SCHOOL OF INDUSTRIAL & ITC ENGINEERING

JACTELK

BACHELOR’S DEGREE IN INFORMATICS ENGINEERING

FINAL DEGREE PROJECT

Author: Illia Nechesa
Supervisor: Jesús Villadangos
Date: Pamplona, 13 February 2020

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Index

Figure Index

1. Introduction

1.1 Current Situation

Tesicnor works with thousands of clients and has tens of projects. Each one, is deployed in a concrete server. All these apps have their own database, which contain many data, such as mail tables, audits (logs, reports, app events). It means, each app stores its audits in its own server.

The project to be developed, is a new big audit management system on a new server, in order to prevent sending all these data reports to individual databases of each app, and to send them to a centralized system. The main idea is to use Apache Kafka, a data bus that can transfer many information in both senses and it’s very efficient.

For the project, it would be a good idea to use Docker, a deployment system, based in containers, which gives us the opportunity to run our app in any device independently of its Operative System.

Another current problem is memory space. Existing audit logs require a lot of space, and each app needs a different schema to store data. The database should be able to have a central point to store the complete set of logs and be able to process in real time logs from different apps (different schema). Non-relational databases are more adapted than relational databases for doing the above tasks. Therefore, we should consider using a non-relational one, such as MongoDB for example

As soon as we finish the backend, we will start developing the frontend. We will almost certainly use Angular or ReactJS. With the UI, the user can access all clients and to all audits of all apps that are subscribed to this system. Finally, remark that for the development of this project, apart from the previously mentioned technologies, other tools such as Jenkins will be used for continuous integration, and other ones such as Jhipster, Spring Boot…

1.2 Current audit system and log management at Tesicnor

Currently, there is one freemium (one server) for each application: Kaiku, Gamesa, Acciona, Cobra, Tdoc, ... Additionally, there are applications such as Tdoc, which has a separate application for each client, with its own server. So, there are several Tdoc applications, and for each one there is one server, and each server has its own database.

There are two main databases in each server: “gestiondocumental”, where all the business data is stored, and its size usually varies between 1 and 2 gigabytes. The other one is called “system”, and this one is a little bit trickier, since all the mails, and the audits, are stored here, and in some cases, they require more than 30 GB.

The current application’s audit management system shows some drawbacks that could be illustrated with an example. Let's say there have been 6 user updates, that is, a user has been modified 6 times. Well, in audits, what is saved in each change is the name of the change, for example (update_user), the date on which the change was made, and who made the change. Now let's imagine, that a change has been wrong. But really, with the data we have, we know in what change of those 6 has the fault occurred? No, since at no time do we save the information that has been changed.

The current idea is that for every change that is made, we will keep it in a non-relational database such as MongoDB. We will store both the previous object and the new object, so that you can see exactly what has been changed.

Apart from the audits, it would be convenient to save the logs as well. We should see how we categorize them, since for example .info or .warning do not need to be stored for a long time, however. error, it is convenient to save and store them for a longer time. Currently Tdoc Logs for example are stored on the server, and in principle, there is no system that manages the deletion of old logs. In addition, there is Apache Tomcat's own Log, which stores not only the logs of the application, but also of the server in the catalina.out file. For all this a rotary log system (Logrotate) is used. Logrotate is the process responsible for rotating, compressing and emailing system logs in Linux.

However, there are other applications, which do not use the apache tomcat service, but work with docker next to NGINX (another server). Docker logs can be accessed through output / docker logs appName, where appName is the name of the application.

1.3 Proposal for the system Architecture

For a better understanding of the system, figure 1 illustrates the architecture of the current one. As we can see, each application, defined as A..Z, has its own audit system (SA), and all logs are stored in their own system, making the only way to access for them, connecting to each server from which you want to take the log files. Moreover, it's the custom audit system is not automated, and does not store all the information we want. The idea is to eliminate these audits from each application and send all of them to a common audit system, a new server, which will be able to manage absolutely everything. From the other side, all the logs also are going to be indexed to one server, so we could visualize all of them from all our applications in only one server.

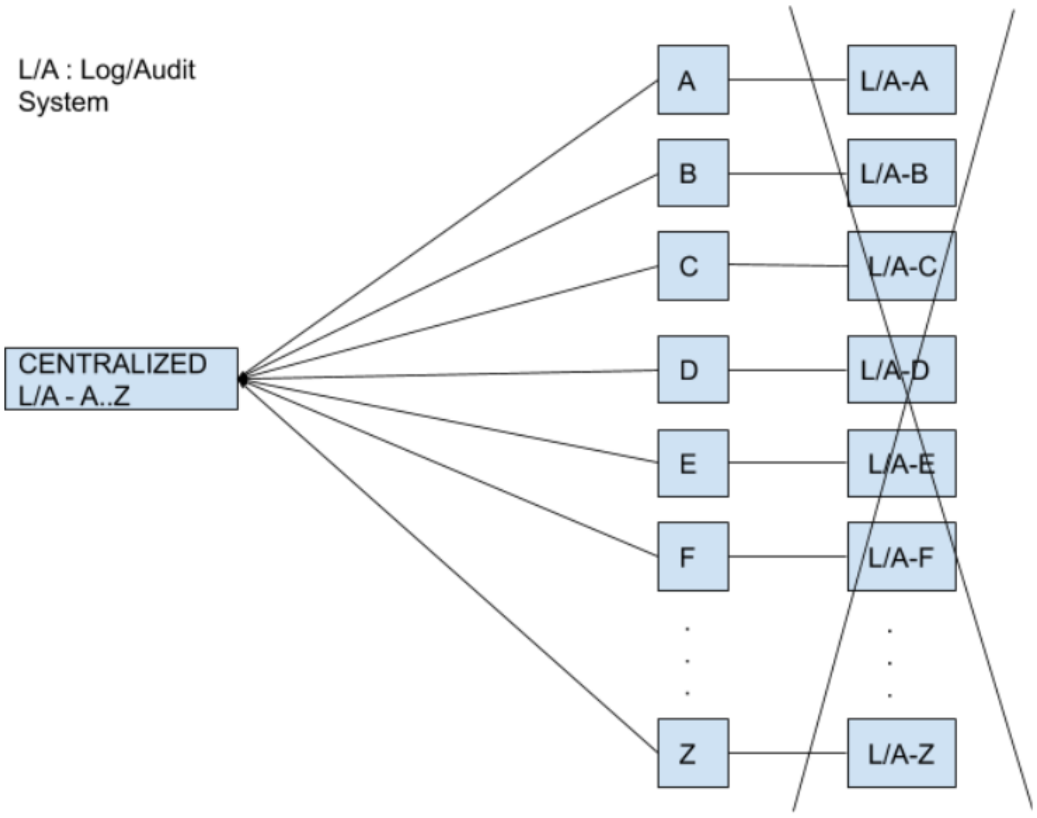


Figure 1 - Audit/Log Generic System Architecture

2. Technologies and Tools

After understanding the project to develop, we have a department meeting to talk about how the development is going to take place.

The first step is to study all the current technologies available we could use for the development of the new system, audit frameworks, log manager frameworks, work methodologies…

2.1 Audits frameworks

2.1.1 Javers

- Lightweight, fully **open-source** Java library for auditing changes in your data **[2].**
- Version and change are first class citizens.
- Use **JSON** for object serialization.
- No need to provide detailed ORM-mapping. Only need to know some high-level facts about our data model.
- Is meant to **keep data versioning records** (Snapshots) in the application's primary database.
- Written in **Java 8.**
- Only matches objects who have the same Id Value.
- The **diff** can be **stored** in a **JSON** - Useful in **MongoDB.**
- Works with **Java Objects.**
- Javers artefacts are published to **Maven Central.**
- **Works** very **well** with **MongoDB** and is **easy** to **integrate** into a **Maven** project.
- There are numerous **commands** with which you can **filter the results** in different ways with **queries**.
- **Main functionalities** → **Diff**, Repository, JQL, **Commit [6]**.

Proof of concept

- **Diff** → Create two objects, and compare them:

```
Diff diff = javers.compare(frodoOld, frodoNew);
System.out.println(diff);
        Diff:
        * new object: Employee/Sméagol
        * new object: Employee/Gandalf
        * changes on Employee/Frodo :
          - 'age' changed from '40' to '41'
          - 'boss' changed from '' to 'Employee/Gandalf'
          - 'position' changed from 'Townsman' to 'Hero'
          - 'primaryAddress.city' changed from 'Shire' to 'Mordor'
          - 'salary' changed from '10000' to '12000'
          - 'skills' collection changes :
            . 'agile coaching' added
          - 'subordinates' collection changes :
            0. 'Employee/Sméagol' added
```

```
System.out.println(javers.getJsonConverter().toJson(diff));
{
  "changes": [
    {
      "changeType": "NewObject",
      "globalId": {
        "entity": "Employee",
        "cdoId": "Gandalf"
      }
    },
    {
      "changeType": "NewObject",
      "globalId": {
        "entity": "Employee",
        "cdoId": "Sméagol"
      }
    },
    {
      "changeType": "ValueChange",
      "globalId": {
        "valueObject": "org.javers.core.examples.model.Address",
        "ownerId": {
          "entity": "Employee",
          "cdoId": "Frodo"
        },
        "fragment": "primaryAddress"
      },
      "property": "city",
      "left": "Shire",
      "right": "Mordor"
    },
    {
      "changeType": "ValueChange",
      "globalId": {
        "entity": "Employee",
        "cdoId": "Frodo"
      },
      "property": "position",
      "left": "Townsman",
      "right": "Hero"
    },
    ...
  ]
}
```

On the left side it is stored the old object, and in the right the new one.

- **Commit** → We want to store all the states of an object, in other words, be aware of all changes made to a specific object. Let's think about the object **robert**. By typing ***javers.commit("user",robert);*** every time the object changes, its state will be stored in **JaversRepository [1]**.

```
22:11:56.572 [main] INFO  org.javers.core.Javers - Commit(id:1.0,
snapshots:1, author:user, changes - NewObject:1), done in 74 millis
(diff:74, persist:0)
22:11:56.594 [main] INFO  org.javers.core.Javers - Commit(id:2.0,
snapshots:1, author:user, changes - ValueChange:2), done in 5 millis
(diff:5, persist:0)

Shadows query:
Person{login='bob', name='Robert C.', position=Developer}
Person{login='bob', name='Robert Martin', position=null}

Snapshots query:
Snapshot{commit:2.0, id:...Person/bob, version:2, (login:bob,
name:Robert C., position:Developer)}
Snapshot{commit:1.0, id:...Person/bob, version:1, (login:bob,
name:Robert Martin)}

Changes query:
Changes:
Commit 2.0 done by user at 13 Apr 2018, 22:11:56 :
* changes on org.javers.core.examples.model.Person/bob :
  - 'name' changed from 'Robert Martin' to 'Robert C.'
  - 'position' changed from '' to 'Developer'
```

2.1.2 Audit4j

- **Open Source** Auditing Framework.
- There are **code examples** in **GitHub.**
- To run, **YAML** and **XML** are **supported.**
- There are **few documentations**, and we could not find **any tutorial** about it **[3].**

2.1.3 Log4j

- Provides information to determine **what actions have been performed**, or attempted to be performed, **by whom**, **when** they did it and **what** the data associated with the action was.
- **Contain** the action being performed and the **data elements** that have been **impacted**.
- Same as previous quoted framework, there are **not examples and tutorials** about this specific framework.

2.1.4 Spring Data JPA Auditing

- Track and log events.
- Discarded option since there is no option to compare old and new objects.

2.2 Data BUS

2.2.1 Kafka

- It is good since we can **connect** a **big number of consumers**, and in our case, this is fundamentally important.
- Writes the messages, assigning them to different categories, named topics .
- **Producers write** those messages into topics, and the **consumers read** it from there **[5]**.
- We think using it is the most reasonable, since we already use it in our applications **[8].**

2.3 Log systems

2.3.1 Prometheus

- Monitoring system and TSDB (Time series Database).
- Made for dynamic cloud environments.
- What **DO NOT** do Prometheus is logging or tracing, individual events, timings with paths.
- If we want logging, use something as Elasticsearch (Literal words from the Prometheus founder) **[4].**

2.3.2 Elasticsearch

- **Kibana** dashboard, where we can perform searches.
- **Logstash**, log manager.
- **Beats**, log sender and light.
- Lots of **widgets** to visualize the results.

There are many more log system managers, and frameworks, such as Splunk, Sumo Logic, Loggly, PaperTrails, InfluxDB, Sensu… however, after researching and reading many articles, we have found, what mostly fits for our requirements is **ELK (Elasticsearch, Logstash, Kibana) [7].**

2.3.3 ELK Stack

- **Elasticsearch** → It is a **distributed** database, distributes all the information on all nodes, therefore it is **fault tolerant**, and has **high availability.** Like distributing the information, it distributes the processing.
- **Logstash** → It is part of **pre-processing before saving** the information in Elasticsearch that we have commented.
- **Kibana** → It's the UI.

So why is Elasticsearch so popular and why everyone uses it? Here its **main** functionalities:

→ **Analyse** application **logs** and **system metrics.**
→ APM → Application Performance Management.
→ **Stores the data it receives** .
→ **Forecast future values** with machine learning.
→ **Predicting future** # of calls.
→ **Anomaly Detection** → The system can detect anomalies, for example, **if the number of visits or accesses is considerably reduced in a certain time**.
→ Every time **something weird occurs**, we are notified.
→ Data is **stored as documents**.
→ A documents data is separated in fields.
→ **JSON** object.
→ Easy to use, and highly scalable.

- Architectures and how it is commonly used.
    - The application communicates with **Elasticsearch**.
    - Can be done with any **HTTP library.**
    - Using the official client libraries is recommended.
    - The web application should keep the data updated.
    - Data will be stored in DB and Elasticsearch.
- Dashboard
    - **Kibana** → Web interface that interacts with Elasticsearch.
    - Kibana can be run on any machine.
    - The web traffic increases significantly → We need to monitor server resources.
    - Metricbeat is the solution, CPU and memory usage.
    - Configure Metric Beat to send data to Elasticsearch → Lightweight shipper that you can install on your servers to periodically collect **metrics** from the operating system and from services running on the serve.

Our company currently works with **Grafana** instead of **Kibana**, so let's see the main differences between these ones.

2.3.3.1 Logs vs Metrics

Grafana is designed for **analysing** and **visualizing metrics** such as system **CPU**, **memory**, **disk**, and **I/O utilization**. Grafana does not allow full-test data querying. **Kibana**, on the other hand, runs on top of Elasticsearch and is used **primarily for analysing log messages**.

If we are building a monitoring system, both can do the job well, though there are still some differences that will be outlined below. For any of the use cases that logs support — **troubleshooting**, **forensics**, **development**, **security**, Kibana is your only option.

2.3.3.2 Setup, installation and configuration

Both are easy **to install and configure**, and support installation on Linux, Mac, Windows, Docker or building from source. **Kibana supports** slightly **more OS**. However, **Grafana** is configured using an **.ini file** which is relatively easier to handle compared to **Kibana's** syntax-sensitive **YAML** configuration files.

2.3.3.3 Data sources and integrations

Grafana was designed to work as a UI for **analysing metrics**. As such, it can **work with multiple** time-series **data stores**, including built-in integrations with Graphite, Prometheus, InfluxDB, MySQL, PostgreSQL, and Elasticsearch, and additional data sources using plugins. **For each data source**, Grafana has a **specific query editor** that is customized for the features and capabilities that are included in that data source. **Kibana** on the other hand, is designed to **work only with Elasticsearch** and thus does not support any other type of data source.

2.3.3.4 Access control and authentication

By default, and unless you are using either the X-Pack (a commercial bundle of ELK - Elasticsearch Logstash Kibana add-ons, including for access control and authentication) or open source solutions such as SearchGuard, your **Kibana dashboards are open and accessible to the public**. In comparison, **Grafana** ships with **built-in user control and authentication mechanisms that allow you to restrict and control access to your dashboards**, including using an external SQL or LDAP server. In addition, Grafana's API can be used for tasks such as saving a specific dashboard, creating users, and updating data sources. You can also create specific API keys and assign them to specific roles.

2.3.3.5 Querying

Querying and searching logs is one of **Kibana's** more **powerful features**. Using either Lucene syntax, the Elasticsearch Query DSL or the experimental Kuery, the data stored in Elasticsearch indices can be searched with results displayed in the main log display area in chronological order. Lucene is quite a powerful querying language but is not intuitive and involves a certain learning curve.

With **Grafana**, users use what is called a **Query Editor for querying**. Each data source has a different Query Editor tailored for the specific data source, meaning that the **syntax used varies according to the data source**. Graphite querying will be different than Prometheus querying, for example.

2.3.3.6 Dashboards and visualizations

Both **Kibana and Grafana** boast **powerful** visualization **capabilities**. **Kibana** offers a **rich variety of visualization types,** allowing you to create pie charts, line charts, data tables, single metric visualizations, geo maps, time series and markdown visualizations, and combine all these into dashboards. Dashboards in Kibana are extremely dynamic and versatile — data can be filtered on the fly, and dashboards can easily be edited and opened in full-page format. Kibana ships with default dashboards for various data sets for easier setup time.

Grafana dashboards are what made Grafana such a **popular visualization tool**. They are infamous for being completely versatile. Visualizations in Grafana are called panels, and users **can create a dashboard containing panels for different data sources**. Grafana supports graph, singlestat, table, heatmap and freetext panel types. Grafana users can make use of a large ecosystem of ready-made dashboards for different data types and sources.

Functionality wise — both Grafana and Kibana offer many customization options that allow users to slice and dice data in any way they want. Users can play around with panel colours, labels, X and Y axis, the size of panels, and plenty more. All in all, though, **Grafana has a wider array of customization options and also makes changing the different setting easier with panel editors and collapsible rows.**

2.3.3.7 Alerts

A **key difference** between Kibana and Grafana is alerts. Since version 4.x, **Grafana** ships with a **built-in alerting engine** that allows users to attach conditional rules to dashboard panels that result in triggered alerts to a notification endpoint of your choice (e.g. email, Slack, PagerDuty, custom webhooks). **Kibana does not come with an out-of-the-box alerting capability**. To add alerting to Kibana users can either opt for a hosted ELK Stack such as Logz.io, implement ElastAlert or use X-Pack.

2.4 Development methodologies

2.4.1 DevOps - Jenkins - Docker - Kubernetes

Nowadays, most companies search for **fast release**, and **always be in production**. There is **lack of integrations tests**, something that is very important. Let's take the example of amazon, it makes a deploy every 11 seconds, and we don't even notice it **[9].**
Remember that **DevOps** is a set of principles for **interactive development**, in order to increase the development speed and the **continuous delivery.**

There are three "Amigos" we should consider, which are :

- **Jenkins** → **Automate** our **life cycle** → Pipelines as code
- **Docker** → In order to **package apps** and all their dependencies
- **Kubernetes** → Universal **execution platform**

We should remember that there are many other technologies we can use, and it's very important not to tie to any in particular and be always prepared for new ones.

Continuous Integration Pipeline

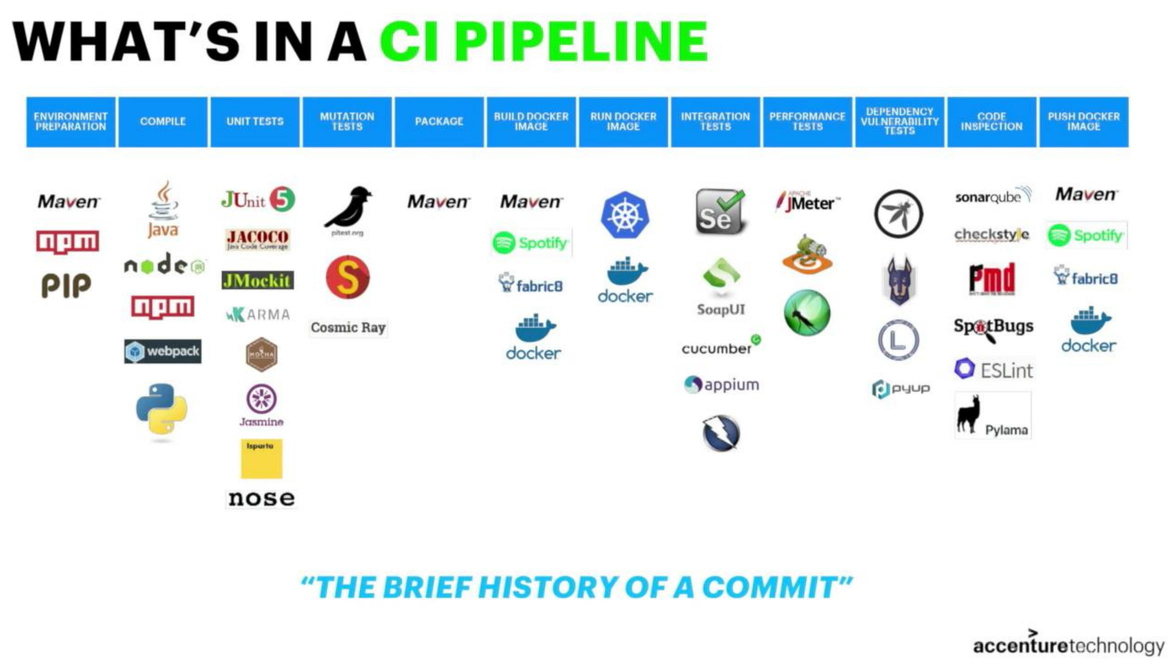


Figure 2 - CI Pipeline

3. Requirements

For a better understanding of the system we want to develop, we have done a use case diagram with the system functionalities we want to implement

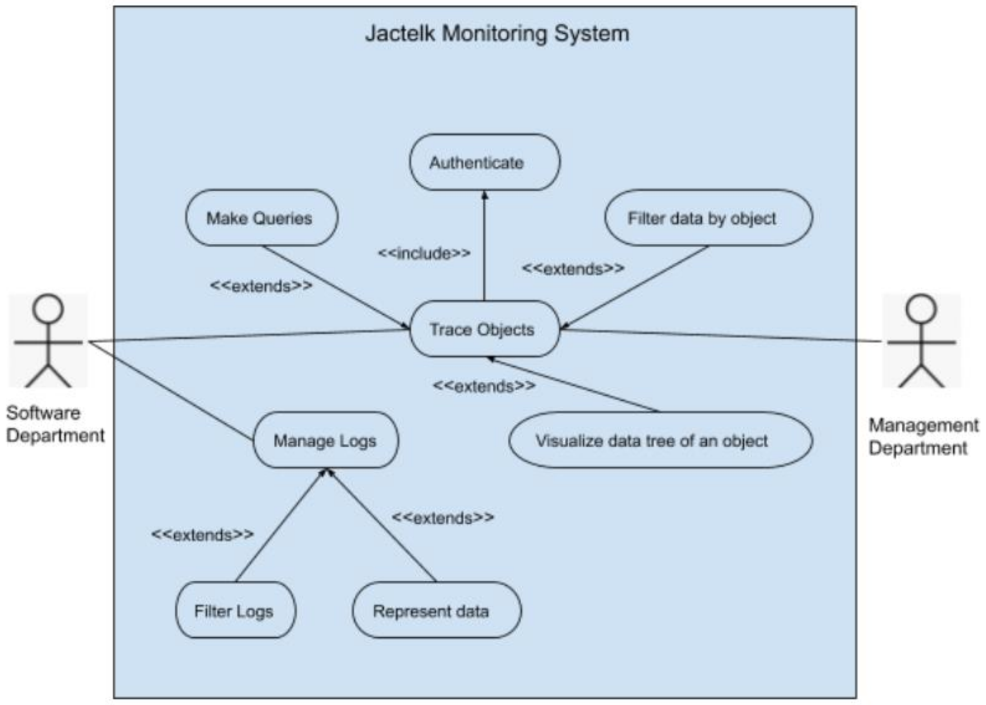


Figure 3 - Use Case Diagram

As we can see, management department will be able only to trace the objects, perform tasks such as making queries, filtering data of a specific object, or even visualizing a tree of an object, were there will be a lot of states of the same objects, and they will be able to see when and which fields were modified. This operation could be also done by the software department. However, to have access for all this data it is required to authenticate before, since we only want specific persons to be able to access this information.

The software department will also manage all the logs of all their applications, to prevent errors, see which services are the used the most, and with this information try to improve the system performance, by refining the already working services. The main features they are going to use is the filtration of logs by application or microservices, and its representation. All this will help to study the system performance.

4. Project Management Methodology

4.1 Agile Development

- **Scrum**

- **Trello** as main software for project management

    - **To do / Doing / Done** lists.
    - **Choose our own workflow.**
    - Create **new** lists.
    - **Assign tasks** to workers
    - Create **checklists** for tasks

- We have managed to develop this project in many iterations, so that we could always see the progress of the project, and to redefine the requisites if it is needed. As the requirements were changed nearly each day, it was impossible to define or to draw any diagram since we did not have certainty that the increment would be correct or not.

