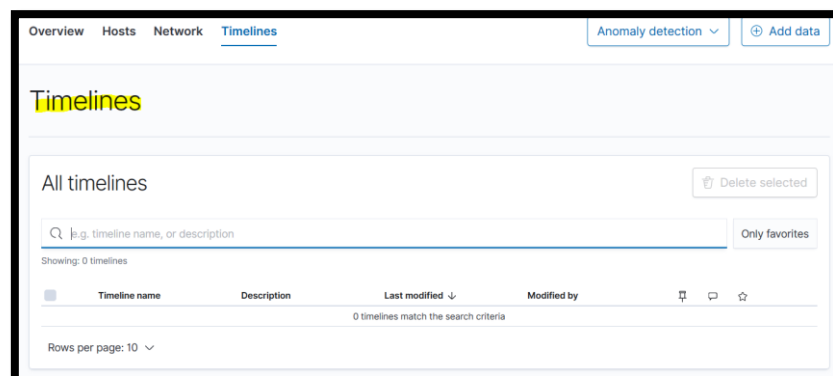


Figure 230: Elastic SIEM Timelines visualization



The Timelines tab on the other hand is a repository of all of the working threads by the investigators and responders where they basically capture and navigate in arbitrary directions of their investigation effort. This happens by either collaborate with their peers or pass that on to subsequent investigators and responders.



Figure 231: Using a custom timeline part 1

After creating a query as aforementioned with the help of the Timelines UI, some suspicious activity is already -hypothetically- identified. Therefore, the concluded events are on a certain timeline that the user can easily rename it and add notes and descriptions. This will help the investigator and responder to pick it up as it is and refine it to take further actions. One last thing to note is that all those timelines can be associated with Kibana spaces, so that different teams and people can get access to. Therefore, this feature is great to facilitate collaboration among peer analysts and with Kibana spaces everything can be isolated and secured.

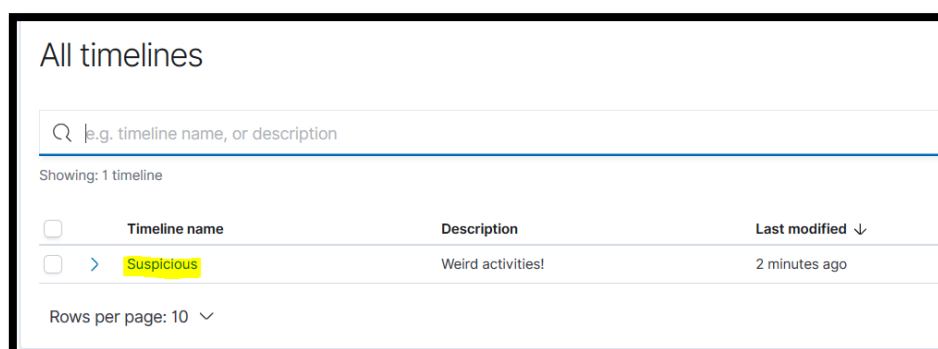


Figure 232: Using a custom timeline part 2



6.5. Elastalert

6.5.1. Installing and configuration

At this point, a way to alert the user about certain events must be used. X-Pack offers an alerting feature but is on paid subscription. Subsequently, an open source tool called Elastalert will be used.

Apart from Elasticsearch and Kibana a client virtual machine will be used as well to send syslog logs through Filebeat. Furthermore, Elastalert will query Elasticsearch and depending on the configuration if it finds a matching event or an inconsistency in data in the Elasticsearch, it will send an alert to the Slack channel/ chosen email. It is important to notice that Elastalert is not installed on the client machine, although it can be installed on any machine that has access to the Elasticsearch. Elastalert, is a standalone tool that doesn't have to be in the ELK server or in the client virtual/ physical machine. Simplified, it is an alerting tool that automatically queries and analyses log data in Elasticsearch clusters.

The installation procedure consists of some prerequisites, as installing the “pip” tool and python3.6. To continue, the latest released version of Elastalert is installed with “pip”.

```
$ pip install elastalert  
  
$ pip install "setuptools>=11.3"  
$ python setup.py install
```

Figure 233: Installing Elastalert

After enabling Elastalert as a service, indexes must be created for the tool to be connected with Elasticsearch and visualized by Kibana.



```
$ elasticsearch-create-index
```

Figure 234: Creating Elastaalert's indices

These indexes are “elastalert_status*”, “elastalert_status_error*”, “elastalert_status_silence*” and “elastalert_status_status*”, as shown below.



Figure 235: elastalert_status* visualization



Figure 236: elastalert_status_error* visualization



Figure 237: elastalert_status_silence* visualization



Figure 238: elastalert_status_status* visualization

Each of these aforementioned events, can be expanded so that the user can see more information about it.



