UNIVERSITÀ DEGLI STUDI

DI MODENA E REGGIO EMILIA

Dottorato di ricerca in Ingegneria dell'Innovazione Industriale
(Industrial Innovation Engineering)

Ciclo XXXII

Big Data for advanced fault diagnosis and monitoring systems: managing system complexity in a distributed environment

Candidato: Claudio Santo Longo

Relatore (Tutor): Prof. Cesare Fantuzzi

Coordinatore del Corso di Dottorato: Prof. Franco Zambonelli

Acknowledgements

This page is dedicated to all the people who supported me and gave me the opportunity to study, do my research and then write this work thesis.

I would like to thank my Tutor Prof. Cesare Fantuzzi for giving me the opportunity to take part on the research and for referencing me for this doctorate course: he has been a real mentor for me, and he has always been available and supportive. He gave me trust and responsibilities (like leading internship students who became my team of our industrial projects, as well as managing European projects): I will never forget this important opportunity.

I would like to thank my research group team members: Marco Sorge, Luca Manfredotti and Marcos De Silva. We have strictly worked together as a real coordinated team. Since the first time, we were very comfortable in working together, even when dealing with several industrial projects and strict deadlines. I've always trusted them as we created an important friendship.

I would like to thank Ing. Lorenzo Tacconi for providing us with very interesting software engineering courses: all our meetings have been always very inspiring and provided us with tools and techniques which helped a lot our work. I recall him as a mentor: during our conversations, we discovered new software and way of working which automatized many working processes.

I would like to thank Prof. Dr. rer. nat. Niggemann, Prof. Dr. Jasperneite, the Big Data Group (Khaled, Nicolas, Florian) and all the personnel of Fraunhofer IOSB-INA (like Mischa, Karin, Astrid, Ken, Andrej, Carsten, Heike): since the first day, they gave me trust, many responsibilities and I had the opportunity to learn a lot from them. This was an amazing experience that I will never forget.

I must express my profound gratitude to my parents Antonio Longo and Maria Pannucci, my family (especially my grandparents Santo Longo, Annamaria Longo, Nunzio Pannucci, Pia Pannucci, my uncle Raffaele Longo and my cousins Alessandro, Mirko and Ludovica Longo), Anipa Duulatova, Bakia Duulatova, Mykhaylo Bazyuk, Oksana Bazyuk, my girlfriend Kader Löle, her family and my close friends (Andrea Bofrone, Antonello Giannetti, Paolo Mecozzi, Claudio Verducci, Lorenzo Di Cintio, Lorenzo Orlando, Koresh Khateri, Damjan Miklic, Riccardo Isola, Flavia Di Noi, Cataldo Saracino, Andrea Cavallo, Andrea Di Matteo, Iginio Chirivì, Philip Merdian, Matteo Martinelli, Ascanio Tracchi, Alberto Vestrali, Ilaria D'angelo, Chiara Biagioni, Andrea Bicceco, Laura Cardellini, Thomas Krahn, Michael Wojtynek).

Their support and care have been important to me, especially during the hard times.

I am proud to have such people close to me.

Abstract

Nowadays, the number of interconnected devices is increasing dramatically: devices used in everyday life are a source of data that can be used for any purpose.
Gaining value from this data is the most important task: data can be used for understanding the interested environmental trends, predicting their behaviour and thus, generating new data. This view can be applied to the management of automated machines, providing with the possibility to analyse their working status, to understand and improve their throughput and to accomplish the necessary maintenance operations in time.

Thus, in a so structured and interconnected environment, high volumes of data are being generated day by day, creating Big Data which are ready to serve deep analyses and develop advanced monitoring solutions.

The work thesis is based on a project developed in collaboration with Elettric80 S.p.A., with the aim of developing a monitoring system for its laser guided AGV systems, to then be part of the company commercial offer. The thesis initially shows the state of the art of maintenance techniques, then introduces the theoretical concepts on which it is based, such as:

- Big Data;
- Message Brokers;
- Software containerisation systems;
- Hardware In the Loop;
- Industry 4.0.

The methodology used in the project is then illustrated: its requirements are collected and analysed. A first conceptual architecture is then defined: this must respect several constraints including the capability to manage a large amount of data, as well as being capable to save them easily on the database. In fact, Elettric80 has several customers who also

manage hundreds of laser guided AGVs and the system must be able to easily handle this complexity.

Based on this architecture, software solutions are chosen to meet the design requirements. The final solution is then explained in all its components: starting from the machine having the task of being able to send data from its sensors, a system installed on a server is responsible for acquiring such data, processing and showing it in a real-time fashion or in terms of batches.