4.2 Iterations

4.2.1 First Iteration

After doing a previous research about the technologies available, we have a meeting, in order to discuss about all the technologies available and how we can use it to develop this project. For the audit management, all are satisfied with the **Javers** option, and exactly with the **commit** function, which is exactly what we need. Since the main purpose of this app is to inform us about all changes to which an object is subjected. The commit function does exactly this, if you change something in an object, and then commit it, it will **save field which was changed**, **and** the **old and new value.**

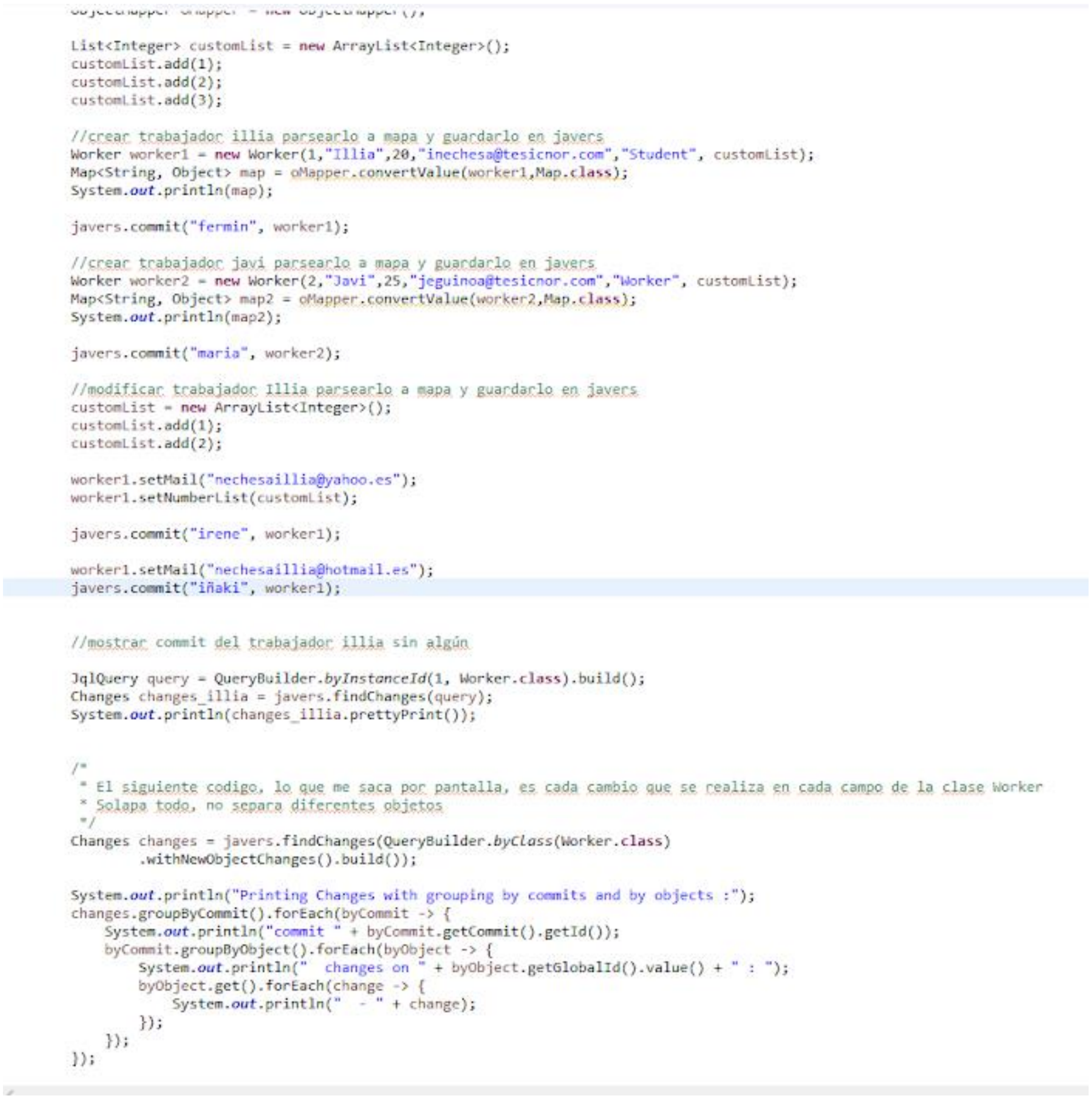
However, there is a problem. Javers, is a framework that compares two objects, or that follows the changes in an object. But where is stored this information? If we have a function which changes something simple, for example the mail of a worker. We would use a setter and that's it, right? But do we need to store the previous object, or a clone of it, in order to compare it later? We thought about this problem all together and reach the conclusion we don't need the diff function, only the commit one. But even with this one, we also want to store the user who performed the action, the date on which it was performed, and maybe more information. Does this information appear in the commit function?

We don’t know also if Javers should be installed in each app, or just in one server , which will receive all the changes. We will research about all this now.

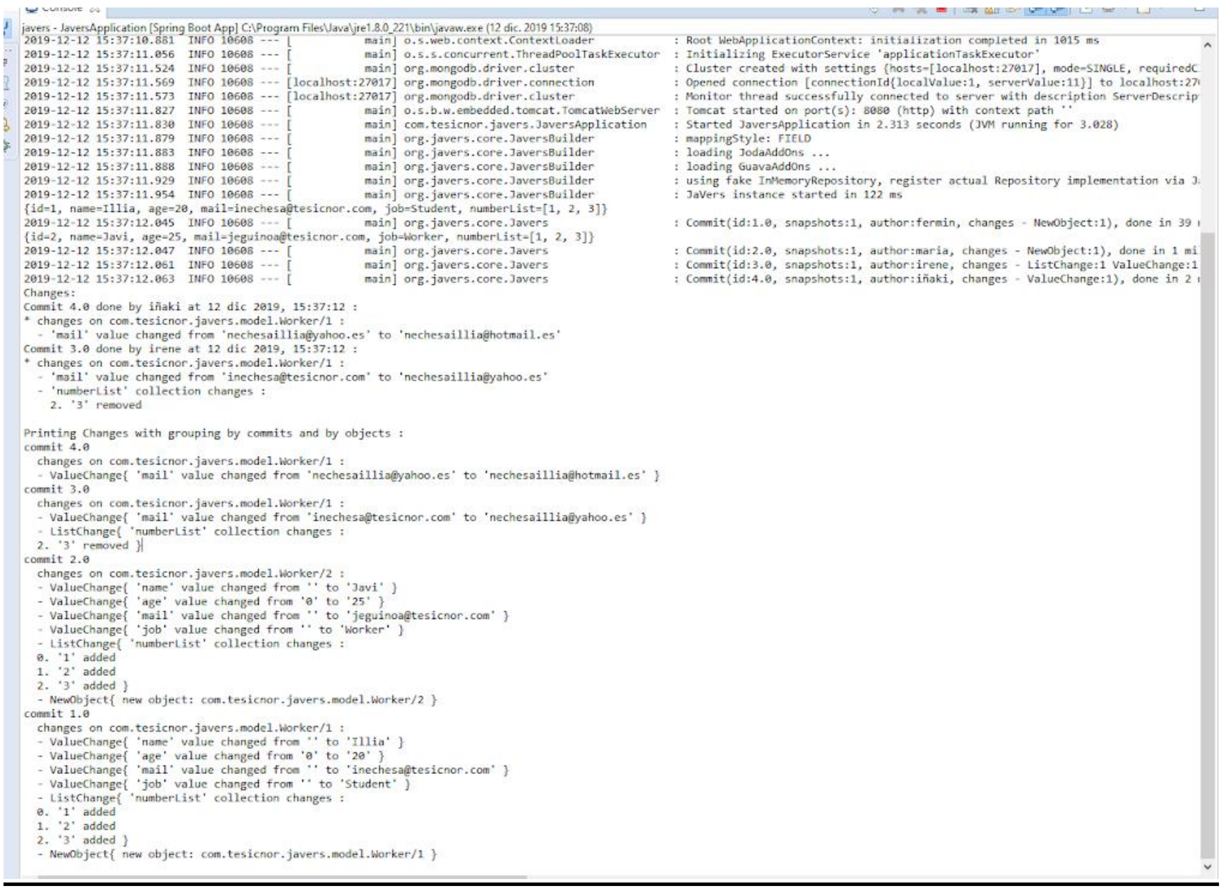
We have been committed to do a deeper research about how we could implement this with our requirements, so we will develop a little project, in which we’ll try to make use of **Javers framework** and see **how it works**.

We’ve created a Maven project, with the Spring Initializer, which helps you to mount all your project automatically, so you don’t need to set it up manually. After that, we create a small class and try new things with it, such as value changes, Javers commits, convert it into a map… and we got some interesting results.

First Tests



```
ObjectMapper oMapper = new ObjectMapper();

List<Integer> customList = new ArrayList<Integer>();
customList.add(1);
customList.add(2);
customList.add(3);

//crear trabajador illia parsearlo a mapa y guardarlo en javers
Worker worker1 = new Worker(1,"Illia",20,"inechesa@tesicnor.com","Student", customList);
Map<String, Object> map = oMapper.convertValue(worker1,Map.class);
System.out.println(map);

javers.commit("fermin", worker1);

//crear trabajador javi parsearlo a mapa y guardarlo en javers
Worker worker2 = new Worker(2,"Javi",25,"jeguinoa@tesicnor.com","Worker", customList);
Map<String, Object> map2 = oMapper.convertValue(worker2,Map.class);
System.out.println(map2);

javers.commit("maria", worker2);

//modificar trabajador Illia parsearlo a mapa y guardarlo en javers
customList = new ArrayList<Integer>();
customList.add(1);
customList.add(2);

worker1.setMail("nechesaillia@yahoo.es");
worker1.setNumberList(customList);

javers.commit("irene", worker1);

worker1.setMail("nechesaillia@hotmail.es");
javers.commit("iñaki", worker1);


//mostrar commit del trabajador illia sin algún

JqlQuery query = QueryBuilder.byInstanceId(1, Worker.class).build();
Changes changes_illia = javers.findChanges(query);
System.out.println(changes_illia.prettyPrint());


/*
 * El siguiente codigo, lo que me saca por pantalla, es cada cambio que se realiza en cada campo de la clase Worker
 * Solapa todo, no separa diferentes objetos
 */
Changes changes = javers.findChanges(QueryBuilder.byClass(Worker.class)
        .withNewObjectChanges().build());

System.out.println("Printing Changes with grouping by commits and by objects :");
changes.groupByCommit().forEach(byCommit -> {
    System.out.println("commit " + byCommit.getCommit().getId());
    byCommit.groupByObject().forEach(byObject -> {
        System.out.println("  changes on " + byObject.getGlobalId().value() + " : ");
        byObject.get().forEach(change -> {
            System.out.println("  - " + change);
        });
    });
});
```



```
javers - JaversApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (12 dic. 2019 15:37:08)
2019-12-12 15:37:10.881  INFO 10608 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1015 ms
2019-12-12 15:37:11.056  INFO 10608 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2019-12-12 15:37:11.524  INFO 10608 --- [           main] org.mongodb.driver.cluster               : Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredC
2019-12-12 15:37:11.569  INFO 10608 --- [localhost:27017] org.mongodb.driver.connection            : Opened connection [connectionId{localValue:1, serverValue:11}] to localhost:27
2019-12-12 15:37:11.573  INFO 10608 --- [localhost:27017] org.mongodb.driver.cluster               : Monitor thread successfully connected to server with description ServerDescrip
2019-12-12 15:37:11.827  INFO 10608 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2019-12-12 15:37:11.830  INFO 10608 --- [           main] com.tesicnor.javers.JaversApplication    : Started JaversApplication in 2.313 seconds (JVM running for 3.028)
2019-12-12 15:37:11.879  INFO 10608 --- [           main] org.javers.core.JaversBuilder            : mappingStyle: FIELD
2019-12-12 15:37:11.883  INFO 10608 --- [           main] org.javers.core.JaversBuilder            : loading JodaAddOns ...
2019-12-12 15:37:11.888  INFO 10608 --- [           main] org.javers.core.JaversBuilder            : loading GuavaAddOns ...
2019-12-12 15:37:11.929  INFO 10608 --- [           main] org.javers.core.JaversBuilder            : using fake InMemoryRepository, register actual Repository implementation via J
2019-12-12 15:37:11.954  INFO 10608 --- [           main] org.javers.core.JaversBuilder            : JaVers instance started in 122 ms
{id=1, name=Illia, age=20, mail=inechesa@tesicnor.com, job=Student, numberList=[1, 2, 3]}
2019-12-12 15:37:12.045  INFO 10608 --- [           main] org.javers.core.Javers                   : Commit(id:1.0, snapshots:1, author:fermin, changes - NewObject:1), done in 39 
{id=2, name=Javi, age=25, mail=jeguinoa@tesicnor.com, job=Worker, numberList=[1, 2, 3]}
2019-12-12 15:37:12.047  INFO 10608 --- [           main] org.javers.core.Javers                   : Commit(id:2.0, snapshots:1, author:maria, changes - NewObject:1), done in 1 mi
2019-12-12 15:37:12.061  INFO 10608 --- [           main] org.javers.core.Javers                   : Commit(id:3.0, snapshots:1, author:irene, changes - ListChange:1 ValueChange:1
2019-12-12 15:37:12.063  INFO 10608 --- [           main] org.javers.core.Javers                   : Commit(id:4.0, snapshots:1, author:iñaki, changes - ValueChange:1), done in 2 
Changes:
Commit 4.0 done by iñaki at 12 dic 2019, 15:37:12 :
* changes on com.tesicnor.javers.model.Worker/1 :
  - 'mail' value changed from 'nechesaillia@yahoo.es' to 'nechesaillia@hotmail.es'
Commit 3.0 done by irene at 12 dic 2019, 15:37:12 :
* changes on com.tesicnor.javers.model.Worker/1 :
  - 'mail' value changed from 'inechesa@tesicnor.com' to 'nechesaillia@yahoo.es'
  - 'numberList' collection changes :
    2. '3' removed

Printing Changes with grouping by commits and by objects :
commit 4.0
  changes on com.tesicnor.javers.model.Worker/1 :
  - ValueChange{ 'mail' value changed from 'nechesaillia@yahoo.es' to 'nechesaillia@hotmail.es' }
commit 3.0
  changes on com.tesicnor.javers.model.Worker/1 :
  - ValueChange{ 'mail' value changed from 'inechesa@tesicnor.com' to 'nechesaillia@yahoo.es' }
  - ListChange{ 'numberList' collection changes :
  2. '3' removed }
commit 2.0
  changes on com.tesicnor.javers.model.Worker/2 :
  - ValueChange{ 'name' value changed from '' to 'Javi' }
  - ValueChange{ 'age' value changed from '0' to '25' }
  - ValueChange{ 'mail' value changed from '' to 'jeguinoa@tesicnor.com' }
  - ValueChange{ 'job' value changed from '' to 'Worker' }
  - ListChange{ 'numberList' collection changes :
  0. '1' added
  1. '2' added
  2. '3' added }
  - NewObject{ new object: com.tesicnor.javers.model.Worker/2 }
commit 1.0
  changes on com.tesicnor.javers.model.Worker/1 :
  - ValueChange{ 'name' value changed from '' to 'Illia' }
  - ValueChange{ 'age' value changed from '0' to '20' }
  - ValueChange{ 'mail' value changed from '' to 'inechesa@tesicnor.com' }
  - ValueChange{ 'job' value changed from '' to 'Student' }
  - ListChange{ 'numberList' collection changes :
  0. '1' added
  1. '2' added
  2. '3' added }
  - NewObject{ new object: com.tesicnor.javers.model.Worker/1 }
```

We run the program with Spring and Mongo, using Maven Central as project manager, and Javers as audit system. We can see that some results such as the commits, work properly. However, there is so much work left, in order to learn how to use properly the Queries to get the exact results that we want.

We have evaluated the first tests in the department, and we are so happy with the results. However, there is one issue. It would be better for our system to not send the Object itself when we do the Javers commit, but to send a map<key,object>, which will contain all the fields, such as **{id=1, name=Illia, age=20, mail=inechesa@tesicnor.com, job=Student, numberList=[1, 2, 3]}**

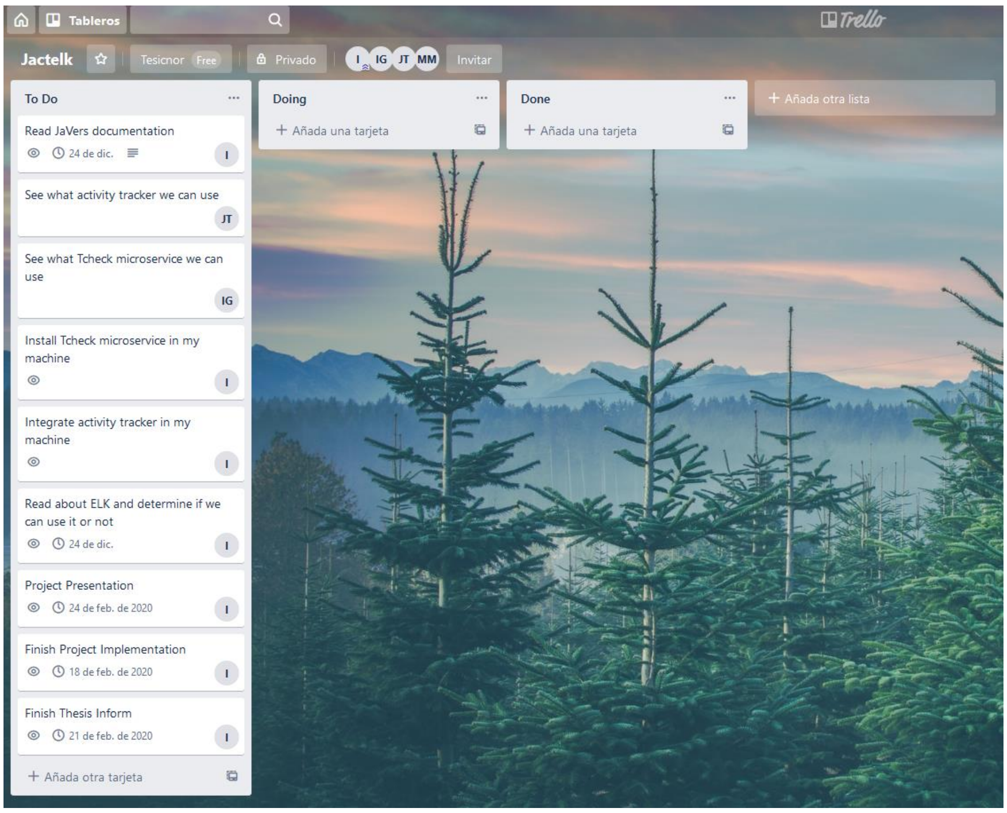
Like that, we do not load the object entirely, and only load the fields of it, so in the future it will be much simpler to read and compare them. Now the thing we need to do, is to investigate if it is possible to do the commit with a map instead of an object.

4.2.2 Second Iteration

After a sprint review, we plan the next one. Our boss is kind of aware, since this weekend he has been reading information about **Javers** library, **ELK**, **Activity trackers**, and find out that this **project** is much bigger than we thought it was. First, we decide to integrate all this in a concrete **microservice** of an application. Why? Because we see that Javers can slow down the app **performance**, and we don't want that. That's why previously to implement the Javers library, we need to integrate an Activity Tracker in this microservice, do performance tests, and then integrate Javers, and repeat the same process, in order to see if there are anomalies in the app performance.

In parallel, we'll be researching about **ELK,** since we need to be sure if it is open source, and if not, see its cost, and determine if we are going to use it or not.

Now, to manage better the project, we create this project in **Trello**, where we'll be creating new tasks to do, in order to set deadlines, and to and give me time to deliver everything on the set date.



To Do
- Read JaVers documentation — 24 de dic. — I
- See what activity tracker we can use — JT
- See what Tcheck microservice we can use — IG
- Install Tcheck microservice in my machine — I
- Integrate activity tracker in my machine — I
- Read about ELK and determine if we can use it or not — 24 de dic. — I
- Project Presentation — 24 de feb. de 2020 — I
- Finish Project Implementation — 18 de feb. de 2020 — I
- Finish Thesis Inform — 21 de feb. de 2020 — I
- + Añada otra tarjeta

Doing
- + Añada una tarjeta

Done
- + Añada una tarjeta

Figure 4 - Trello Initial

First, we will read all the official documentation about Javers, and make a small presentation about it next week, so that we can see exactly how we can use it.

Javers Features Overview

- Designed for **auditing changes**
- Commit changes on your object **graph** to a specialized **repository (JaversRepository)**
- Browse **change history** with **diffs** and **snapshots**
- **Diff** is the **main way** to deeply **compare two object** graphs
- *javers.compare()* , list of **atomic Changes**. There are several **types** of Changes, see the **inheritance hierarchy** of **Change class** to get the **complete list**.
- **JaversRepository** is the central part of our **data auditing engine**
- You **don't need to commit every object**. Javers navigates through the object graph, starting from the object passed to javers.commit() and **deeply compares the whole structure** with the previous version stored in JaversRepository
- **Spring Data**
- **Javers Query Language** → **JQL**
- Three **views** → **Changes**, **Shadows**, **Snapshots** → *javers.find*()* → Detailed **history**
- **JaversRepository** easily implemented with MongoDB

You don't need to commit every object. Javers navigates through the object graph, starting from the object passed to javers.commit() and deeply compares the whole structure with the previous version stored in JaversRepository. Thanks to this approach, you can commit large structures, like trees, graphs and DDD aggregates with a single commit() call.

So let's look at this deeply. Javers framework commit changes to **Javers Repository**. With which **purpose**? **Store** Javers **commit** in your **database** alongside your domain data. Emphasize that JSON format is used to serialize data. Javers-core oversees mapping domain objects to JSON. Javers commit holds a **list of domain object snapshots** and a **list of changes (a diff)**. Only snapshots are persisted in a database. When Javers commit is being read from a database, snapshots are deserialized from JSON and the diff is re-calculated by comparing snapshot pairs.

By default, Javers come with **in-memory repository implementation**...great for testing. Nevertheless, for production environment. There are plenty of databases supported by Javers. Those are **MongoDB**, **H2**, **PostgreSQL**, **MariaDB**, **Oracle**, **Microsoft SQL Server.**
In our case, we opt for using **MongoDB**. We only need to write some lines in order to make it work. First, we need to **add** javers-persistence-mongo **module** to our **classpath**, and then **provide** a working **Mongo client** and that's it.

Javers create two collection in MongoDB:
- **jv_head_id** → One document with the last CommitID
- **jv_snapshots** → Domain object snapshot. Each document contains snapshot data and commit medatada.

ELK Features Overview

ELK is the old name. Since after adding **Beats** , the commonly used name is **Elastic Stack**. First, we need to know that if we want **hosting** from elastic.co, they charge us according to several variables. The pricing of it it's in their web. However, if we want to use **open source**, stand up our **servers**, and manage our own **deployment**, we can do it without any problem **[10]**.

Logstash is a log aggregator that collects data from various input sources.

Logstash along with family of Beats **collect** and **parse logs**(say NGINX logs for SEO and analysis of web traffic), and then this information is **indexed** and **stored by Elasticsearch**. Finally, **Kibana presents** the **data** in visualizations enabling us to provide decision-making insights. Isn't it amazing?

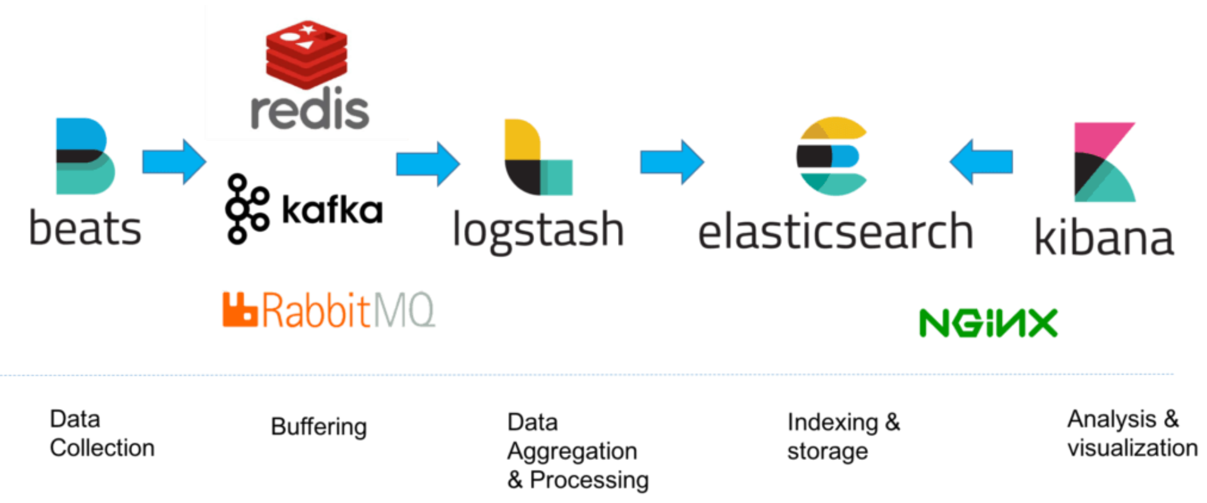


Figure 5 - ELK Schema

This is of course a simplified **diagram** for the sake of illustration. A **full production-grade architecture** will consist of **multiple Elasticsearch nodes**, perhaps **multiple Logstash instances**, an archiving mechanism, an alerting plugin and a full replication across regions or segments of your data centre for high availability **[11].**

4.2.3 Third Iteration

During the meeting we have gained further knowledge of the project. We have discussed the main features we could use, explained how Javers commits are stored, where they are stored, and how we could implement them in the project. Also, we have discussed how relational objects are stored in the Database. The main question was the security of uploading this to MongoDB. MongoDB, by default only let's access to its collections from localhost, and we want to access them with different IP addresses. That's why we should investigate if it is possible to add some exceptions so that accessing from a specific IP with a user and password would be possible.

About ELK stack, the only question is if we can send data (logs), from different servers to a specific one. Since most of our applications are in different servers. Another thing we need to look for, is how to store docker logs, since they are not stored in any path. Other logs we should consider are Keycloak, Tomcat, Apache2, MySQL, Kafka, Zookeeper...

Regarding Mongo

- MongoDB does not allow remote connections. By default, it only connects to the local interface restricting all other connections.

- There is a way to fix this, but all our servers are supposed to be on the same LAN. In this case, being on the same LAN would only have to link MongoDB with its own private IP interface.

- If we want remote access from a public interface, we can also add it to the configuration file.

- But, as it puts in the link below, it could be configured in the MongoDB, with iptables, by default everything that goes or receive or send, we drop it. And we only add INPUT OUTPUT rules to the IPs and ports that we want not.

Regarding ELK

- If we want to send logs from different servers, we must install Filebeat on each of them.

- You need to send the logs for all those servers for whom you want to visualize the logs on Kibana. You need to install the Filebeat agent on every server to send the logs over Elasticsearch. You need to use the same version of ELK (Elasticsearch, Logstash and Kibana) and Filebeat.

- Apparently, Beats is the family of loggers. Among which are Filebeat, Metricbeat, Packetbeat, Wingobeat, Auditbeat, Heartbeat, Functionbeat.

- Filebeat is the one that collects the logs, but others can be interesting, the ones that I have marked. Metricbeat that collects the metrics, performance so to speak, and Auditbeat that serves to audit the data.

Regarding docker logs

- Each docker daemon has its own log driver by default, which each container uses unless another log driver is configured.

- The easiest method is to configure our Logstash in such a way that by changing the log driver of the container to syslog, we receive them in Logstash. I do not know if I have understood well, I leave a link to the simple method that I found:

After a daily meeting, we discuss the following:

- MongoDB can be configured in order to let remote IP addresses connect to it. We only need to configure the *mongod.conf* file and enable the remote access. When we are going to install mongo, we will configure it.

- Very interesting about beats, we will start with Filebeat. We must determine which would be the best strategy. Install it in the same docker image where the service is running, or in a docker apart. Metricbeat and Auditbeat are also interesting, however we i think we should let them for later

We've read all the information given and reached the conclusion that it is possible and very simple to connect to MongoDB from a remote address changing some file lines from the MongoDB configuration.

Open MongoDB configuration file /etc/mongod.conf and change bindIp by adding required LAN interfaces or configure it to bind to all interfaces, for example:

```
# network interfaces

net:

  port: 27017

  bindIp:  127.0.0.1,192.168.0.100  #  Enter  0.0.0.0,::  to  bind  to  all
interfaces
```

So now the only thing left is to see where we need to install Filebeat, in the same docker where the application is running, or in a docker apart?

What we are going to do with MongoDB, is to bin all interfaces configuring the *mongod.conf* file, and set a user and a password in MongoDB, with a linked IP, so that only the one who access from a specific IP address with the correct credentials, can have access to our database **[12].**

Tcheck-Microservice

The main idea is to implement all those things which we've talked about in a working Microservice of Tcheck, which is one of the biggest applications of our enterprise. Through a mobile or a tablet, with this application, you can manage and review personal protective equipment. This tool is aimed at companies and manufacturers. The application can perform numerous functions, for example, generates review reports, has digitized information on the cycle of team life, receive alerts about reviews and expiration dates, and plan investments for equipment renovation.