Expanded document

Table JSON

@timestamp	Jan 9, 2020 @ 14:01:33.565	
t _id	7pouim8Br2QBYPs8Yyt_	Unique event ID
t _index	elastalert_status_status	
# _score	-	
t _type	_doc	
@endtime	Jan 9, 2020 @ 14:01:32.770	
# hits	263	How many times it has matched the event
# matches	263	
t rule_name	blacklist	
@starttime	Jan 9, 2020 @ 13:53:01.934	
# time_taken	0.795	

Figure 239: Examining a document with elastalert_status_status* index

The configuration part consists only of a file called “config.yaml” which is configured as presented below.



```
# This is the folder that contains the rule yaml files
# Any .yaml file will be loaded as a rule
rules_folder: example_rules where the rules are saved

# How often ElastAlert will query Elasticsearch
# The unit can be anything from weeks to seconds
run_every:
  minutes: 1 frequency that is used

# ElastAlert will buffer results from the most recent
# period of time, in case some log sources are not in real time
buffer_time:
  minutes: 15

# The Elasticsearch hostname for metadata writeback
# Note that every rule can have its own Elasticsearch host
es_host:  Elasticsearch Host IP

# The Elasticsearch port
es_port:  Elasticsearch Host Port

# The index on es host which is used for metadata storage
# This can be a unmapped index, but it is recommended that you run
# elastalert-create-index to set a mapping
writeback_index: elastalert_status The indexes that are used for
writeback_alias: elastalert_alerts Elastalert where data are stored

# If an alert fails for some reason, ElastAlert will retry
# sending the alert until this time period has elapsed
alert_time_limit:
  days: 2
```

Figure 240: Elastalert configuration

Lastly in order to start the Elastalert service the following command must be used, where “verbose” is an option that provides additional display info level messages.

```
elastalert --start NOW --verbose
```

Figure 241: Starting Elastalert

6.5.2. Elastalert rules

6.5.2.1. Arguments

Elastalert, is comprised of rules that are saved inside a folder called “example_rules”. These easy-to-write rules are the main component used for the alert creating. The specific path for the rules, is shown in the screenshot below.



```
root@elksiem2:~/elastalert/example_rules# ls
blacklist1.yaml  frequency1.yaml
```

Figure 242: Elastalert rules directory

They are used for many different reasons, like blacklisting and event spiking. All these rules are described inside of “.txt” files and have specific required arguments.

To be more precise in the following table all required arguments are analysed.

es_host	IP of the host where the ELK stack is running on
es_port	The port that Elasticsearch is listening on
num_events	The number of events that will trigger the event
timeframe	The period of time that the number of events should be done
filter	Filter, query that specifies what to match in order to fetch from a specific rule
type	The rule type (e.g. spike)
alert	Where to send the notification about the alert (slack, email etc.)
index	The Elasticsearch index that will be used to send data to it

Every rule can run independently with the following command.

```
root@elksiem2:~/elastalert# python3.6 -m elastalert.elastalert --rule example_rules/frequency1.yaml
1 rules loaded
```

Figure 243: Running a rule independently



But since it is practically impossible to depend on multiple sessions of terminal, a file is created that will automatically run every rule on the background.

```
elastalert.egg-info LICENSE README.md setup.cfg tests
example_rules Makefile requirements-dev.txt setup.py tox.ini
get-pip.py pytest.ini requirements.txt supervisord.conf.example zdaemon.conf
```

Figure 244: File that runs all rules in the background part 1

```
<runner>
  program python3.6 -m elastalert.elastalert --conf config.yaml
  socket-name /tmp/elastalert.zdsocket
  forever true
</runner>
```

Figure 245: File that runs all rules in the background part 2

```
root@elksiem2:~/elastalert# zdaemon -C zdaemon.conf start
.
daemon process started, pid=10110
```

Figure 246: File that runs all rules in the background part 3

6.5.2.2. Example: Frequency – Winlogbeat

For this example, a frequency rule will be created. This rule will alert on a specific Slack chat when a machine that has Winlogbeat installed, types the command “ipconfig” on the command prompt.



```
name: frequency1
type: frequency
index: winlogbeat-*
num_events: 1
timeframe:
  minutes: 30
filter:
- term:
    winlog.event_data.CommandLine: "ipconfig"
# (Required)
# The alert is use when a match is found
alert:
#- "email"
- "slack"
slack:
slack_webhook_url: "https://hooks.slack.com/services/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"
# (required, email specific)
# a list of email addresses to send alerts to
#email:
#- "elastalert@example.com"
```

Figure 247: Elastalert example 1 configuration file

For this to work, Winlogbeat must be installed on the client’s machine so that it can collect logs from the command prompt. This is performed by firstly navigating to the Group Policy Editor. There, by following the path:

Local Computer Policy -> Computer Configuration -> Windows Settings -> Security Settings -> Advanced Audit Policy Configuration -> System Audit Policies -> Detailed Tacking

Figure 248: Group Policy Editor changes part 1

”Audit Process Creation” is selected and then a box appears where the “Success” and the “Failure” radio buttons must be checked.