Solutions such as NoSQL databases (Apache Cassandra) and Message Brokers (Apache Kafka) are milestones of the architecture, as they allow you to easily manage huge amounts of data coming from all the machines of the customer, to analyse and save them safely. Certain types of analyses were defined with Elettric80 during the development of the software solution: analyses such as the quality of navigation and data coming from the machine sensors were implemented. A web dashboard will then have the task of showing the collected and analysed data. Finally, additional solutions have been implemented in order to make this architecture solid: checks are carried out so that all the components work properly and act in case they are not working as expected (automatic troubleshooting and technician alerting).

Finally, the achieved results are explained and commented as well as the way the project has been leaded shown. Future works are explained in the last chapter.

Sommario

Al giorno d'oggi, il numero di dispositivi interconnessi sta aumentando notevolmente: dispositivi utilizzati nella vita di tutti i giorni sono una importante sorgente di dati che può essere usata per qualunque scopo. Acquisire valore da questi dati è il compito più importante: i dati possono essere usati per conoscere i trend all'interno dell'ambiente di interesse, predirne il comportamento e perciò generare nuovi dati. Questa visione può essere applicata alla gestione delle macchine automatiche, dando la possibilità di analizzare il loro stato di funzionamento, comprenderne e migliorarne il throughput ed effettuare le necessarie operazioni di manutenzione in tempo.

Perciò, in un ambiente così strutturato ed interconnesso, un alto volume di dati viene generato giorno dopo giorno, creando i Big Data: questi sono a loro volta utilizzati per effettuare analisi approfondite e realizzare avanzati sistemi di monitoraggio.

Il lavoro di tesi si basa su un progetto sviluppato in collaborazione con Elettric80 S.p.A., con l'obiettivo di sviluppare un sistema di monitoraggio per sistemi AGV a guida laser, per poi corredare l'offerta commerciale dell'azienda. La tesi mostra inizialmente lo stato dell'arte ad oggi delle tecniche di manutenzione, dopodiché introduce i concetti teorici su cui essa si basa, quali:

- Big Data;
- Message Brokers;
- Sistemi di containerizzazione software;
- Hardware In the Loop;
- Industria 4.0.

La metodologia utilizzata nel progetto di tesi viene quindi illustrata: i requisiti di progetto vengono raccolti ed analizzati. Una prima architettura concettuale viene quindi definita: questa deve rispettare diversi vincoli tra cui anche quello di poter gestire una grande mole di dati, nonché poterli salvare agilmente su database. Infatti, Elettric80 ha diversi clienti che gestiscono anche centinaia di AGV a guida laser ed il

sistema dovrà essere in grado di poter gestire agevolmente tale complessità.

Basandosi su questa architettura, le soluzioni software sono scelte in modo da poter soddisfare i requisiti di progetto. La soluzione finale viene quindi spiegata in tutte le sue componenti: partendo dalla macchina avente il compito di poter inviare dati provenienti dai propri sensori, un sistema installato su un server ha l'onere di acquisire tali dati, processarli e poterli mostrare in real-time o in batches.

Soluzioni come NoSQL databases (Apache Cassandra) e Message Brokers (Apache Kafka) sono punti cardini dell'architettura, in quanto permettono di poter gestire agevolmente le enormi moli di dati provenienti da tutte le macchine presenti presso il cliente, di poter quindi analizzarli e salvarli in modo sicuro. Determinate tipologie di analisi sono state definite con Elettric80 durante lo sviluppo della soluzione software: analisi quali la qualità della navigazione e dei dati provenienti dai sensori macchina sono state implementate. Una web dashboard avrà quindi il compito di poter mostrare i dati raccolti ed analizzati. Infine, soluzioni aggiuntive sono state implementate in modo da poter rendere tale architettura solida: controlli vengono effettuati affinché tutti i componenti funzionino correttamente ed azioni automatiche vengono intraprese nel caso in cui non funzioni inaspettatamente (risoluzione automatica delle problematiche ed allertamento dei tecnici).

Infine, i risultati raggiunti vengono quindi spiegati e commentati, nonché l'organizzazione del lavoro mostrata. I lavori futuri vengono quindi illustrati nell'ultimo capitolo.

1 Contents

1 Introduction

With the advent of industry 4.0, companies have started wondering how to improve their industrial plants' throughput and how to connect their industrial machines with their IT systems. That's because, in an increasingly globalised world, complex systems like machines and plants, are becoming way more delocalised: this increments the complexity grade which the companies must control.
The quality of the final product, as well as the machine productivity, must be constantly monitored: throughput might start being lower than it has used to be or the machines might start requiring more maintenance.

A system capable of managing these complexities, overseeing the machines working status and the product quality is an important asset: it allows the companies to better manage machines and plants in one time, helping them in taking faster decisions. Systems like this might allow, for example, to determine if the possible decremented throughput is caused by a lower quality of the used raw materials in production or by maintenance processes made incorrectly, giving to the companies with the possibility to act in time.

This work thesis is based on a project (SIRO) [1] developed with Elettric80 S.p.A., an international company which offers different types of automated systems: palletising solutions, wrapping machines, AS/RS storage systems, Laser Guided Vehicles and Warehouse Management Software.