First, we're going to implement an activity tracker, before and after implementing Javers. in order to determine if it slows downs the system or consumes many resources. But before that, it is mandatory to understand completely how this microservice works, and that's why we are going to have a meeting in which my colleagues who are working with this app, will explain me its functionality.

4.2.4 Fourth Iteration

We've been discussing with my colleagues the main steps to follow. I've understood completely how tcheck microservices run with docker. Tcheck is in one determinate server, which contains many containers, which run different tcheck services. We'll be using only one of them. The main questions raised in the meeting: Are the logs stored in Kibana? If the logs are erased from the docker container, could we visualize them in any part? With Filebeat we define the container id from which we want to take the logs, however, each time we update the app service, the container id changes. This means we must change the configuration file of Filebeat each time we update our application?

Now we are going to start talking about setting up all the things in order to run the project. We've created a new account in *pre.tcheck.tesicnor.com* with my credentials. We've made a SSH tunnel from the database of tcheck to my local machine, and installed postman in order to make tests without having a UI. The development of the project will be done in pre-production environment. Filebeat will be installed in the same server where the microservice runs, and ELK will be installed in a server apart.

To organize a little bit this, amount of information, the best idea is to start installing an activity tracer in our microservice, and checkout the performance differences with and without Javers. The activity tracer we'll use are called Sleuth and Zipkin.

Spring Cloud Sleuth

- Adding Sleuth dependency in the microservices, we endow the environment with an request identification automatic mechanism, since it adds some fields to them in order to identify them.
- The most important are **traceID** and **spanID**
- **Trace**: Set of spans which form a calls tree structure and creates the trace of the request.
- **Span**: Has start and end, with which can trace the time of response of a request.
- **Annotation**: Used to record an event in time. The most important are :
    - **cs**(Client sent) → The client made a request, start of the span.
    - **sr**(Server received) → The server received the request and starts its processing. ***timestampss - timestampcs = network latency***.
    - **ss**(Server sent) → Send the response from the server to the client. ***timestampss - timestampsr = server request processing time.***
    - **cr**(Client received) → End of span, the client received the response correctly from the server. ***timestampcs - timestampcr = total time of the request.***

Zipkin

- Zipkin leans on Sleuth, who has **two possible ways of sending traces** to the server, with **HTTP headers** , or other **messaging services**.
- The Zipkin client(equipment service) sends the trace in the HTTP header to the Zipkin Server.
- If any way to store it is defined, then it leaves it in memory. But we could use a Database such as **Elasticsearch** to store it.

After doing this little research, we tried to setup all the microservice in my local machine and run it. However, there is a problem. We don't know why, but it does not run properly, and we obtain an error. We've been an entire day trying to make it work, the two main tcheck developers and me, and we didn't find a way to make it work. That's why we've talked with the head of the Systems department, and he told the best thing we can do, is to format the pc entirely. So today we are going to format my local machine and install all the needed software from zero.

After formatting the computer, we installed all the necessary software in order to be able to build the project.

- Eclipse IDE for Java EE → For developing my project
- Postman → Make request tests
- MySQL Workbench → Local Database
- Java 8 (jdk and jre)
- Office 365 → Documentation
- Docker Desktop → Run the microservice
- KeePass → Manage all the keys
- MongoDB → Non-Relational database
- MTPuTTY → Connect to different servers
- PuTTY → Create SSH Tunnels
- Sublime Text → Lightweight editor
- TortoiseSVN → Remote Repository
- WinSCP → Navigate through our servers.

Now, we try to run the microservice again, and see if we have luck this time. Finally, after several tries, the microservice is running correctly.

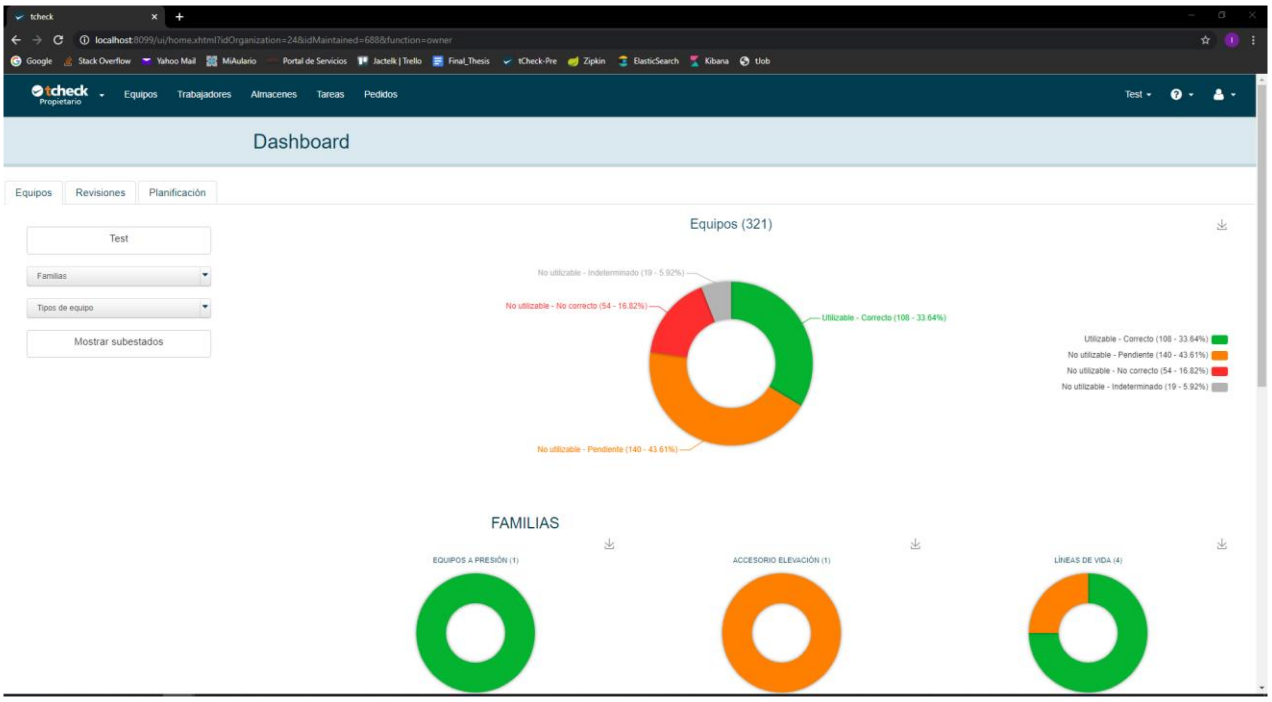

tcheck
localhost:8099/ui/home.xhtml?idOrganization=24&idMaintained=688&function=owner
Google
Stack Overflow
Yahoo Mail
MiAulario
Portal de Servicios
Jactelk | Trello
Final_Thesis
tCheck-Pre
Zipkin
ElasticSearch
Kibana
tJob
tcheck
Propietario
Equipos
Trabajadores
Almacenes
Tareas
Pedidos
Test
Dashboard
Equipos
Revisiones
Planificación
Test
Familias
Tipos de equipo
Mostrar subestados
Equipos (321)
No utilizable - Indeterminado (19 - 5.92%)
No utilizable - No correcto (54 - 16.82%)
Utilizable - Correcto (108 - 33.64%)
No utilizable - Pendiente (140 - 43.61%)
Utilizable - Correcto (108 - 33.64%) 🟩
No utilizable - Pendiente (140 - 43.61%) 🟧
No utilizable - No correcto (54 - 16.82%) 🟥
No utilizable - Indeterminado (19 - 5.92%) ⬜
FAMILIAS
EQUIPOS A PRESIÓN (1)
ACCESORIO ELEVACIÓN (1)
LÍNEAS DE VIDA (4)

Figure 6 - Tcheck Dashboard

First, let's think about how we can integrate Zipkin in our project. For that purpose, we're going to look at a Zipkin talk, where we are going to study the following points:

- How to correlate logs with Spring Cloud Sleuth?
- How to visualize latency with Spring Cloud Sleuth and Zipkin?
- How to use Pivotal Cloud Foundry to deploy and integrate apps with Spring Cloud Services and Zipkin?
- How to integrate Spring Cloud Sleuth and Zipkin in an existing app?
- How to investigate issues using PCF Metrics and Zipkin?

Tracing

- Span → Operation bounded in time
- Trace → Set of spans
- Logs → Things that happen in time
- Tags → Key value pair

Let’s think we have the following architecture:

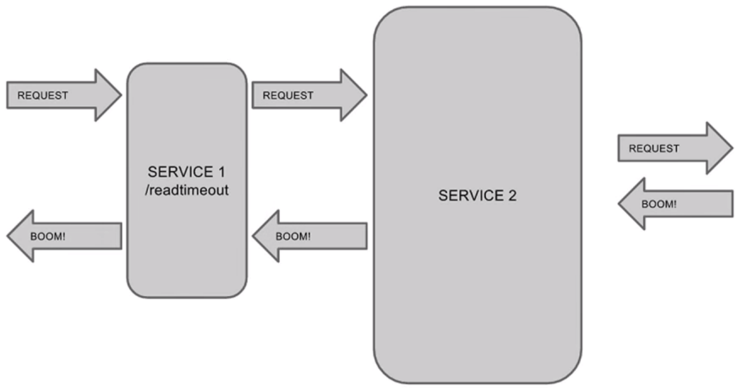


Figure 7 - Zipkin Explanation 1

Now we are going to create a Spring Boot Service:

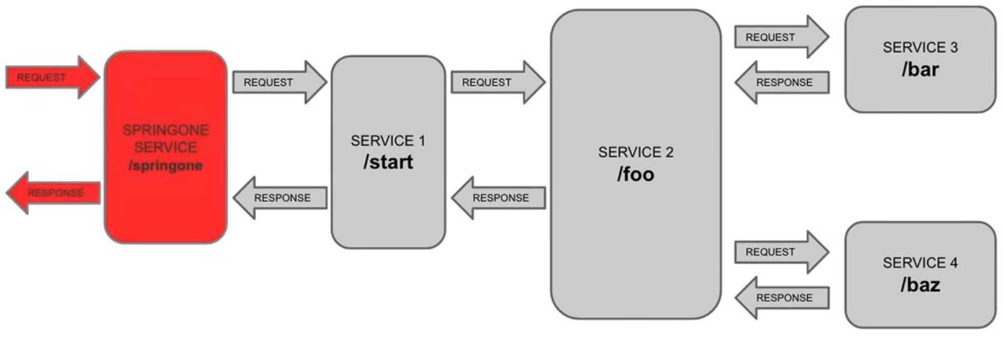
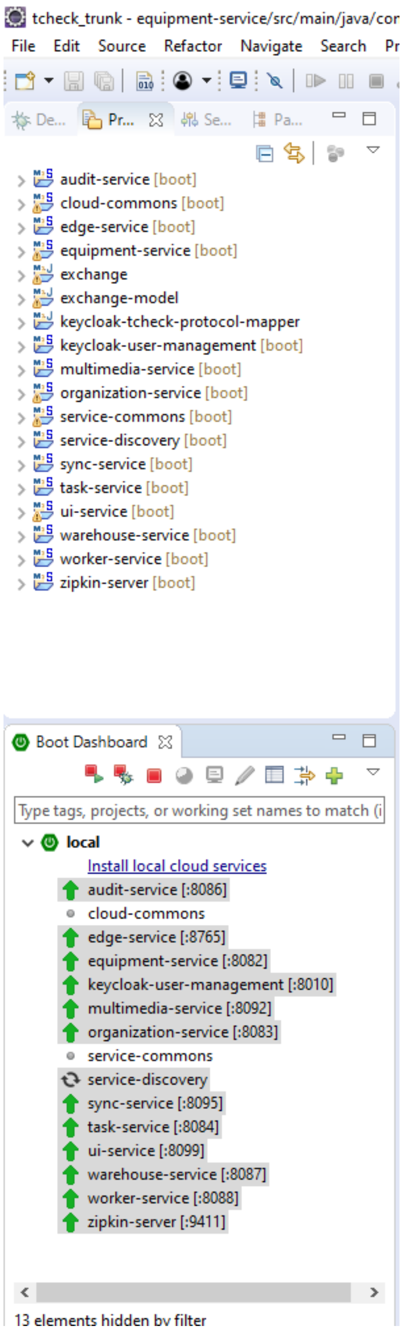


Figure 8 - Zipkin Explanation 2

Zipkin Implementation

Frist of all, we must understand that Zipkin is another microservice which will be running in tcheck, simultaneously with all the other microservices. This one will be the Zipkin Server, which by default runs in port 9411 **[16]**.


tcheck_trunk - equipment-service/src/main/java/cor

audit-service [boot]
cloud-commons [boot]
edge-service [boot]
equipment-service [boot]
exchange
exchange-model
keycloak-tcheck-protocol-mapper
keycloak-user-management [boot]
multimedia-service [boot]
organization-service [boot]
service-commons [boot]
service-discovery [boot]
sync-service [boot]
task-service [boot]
ui-service [boot]
warehouse-service [boot]
worker-service [boot]
zipkin-server [boot]

Boot Dashboard

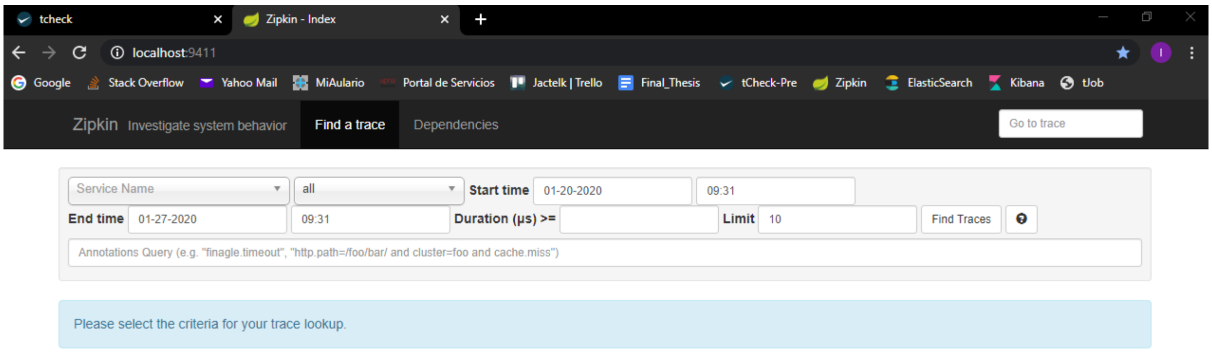
Type tags, projects, or working set names to match (i

local
Install local cloud services
audit-service [:8086]
cloud-commons
edge-service [:8765]
equipment-service [:8082]
keycloak-user-management [:8010]
multimedia-service [:8092]
organization-service [:8083]
service-commons
service-discovery
sync-service [:8095]
task-service [:8084]
ui-service [:8099]
warehouse-service [:8087]
worker-service [:8088]
zipkin-server [:9411]

13 elements hidden bv filter

Figure 9 - Tcheck Project

The microservice we are going to work with is the *equipment-service*. And it's here where we will add the Zipkin code from the client side. In order to send to the server all, the endpoint, and with this information, determine the performance and the time that it is needed to perform each action, with and without Javers.



Service Name | all | Start time 01-20-2020 09:31
End time 01-27-2020 09:31 | Duration (μs) >= | Limit 10

Annotations Query (e.g. "finagle.timeout", "http.path=/foo/bar/ and cluster=foo and cache.miss")

Please select the criteria for your trace lookup.

Figure 10 - Zipkin Page

After writing all the necessary code in the client side, we only obtain one endpoint in the ZipkinUI, and after a lot of time researching, we could not obtain any answer why that happens. We have also discussed with our colleagues, and no one could find the reason why that happens. After some days trying to solve this issue, we have chosen to start with the implementation of Javers, and when we finish that, return to the implementation of Zipkin, since I would just waste a lot of time searching for the problem.

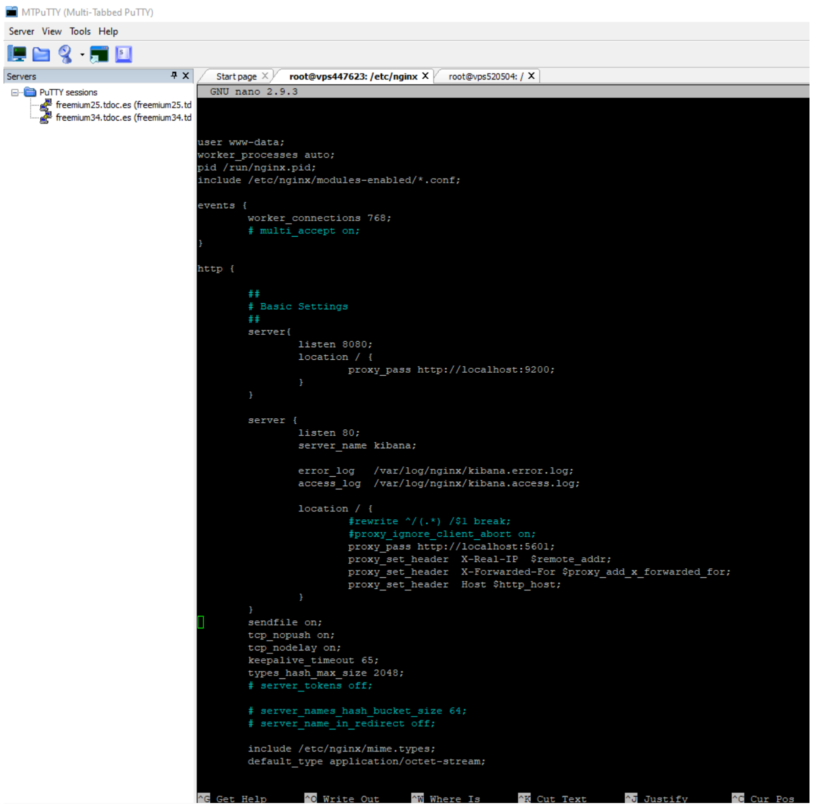
At this point, we implemented all the Javers framework in the *equipment-service.* Most things seemed to work find, but there were 2 exceptions which occur sometimes. The first one, happens when I create an equipment, Javers says it is an ENTITY ERROR with null id. The second one happens when we try to edit a certain object , we obtain a SNAPSHOT SERIALIZATION ERROR, which says there is a duplicated field. We were a little bit saturated with that, and decided to change completely, and to start with the Elastic Stack implementation in a server.

4.2.5 Fifth Iteration

The Systems development team from my company, gave us an empty OVH server, so we can use it to deploy all the things we needed to make the project work. Services such as Elasticsearch, Kibana, NGINX, MongoDB…

First, to access this server we need to create an SSH tunnel in order to work with it from my local machine. This was performed with Putty, and, we've used WinSCP to copy/delete files in the server. This could be done from the command line, but like that it is much easier. Remember that for the correct working of Tcheck, we have previously downloaded all Tcheck pre-production database to my local machine, so that we could work with it without any problem. In this new server, we need to install all the basic technologies, such as Java JDK for example. After that, we start with the installation of Elasticsearch and Kibana as services. This was done without much problems, however, after trying to access them, as freemium25.tdoc.es:9200 (Elasticsearch default port) and freemium25.tdoc.es:5601 (Kibana default port) , the browser says that this page is not reachable.

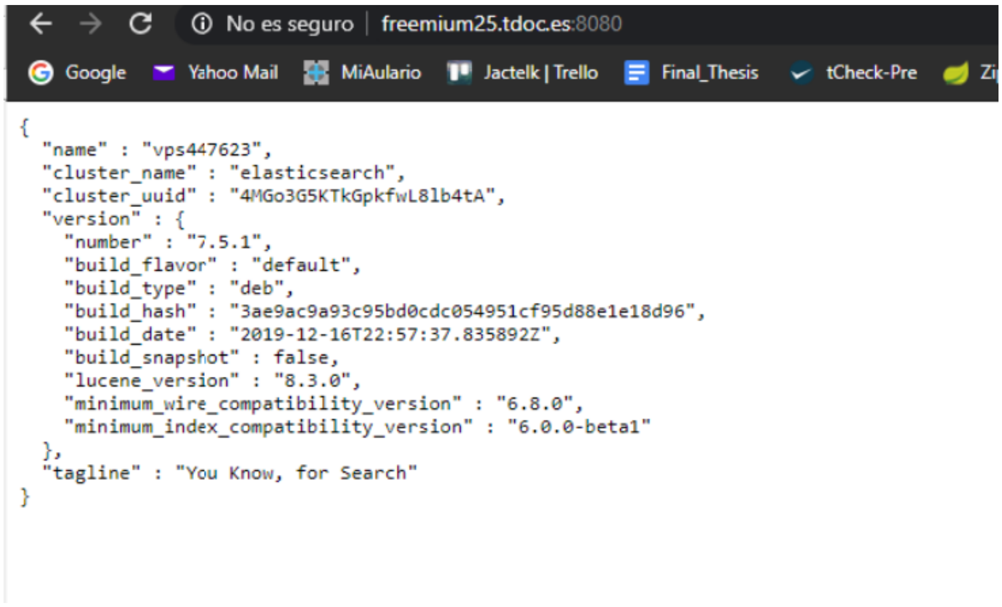
This is because I need a web serving service, such as NGINX for example, which we installed and configured in the following way in order to make my services Elasticsearch and Kibana accessible:


MTPuTTY (Multi-Tabbed PuTTY)
Server View Tools Help
Servers
PuTTY sessions
freemium25.tdoc.es (freemium25.td
freemium34.tdoc.es (freemium34.td
Start page
root@vps447623: /etc/nginx
root@vps520504: /
GNU nano 2.9.3

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##
        server{
                listen 8080;
                location / {
                        proxy_pass http://localhost:9200;
                }
        }

        server {
                listen 80;
                server_name kibana;

                error_log   /var/log/nginx/kibana.error.log;
                access_log  /var/log/nginx/kibana.access.log;

                location / {
                        #rewrite ^/(.*) /$1 break;
                        #proxy_ignore_client_abort on;
                        proxy_pass http://localhost:5601;
                        proxy_set_header  X-Real-IP  $remote_addr;
                        proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
                        proxy_set_header  Host $http_host;
                }
        }
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

^G Get Help     ^O Write Out     ^W Where Is     ^K Cut Text     ^J Justify     ^C Cur Pos
```

Figure 11 - NGINX Config

Now, with this configuration, by accessing to freemium25.tdoc.es:8080 we access to the Elasticsearch website:

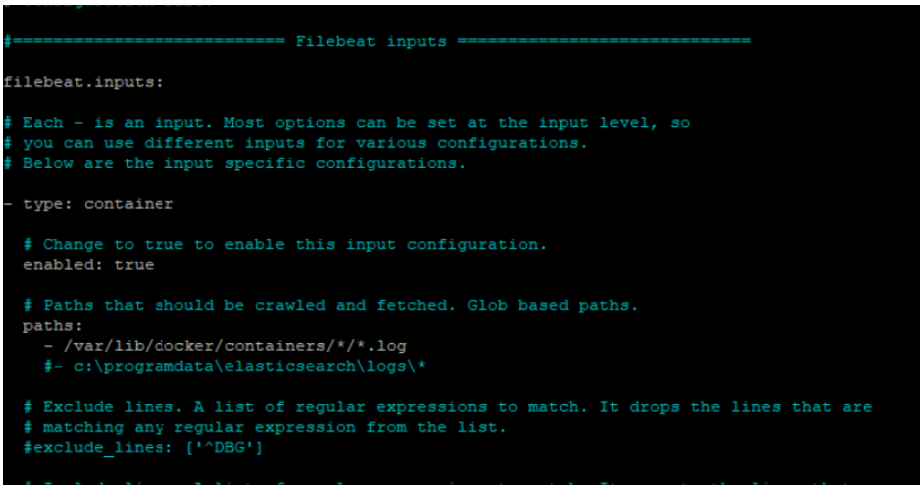


```
{
  "name" : "vps447623",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "4MGo3G5KTkGpkfwL8lb4tA",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 12 - Elasticsearch

With freemium25.tdoc.es we access directly to Kibana, since the default port for a website is 80, we do not need to write it in the URL. However, now, Elasticsearch is not receiving any kind of data, so Kibana cannot represent any data. This is because we now need to install Filebeat in the client from which we want to send the data **[13]**. In our case, Tcheck pre-production is deployed in server 34, freemium34.tdoc.es, so it is here where we will install **[14].**

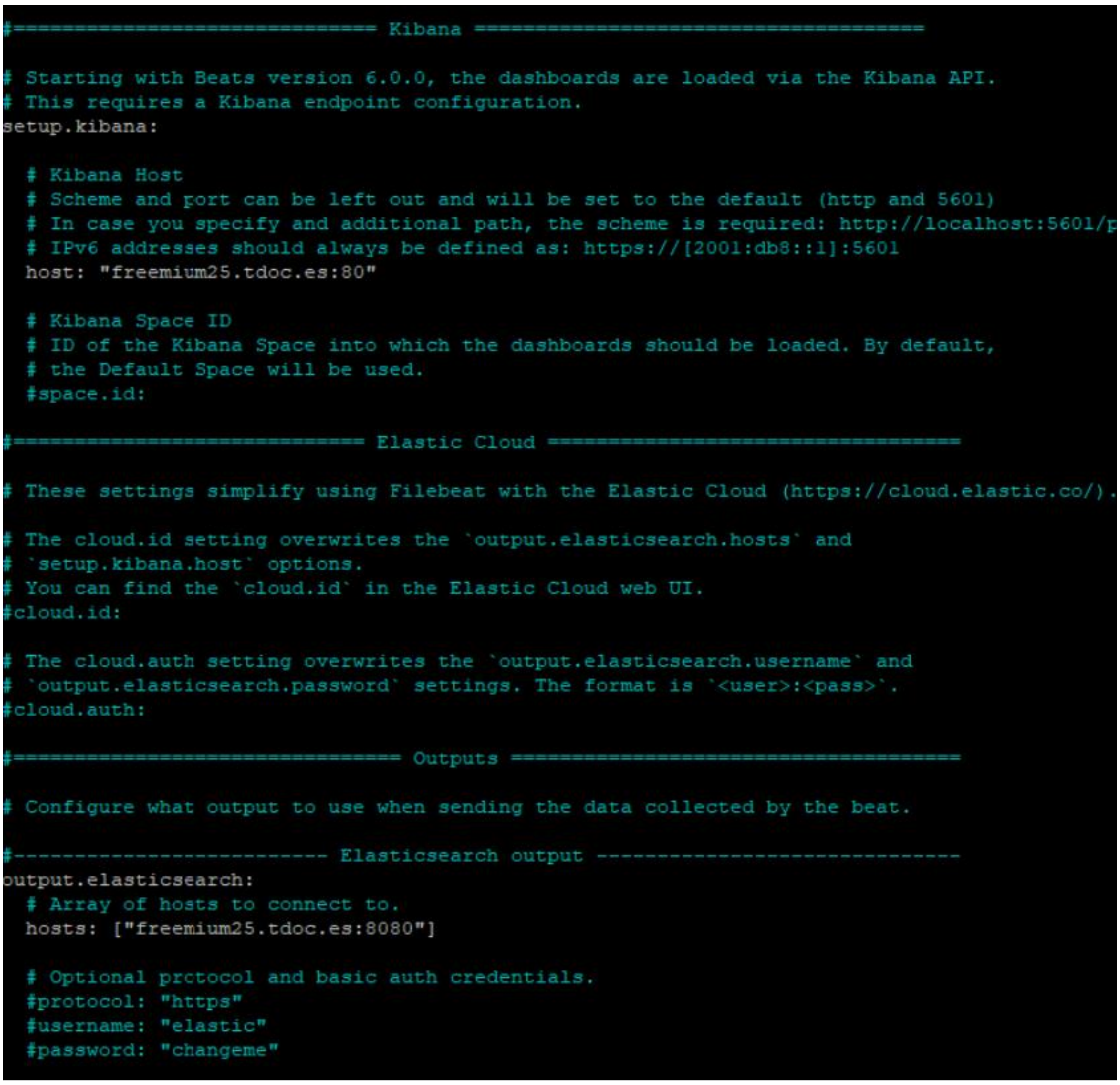
We just downloaded a tar.gz folder, decompress it, and execute it. This didn't work obviously, since I did not configure any Filebeat inputs, or Filebeat outputs, either the Kibana host. This must be configured in the *filebeat.yml* file, in the following way in order to make it work, and to interact with our Elasticsearch and Kibana hosts:



```
#========================== 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: container

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

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

  # Exclude lines. A list of regular expressions to match. It drops the lines that are
  # matching any regular expression from the list.
  #exclude_lines: ['^DBG']
```