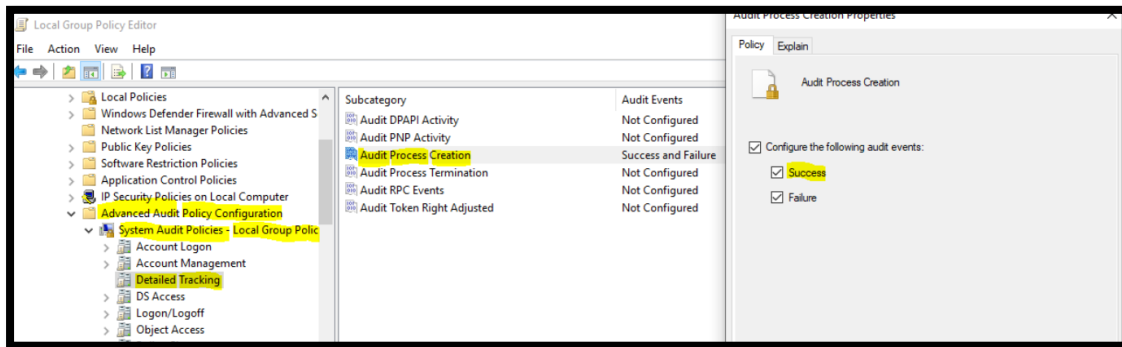


Figure 249: Group Policy Editor changes part 2

Afterwards, the command line process creation must be enabled. This is succeeded by navigating to another directory of the Group Policy Editor. The path that is used is the following:

Local Computer Policy -> Computer Configuration -> Administrative Templates -> System -> Audit Process Creation -> Include command line in process creation events

Figure 250: Group Policy Editor changes part 3

At this option the “Enabled” field must be selected.

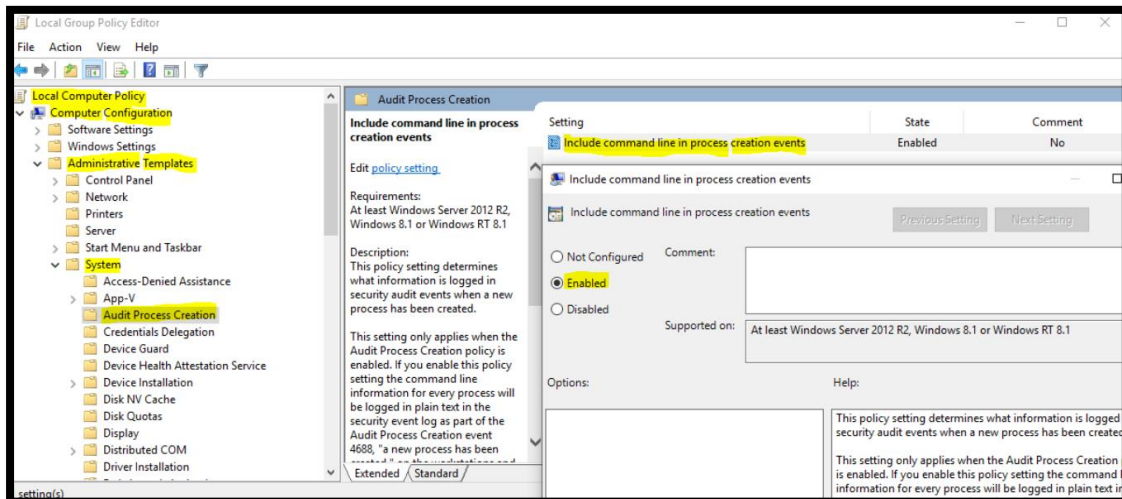


Figure 251: Group Policy Editor changes part 4

In order to make sure that logs are gathered for the command prompt by Winlogbeat, the service is restarted, and the group policies are updated through the cmd by using the “gpupdate” command. Now on the Event Viewer, in the Security tab, after opening cmd and pressing “ipconfig” the following events are collected.