Figure 13 - Filebeat Inputs

With the following path: /var/lib/docker/containers/*/*.log, we tell Filebeat where it should take the data from. We can indicate all the paths we want, there is no restriction. In our case, we are collecting all the .log files from all the docker containers in this service. Remember Tcheck runs in this server, so all those containers are microservices from Tcheck. But how Filebeat knows where to send all those logs? Configuring the Filebeat outputs to our hosts is the solution, and we do it in the following way:



```
#============================== Kibana =====================================

# Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the default (http and 5601)
  # In case you specify and additional path, the scheme is required: http://localhost:5601/p
  # IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
  host: "freemium25.tdoc.es:80"

  # Kibana Space ID
  # ID of the Kibana Space into which the dashboards should be loaded. By default,
  # the Default Space will be used.
  #space.id:

#============================= Elastic Cloud ==================================

# These settings simplify using Filebeat with the Elastic Cloud (https://cloud.elastic.co/).

# The cloud.id setting overwrites the `output.elasticsearch.hosts` and
# `setup.kibana.host` options.
# You can find the `cloud.id` in the Elastic Cloud web UI.
#cloud.id:

# The cloud.auth setting overwrites the `output.elasticsearch.username` and
# `output.elasticsearch.password` settings. The format is `<user>:<pass>`.
#cloud.auth:

#================================ Outputs =====================================

# Configure what output to use when sending the data collected by the beat.

#-------------------------- Elasticsearch output ------------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["freemium25.tdoc.es:8080"]

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

Figure 14 - Filebeat Outputs

With this config, the connection between Filebeat, Elasticsearch and Kibana works correctly. At this point, we organized a meeting with our my colleagues and our boss in order to show them how the stack is working, and seizing the opportunity , explain them how the project is progressing, the issues I found during the implementation, and to determine our future steps.

4.2.6 Sixth Iteration

All the department is happy with the big progress, since the Elastic Stack was completely implemented, and working correctly. However, there were some points we should think about. The first one, is to determine if all those logs that we send are stored somewhere in Elasticsearch or not. If we erase all those logs from the client side, are we going to continue visualizing them in Kibana or not? Also, it would be a good idea to deploy Filebeat as a container and not to execute it with the ./filebeat command.

From the other side, we have also talked about researching another service to measure performance since Zipkin is not working as expected. About Javers issues we decided to treat them later.

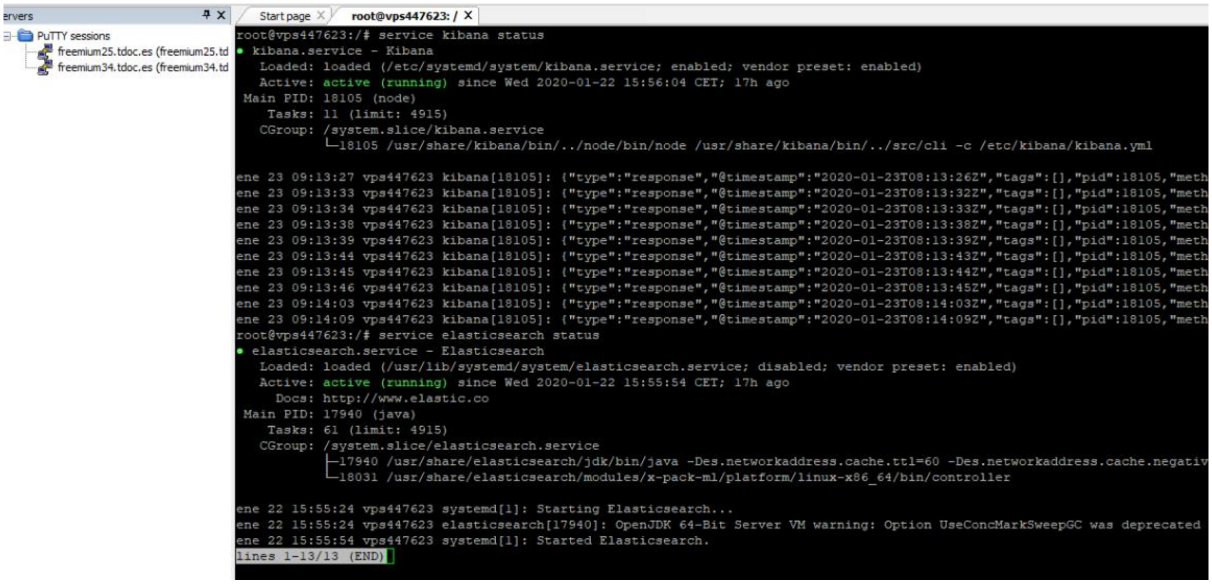
Our next step after this, is to deploy Filebeat in a Docker container, and to add this service to the docker-compose.yml file. We've done it in a moment and configured all the fields and variables correctly. However, for whatever reason it does not take the config file (filebeat.yml), and executes it with the default config file stored in the docker Filebeat image, even if we indicate in the service that the config file he must take is the one we tell him.

We've created the docker container and run it, it works, but the configuration is not correct, and that's why we decided to leave Filebeat as it was. Nevertheless, we realized that at that moment we execute Filebeat from an executable file, with the ./filebeat command. And when we exit from the server, it just stops. Obviously, we do not want that. We want Filebeat to work in the same way that Elasticsearch and Kibana work, as services. So, we deleted all the Filebeat folders in the tcheck server, and installed it as a service and configure it in the same way as we did with Elasticsearch and Kibana in the server 25, forcing them to run always. And if the server restarts because of whatever reason, those services will restart automatically.

But here comes another issue. After installing Filebeat service, it could not connect to Elasticsearch and Kibana, and the reason was that the version of those 2 last ones, where 6.x, and the one from Filebeat 7.x . The problem is that you cannot upgrade from version 6 to version 7, and that's why we were forced to delete all the services we've created in server 25 (Elasticsearch and Kibana services), and to reinstall them in the latest versions.

After doing that, and configuring all correctly, all seemed to work correctly, and here are some screenshots about all the services running, and the Elasticsearch and the data represented in Kibana:

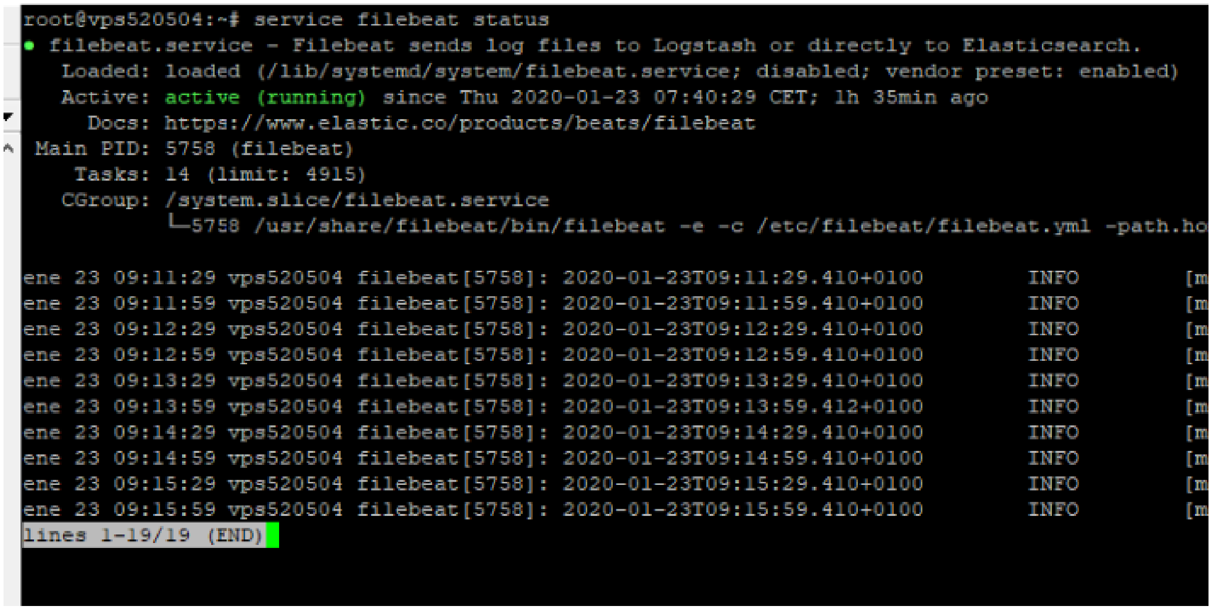
Server 25 where Kibana and Elasticsearch are deployed:


ervers
PuTTY sessions
freemium25.tdoc.es (freemium25.td
freemium34.tdoc.es (freemium34.td
Start page
root@vps447623: /

```
root@vps447623:/# service kibana status
● kibana.service - Kibana
   Loaded: loaded (/etc/systemd/system/kibana.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-01-22 15:56:04 CET; 17h ago
 Main PID: 18105 (node)
    Tasks: 11 (limit: 4915)
   CGroup: /system.slice/kibana.service
           └─18105 /usr/share/kibana/bin/../node/bin/node /usr/share/kibana/bin/../src/cli -c /etc/kibana/kibana.yml

ene 23 09:13:27 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:26Z","tags":[],"pid":18105,"meth
ene 23 09:13:33 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:32Z","tags":[],"pid":18105,"meth
ene 23 09:13:34 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:33Z","tags":[],"pid":18105,"meth
ene 23 09:13:38 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:38Z","tags":[],"pid":18105,"meth
ene 23 09:13:39 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:39Z","tags":[],"pid":18105,"meth
ene 23 09:13:44 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:43Z","tags":[],"pid":18105,"meth
ene 23 09:13:45 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:44Z","tags":[],"pid":18105,"meth
ene 23 09:13:46 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:13:45Z","tags":[],"pid":18105,"meth
ene 23 09:14:03 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:14:03Z","tags":[],"pid":18105,"meth
ene 23 09:14:09 vps447623 kibana[18105]: {"type":"response","@timestamp":"2020-01-23T08:14:09Z","tags":[],"pid":18105,"meth
root@vps447623:/# service elasticsearch status
● elasticsearch.service - Elasticsearch
   Loaded: loaded (/usr/lib/systemd/system/elasticsearch.service; disabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-01-22 15:55:54 CET; 17h ago
     Docs: http://www.elastic.co
 Main PID: 17940 (java)
    Tasks: 61 (limit: 4915)
   CGroup: /system.slice/elasticsearch.service
           ├─17940 /usr/share/elasticsearch/jdk/bin/java -Des.networkaddress.cache.ttl=60 -Des.networkaddress.cache.negativ
           └─18031 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/controller

ene 22 15:55:24 vps447623 systemd[1]: Starting Elasticsearch...
ene 22 15:55:24 vps447623 elasticsearch[17940]: OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated
ene 22 15:55:54 vps447623 systemd[1]: Started Elasticsearch.
lines 1-13/13 (END)
```

Figure 15 - Elasticsearch and Kibana Services

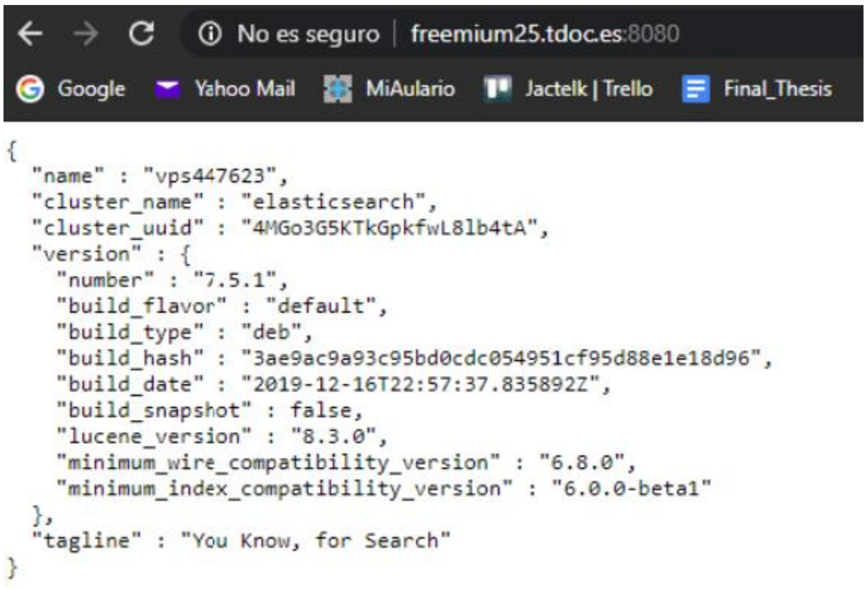
Server 34 where Filebeat is deployed:



```
root@vps520504:~# service filebeat status
● filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch.
   Loaded: loaded (/lib/systemd/system/filebeat.service; disabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-01-23 07:40:29 CET; 1h 35min ago
     Docs: https://www.elastic.co/products/beats/filebeat
 Main PID: 5758 (filebeat)
    Tasks: 14 (limit: 4915)
   CGroup: /system.slice/filebeat.service
           └─5758 /usr/share/filebeat/bin/filebeat -e -c /etc/filebeat/filebeat.yml -path.ho

ene 23 09:11:29 vps520504 filebeat[5758]: 2020-01-23T09:11:29.410+0100        INFO        [m
ene 23 09:11:59 vps520504 filebeat[5758]: 2020-01-23T09:11:59.410+0100        INFO        [m
ene 23 09:12:29 vps520504 filebeat[5758]: 2020-01-23T09:12:29.410+0100        INFO        [m
ene 23 09:12:59 vps520504 filebeat[5758]: 2020-01-23T09:12:59.410+0100        INFO        [m
ene 23 09:13:29 vps520504 filebeat[5758]: 2020-01-23T09:13:29.410+0100        INFO        [m
ene 23 09:13:59 vps520504 filebeat[5758]: 2020-01-23T09:13:59.412+0100        INFO        [m
ene 23 09:14:29 vps520504 filebeat[5758]: 2020-01-23T09:14:29.410+0100        INFO        [m
ene 23 09:14:59 vps520504 filebeat[5758]: 2020-01-23T09:14:59.410+0100        INFO        [m
ene 23 09:15:29 vps520504 filebeat[5758]: 2020-01-23T09:15:29.410+0100        INFO        [m
ene 23 09:15:59 vps520504 filebeat[5758]: 2020-01-23T09:15:59.410+0100        INFO        [m
lines 1-19/19 (END)
```

Figure 16 - Filebeat Service

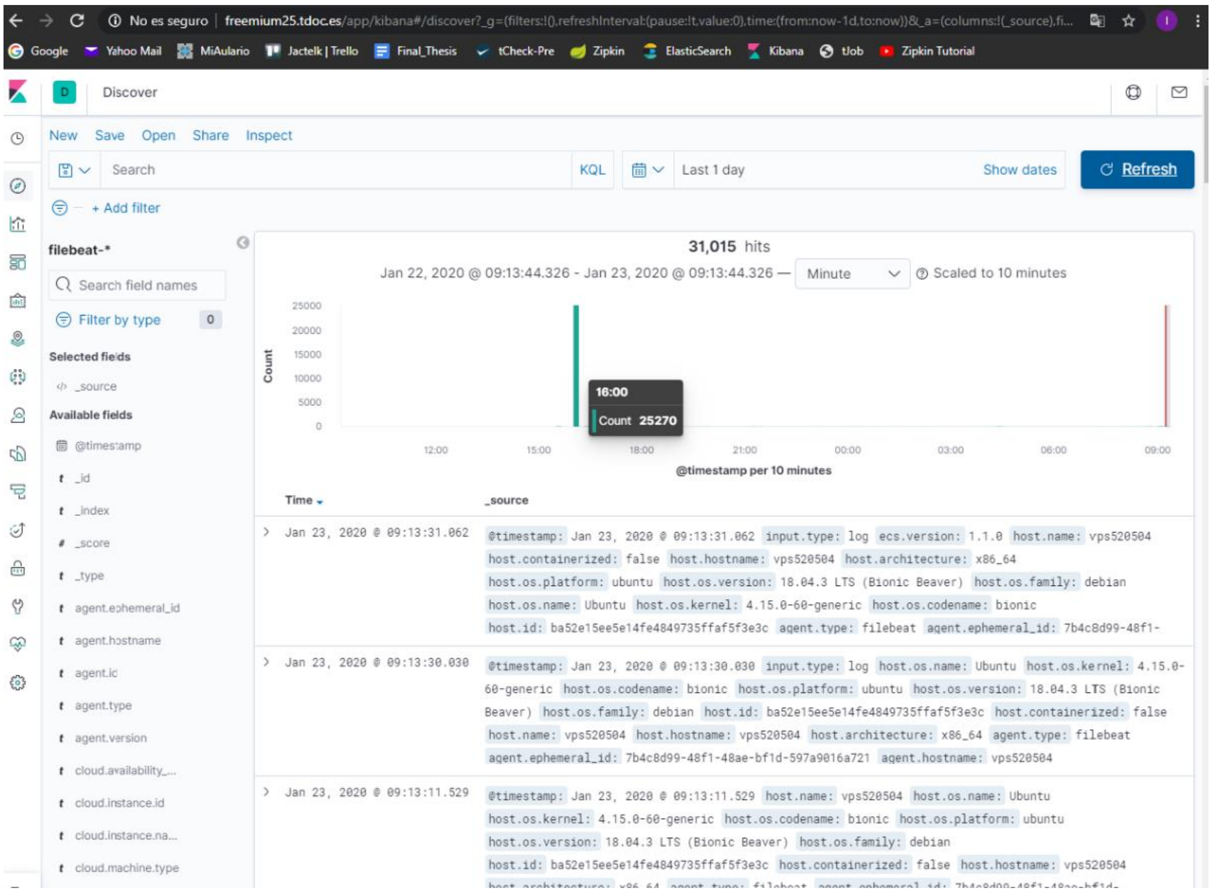
Elasticsearch running in freemium25.tdoc.es:8080



```
{
  "name" : "vps447623",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "4MGo3G5KTkGpkfwL8lb4tA",
  "version" : {
    "number" : "7.5.1",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "3ae9ac9a93c95bd0cdc054951cf95d88e1e18d96",
    "build_date" : "2019-12-16T22:57:37.835892Z",
    "build_snapshot" : false,
    "lucene_version" : "8.3.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 17 - Elasticsearch Server 25

Some Kibana screenshots



```
> Jan 23, 2020 @ 09:13:31.062   @timestamp: Jan 23, 2020 @ 09:13:31.062 input.type: log ecs.version: 1.1.0 host.name: vps520504
                                host.containerized: false host.hostname: vps520504 host.architecture: x86_64
                                host.os.platform: ubuntu host.os.version: 18.04.3 LTS (Bionic Beaver) host.os.family: debian
                                host.os.name: Ubuntu host.os.kernel: 4.15.0-60-generic host.os.codename: bionic
                                host.id: ba52e15ee5e14fe4849735ffaf5f3e3c agent.type: filebeat agent.ephemeral_id: 7b4c8d99-48f1-

> Jan 23, 2020 @ 09:13:30.030   @timestamp: Jan 23, 2020 @ 09:13:30.030 input.type: log host.os.name: Ubuntu host.os.kernel: 4.15.0-
                                60-generic host.os.codename: bionic host.os.platform: ubuntu host.os.version: 18.04.3 LTS (Bionic
                                Beaver) host.os.family: debian host.id: ba52e15ee5e14fe4849735ffaf5f3e3c host.containerized: false
                                host.name: vps520504 host.hostname: vps520504 host.architecture: x86_64 agent.type: filebeat
                                agent.ephemeral_id: 7b4c8d99-48f1-48ae-bf1d-597a9016a721 agent.hostname: vps520504

> Jan 23, 2020 @ 09:13:11.529   @timestamp: Jan 23, 2020 @ 09:13:11.529 host.name: vps520504 host.os.name: Ubuntu
                                host.os.kernel: 4.15.0-60-generic host.os.codename: bionic host.os.platform: ubuntu
                                host.os.version: 18.04.3 LTS (Bionic Beaver) host.os.family: debian
                                host.id: ba52e15ee5e14fe4849735ffaf5f3e3c host.containerized: false host.hostname: vps520504
```

Figure 18 - Kibana Server 25

In the image above, we can see all the logs that are arriving in real-time , and can see a detailed info about each one. We'll investigate later other functionalities that could be useful for our company.

For example, as a fast example , we can visualize from which microservice, most quantity of logs is arriving.

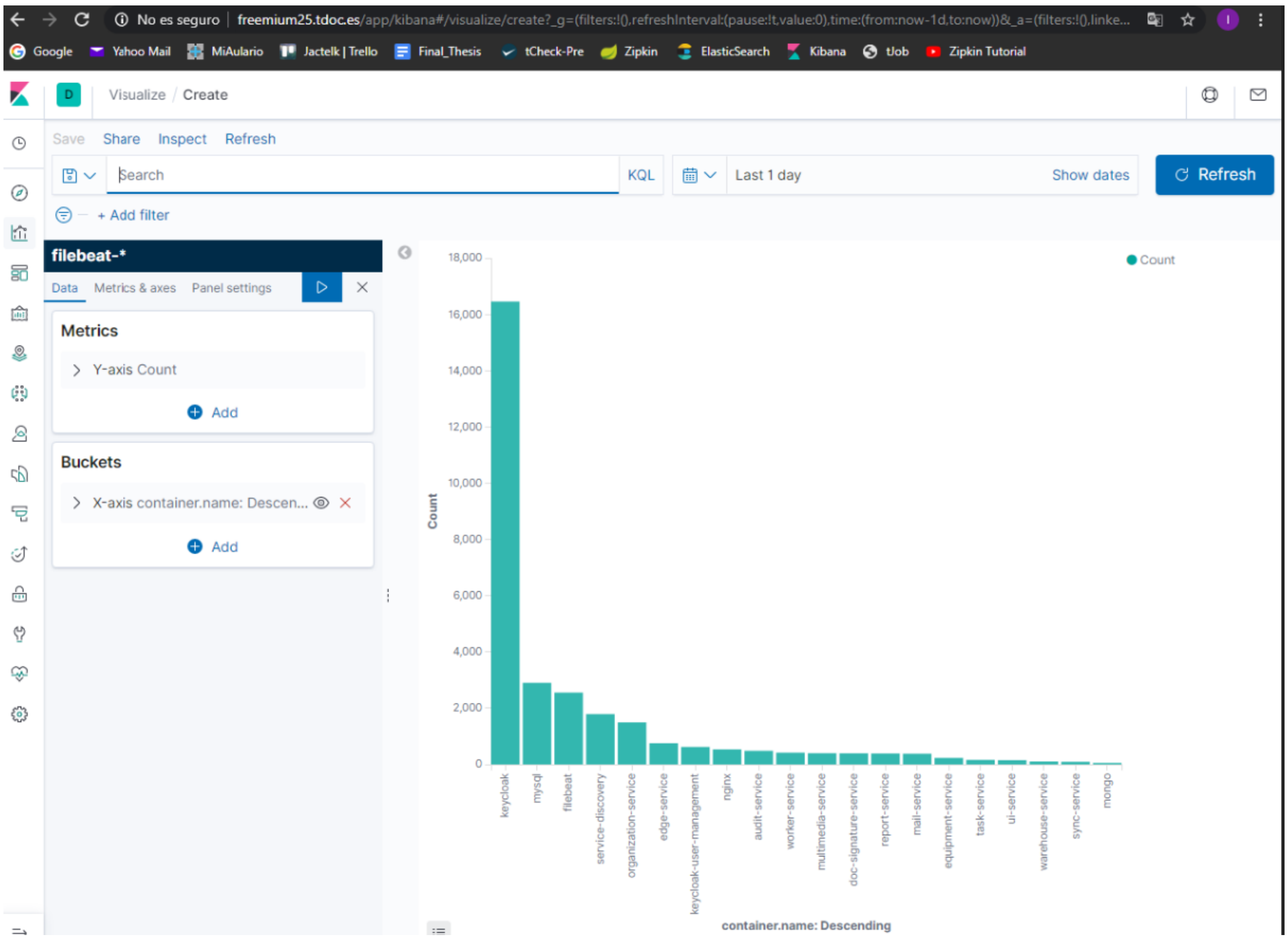
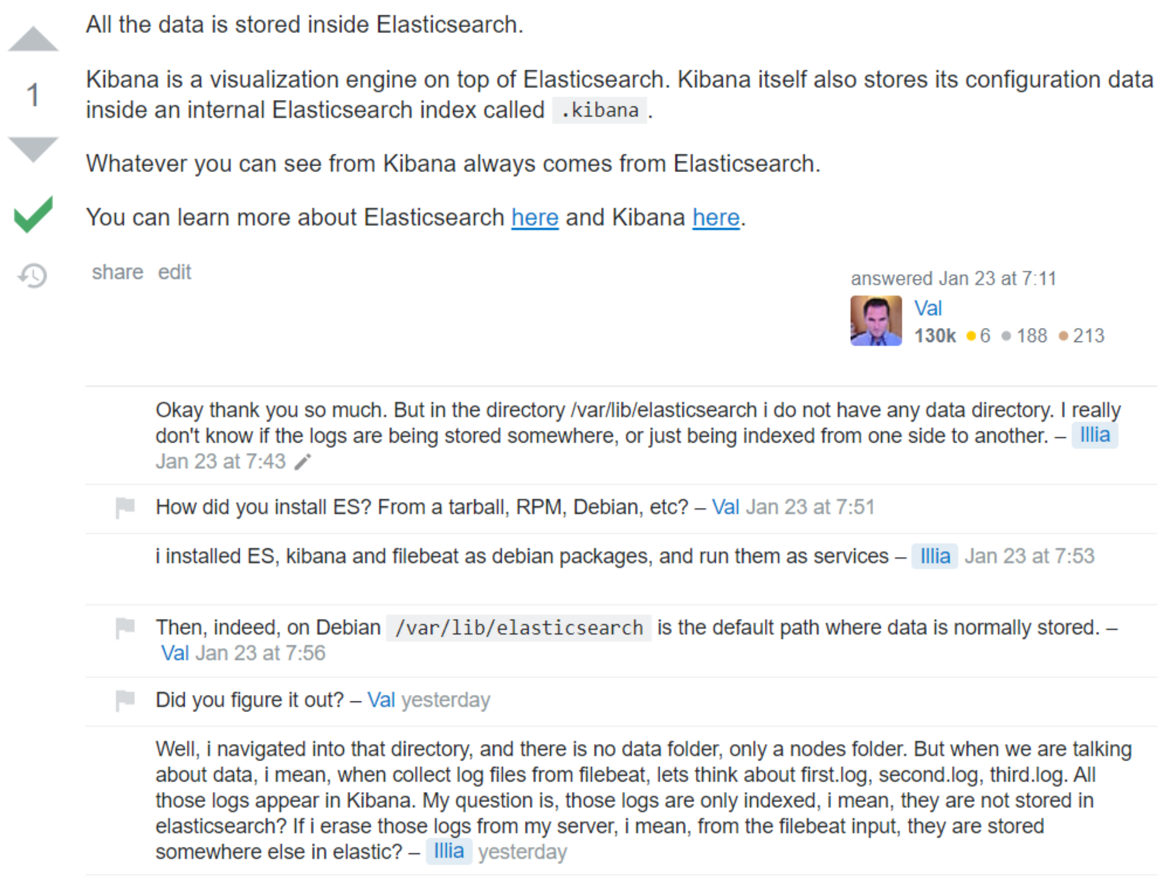


Figure 19 - Kibana Filter Container Name

Keycloak in our case is the one that most logs generate

Now the question is, where all those logs are stored? Or they are just indexed from the client side, and the server just represent them?