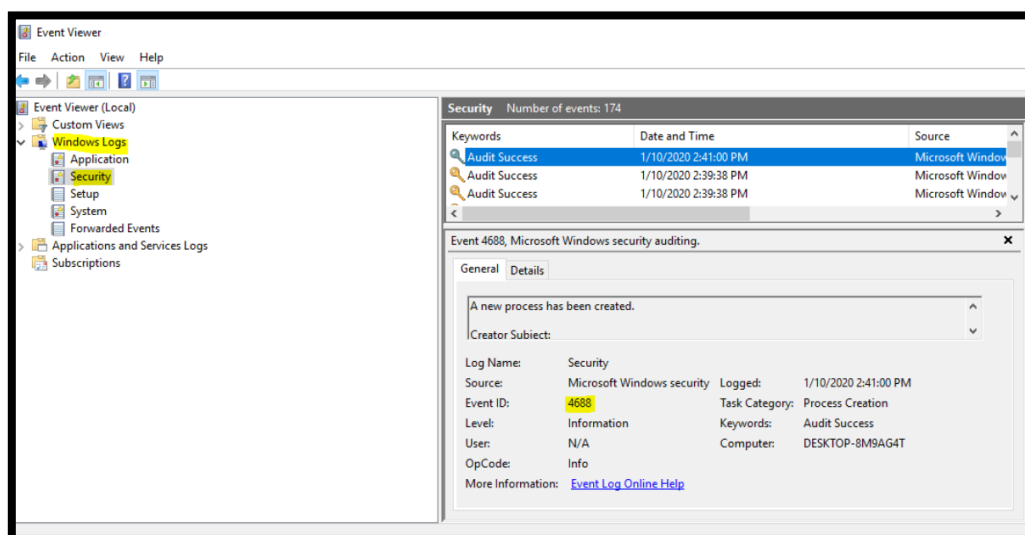


Figure 252: Group Policy Editor proof part 1

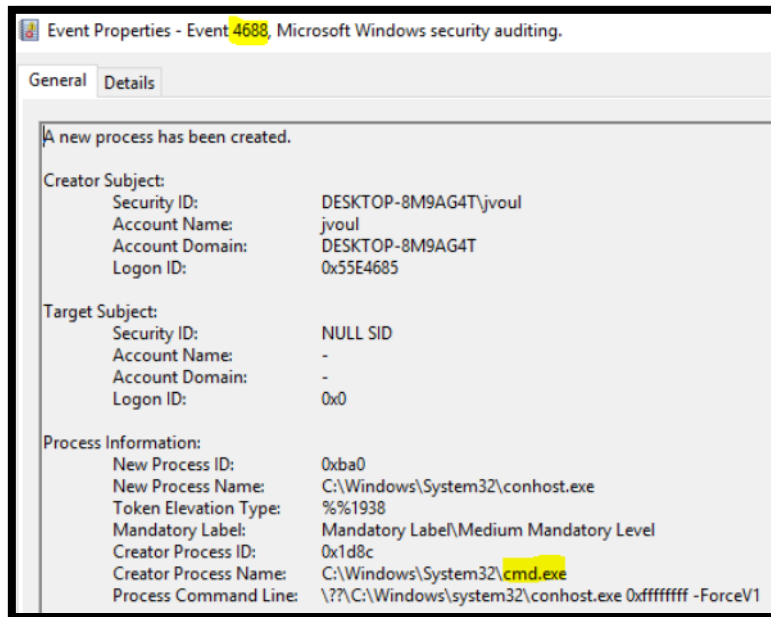


Figure 253: Group Policy Editor proof part 2

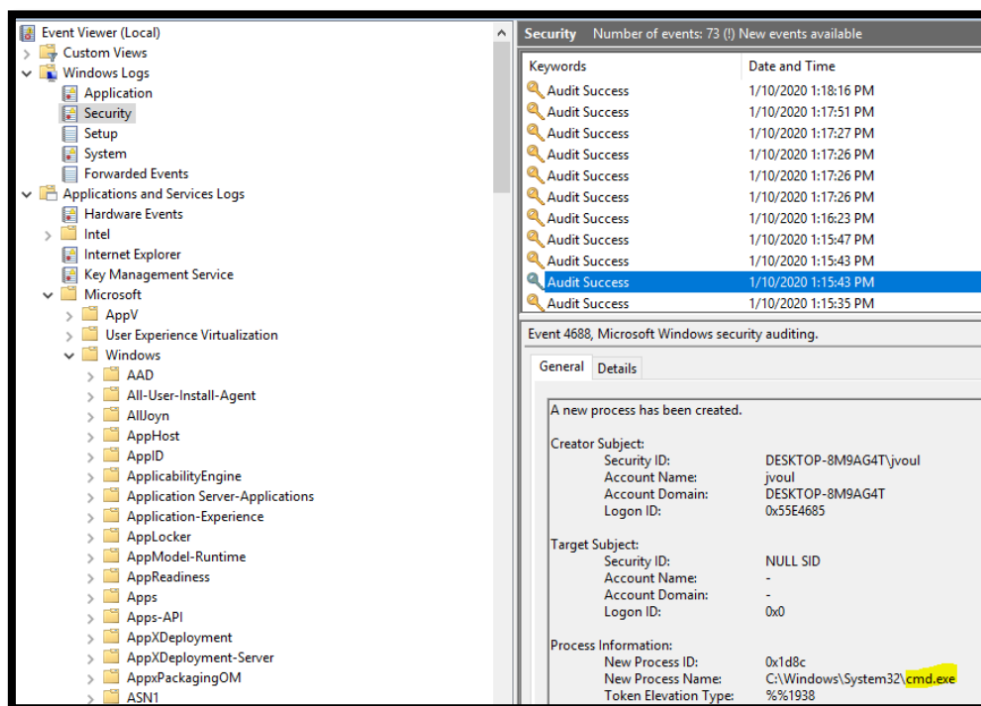


Figure 254: Group Policy Editor proof part 3

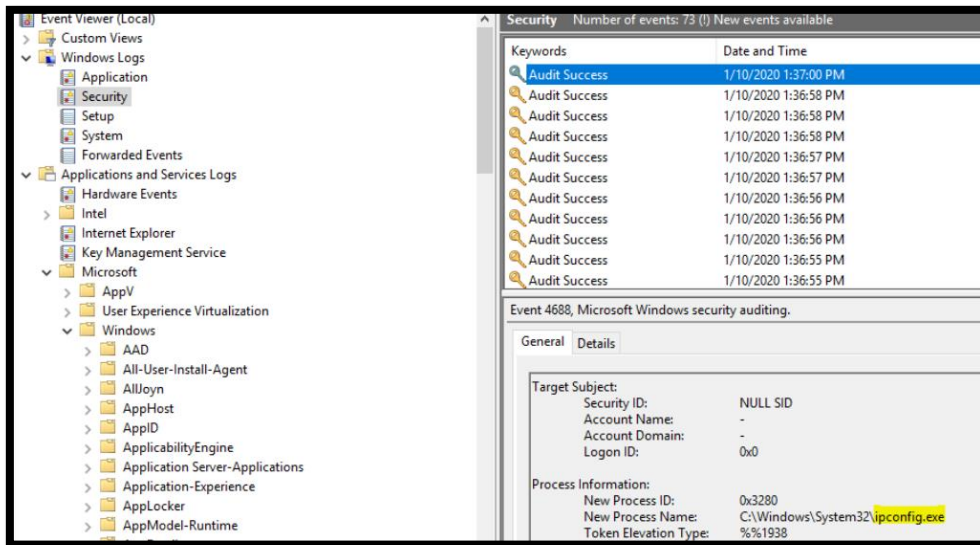


Figure 255: Group Policy Editor proof part 4

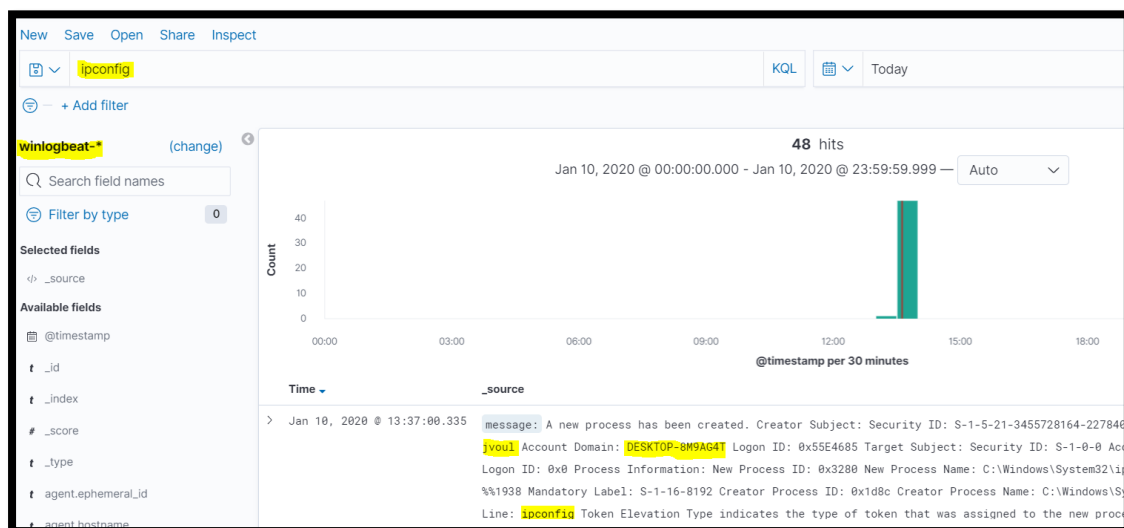


Figure 256: Proof that the rule works (Example 1) part 1



```
alerts - Jan 12th View in channel
elastalert (APR) 7:37 PM
frequency1
frequency1
At least 1 events occurred between 2020-01-12 17:06 UTC and 2020-01-12 17:36 UTC
@timestamp: 2020-01-12T17:36:58.113Z
_id: 4qvUmm8Br2QBYPs8Is9f
index: winlogbeat-7.5.0-2020.01.10-000001
_type: _doc
agent: {
  "ephemeral_id": "51d9213d-8161-481d-87de-9868e7da01c4",
  "hostname": "...",
  "id": "e5a55bd6-b8b8-41ee-a3ee-991c96de285b",
  "type": "winlogbeat",
  "version": "7.5.0"
}
Creator Subject:
Security ID: S-1-5-21-3455728164-2278404548-1401613097-1001
Account Name: PC account name
Account Domain: Name of the device
Logon ID: 0x832E53A
Process Information:
New Process ID: 0x27e0
New Process Name: C:\Windows\System32\ipconfig.exe
Token Elevation Type: %1938
Mandatory Label: S-1-16-8192
Creator Process ID: 0x1104
Creator Process Name: C:\Windows\System32\cmd.exe
Process Command Line: ipconfig
```