All the information we found in the internet did not helped us, so we decided to use StackOverflow to ask the question and this is what I got:



1

All the data is stored inside Elasticsearch.

Kibana is a visualization engine on top of Elasticsearch. Kibana itself also stores its configuration data inside an internal Elasticsearch index called `.kibana`.

Whatever you can see from Kibana always comes from Elasticsearch.

You can learn more about Elasticsearch here and Kibana here.

share edit

answered Jan 23 at 7:11
Val
130k ●6 ●188 ●213

Okay thank you so much. But in the directory /var/lib/elasticsearch i do not have any data directory. I really don't know if the logs are being stored somewhere, or just being indexed from one side to another. – Illia Jan 23 at 7:43

How did you install ES? From a tarball, RPM, Debian, etc? – Val Jan 23 at 7:51

i installed ES, kibana and filebeat as debian packages, and run them as services – Illia Jan 23 at 7:53

Then, indeed, on Debian `/var/lib/elasticsearch` is the default path where data is normally stored. – Val Jan 23 at 7:56

Did you figure it out? – Val yesterday

Well, i navigated into that directory, and there is no data folder, only a nodes folder. But when we are talking about data, i mean, when collect log files from filebeat, lets think about first.log, second.log, third.log. All those logs appear in Kibana. My question is, those logs are only indexed, i mean, they are not stored in elasticsearch? If i erase those logs from my server, i mean, from the filebeat input, they are stored somewhere else in elastic? – Illia yesterday

Figure 20 - Stackoverflow Elastic Stack

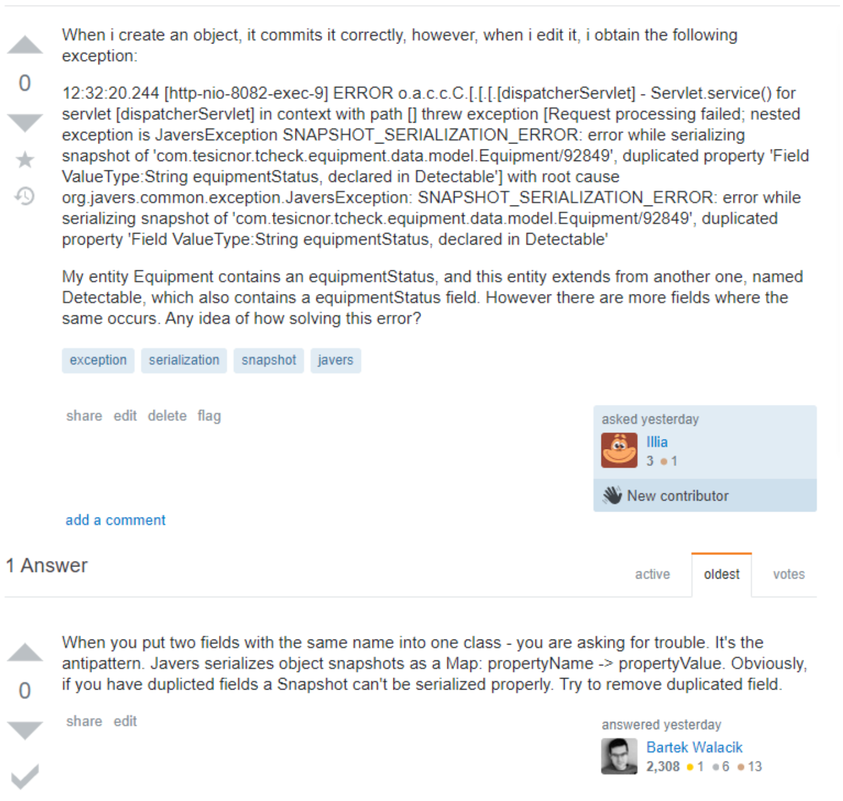
The thing is that in my /var/lib/elasticsearch path, there isn't any data folder. To solve this issue and to not waste a lot of time with this question, we decided leaving these services running some days, and look if the server memory usage increases. If it does, it means that all those logs are stored somewhere in Elasticsearch, if not, it would mean that those logs are not stored, and just being indexed. So, from this moment, we finish developing the Elastic Stack, and will try to make Javers work in my machine, send all the Javers commits to a mongo which we need to deploy in the server 25, where Kibana and Elasticsearch are running. With that, we would finish the project implementation.

4.2.7 Seventh Iteration

Remember there were two errors here. The first one was the ENTITY INSTANCE WITH NULL ID. After some time researching, we found the solution. In our equipment microservice we use hibernate. If you are using Hibernate, there is common problem with lazy loading proxies. For certain queries, hibernate doesn't return the real domain objects but dynamic proxy objects which are essentially empty (hence null ID). This technique maybe looks smart but makes your objects garbage until they are initialized by Hibernate. Javers provides HibernateUnproxyObjectAccessHook which does the cleaning: initializing and un-proxying of your domain objects.

JaversBuilder.javers().withObjectAccessHook(new HibernateUnproxyObjectAccessHook()).build()

Adding those lines above in JaversBuilder, we solve this issue, since now , the object will be committed to the Javers repository, only when it will be completely created.
Now the second problem is when we try to update or edit any equipment, we get a SNAPSHOT SERIALIZATION ERROR. We really didn't find any info in the internet, so we've posted a new question in StackOverflow and got a response from the Javers creator. Here is what he said:



When i create an object, it commits it correctly, however, when i edit it, i obtain the following exception:

12:32:20.244 [http-nio-8082-exec-9] ERROR o.a.c.c.C.[.[.[.[dispatcherServlet] - Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is JaversException SNAPSHOT_SERIALIZATION_ERROR: error while serializing snapshot of 'com.tesicnor.tcheck.equipment.data.model.Equipment/92849', duplicated property 'Field ValueType:String equipmentStatus, declared in Detectable'] with root cause org.javers.common.exception.JaversException: SNAPSHOT_SERIALIZATION_ERROR: error while serializing snapshot of 'com.tesicnor.tcheck.equipment.data.model.Equipment/92849', duplicated property 'Field ValueType:String equipmentStatus, declared in Detectable'

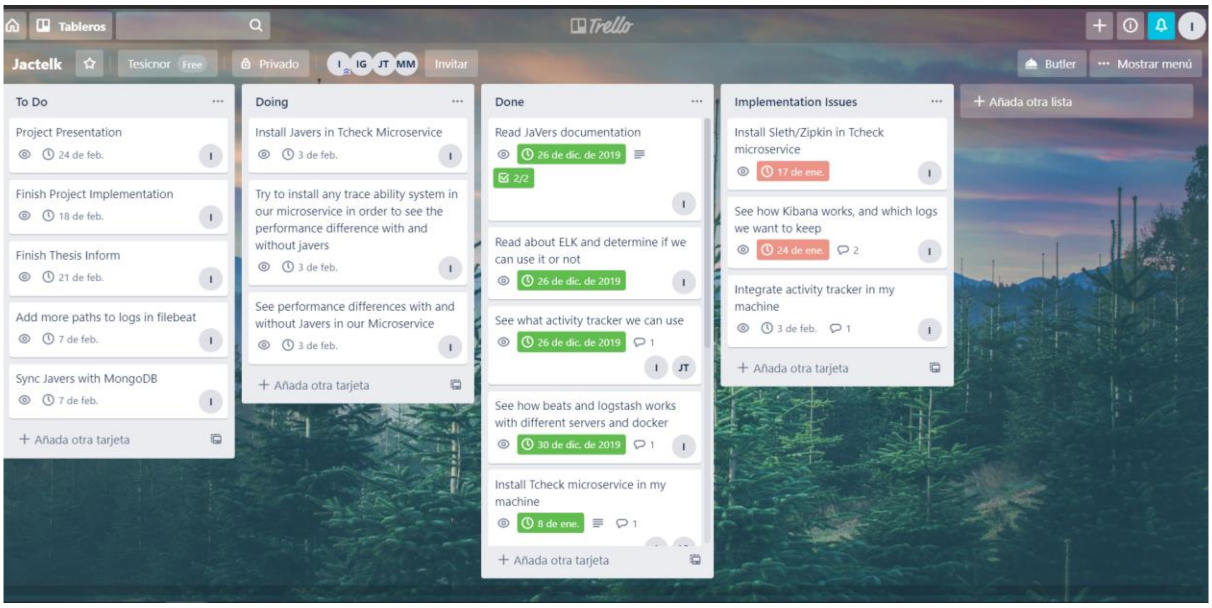
My entity Equipment contains an equipmentStatus, and this entity extends from another one, named Detectable, which also contains a equipmentStatus field. However there are more fields where the same occurs. Any idea of how solving this error?

When you put two fields with the same name into one class - you are asking for trouble. It's the antipattern. Javers serializes object snapshots as a Map: propertyName -> propertyValue. Obviously, if you have duplicted fields a Snapshot can't be serialized properly. Try to remove duplicated field.

Figure 21 - StackOverflow Javers

I think he did not understand well the problem. Since our class does not have any duplicated field. However, we will try to solve this issue, and like that, Javers implementation would be done, and only sending all this data to MongoDB server will be the last task to perform from our side.

Now, this is how Trello looks:



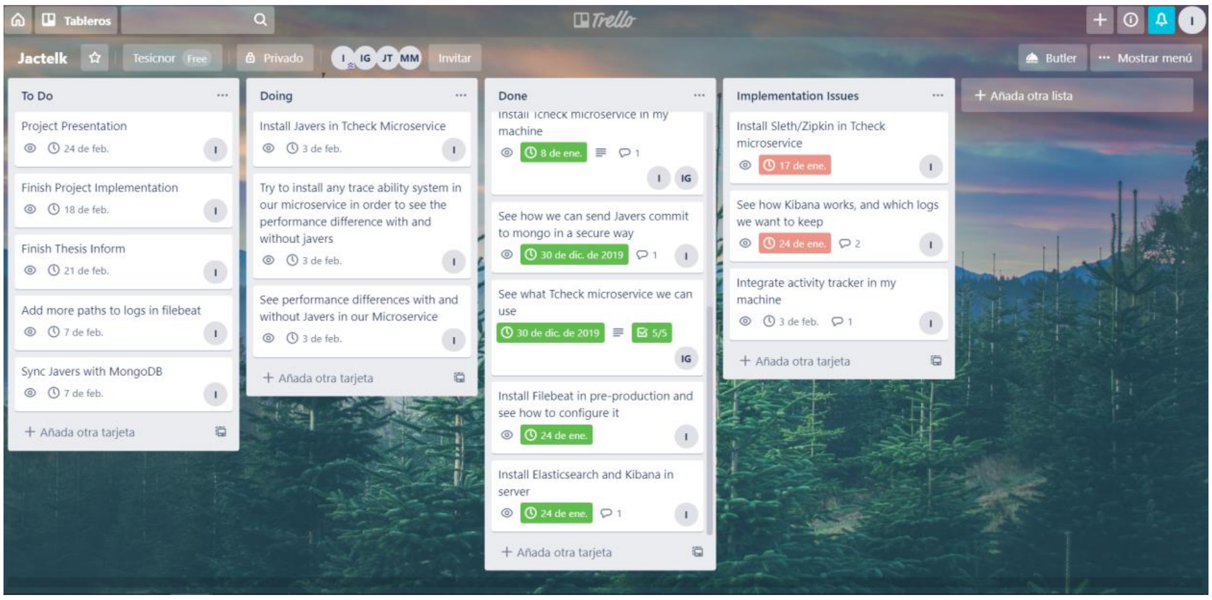
To Do
- Project Presentation — 24 de feb.
- Finish Project Implementation — 18 de feb.
- Finish Thesis Inform — 21 de feb.
- Add more paths to logs in filebeat — 7 de feb.
- Sync Javers with MongoDB — 7 de feb.
- + Añada otra tarjeta

Doing
- Install Javers in Tcheck Microservice — 3 de feb.
- Try to install any trace ability system in our microservice in order to see the performance difference with and without javers — 3 de feb.
- See performance differences with and without Javers in our Microservice — 3 de feb.
- + Añada otra tarjeta

Done
- Read JaVers documentation — 26 de dic. de 2019 — 2/2
- Read about ELK and determine if we can use it or not — 26 de dic. de 2019
- See what activity tracker we can use — 26 de dic. de 2019 — 1
- See how beats and logstash works with different servers and docker — 30 de dic. de 2019 — 1
- Install Tcheck microservice in my machine — 8 de ene. — 1
- + Añada otra tarjeta

Implementation Issues
- Install Sleth/Zipkin in Tcheck microservice — 17 de ene.
- See how Kibana works, and which logs we want to keep — 24 de ene. — 2
- Integrate activity tracker in my machine — 3 de feb. — 1
- + Añada otra tarjeta

Figure 22 - Trello Dashboard 1



To Do
- Project Presentation — 24 de feb.
- Finish Project Implementation — 18 de feb.
- Finish Thesis Inform — 21 de feb.
- Add more paths to logs in filebeat — 7 de feb.
- Sync Javers with MongoDB — 7 de feb.
- + Añada otra tarjeta

Doing
- Install Javers in Tcheck Microservice — 3 de feb.
- Try to install any trace ability system in our microservice in order to see the performance difference with and without javers — 3 de feb.
- See performance differences with and without Javers in our Microservice — 3 de feb.
- + Añada otra tarjeta

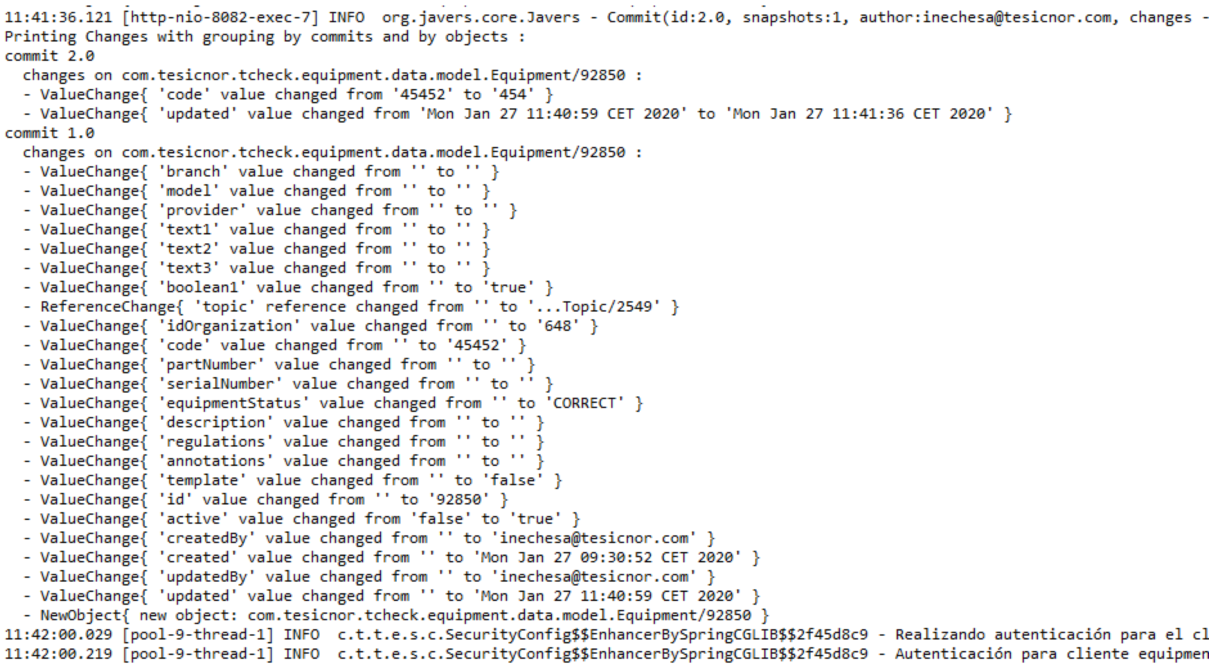
Done
- Install Tcheck microservice in my machine — 8 de ene. — 1
- See how we can send Javers commit to mongo in a secure way — 30 de dic. de 2019 — 1
- See what Tcheck microservice we can use — 30 de dic. de 2019 — 5/5
- Install Filebeat in pre-production and see how to configure it — 24 de ene.
- Install Elasticsearch and Kibana in server — 24 de ene. — 1
- + Añada otra tarjeta

Implementation Issues
- Install Sleth/Zipkin in Tcheck microservice — 17 de ene.
- See how Kibana works, and which logs we want to keep — 24 de ene. — 2
- Integrate activity tracker in my machine — 3 de feb. — 1
- + Añada otra tarjeta

Figure 23 - Trello Dashboard 2

Solving bugs

After inspecting the code, we realized that in the Equipment class we have a class with an equipmentStatus field, but in the same class, in the Equipment constructor we use the equipmentStatus from the builder. The thing is that Equipment class extends from Detectable, which contains also an equipmentStatus field. Javers is confused with this combination and says equipmentStatus field is duplicated. To solve this issue, in the Equipment constructor, we've changed *this.equipmentStatus = builder.equipmentStatus;* with *this.setEquipmentStatus(builder.equipmentStatus);* in order to prevent this duplication field error. With this change, the issue is finally solved, and the creation and edition of equipment is working correctly with Javers.

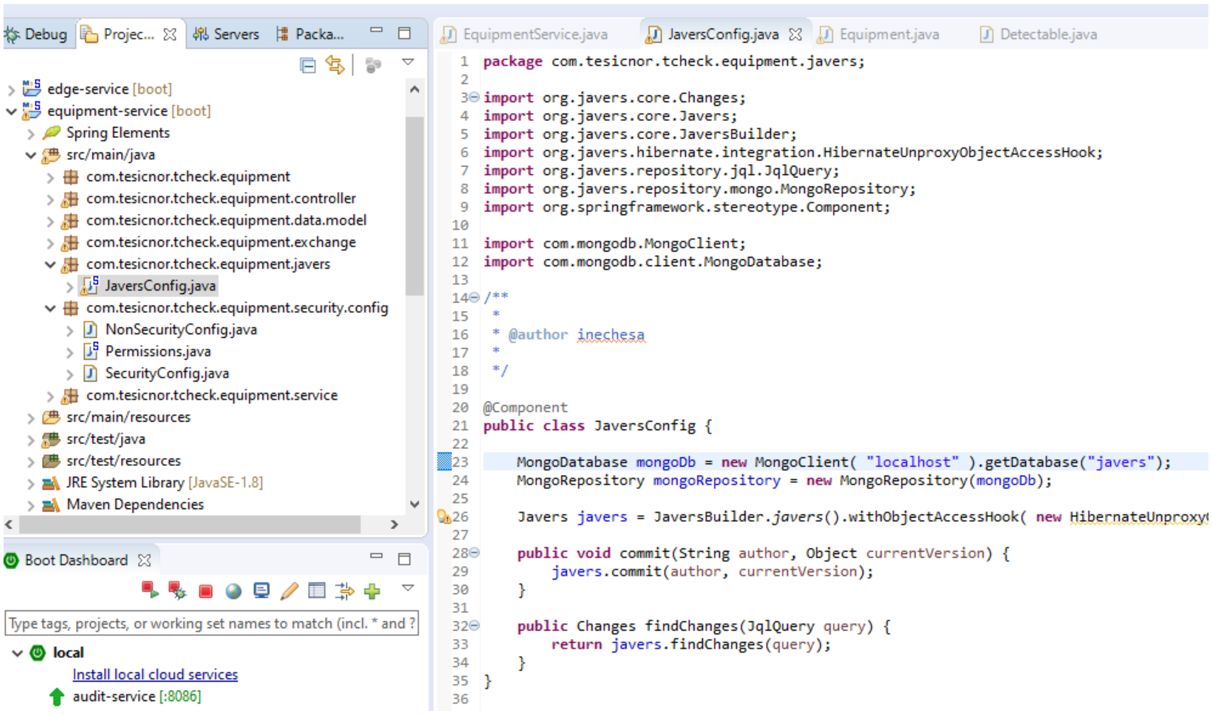


```
11:41:36.121 [http-nio-8082-exec-7] INFO  org.javers.core.Javers - Commit(id:2.0, snapshots:1, author:inechesa@tesicnor.com, changes -
Printing Changes with grouping by commits and by objects :
commit 2.0
  changes on com.tesicnor.tcheck.equipment.data.model.Equipment/92850 :
  - ValueChange{ 'code' value changed from '45452' to '454' }
  - ValueChange{ 'updated' value changed from 'Mon Jan 27 11:40:59 CET 2020' to 'Mon Jan 27 11:41:36 CET 2020' }
commit 1.0
  changes on com.tesicnor.tcheck.equipment.data.model.Equipment/92850 :
  - ValueChange{ 'branch' value changed from '' to '' }
  - ValueChange{ 'model' value changed from '' to '' }
  - ValueChange{ 'provider' value changed from '' to '' }
  - ValueChange{ 'text1' value changed from '' to '' }
  - ValueChange{ 'text2' value changed from '' to '' }
  - ValueChange{ 'text3' value changed from '' to '' }
  - ValueChange{ 'boolean1' value changed from '' to 'true' }
  - ReferenceChange{ 'topic' reference changed from '' to '...Topic/2549' }
  - ValueChange{ 'idOrganization' value changed from '' to '648' }
  - ValueChange{ 'code' value changed from '' to '45452' }
  - ValueChange{ 'partNumber' value changed from '' to '' }
  - ValueChange{ 'serialNumber' value changed from '' to '' }
  - ValueChange{ 'equipmentStatus' value changed from '' to 'CORRECT' }
  - ValueChange{ 'description' value changed from '' to '' }
  - ValueChange{ 'regulations' value changed from '' to '' }
  - ValueChange{ 'annotations' value changed from '' to '' }
  - ValueChange{ 'template' value changed from '' to 'false' }
  - ValueChange{ 'id' value changed from '' to '92850' }
  - ValueChange{ 'active' value changed from 'false' to 'true' }
  - ValueChange{ 'createdBy' value changed from '' to 'inechesa@tesicnor.com' }
  - ValueChange{ 'created' value changed from '' to 'Mon Jan 27 09:30:52 CET 2020' }
  - ValueChange{ 'updatedBy' value changed from '' to 'inechesa@tesicnor.com' }
  - ValueChange{ 'updated' value changed from '' to 'Mon Jan 27 11:40:59 CET 2020' }
  - NewObject{ new object: com.tesicnor.tcheck.equipment.data.model.Equipment/92850 }
11:42:00.029 [pool-9-thread-1] INFO  c.t.t.e.s.c.SecurityConfig$$EnhancerBySpringCGLIB$$2f45d8c9 - Realizando autenticación para el cl
11:42:00.219 [pool-9-thread-1] INFO  c.t.t.e.s.c.SecurityConfig$$EnhancerBySpringCGLIB$$2f45d8c9 - Autenticación para cliente equipmen
```

Now the only thing left is to send all those commit to MongoDB, since now they are being stored in Javers Temporal Repository. Firstly, we're going to mount it locally, and when it works correctly, we will deploy it in the same server where Kibana and Elasticsearch are running, in the server 25.

Mounting MongoDB locally

At this point, remember that Javers instance is declared in the *EquipmentService* class. It means that in each class we want to use our Javers instance, we must initialize a new one each time. Same with mongo, which need to be added in the Javers builder in order to connect to a Mongo Client. Therefore, we decided to create a new package in which class we would have a Javers component which could be used in any class in the service, without having to configure it in each class we want to use it.


Debug
Projec...
Servers
Packa...
edge-service [boot]
equipment-service [boot]
Spring Elements
src/main/java
com.tesicnor.tcheck.equipment
com.tesicnor.tcheck.equipment.controller
com.tesicnor.tcheck.equipment.data.model
com.tesicnor.tcheck.equipment.exchange
com.tesicnor.tcheck.equipment.javers
JaversConfig.java
com.tesicnor.tcheck.equipment.security.config
NonSecurityConfig.java
Permissions.java
SecurityConfig.java
com.tesicnor.tcheck.equipment.service
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.8]
Maven Dependencies
Boot Dashboard
Type tags, projects, or working set names to match (incl. * and ?
local
Install local cloud services
audit-service [:8086]
EquipmentService.java
JaversConfig.java
Equipment.java
Detectable.java

```java
package com.tesicnor.tcheck.equipment.javers;

import org.javers.core.Changes;
import org.javers.core.Javers;
import org.javers.core.JaversBuilder;
import org.javers.hibernate.integration.HibernateUnproxyObjectAccessHook;
import org.javers.repository.jql.JqlQuery;
import org.javers.repository.mongo.MongoRepository;
import org.springframework.stereotype.Component;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

/**
 *
 * @author inechesa
 *
 */

@Component
public class JaversConfig {

    MongoDatabase mongoDb = new MongoClient( "localhost" ).getDatabase("javers");
    MongoRepository mongoRepository = new MongoRepository(mongoDb);

    Javers javers = JaversBuilder.javers().withObjectAccessHook( new HibernateUnproxy

    public void commit(String author, Object currentVersion) {
        javers.commit(author, currentVersion);
    }

    public Changes findChanges(JqlQuery query) {
        return javers.findChanges(query);
    }
}

```

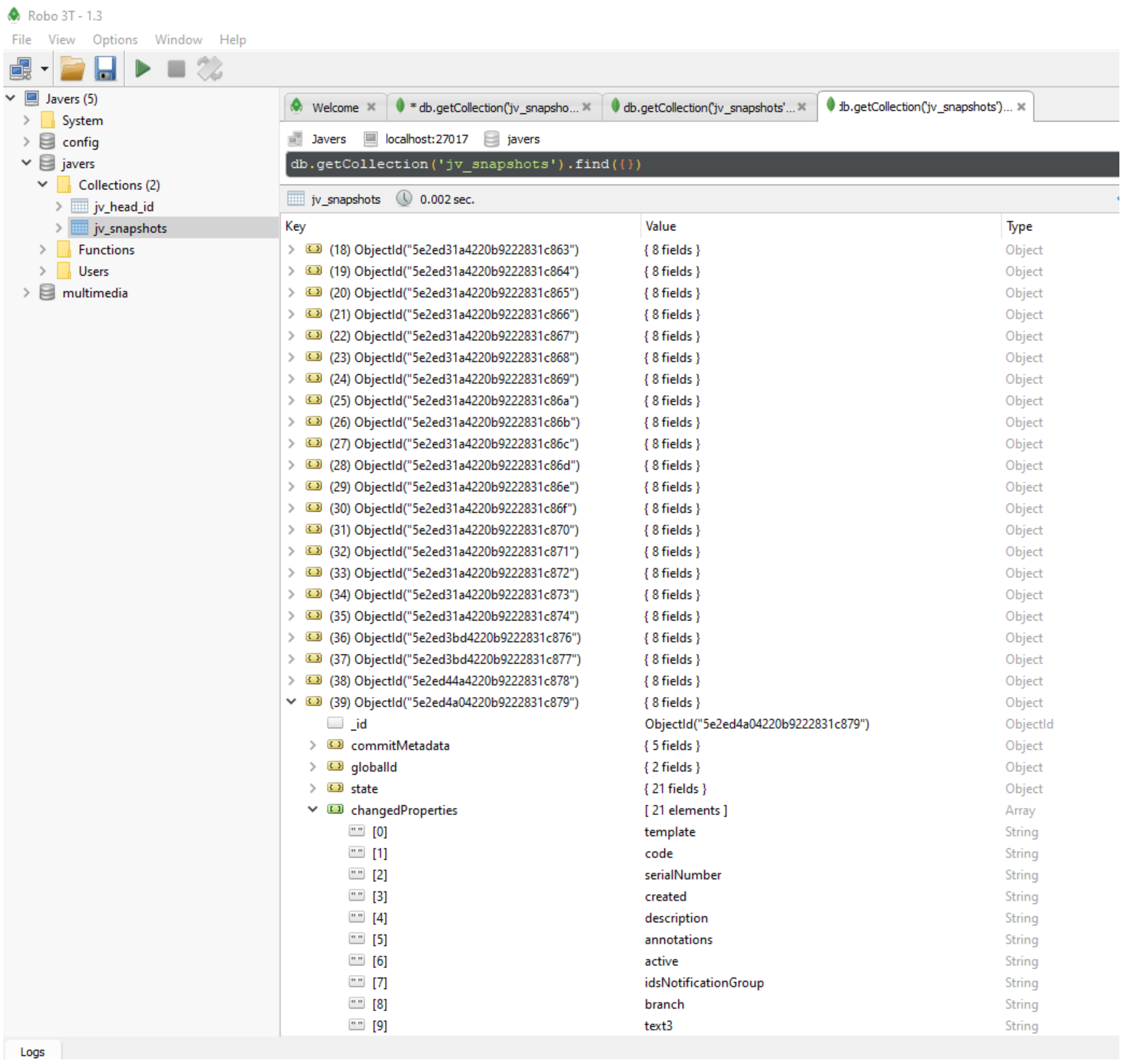
Now, in each class we want to use Javers, we just need to set the JaversConfig, and add the @Autowired annotation to make it work properly.

```java
private JaversConfig javers;

@Autowired
public void setJaversConfig(JaversConfig javers) {
    this.javers = javers;
}
```

Like that, we can use all the Javers methods we have implemented in the JaversConfig Class.

As you can see in the images above, we already configured a local MongoDB, and all he commits from now, are going to be stored in my local Javers database. After doing some tests, it works correct.

In order to visualize the data stored in local mongo, we use the Robo 3T tool, which is very useful for this kind of situations, when we don't have an UI to visualize the Data:



```
db.getCollection('jv_snapshots').find({})
```

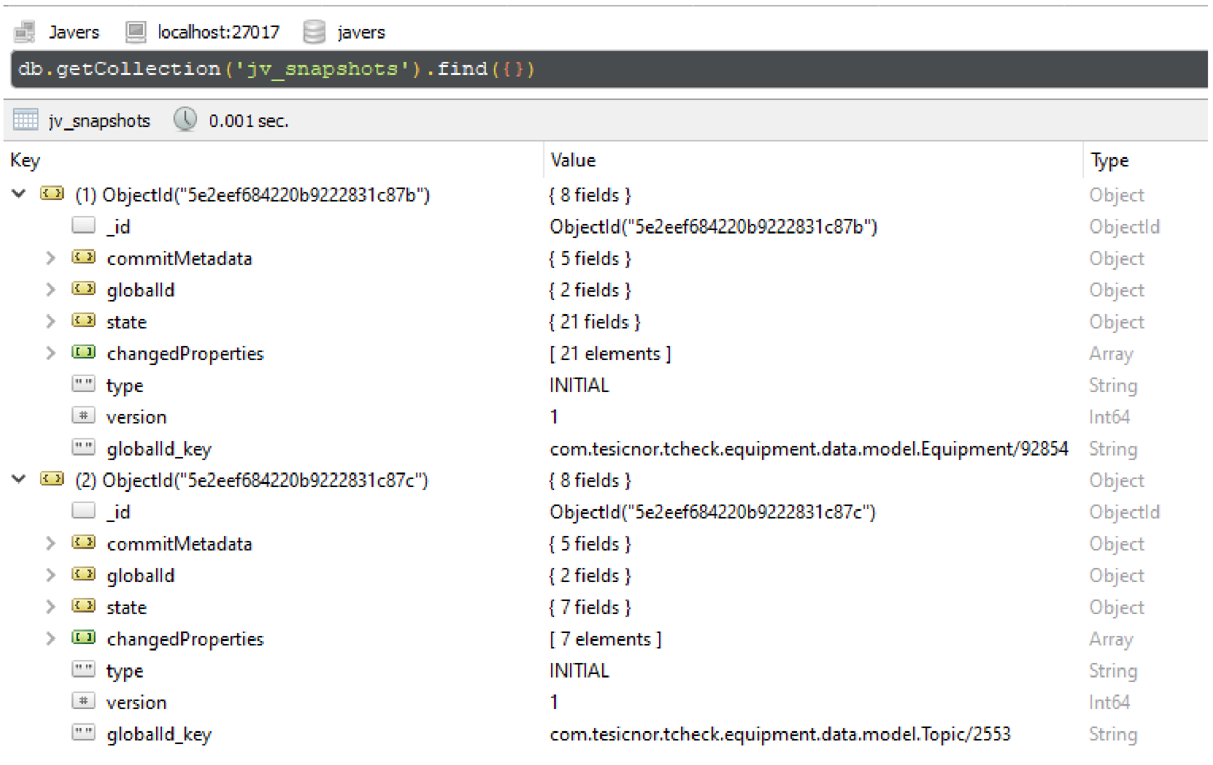
Key	Value	Type
(18) ObjectId("5e2ed31a4220b9222831c863")	{ 8 fields }	Object
(19) ObjectId("5e2ed31a4220b9222831c864")	{ 8 fields }	Object
(20) ObjectId("5e2ed31a4220b9222831c865")	{ 8 fields }	Object
(21) ObjectId("5e2ed31a4220b9222831c866")	{ 8 fields }	Object
(22) ObjectId("5e2ed31a4220b9222831c867")	{ 8 fields }	Object
(23) ObjectId("5e2ed31a4220b9222831c868")	{ 8 fields }	Object
(24) ObjectId("5e2ed31a4220b9222831c869")	{ 8 fields }	Object
(25) ObjectId("5e2ed31a4220b9222831c86a")	{ 8 fields }	Object
(26) ObjectId("5e2ed31a4220b9222831c86b")	{ 8 fields }	Object
(27) ObjectId("5e2ed31a4220b9222831c86c")	{ 8 fields }	Object
(28) ObjectId("5e2ed31a4220b9222831c86d")	{ 8 fields }	Object
(29) ObjectId("5e2ed31a4220b9222831c86e")	{ 8 fields }	Object
(30) ObjectId("5e2ed31a4220b9222831c86f")	{ 8 fields }	Object
(31) ObjectId("5e2ed31a4220b9222831c870")	{ 8 fields }	Object
(32) ObjectId("5e2ed31a4220b9222831c871")	{ 8 fields }	Object
(33) ObjectId("5e2ed31a4220b9222831c872")	{ 8 fields }	Object
(34) ObjectId("5e2ed31a4220b9222831c873")	{ 8 fields }	Object
(35) ObjectId("5e2ed31a4220b9222831c874")	{ 8 fields }	Object
(36) ObjectId("5e2ed3bd4220b9222831c876")	{ 8 fields }	Object
(37) ObjectId("5e2ed3bd4220b9222831c877")	{ 8 fields }	Object
(38) ObjectId("5e2ed44a4220b9222831c878")	{ 8 fields }	Object
(39) ObjectId("5e2ed4a04220b9222831c879")	{ 8 fields }	Object
_id	ObjectId("5e2ed4a04220b9222831c879")	ObjectId
commitMetadata	{ 5 fields }	Object
globalId	{ 2 fields }	Object
state	{ 21 fields }	Object
changedProperties	[21 elements]	Array
[0]	template	String
[1]	code	String
[2]	serialNumber	String
[3]	created	String
[4]	description	String
[5]	annotations	String
[6]	active	String
[7]	idsNotificationGroup	String
[8]	branch	String
[9]	text3	String

Figure 24 - Robo 3T Javers Database

Nevertheless, there are too much snapshots here. This is because all the tests we've been doing with Javers were being stored in the JaversRepository. And after establishing the connection with mongo, it moved all the existing snapshots in the JaversRepository to the mongo database.

We need to empty my JaversRepository to solve that. After, we'll deploy the MongoDB in another server, in freemium25.tdoc.es, where Elasticsearch and Kibana are running.

Dropping all the snapshots and collections from MongoDB deletes the snapshots and collections from JaversRepository. We try another time to create a new equipment which has a specific topic (another class), and we can see that Javers creates the initial snapshots of each object, as it was expected:



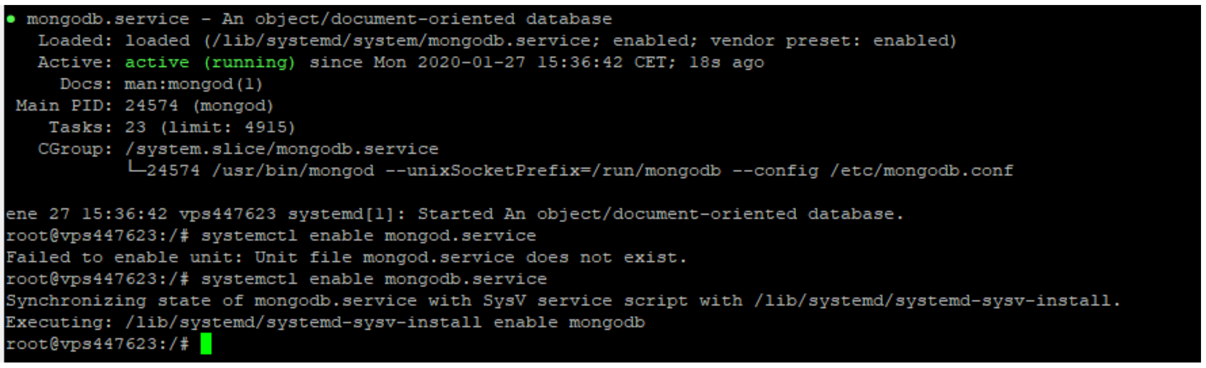
```
Javers   localhost:27017   javers
db.getCollection('jv_snapshots').find({})
jv_snapshots   0.001 sec.
```

Key	Value	Type
(1) ObjectId("5e2eef684220b9222831c87b")	{ 8 fields }	Object
_id	ObjectId("5e2eef684220b9222831c87b")	ObjectId
commitMetadata	{ 5 fields }	Object
globalId	{ 2 fields }	Object
state	{ 21 fields }	Object
changedProperties	[21 elements]	Array
type	INITIAL	String
version	1	Int64
globalId_key	com.tesicnor.tcheck.equipment.data.model.Equipment/92854	String
(2) ObjectId("5e2eef684220b9222831c87c")	{ 8 fields }	Object
_id	ObjectId("5e2eef684220b9222831c87c")	ObjectId
commitMetadata	{ 5 fields }	Object
globalId	{ 2 fields }	Object
state	{ 7 fields }	Object
changedProperties	[7 elements]	Array
type	INITIAL	String
version	1	Int64
globalId_key	com.tesicnor.tcheck.equipment.data.model.Topic/2553	String

Figure 25 - Javers Snapshots

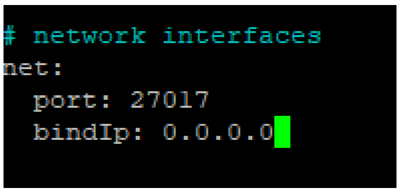
MongoDB Server Installation

Install mongo as a service and configured it to start on system start



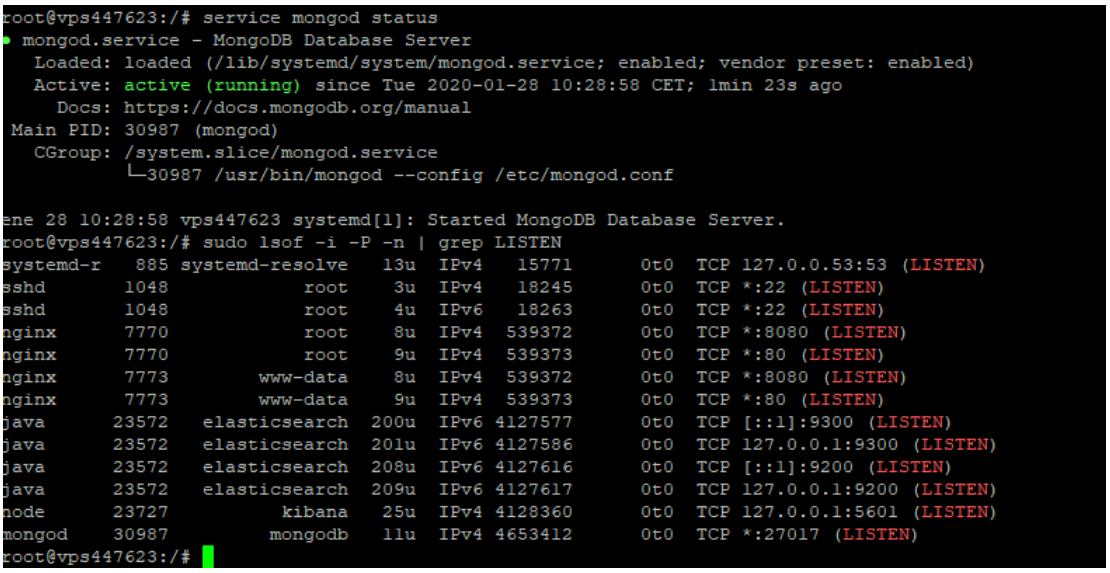
```
● mongodb.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-01-27 15:36:42 CET; 18s ago
     Docs: man:mongod(1)
 Main PID: 24574 (mongod)
    Tasks: 23 (limit: 4915)
   CGroup: /system.slice/mongodb.service
           └─24574 /usr/bin/mongod --unixSocketPrefix=/run/mongodb --config /etc/mongodb.conf

ene 27 15:36:42 vps447623 systemd[1]: Started An object/document-oriented database.
root@vps447623:/# systemctl enable mongod.service
Failed to enable unit: Unit file mongod.service does not exist.
root@vps447623:/# systemctl enable mongodb.service
Synchronizing state of mongodb.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mongodb
root@vps447623:/# 
```

As we want to access mongo not only from localhost, but from any machine, we edit the /etc/mongod.conf, and change the bindIp value from 127.0.0.1 to broadcast.

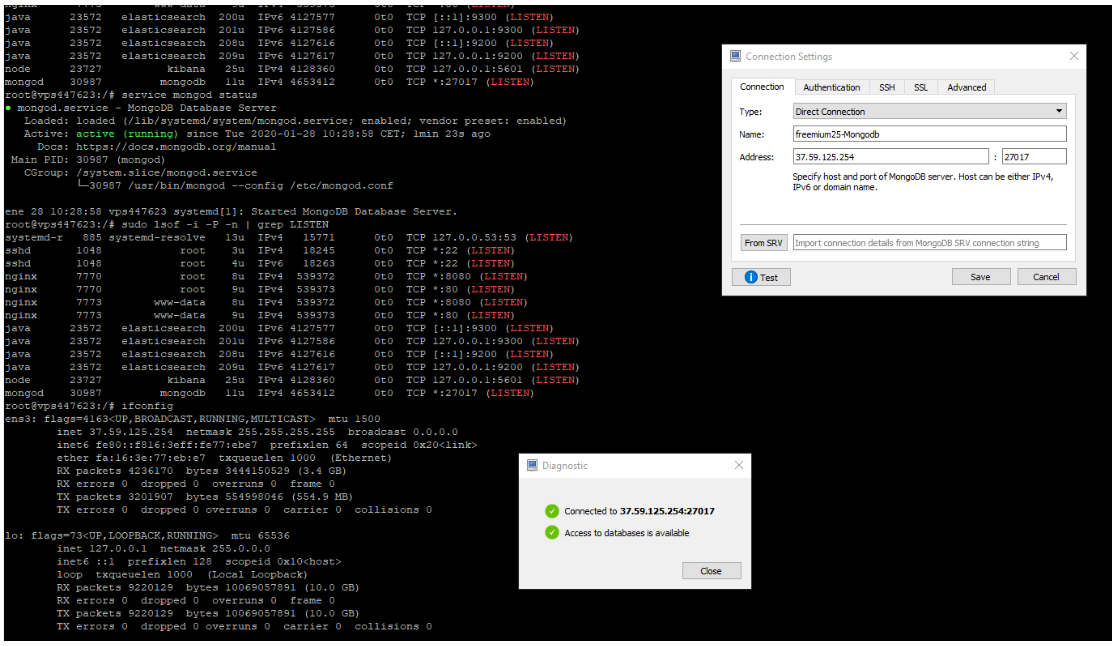


```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0
```

Now in the image below we can see that MongoDB is listening from all interfaces , in port 27017 (default from MongoDB)



```
root@vps447623:/# service mongod status
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-01-28 10:28:58 CET; 1min 23s ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 30987 (mongod)
   CGroup: /system.slice/mongod.service
           └─30987 /usr/bin/mongod --config /etc/mongod.conf

ene 28 10:28:58 vps447623 systemd[1]: Started MongoDB Database Server.
root@vps447623:/# sudo lsof -i -P -n | grep LISTEN
systemd-r   885 systemd-resolve   13u  IPv4   15771      0t0  TCP 127.0.0.53:53 (LISTEN)
sshd       1048            root    3u  IPv4   18245      0t0  TCP *:22 (LISTEN)
sshd       1048            root    4u  IPv6   18263      0t0  TCP *:22 (LISTEN)
nginx      7770            root    8u  IPv4  539372      0t0  TCP *:8080 (LISTEN)
nginx      7770            root    9u  IPv4  539373      0t0  TCP *:80 (LISTEN)
nginx      7773        www-data    8u  IPv4  539372      0t0  TCP *:8080 (LISTEN)
nginx      7773        www-data    9u  IPv4  539373      0t0  TCP *:80 (LISTEN)
java      23572   elasticsearch  200u  IPv6 4127577      0t0  TCP [::1]:9300 (LISTEN)
java      23572   elasticsearch  201u  IPv6 4127586      0t0  TCP 127.0.0.1:9300 (LISTEN)
java      23572   elasticsearch  208u  IPv6 4127616      0t0  TCP [::1]:9200 (LISTEN)
java      23572   elasticsearch  209u  IPv6 4127617      0t0  TCP 127.0.0.1:9200 (LISTEN)
node      23727          kibana   25u  IPv4 4128360      0t0  TCP 127.0.0.1:5601 (LISTEN)
mongod    30987         mongodb   11u  IPv4 4653412      0t0  TCP *:27017 (LISTEN)
root@vps447623:/#
```



```
java      23572   elasticsearch  200u  IPv6 4127577      0t0  TCP [::1]:9300 (LISTEN)
java      23572   elasticsearch  201u  IPv6 4127586      0t0  TCP 127.0.0.1:9300 (LISTEN)
java      23572   elasticsearch  208u  IPv6 4127616      0t0  TCP [::1]:9200 (LISTEN)
java      23572   elasticsearch  209u  IPv6 4127617      0t0  TCP 127.0.0.1:9200 (LISTEN)
node      23727          kibana   25u  IPv4 4128360      0t0  TCP 127.0.0.1:5601 (LISTEN)
mongod    30987         mongodb   11u  IPv4 4653412      0t0  TCP *:27017 (LISTEN)
root@vps447623:/# service mongod status
● mongod.service - MongoDB Database Server
   Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-01-28 10:28:58 CET; 1min 23s ago
     Docs: https://docs.mongodb.org/manual
 Main PID: 30987 (mongod)
   CGroup: /system.slice/mongod.service
           └─30987 /usr/bin/mongod --config /etc/mongod.conf

ene 28 10:28:58 vps447623 systemd[1]: Started MongoDB Database Server.
root@vps447623:/# sudo lsof -i -P -n | grep LISTEN
systemd-r   885 systemd-resolve   13u  IPv4   15771      0t0  TCP 127.0.0.53:53 (LISTEN)
sshd       1048            root    3u  IPv4   18245      0t0  TCP *:22 (LISTEN)
sshd       1048            root    4u  IPv6   18263      0t0  TCP *:22 (LISTEN)
nginx      7770            root    8u  IPv4  539372      0t0  TCP *:8080 (LISTEN)
nginx      7770            root    9u  IPv4  539373      0t0  TCP *:80 (LISTEN)
nginx      7773        www-data    8u  IPv4  539372      0t0  TCP *:8080 (LISTEN)
nginx      7773        www-data    9u  IPv4  539373      0t0  TCP *:80 (LISTEN)
java      23572   elasticsearch  200u  IPv6 4127577      0t0  TCP [::1]:9300 (LISTEN)
java      23572   elasticsearch  201u  IPv6 4127586      0t0  TCP 127.0.0.1:9300 (LISTEN)
java      23572   elasticsearch  208u  IPv6 4127616      0t0  TCP [::1]:9200 (LISTEN)
java      23572   elasticsearch  209u  IPv6 4127617      0t0  TCP 127.0.0.1:9200 (LISTEN)
node      23727          kibana   25u  IPv4 4128360      0t0  TCP 127.0.0.1:5601 (LISTEN)
mongod    30987         mongodb   11u  IPv4 4653412      0t0  TCP *:27017 (LISTEN)
root@vps447623:/# ifconfig
ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 37.59.125.254  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::f816:3eff:fe77:ebe7  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:77:eb:e7  txqueuelen 1000  (Ethernet)
        RX packets 4236170  bytes 3444150529 (3.4 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3201907  bytes 554998046 (554.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 9220129  bytes 10069057891 (10.0 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9220129  bytes 10069057891 (10.0 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Connection Settings

Type: Direct Connection
Name: freemium25-Mongodb
Address: 37.59.125.254 : 27017
Specify host and port of MongoDB server. Host can be either IPv4, IPv6 or domain name.

Import connection details from MongoDB SRV connection string

Diagnostic

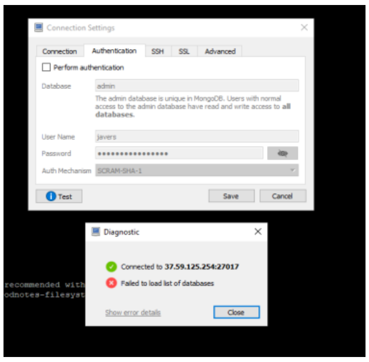
✓ Connected to **37.59.125.254:27017**
✓ Access to databases is available

The connection with Robo 3T to our server where mongo is running, works properly

However, we want to have security. We want users to authenticate in order to have access to the database. That's why we enable the security and authentication in mongo configuration file and add a new user in mongo with root role, establishing a specific password for it.

```
> db.createUser({user:"javers", pwd:"*************", roles:[{role:"root", db:"admin"}]})
```

With this configuration, and restarting the service, we try to connect to the database without authentication we get an authentication error. Nevertheless, if we perform the authentication, we have a complete access to the database.



☐ Perform authentication

Database: admin

The admin database is unique in MongoDB. Users with normal access to the admin database have read and write access to **all databases.**

User Name: javers

Password: ••••••••••••••••

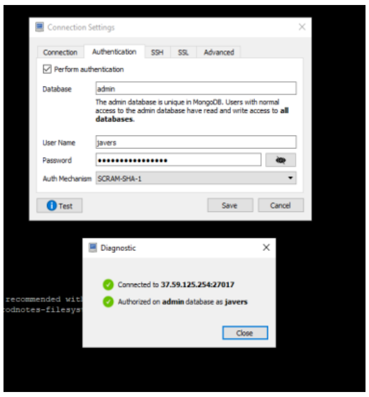
Auth Mechanism: SCRAM-SHA-1

Connected to **37.59.125.254:27017**

Failed to load list of databases

```
recommended with
odnotes-filesyst
```

Figure 26 - Mongo Failed Authentication



☑ Perform authentication

Database: admin

The admin database is unique in MongoDB. Users with normal access to the admin database have read and write access to **all databases.**

User Name: javers

Password: ••••••••••••••••

Auth Mechanism: SCRAM-SHA-1

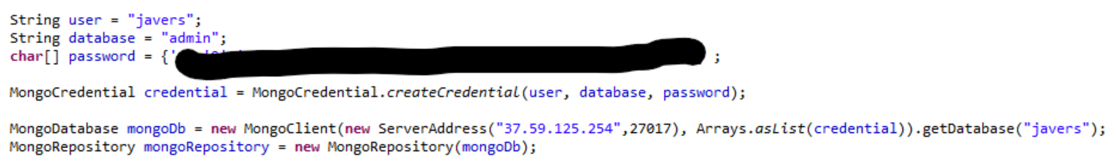
Connected to **37.59.125.254:27017**

Authorized on **admin** database as **javers**

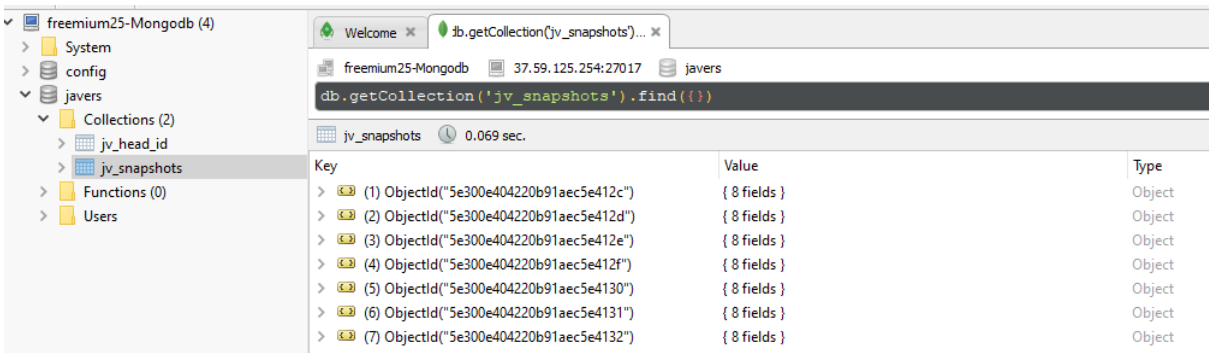
```
recommended wit
odnotes-filesys
```

Figure 27 - Mongo Success Authentication

In T-check microservice, we need to configure correctly the mongoclient with the credentials so that Javers can have access to the database.