Figure 257: Proof that the rule works (Example 1) part 2

6.5.2.3. Example: Frequency – Packetbeat

Another use case example would be if a user visited a restricted site for which the security expert would want to be notified for. The site that will be used for this example will be “youtube.com”. So, every time a device navigates to the youtube.com domain, the expert will be alerted. This procedure will be performed by using a device with Packetbeat installed and running.

```
type: frequency
# (Required)
# Index to search wildcard supported
index: packetbeat-*
# (Required, frequency specific)
# Alert when this many documents matching the query occur within a timeframe
num_events: 5
# (Required, frequency specific)
# num_events must occur within this amount of time to trigger an alert
timeframe:
  minutes: 30
# (Required)
# A list of Elasticsearch filters used for find events
# These filters are joined with AND and nested in a filtered query
# For more info: http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/query$
filter:
  term:
    dns.question.registered_domain: " .com"
# (Required)
# The alert is use when a match is found
alert:
  #- "email"
  - "slack"
slack:
  slack webhook url: "https://hooks.slack.com/services/T2KF4P7D3/BS75PLJ22/F15aXcad3UbnC0qgIne$
# (required, email specific)
# a list of email addresses to send alerts to
email:
  #- "elastalert@example.com"
```

Figure 258: Elastalert example 2 configuration file



Figure 259: Proof that the rule works (Example 2) part 1

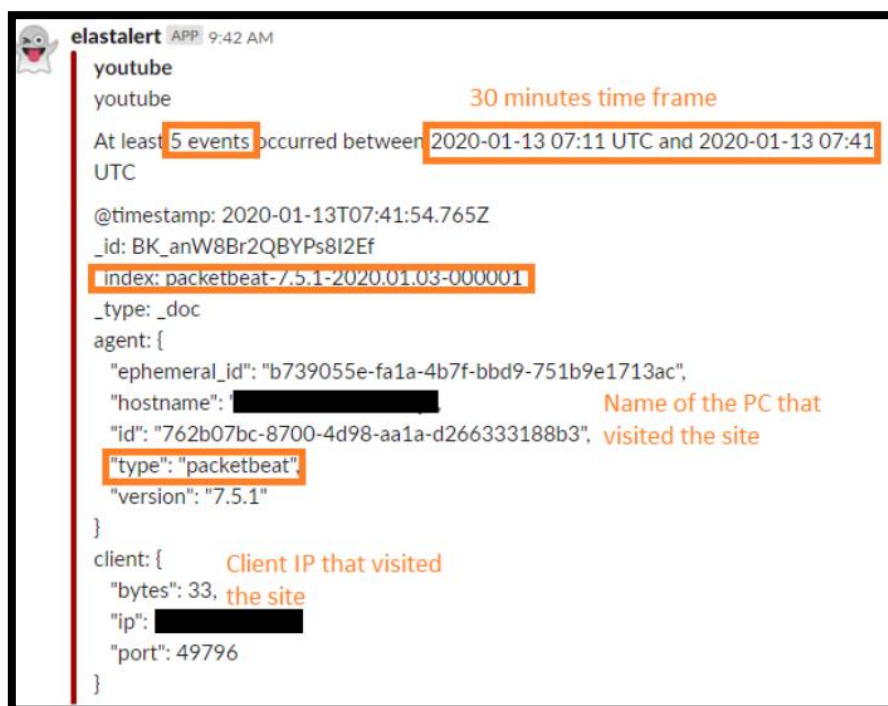


Figure 260: Proof that the rule works (Example 2) part 2

6.5.2.4. Example: Whitelist - Packetbeat

A Whitelist rule, as the name clearly states uses the basic concept of whitelisting. It is the practice of explicitly allowing some identified entities access to a particular IP, in



our case. This rule, much like the next example, requires three additional options; “compare_key”, “whitelist” and “ignore_null”. The first argument specifies the information that will be compared in order for the event to be triggered. Afterwards, the “whitelist” option lists all whitelisted values or a list of paths to flat files. Lastly, the “ignore_null” variable is used so that events without a “compare_key” field will not match.

```
name: whitelist
type: whitelist
index: packetbeat-*
compare_key: source.ip
ignore_null: true
whitelist:
- "!file /tmp/whitelist.txt"
realert:
  hours: 1
filter:
  query:
    query_string:
      query: "*"
alert:
- "slack"
slack:
  slack_webhook_url: "https://hooks.slack.com/services/T2KF4P7D3/BS75PLJ22/F15aXcS"
```

The index used

The argument sent by packetbeat that will be compared

File that specifies which IPs are allowed to access

Query everything

Figure 261: Elastalert example 3 configuration file



```
elastalert APP 8:53 AM
whitelist
whitelist

@timestamp: 2020-01-13T06:37:59.122Z
_id: 662fnW8Br2QBYPs848ls
_index: packetbeat-7.5.1-2020.01.03-000001
_type: _doc
agent: {
  "ephemeral_id": "77111ddb-43dd-49ab-a89b-468a70cbfb27",
  "hostname": ██████████
  "id": "2080d7cd-67bf-4813-a24f-23584ee25855",
  "type": "packetbeat"
  "version": "7.5.1"
}
client: {
  "bytes": 32, matches the criteria
  "ip": ██████████
}
destination: {
  "ip": ██████████
}
```