```
String user = "javers";
String database = "admin";
char[] password = {[redacted];

MongoCredential credential = MongoCredential.createCredential(user, database, password);

MongoDatabase mongoDb = new MongoClient(new ServerAddress("37.59.125.254",27017), Arrays.asList(credential)).getDatabase("javers");
MongoRepository mongoRepository = new MongoRepository(mongoDb);
```



```
freemium25-Mongodb (4)
  System
  config
  javers
    Collections (2)
      jv_head_id
      jv_snapshots
    Functions (0)
    Users
```

Welcome | db.getCollection('jv_snapshots')...

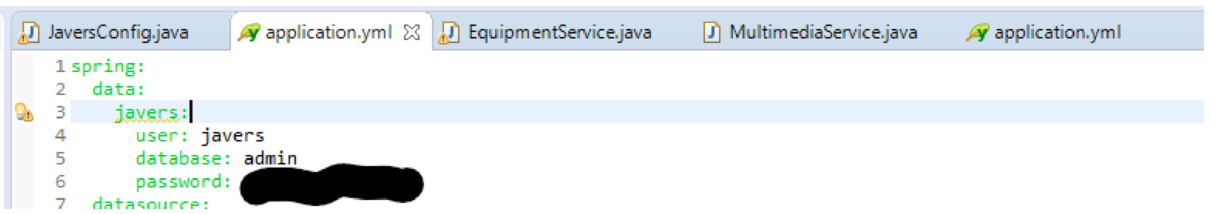
freemium25-Mongodb | 37.59.125.254:27017 | javers

```
db.getCollection('jv_snapshots').find({})
```

jv_snapshots 0.069 sec.

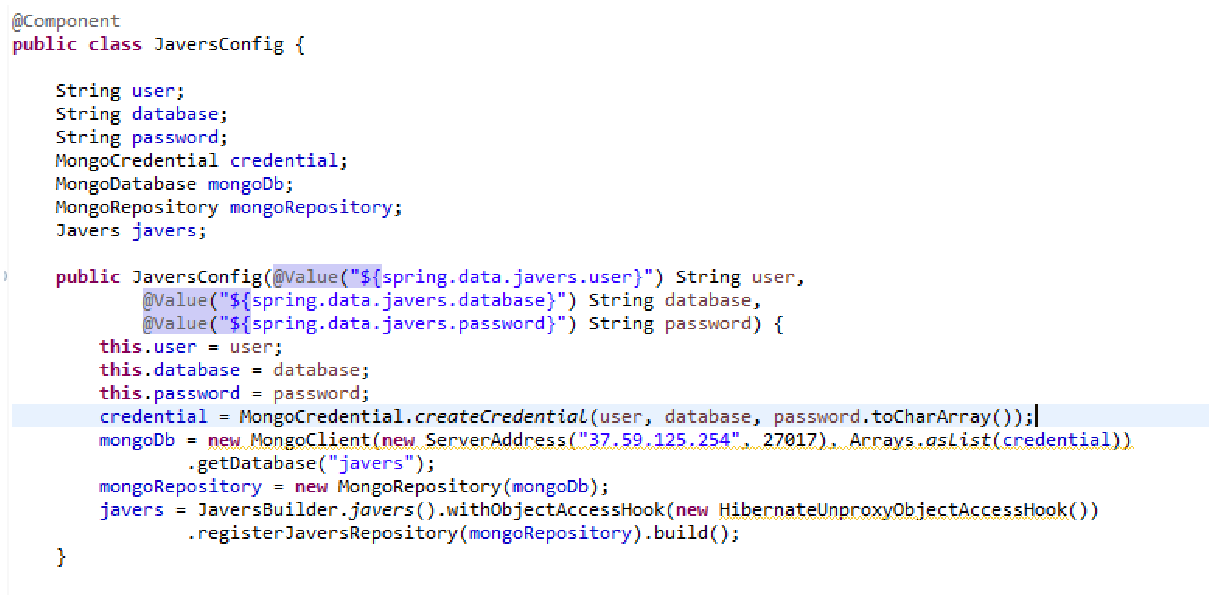
Key	Value	Type
(1) ObjectId("5e300e404220b91aec5e412c")	{ 8 fields }	Object
(2) ObjectId("5e300e404220b91aec5e412d")	{ 8 fields }	Object
(3) ObjectId("5e300e404220b91aec5e412e")	{ 8 fields }	Object
(4) ObjectId("5e300e404220b91aec5e412f")	{ 8 fields }	Object
(5) ObjectId("5e300e404220b91aec5e4130")	{ 8 fields }	Object
(6) ObjectId("5e300e404220b91aec5e4131")	{ 8 fields }	Object
(7) ObjectId("5e300e404220b91aec5e4132")	{ 8 fields }	Object

Just for tests, we've done it by hardcoding the user and pass, which is so insecure. But now that it works, we will set the credentials in the config file.



JaversConfig.java | application.yml | EquipmentService.java | MultimediaService.java | application.yml

```
spring:
  data:
    javers:
      user: javers
      database: admin
      password: [redacted]
  datasource:
```



```
@Component
public class JaversConfig {

    String user;
    String database;
    String password;
    MongoCredential credential;
    MongoDatabase mongoDb;
    MongoRepository mongoRepository;
    Javers javers;

    public JaversConfig(@Value("${spring.data.javers.user}") String user,
            @Value("${spring.data.javers.database}") String database,
            @Value("${spring.data.javers.password}") String password) {
        this.user = user;
        this.database = database;
        this.password = password;
        credential = MongoCredential.createCredential(user, database, password.toCharArray());
        mongoDb = new MongoClient(new ServerAddress("37.59.125.254", 27017), Arrays.asList(credential))
                .getDatabase("javers");
        mongoRepository = new MongoRepository(mongoDb);
        javers = JaversBuilder.javers().withObjectAccessHook(new HibernateUnproxyObjectAccessHook())
                .registerJaversRepository(mongoRepository).build();
    }
```

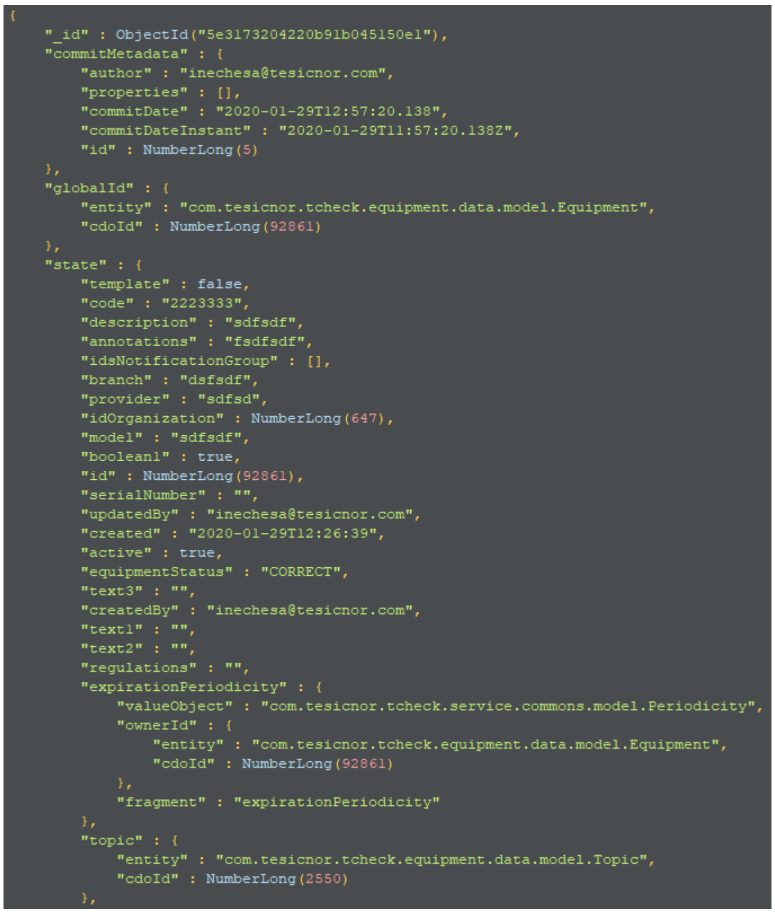
Now, all works perfectly, as it should.

4.2.8 Eighth Iteration

Javers Snapshots

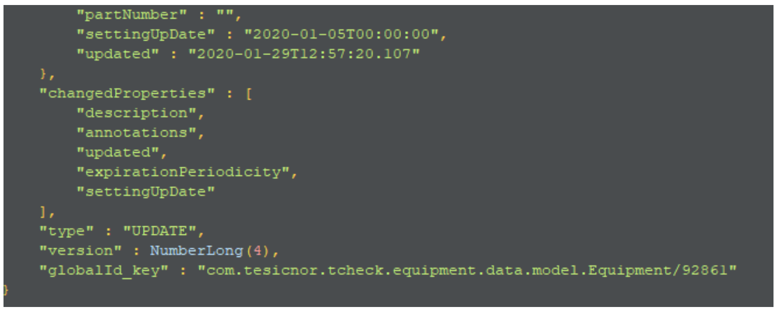
During the Spring Review, we discussed the next steps we should follow. From one side, we need to do some tests with Javers, see how it handles 1-N and M-N relationships. From the other side, we need to finally implement a tracing system, in order to look how the implementation of Javers framework impact in the application performance.

First, let’s see how the jv_snapshots are stored. We have created an object and edit it several times. Each snapshot is stored in a JSON format. The image below is a JSON of an Object commit.



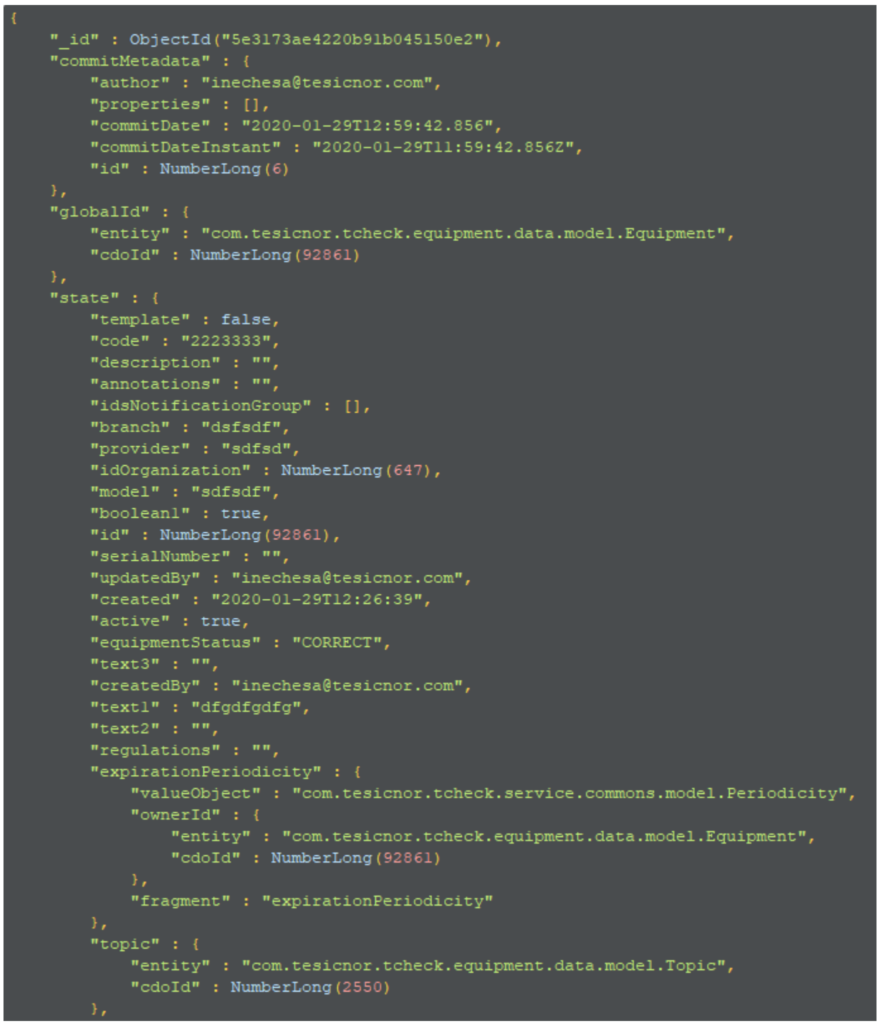
```
{
    "_id" : ObjectId("5e3173204220b91b045150e1"),
    "commitMetadata" : {
        "author" : "inechesa@tesicnor.com",
        "properties" : [],
        "commitDate" : "2020-01-29T12:57:20.138",
        "commitDateInstant" : "2020-01-29T11:57:20.138Z",
        "id" : NumberLong(5)
    },
    "globalId" : {
        "entity" : "com.tesicnor.tcheck.equipment.data.model.Equipment",
        "cdoId" : NumberLong(92861)
    },
    "state" : {
        "template" : false,
        "code" : "2223333",
        "description" : "sdfsdf",
        "annotations" : "fsdfsdf",
        "idsNotificationGroup" : [],
        "branch" : "dsfsdf",
        "provider" : "sdfsd",
        "idOrganization" : NumberLong(647),
        "model" : "sdfsdf",
        "boolean1" : true,
        "id" : NumberLong(92861),
        "serialNumber" : "",
        "updatedBy" : "inechesa@tesicnor.com",
        "created" : "2020-01-29T12:26:39",
        "active" : true,
        "equipmentStatus" : "CORRECT",
        "text3" : "",
        "createdBy" : "inechesa@tesicnor.com",
        "text1" : "",
        "text2" : "",
        "regulations" : "",
        "expirationPeriodicity" : {
            "valueObject" : "com.tesicnor.tcheck.service.commons.model.Periodicity",
            "ownerId" : {
                "entity" : "com.tesicnor.tcheck.equipment.data.model.Equipment",
                "cdoId" : NumberLong(92861)
            },
            "fragment" : "expirationPeriodicity"
        },
        "topic" : {
            "entity" : "com.tesicnor.tcheck.equipment.data.model.Topic",
            "cdoId" : NumberLong(2550)
        },
```

Figure 28 - Snapshot JSON Format

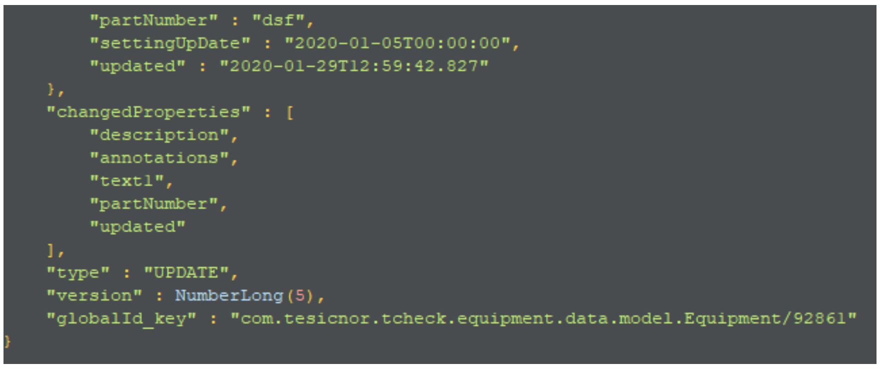


```
        "partNumber" : "",
        "settingUpDate" : "2020-01-05T00:00:00",
        "updated" : "2020-01-29T12:57:20.107"
    },
    "changedProperties" : [
        "description",
        "annotations",
        "updated",
        "expirationPeriodicity",
        "settingUpDate"
    ],
    "type" : "UPDATE",
    "version" : NumberLong(4),
    "globalId_key" : "com.tesicnor.tcheck.equipment.data.model.Equipment/92861"
}
```

After editing the same object, and commit it, another snapshot is created with a new state, and we can see the changes between the previous status and the new status. The edited object JSON is the one below:



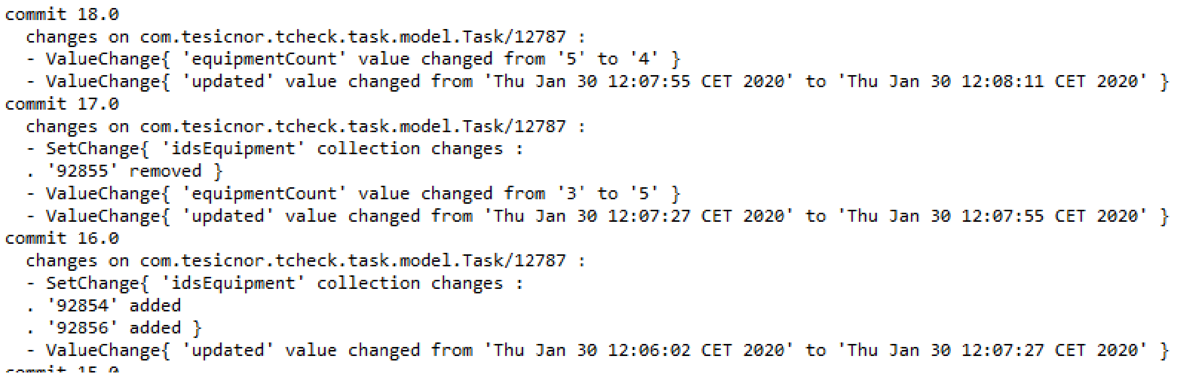
```
{
    "_id" : ObjectId("5e3173ae4220b91b045150e2"),
    "commitMetadata" : {
        "author" : "inechesa@tesicnor.com",
        "properties" : [],
        "commitDate" : "2020-01-29T12:59:42.856",
        "commitDateInstant" : "2020-01-29T11:59:42.856Z",
        "id" : NumberLong(6)
    },
    "globalId" : {
        "entity" : "com.tesicnor.tcheck.equipment.data.model.Equipment",
        "cdoId" : NumberLong(92861)
    },
    "state" : {
        "template" : false,
        "code" : "2223333",
        "description" : "",
        "annotations" : "",
        "idsNotificationGroup" : [],
        "branch" : "dsfsdf",
        "provider" : "sdfsd",
        "idOrganization" : NumberLong(647),
        "model" : "sdfsdf",
        "boolean1" : true,
        "id" : NumberLong(92861),
        "serialNumber" : "",
        "updatedBy" : "inechesa@tesicnor.com",
        "created" : "2020-01-29T12:26:39",
        "active" : true,
        "equipmentStatus" : "CORRECT",
        "text3" : "",
        "createdBy" : "inechesa@tesicnor.com",
        "text1" : "dfgdfgdfg",
        "text2" : "",
        "regulations" : "",
        "expirationPeriodicity" : {
            "valueObject" : "com.tesicnor.tcheck.service.commons.model.Periodicity",
            "ownerId" : {
                "entity" : "com.tesicnor.tcheck.equipment.data.model.Equipment",
                "cdoId" : NumberLong(92861)
            },
            "fragment" : "expirationPeriodicity"
        },
        "topic" : {
            "entity" : "com.tesicnor.tcheck.equipment.data.model.Topic",
            "cdoId" : NumberLong(2550)
        },
```



```
        "partNumber" : "dsf",
        "settingUpDate" : "2020-01-05T00:00:00",
        "updated" : "2020-01-29T12:59:42.827"
    },
    "changedProperties" : [
        "description",
        "annotations",
        "text1",
        "partNumber",
        "updated"
    ],
    "type" : "UPDATE",
    "version" : NumberLong(5),
    "globalId_key" : "com.tesicnor.tcheck.equipment.data.model.Equipment/92861"
}
```

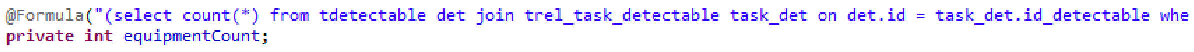
We can see in the “changedProperties” fields, which properties have been changed, and then locate them in the state and see the previous and the new values.

Now we are going to do a test that involves a one-to-many relationship, to see how Javers. We use the task microservice, since one task may have assigned or not many equipment. However, after adding Javers to task-service, we’ve seen a weird behaviour. When we add some equipment to a task, and save it, the commit adds that equipment to the task, but the equipment count is not updated until the next commit. In the image below we can see in commit 16, we added two equipment to the task, but the equipment count only increases in the next commit, the 17 one.



```
commit 18.0
  changes on com.tesicnor.tcheck.task.model.Task/12787 :
  - ValueChange{ 'equipmentCount' value changed from '5' to '4' }
  - ValueChange{ 'updated' value changed from 'Thu Jan 30 12:07:55 CET 2020' to 'Thu Jan 30 12:08:11 CET 2020' }
commit 17.0
  changes on com.tesicnor.tcheck.task.model.Task/12787 :
  - SetChange{ 'idsEquipment' collection changes :
  . '92855' removed }
  - ValueChange{ 'equipmentCount' value changed from '3' to '5' }
  - ValueChange{ 'updated' value changed from 'Thu Jan 30 12:07:27 CET 2020' to 'Thu Jan 30 12:07:55 CET 2020' }
commit 16.0
  changes on com.tesicnor.tcheck.task.model.Task/12787 :
  - SetChange{ 'idsEquipment' collection changes :
  . '92854' added
  . '92856' added }
  - ValueChange{ 'updated' value changed from 'Thu Jan 30 12:06:02 CET 2020' to 'Thu Jan 30 12:07:27 CET 2020' }
```

After researching a little bit, we’ve found the problem. The equipmentCount is a variable which is taken from the Database before executing the save action. Due to this, when we do the Javers commit, the equipmentCount field is always the previous one of the actions we did before, since it does not update it until the next iteration, taking it from the database.



```
@Formula("(select count(*) from tdetectable det join trel_task_detectable task_det on det.id = task_det.id_detectable whe
private int equipmentCount;
```

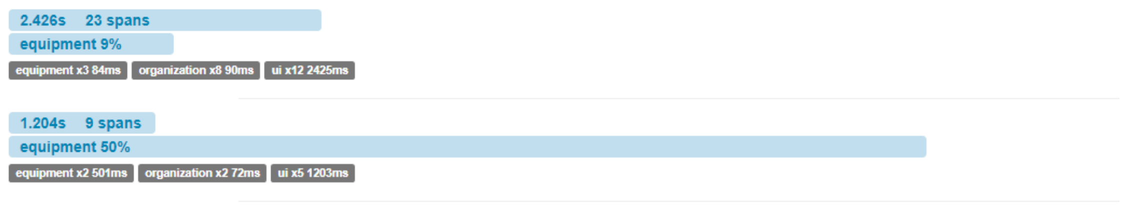
Zipkin Sleuth

After several tries to make Zipkin work, we finally did it. Here are some tests we could do, and now we should try the performance, with, and without Javers

We tried to measure the performance in methods where we use the Javers commit function, and we obtained the following results:

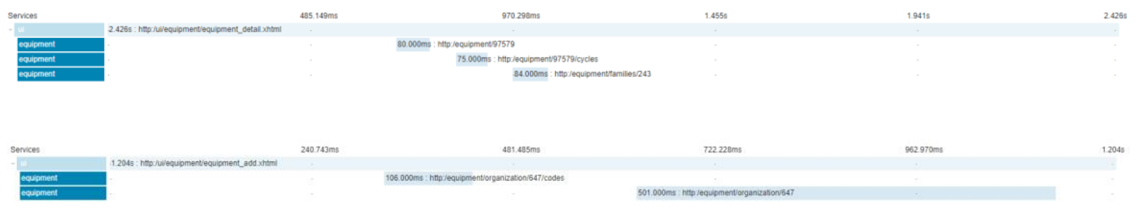
Create Equipment

<u>With Javers:</u>



2.426s 23 spans
equipment 9%
equipment x3 84ms | organization x8 90ms | ui x12 2425ms

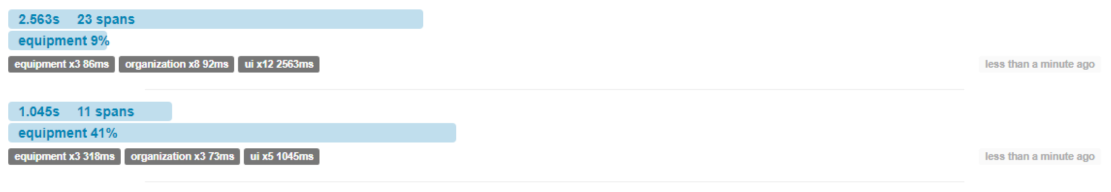
1.204s 9 spans
equipment 50%
equipment x2 501ms | organization x2 72ms | ui x5 1203ms



```
Services                      485.149ms        970.298ms        1.455s        1.941s        2.426s
- ui          2.426s : http:/ui/equipment/equipment_detail.xhtml
  equipment                              80.000ms : http:/equipment/97579
  equipment                                   75.000ms : http:/equipment/97579/cycles
  equipment                                        84.000ms : http:/equipment/families/243

Services                      240.743ms        481.485ms        722.228ms        962.970ms        1.204s
- ui          1.204s : http:/ui/equipment/equipment_add.xhtml
  equipment                             106.000ms : http:/equipment/organization/647/codes
  equipment                                                 501.000ms : http:/equipment/organization/647
```

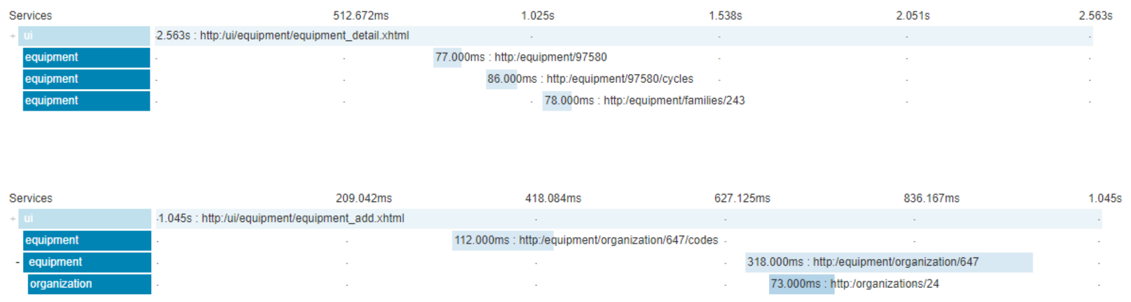
The trace to consider is the second one, the one that is in the bottom, since it represents the time it takes to add the new equipment. The one in the top is just the time it takes to represent the information in the user interface.

<u>Without Javers:</u>



2.563s 23 spans
equipment 9%
equipment x3 86ms | organization x8 92ms | ui x12 2563ms
less than a minute ago

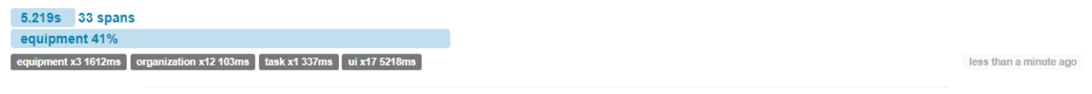
1.045s 11 spans
equipment 41%
equipment x3 318ms | organization x3 73ms | ui x5 1045ms
less than a minute ago



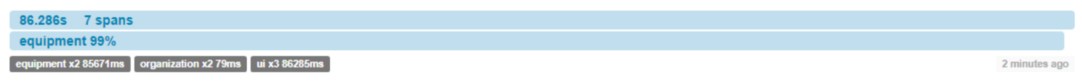
```
Services                      512.672ms        1.025s        1.538s        2.051s        2.563s
- ui          2.563s : http:/ui/equipment/equipment_detail.xhtml
  equipment                              77.000ms : http:/equipment/97580
  equipment                                   86.000ms : http:/equipment/97580/cycles
  equipment                                        78.000ms : http:/equipment/families/243

Services                      209.042ms        418.084ms        627.125ms        836.167ms        1.045s
- ui          1.045s : http:/ui/equipment/equipment_add.xhtml
  equipment                             112.000ms : http:/equipment/organization/647/codes
- equipment                                                 318.000ms : http:/equipment/organization/647
   organization                                                 73.000ms : http:/organizations/24
```

Create 200 equipment with template

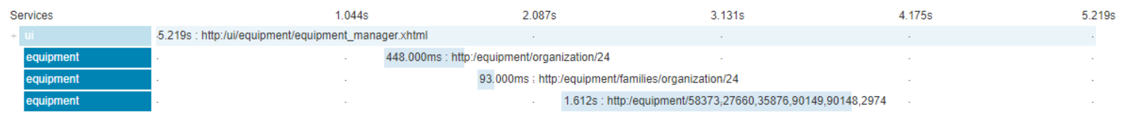
With Javers:



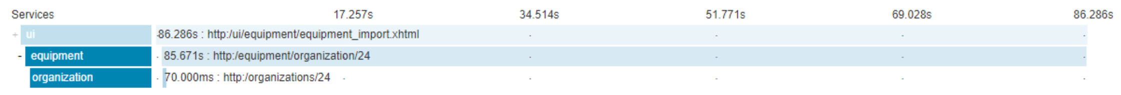
```
5.219s  33 spans
equipment 41%
equipment x3 1612ms  organization x12 103ms  task x1 337ms  ui x17 5218ms        less than a minute ago
```



```
86.286s  7 spans
equipment 99%
equipment x2 85671ms  organization x2 79ms  ui x3 86285ms        2 minutes ago
```

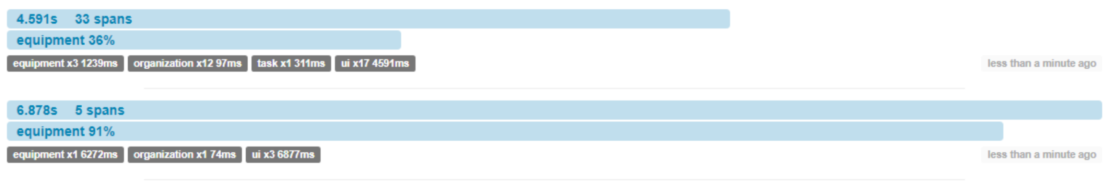


```
Services        1.044s        2.087s        3.131s        4.175s        5.219s
+ ui          5.219s : http:/ui/equipment/equipment_manager.xhtml
  equipment                 448.000ms : http:/equipment/organization/24
  equipment                       93.000ms : http:/equipment/families/organization/24
  equipment                             1.612s : http:/equipment/58373,27660,35876,90149,90148,2974
```

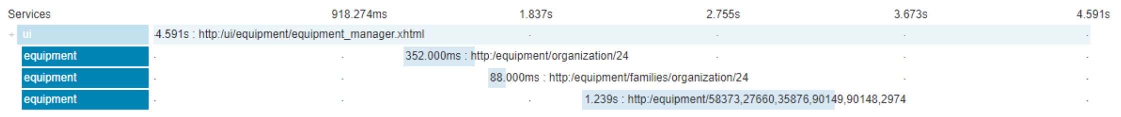


```
Services        17.257s        34.514s        51.771s        69.028s        86.286s
+ ui            86.286s : http:/ui/equipment/equipment_import.xhtml
- equipment      85.671s : http:/equipment/organization/24
  organization   70.000ms : http:/organizations/24
```

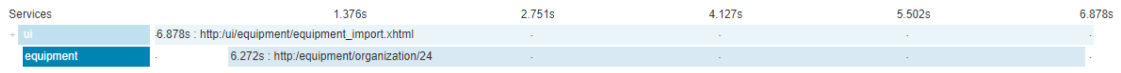
Without Javers:



```
4.591s  33 spans
equipment 36%
equipment x3 1239ms  organization x12 97ms  task x1 311ms  ui x17 4591ms        less than a minute ago

6.878s  5 spans
equipment 91%
equipment x1 6272ms  organization x1 74ms  ui x3 6877ms        less than a minute ago
```

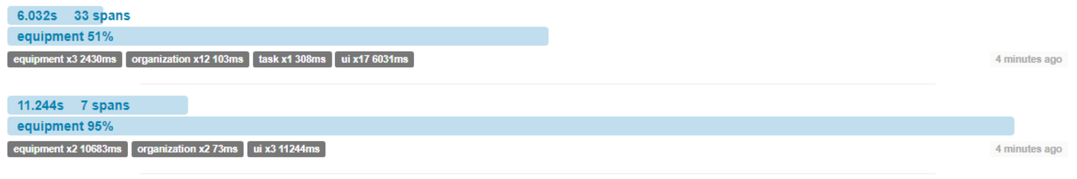


```
Services        918.274ms        1.837s        2.755s        3.673s        4.591s
+ ui          4.591s : http:/ui/equipment/equipment_manager.xhtml
  equipment                 352.000ms : http:/equipment/organization/24
  equipment                       88.000ms : http:/equipment/families/organization/24
  equipment                             1.239s : http:/equipment/58373,27660,35876,90149,90148,2974
```



```
Services        1.376s        2.751s        4.127s        5.502s        6.878s
+ ui          6.878s : http:/ui/equipment/equipment_import.xhtml
  equipment          6.272s : http:/equipment/organization/24
```

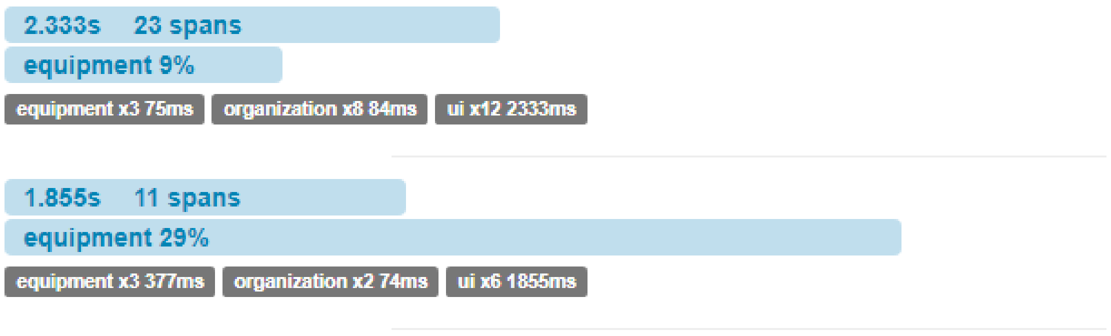
Here we have a problem, since there is a huge difference, therefore we execute the commit function asynchronously. Doing this, we obtained very good results



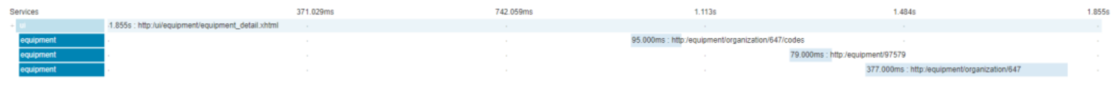
```
6.032s  33 spans
equipment 51%
equipment x3 2430ms  organization x12 103ms  task x1 308ms  ui x17 6031ms        4 minutes ago

11.244s  7 spans
equipment 95%
equipment x2 10683ms  organization x2 73ms  ui x3 11244ms        4 minutes ago
```

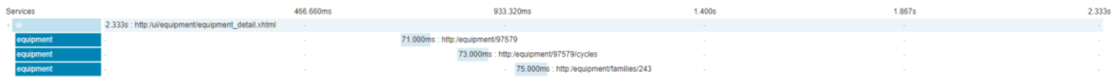
Modify equipment

<u>With Javers:</u>

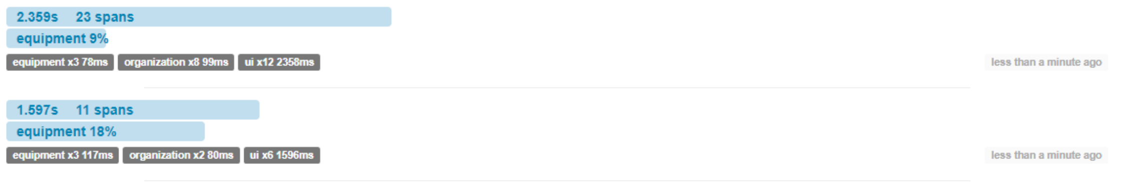

2.333s 23 spans
equipment 9%
equipment x3 75ms | organization x8 84ms | ui x12 2333ms

1.855s 11 spans
equipment 29%
equipment x3 377ms | organization x2 74ms | ui x6 1855ms

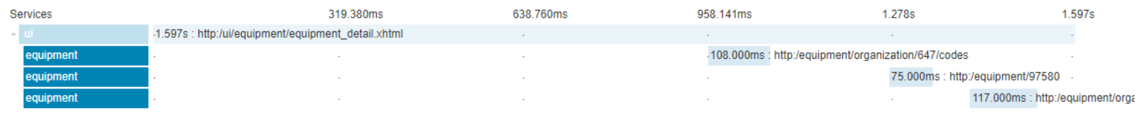

```
Services                 371.029ms      742.059ms      1.113s      1.484s      1.855s
ui         1.855s : http:/ui/equipment/equipment_detail.xhtml
equipment                                  95.000ms : http:/equipment/organization/647/codes
equipment                                               79.000ms : http:/equipment/97579
equipment                                                   377.000ms : http:/equipment/organization/647
```

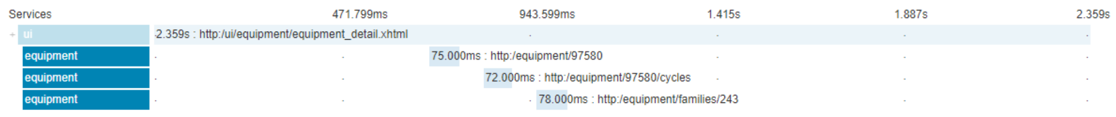

```
Services                 466.660ms      933.320ms      1.400s      1.867s      2.333s
ui         2.333s : http:/ui/equipment/equipment_detail.xhtml
equipment                      71.000ms : http:/equipment/97579
equipment                         73.000ms : http:/equipment/97579/cycles
equipment                            75.000ms : http:/equipment/families/243
```

<u>Without Javers:</u>


2.359s 23 spans
equipment 9%
equipment x3 78ms | organization x8 99ms | ui x12 2358ms
less than a minute ago

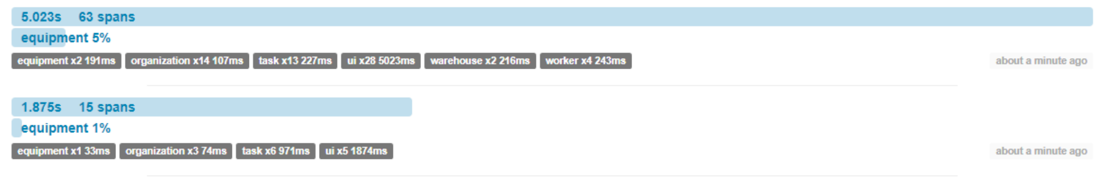
1.597s 11 spans
equipment 18%
equipment x3 117ms | organization x2 80ms | ui x6 1596ms
less than a minute ago


```
Services                 319.380ms      638.760ms      958.141ms      1.278s      1.597s
ui         1.597s : http:/ui/equipment/equipment_detail.xhtml
equipment                                  108.000ms : http:/equipment/organization/647/codes
equipment                                                  75.000ms : http:/equipment/97580
equipment                                                        117.000ms : http:/equipment/orga
```

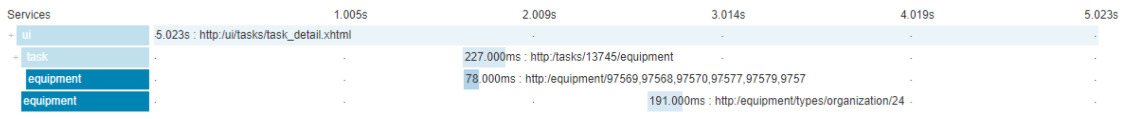

```
Services                 471.799ms      943.599ms      1.415s      1.887s      2.359s
ui         2.359s : http:/ui/equipment/equipment_detail.xhtml
equipment                      75.000ms : http:/equipment/97580
equipment                         72.000ms : http:/equipment/97580/cycles
equipment                            78.000ms : http:/equipment/families/243
```

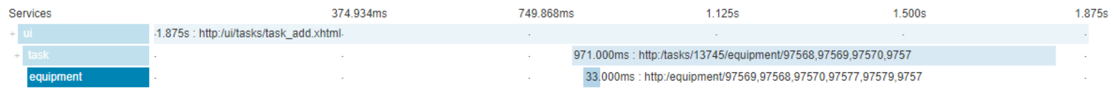
Assign equipment

<u>With Javers:</u>

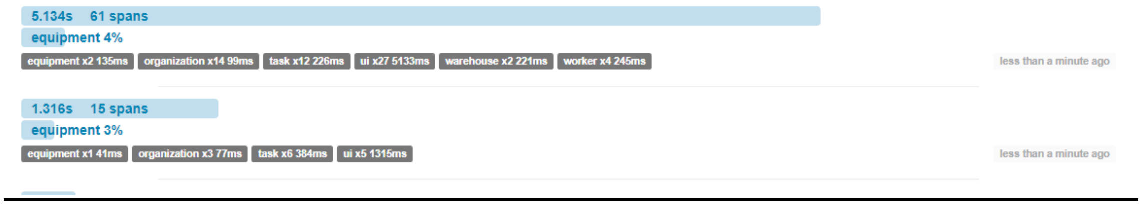

5.023s 63 spans
equipment 5%
equipment x2 191ms | organization x14 107ms | task x13 227ms | ui x28 5023ms | warehouse x2 216ms | worker x4 243ms
about a minute ago

1.875s 15 spans
equipment 1%
equipment x1 33ms | organization x3 74ms | task x6 971ms | ui x5 1874ms
about a minute ago

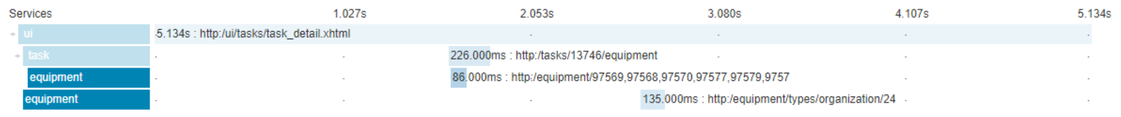

```
Services      1.005s      2.009s      3.014s      4.019s      5.023s
ui            5.023s : http:/ui/tasks/task_detail.xhtml
task          227.000ms : http:/tasks/13745/equipment
equipment     78.000ms : http:/equipment/97569,97568,97570,97577,97579,9757
equipment     191.000ms : http:/equipment/types/organization/24
```

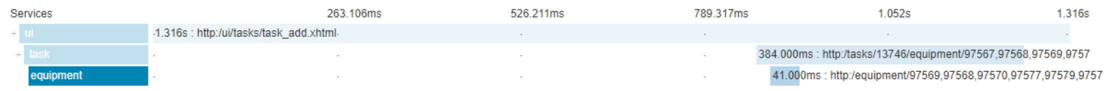

```
Services      374.934ms      749.868ms      1.125s      1.500s      1.875s
ui            1.875s : http:/ui/tasks/task_add.xhtml
task          971.000ms : http:/tasks/13745/equipment/97568,97569,97570,9757
equipment     33.000ms : http:/equipment/97569,97568,97570,97577,97579,9757
```

<u>Without Javers:</u>


5.134s 61 spans
equipment 4%
equipment x2 135ms | organization x14 99ms | task x12 226ms | ui x27 5133ms | warehouse x2 221ms | worker x4 245ms
less than a minute ago

1.316s 15 spans
equipment 3%
equipment x1 41ms | organization x3 77ms | task x6 384ms | ui x5 1315ms
less than a minute ago


```
Services      1.027s      2.053s      3.080s      4.107s      5.134s
ui            5.134s : http:/ui/tasks/task_detail.xhtml
task          226.000ms : http:/tasks/13746/equipment
equipment     86.000ms : http:/equipment/97569,97568,97570,97577,97579,9757
equipment     135.000ms : http:/equipment/types/organization/24
```


```
Services      263.106ms      526.211ms      789.317ms      1.052s      1.316s
ui            1.316s : http:/ui/tasks/task_add.xhtml
task          384.000ms : http:/tasks/13746/equipment/97567,97568,97569,9757
equipment     41.000ms : http:/equipment/97569,97568,97570,97577,97579,9757
```

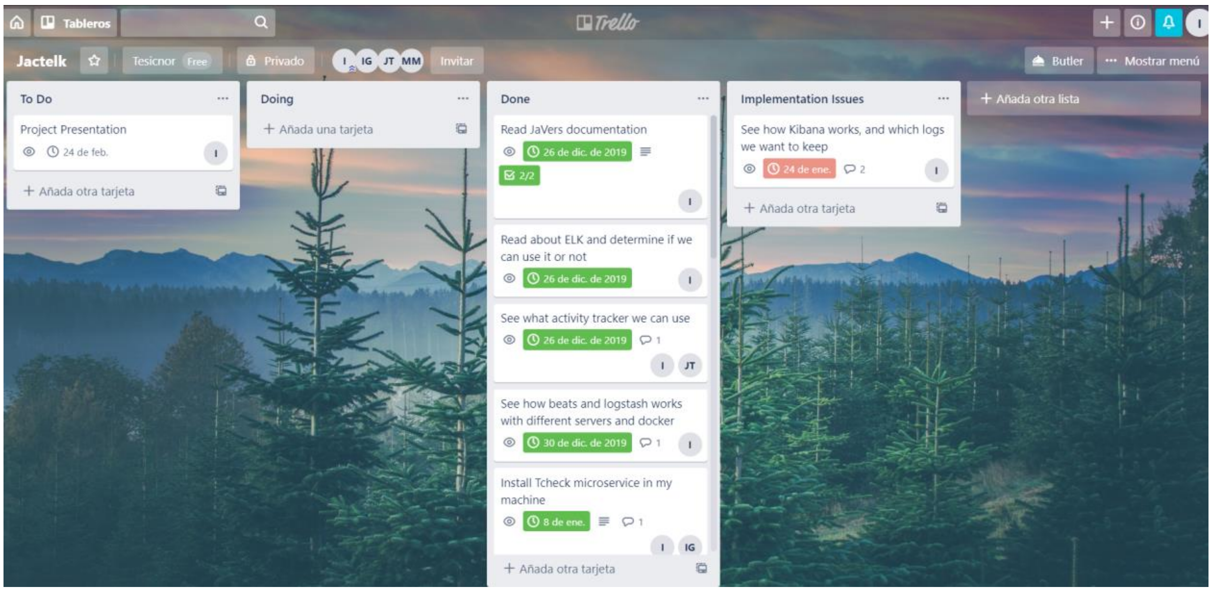
After measuring the performance with and without Javers, we've seen that it does not affect drastically the system performance. This is since Javers is a lightweight library, which does not require too much resources to work.

5. Final Results

The final results are the following:

- Elastic Stack and MongoDB services running correctly in server 25.
- Javers implemented in two microservices of Tcheck.
- Javers audited objects are stored in a MongoDB database in server 25.
  - Need authentication.
- Javers does not affect to the system performance.
- We can visualize the Javers data (snapshots) with a connection to MongoDB with Robo 3T program.
- Elastic Stack working correctly.
  - Only logs from Tcheck at the moment.
- Can visualize and filter logs with Kibana.

Trello Dashboard:



To Do
- Project Presentation — 24 de feb. — I
- + Añada otra tarjeta

Doing
- + Añada una tarjeta

Done
- Read JaVers documentation — 26 de dic. de 2019 — 2/2 — I
- Read about ELK and determine if we can use it or not — 26 de dic. de 2019 — I
- See what activity tracker we can use — 26 de dic. de 2019 — 1 — I, JT
- See how beats and logstash works with different servers and docker — 30 de dic. de 2019 — 1 — I
- Install Tcheck microservice in my machine — 8 de ene. — 1 — I, IG
- + Añada otra tarjeta

Implementation Issues
- See how Kibana works, and which logs we want to keep — 24 de ene. — 2 — I
- + Añada otra tarjeta

Figure 29 - Trello Dashboard Final

6. Conclusions

In summary, I have done a deep research about all the latest technologies and frameworks available for log management and data auditing. I've implemented Javers auditing system for some of the microservices of Tcheck, using Zipkin to measure the system performance with and without Javers, which information is being stored in a MongoDB in one of our servers, and is only accessible if we perform an authentication. In the same server, I have installed Elasticsearch and Kibana, using NGINX as a reverse proxy to manage incoming traffic. Elasticsearch is receiving information from the server where I have installed Filebeat.

Currently, the system is used in my department by all the developers. They can now visualize all the objects that are being modified. Before, we could only visualize that an object has been modified, and that's it. However, now we can see by whom the object was modified, at which time, and the changed properties that has been modified, among a lot of other properties, such as type of commit, object trace, state…

From the other side, all logs are accessible from one URL, which is www.freemium25.tdoc.es , and we do not need to access each server with a SSH tunnel in order to visualize the logs of a specific application. With Kibana dashboard you have many features to filter the logs as you want.

During the project implementation many problems have arisen, however we have managed to solve them every time. We have noticed also that the initial requisites were over the scope, and we redefined them many times.

We have followed an agile development methodology, organizing meetings frequently, and presenting increments each iteration, so the project was not stuck at any time. When something did not work, we start doing another thing in order to not waste time. This is a very good idea to refresh our mind and may help you at the time you return to the previous work, since new ideas could come to your mind to solve the issue you could not solve before.

We have understood completely a project's life cycle, learned how to manage the issues that could appear at any time, and saw the importance of team working and understanding between all the members of the team. Listening to other opinions, other ways of thinking, makes you see other perspectives, and to be more open-minded at the time of taking decisions.

7. Future lines

At the moment, Javers is auditing the data only from some services from Tcheck. Specifically, from *equipment-service* and *task-service.* However, the initial idea is to audit all the microservices from all of our applications in order to make it easier to manage all this information. Remember that Javers snapshots are stored in our mongo database, which runs in server 25. The next step will be including the Javers config package in all the microservices from all our applications. That requires some time, since we don't want to audit all the functions, just the ones which interest us, and for that, we need to think carefully which ones we want to store.

When the implementation of Javers framework in all our applications will be done, the main idea is to create a simple UI, with some basics queries, so that all our team could use it, without writing the queries manually in Robo 3T, which is the program we use now in order to visualize all the information, all the snapshots which contain the information of the commits, the object status and the changed properties among others.

Nevertheless, all this information is being stored, and the memory usage increases. We need to discuss how much time we should store the commit data. One week, one month, or even one year, are the questions which should be solved as soon as possible.

Regarding Elastic Stack, the situation is similar. Currently, the stored logs come from all the microservices from Tcheck. But we want also to visualize all the logs from all the other applications. This is quite easy to be done, it just requires some time to install Filebeat in the servers where the applications we want to trace, and to configure the config file, telling the paths from which we want to take the logs, and configuring the outputs such as Elasticsearch and Kibana.

However, all this data is also being stored and not being erased. So, we also need to think carefully how much time we want this data to be stored, and configured an auto-delete system in a specific period on time , in order to prevent memory saturation. The main idea is to determine the retention policy we want to follow.

We have written all the documentation about this project in "Confluence", which is a great tool used to help teams collaborate and share knowledge efficiently. This will make easier the project to be maintained and develop new features in the future.

8. Bibliography

[1]
https://javers.org/documentation/repository-examples/ → Repository examples and queries with Javers.

[2]
https://www.paradigmadigital.com/dev/introduccion-javers/ → Spanish link where you can understand perfectly what Javers is and how to use it.

[3]
http://github.com/audit4j/audit4j-demo → Audit4j code examples.

[4]
https://www.youtube.com/watch?v=5O1djJ13gRU → Prometheus log manager explanation.

[5]
https://youtu.be/B7t6F7erk2I → Kafka explanation

[6]
https://javers.org/ → JaVers Library

[7]
https://www.koliseo.com/events/commit-2018/r4p/5630471824211968/agenda#/5116072650866688/5676247317217280 → Logging systems

[8]
https://www.koliseo.com/events/commit-2018/r4p/5630471824211968/agenda#/5734118109216768/5677007392210944 → Kafka explanation

[9]
https://www.koliseo.com/events/commit-2018/r4p/5630471824211968/agenda#/5116072650866688/5684448624377856
https://www.koliseo.com/events/commit-2018/r4p/5630471824211968/agenda#/5734118109216768/5174325493628928 → Continuous Integration and Development

[10]
https://stackoverflow.com/questions/33242197/is-elasticsearch-is-free-or-costly →ELK Stack

[11]
https://logz.io/learn/complete-guide-elk-stack/ → ELK complete guid

[12]
https://www.mkyong.com/mongodb/mongodb-allow-remote-access/
https://www.shellhacks.com/mongodb-allow-remote-access/
https://stackoverflow.com/questions/48624476/connecting-to-remote-mongo-server
→ Bind another IP with MongoDB

[13]
https://discuss.elastic.co/t/how-to-collect-the-logs-from-multiple-machines-to-my-server-efficiently/135902/6 → Bind many servers/machiens to one elasticstack

[14]
https://discuss.elastic.co/t/sending-logs-from-a-remote-machine/151330 → Send logs from different machiens

[15]
https://stackoverflow.com/questions/41846988/sending-docker-container-logs-to-elk-stack-by-configuring-the-logging-drivers → Send docker logs to logstash

[16]
https://www.paradigmadigital.com/dev/trazabilidad-distribuida-spring-cloud-sleuth-zipkin/
→ Activity tracker which uses jhipster

[17]
https://www.youtube.com/watch?v=vpFL4MZ0jII → Implementing Microservices tracing with Spring Cloud and Zipkin