Figure 262: Proof that the rule works (Example 3) part 1

Time	_source
> Jan 13, 2020 @ 11:20:36.813	rule_name: whitelist endtime: Jan 13, 2020 @ 11:20:30.165 starttime: Jan 13, 2020 @ 11:05:30.165 matches: 65 hits: 12,437 @timestamp: Jan 13, 2020 @ 11:20:36.813 time_taken: 6.648 _id: cLI0nm8Br2QBYPs8eRfO _type: _doc _index: elastalert_status_status _score: -
> Jan 13, 2020 @ 11:19:42.135	rule_name: whitelist endtime: Jan 13, 2020 @ 11:19:31.457 starttime: Jan 13, 2020 @ 11:04:31.457 matches: 833 hits: 12,737 @timestamp: Jan 13, 2020 @ 11:19:42.135 time_taken: 10.678 _id: YrIznm8Br2QBYPs8pBc4 _type: _doc _index: elastalert_status_status _score: -

Figure 263: Proof that the rule works (Example 3) part 2

6.5.2.5. Example: Blacklist - Packetbeat

At this point a Blacklist rule will be created. It will check a certain field against a blacklist, and match if it is in the blacklist. This rule opposed to the earlier examples requires two additional options; “compare_key” and “blacklist”. These options specify the key that is used to compare with the contents of the blacklist in order to match with a certain event. Also, the second option includes all the files that are embodied in the blacklist.



```
name: blacklist
type: blacklist
index: packetbeat-*
compare_key: destination.ip
blacklist:
- "!file /tmp/blacklist.txt"
realert:
  hours: 1
filter:
- query:
  query_string:
    query: "*"
alert:
- "slack"
slack:
  slack_webhook_url: "https://hooks.slack.com/services/T2KF4P7D3/BS75PLJ22/F15aXc"
```

```
root@elksiem2:/tmp# nano blacklist.txt
```

Figure 264: Elastalert example 4 configuration file

```
elastalert APP 8:46 AM
blacklist
blacklist
@timestamp: 2020-01-13T06:45:29.115Z
_id: vq6mnW8Br2QBYPs8wQI5
_index: packetbeat-7.5.1-2020.01.03-000001
_type: _doc
agent: {
  "ephemeral_id": "77111ddb-43dd-49ab-a89b-468a70cbfb27",
  "hostname": "██████████",
  "id": "2080d7cd-67bf-4813-a24f-23584ee25855",
  "type": "packetbeat",
  "version": "7.5.1"
}
client: { Client that triggered
  "bytes": 32, the event
  "ip": "██████████"
}
destination: IP which is
  "ip": "██████████" used inside the
} blacklist txt file
```

Figure 265: Proof that the rule works (Example 4) part 1



```
> Jan 13, 2020 @ 11:20:42.939 rule_name: blacklist endtime: Jan 13, 2020 @ 11:20:42.832 starttime: Jan 13, 2020 @ 11:05:42.832 matches: 0 hits: 243
@timestamp: Jan 13, 2020 @ 11:20:42.939 time_taken: 0.107 _id: cbI0nm8Br20BYPs8KRe. _type: _doc _index: elasticsearch_status_status
_score: -

> Jan 13, 2020 @ 11:19:45.991 rule_name: blacklist endtime: Jan 13, 2020 @ 11:19:45.849 starttime: Jan 13, 2020 @ 11:04:45.849 matches: 12 hits: 260
@timestamp: Jan 13, 2020 @ 11:19:45.991 time_taken: 0.142 _id: Y7Iznm8Br20BYPs8sxdJ. _type: _doc _index: elasticsearch_status_status
_score: -
```

Figure 266: Proof that the rule works (Example 4) part 2

6.5.2.6. Example: Spike - Metricbeat

The last example use case will be a spike rule for CPU usage of a device. The beat which is needed is Metricbeat and doesn't need any modification since it was installed earlier. It has multiple -unknown to other rules- options that are explained thoroughly in the screenshot below. Proof that the data is sent to Elasticsearch and shown in Kibana is also attached.

```
name: Metricbeat CPU Spike Rule
type: metric_aggregation
#es_host:
#es_port:
index: metricbeat-*
buffer_time:
  hours: 1
metric_agg_key: system.cpu.user.pct
metric_agg_type: avg
query_key: beat.hostname
doc_type: metricsets
bucket_interval:
  minutes: 5
sync_bucket_interval: true
#allow_buffer_time_overlap: true
#use_run_every_query_size: true
min_threshold: 0.1
max_threshold: 0.8
filter:
  - term:
      metricset.name: cpu
# (Required)
# The alert is use when a match is found
alert:
  - "debug"
alert:
  - "slack"
```

This rule matches when the value of a metric within the calculation window is higher or lower than a threshold

The name of the field over which the metric value will be calculated

The type of metric aggregation (min, max, avg etc.)

Group metric calculations by this field

Specify the _type of document to search for

If the calculated metric value is less than this number, an alert will be triggered

If the calculated metric value is greater than this number, an alert will be triggered

Figure 267: Elastaalert example 5 configuration file



```
Jan 8, 2020 @ 11:19:06.245 agent.hostname: DESKTOP-8M9A54T @timestamp: Jan 8, 2020 @ 11:19:06.245 agent.id: 9eeef044-1de9-46df-92bf-efc4cd56d8ee agent.version: 7.5.0 agent.type: metricbeat
agent.ephemeral_id: da14dc26-c919-4f2d-8b3c-c3f11bc84bd2 ecs.version: 1.1.0 process.name: dm.exe process.pid: 1,112 process.ppid: 848 process.pgid: 0 process.args: dm.exe
user.name: Window Manager\DMW-1 metricset.name: process metricset.period: 10,000 system.process.state: running system.process.memory.size: 118,800,384
system.process.memory.rss.bytes: 47,198,208 system.process.memory.rss.pct: 0.011 system.process.memory.share: 0 system.process.cmdline: dm.exe
system.process.cpu.start_time: Dec 31, 2019 @ 14:58:11.992 system.process.cpu.total.pct: 0.078 system.process.cpu.total.norm.pct: 0.02 system.process.cpu.total.value: 2,848,8
```

Expanded document [View surrounding documents](#) [View single document](#)

Table	JSON
@timestamp	Jan 8, 2020 @ 11:19:06.245
_id	b5xzh08BALa7FB_3Tqyh
_index	metricbeat-7.5.0-2020.01.08-000001
_score	-
_type	_doc
agent.ephemeral_id	da14dc26-c919-4f2d-8b3c-c3f11bc84bd2
agent.hostname	DESKTOP-8M9A54T

Figure 268: Proof that the rule works (Example 5)

7. Concluding Remarks – Empirical Findings

Cyber threats and cybercrime in general are currently serious problems, which will remain in the coming years. Some scenarios were described and analysed in this Thesis, that will help mainly with monitoring, forensics and accountability issues.

I showed that it is fairly difficult to make a perfectly secure system, since it needs massive configuration. The reason is that cybercriminals and researchers/ security experts are involved in an arms race; every time a new detection technique is developed, miscreants come up with more advanced attack and malicious strategies.

In this Thesis, I showed that GDPR, although fairly new can be easily integrated inside a platform like Elastic Stack; important since it is mainly used as a search engine suite.

Furthermore, I found out that Beats are a better solution contrary to Wazuh, since they need less configuration as they work natively inside ELK. It is highly notable that for every single log file Wazuh couldn't handle, a new ruleset had to be created, which is not an issue with Beats. As far as availability goes, Beats in older versions were heavily resources dependent, but now are fairly light making them equally as light as the Wazuh platform.

In the future, I plan to keep studying internet threats. As these threats, vulnerabilities and exploits become trickier and more sophisticated, the defending mechanisms must be always spot on and up to date. The security expert must always be informed and well trained in order to address the needs of this IT sector; cybersecurity.

8. Bibliography

Books

- [A]: Elasticsearch: The Definitive Guide, Clinton Gormley and Zachary Tong
- [B]: Elasticsearch Server, Rafal Kuc and Marek Rogozinski
- [C]: Elasticsearch Cookbook Second Edition, Alberto Paro
- [D]: Mastering Elasticsearch, Rafal Kuc and Marek Rogozinski
- [E]: Learning Elasticsearch, Abhishek Andhavarapu
- [F]: Elasticsearch Indexing, Huseyin Akdogan
- [G]: Elasticsearch: A Complete Guide Learning Path. End-to-end search and Analytics, Bharvi Dixit, Rafal Kuc, Marek Rogozinski, Saurabh Chhajed
- [H]: The Logstash book. Log management made easy, James Turnbull
- [I]: Kibana 7, Quick Start Guide, Anurag Srivastava
- [J]: Kibana Essentials, Yuvraj Gupta
- [K]: Ossec Host-based Intrusion Detection, Brad Lhotsky
- [L]: Learning ELK Stack, Saurabh Chhajed

Papers and Presentations

- [1]: Elasticsearch: an advanced and quick search technique to handle voluminous data, Manda Sai Divya and Shiv Kumar Goyal
- [2]: Using Elasticsearch, Logstash and Kibana to create realtime dashboards, Alexander Reelsen
- [3]: Review on efficient log analysis to evaluate multiple honeypots with ELK, Ibrahim Yahya Mohammed AL-Mahbashi, Prashant Chauhan, Shivi Shukla and M.B. Potdar
- [4]: Log Analysis with the ELK Stack (Elasticsearch, Logstash and Kibana), Gary Smith, Pacific Northwest National Laboratory
- [5]: Log Analysis using OSSEC, Daniel B. Cid
- [6]: Building a real-world logging infrastructure with Logstash, Elasticsearch and Kibana, Patrick Kleindienst
- [7]: OSSEC Host-Based Intrusion Detection Guide, Andrew Hay, Daniel Cid, Rory Bray
- [8]: Testbed Design for Evaluation of Active Cyber Defence Systems, Srikumar Sridhar
- [9]: Intrusion Detection with OSSEC, Ali Anafcheh

Documentations



[i]: <https://documentation.wazuh.com/3.9/user-manual/index.html>

[ii]: <https://www.elastic.co/guide/index.html>

[iii]: <https://www.ossec.net/docs/>