The aim of the project is to develop a machine monitoring solution responsible for acquiring, analysing and monitoring sensor data coming from fleets of Laser guided autonomous Driving Vehicles (LGV) [Figure 1].

Figure 1 Elettric80 Laser Guided Vehicles. (snippet taken by Claudio Santo Longo from Elettric80 website, 2018 Elettric80 S.p.A., https://www.elettric80.com/)

Unlike the AGV machines for pallet handling, the LGV machines are capable of freely driving inside warehouses with a high degree of flexibility, which is provided by the usage of the laser-scanner localisation technology.

The developed machine monitoring solution is also enriched with the knowledge that the Author has achieved during his period abroad, as a Visiting Student: this enrichment is shown in the last chapters, as future works.

This work thesis is structured as follows:

- Chapter 2 delivers a survey over the traditional methodologies used to diagnose faults, to monitor the asset health status and plan the maintenance processes;
- Chapter 3 shows which are the concepts on which the software solution is based, showing the already used technologies;
- Chapter 4 starts with the project requirements acquisition, shows a conceptual architecture that has been developed basing on them and on the chosen technologies;
- Chapter 5 shows the developed machine monitoring solution, explaining all its system components;
- Chapter 6 shows the results;
- Chapter 7 discusses the achieved results, explaining the outcome of the research;
- Chapter 8 concludes the work thesis by summing up what has been shown and proposes new additional improvements.

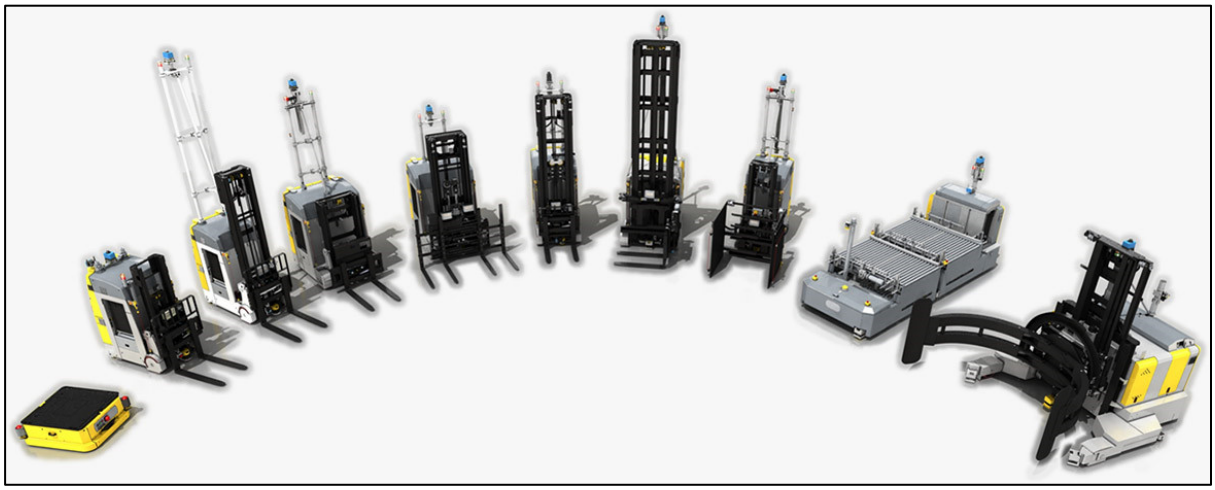

Figure 2 Elettric80 Laser Guided Vehicles. (snippet taken by Claudio Santo Longo from Elettric80 website, 2019 Elettric80 S.p.A., https://www.elettric80.com/)

2 Background on traditional fault diagnosis and monitoring systems

In a local and global context, efficiency and productivity are two strategic points which lead to the success of a company: these factors have a serious impact on the financial status of an industry and must be taken under control over time.

In literature, different ways of analysing and dealing with the incoming faults in automated machines are explained and their pros and cons analysed. These methodologies are chosen basing on the way the company wants to manage the maintenance processes and the action it must take in order to apply them. All the different types of maintenance methodologies are, of course, used to increase and maintain the availability of the company assets: the chosen strategy has impacts on the organisation's budget and can be applied only basing on the available resources, technicians experience and maintenance goals. A company that has high-budget capabilities will have a dedicated team for the maintenance processes. A small company sometimes cannot afford these costs and will rely on corrective maintenance or on outsourcing the maintenance [2]. Outsourcing is a strategy adopted when there is no competitive nor strategic advantage in doing in-house maintenance: risks might be too many and the company cannot afford a dedicated team for maintaining the assets, as it might be too much expensive or not really needed because maintenance is done only sporadically (for example, outsourcing the maintenance of "Heating, Ventilation and Air-Conditioning" systems (HVAC) is very common and cost-effective [Figure 3]).

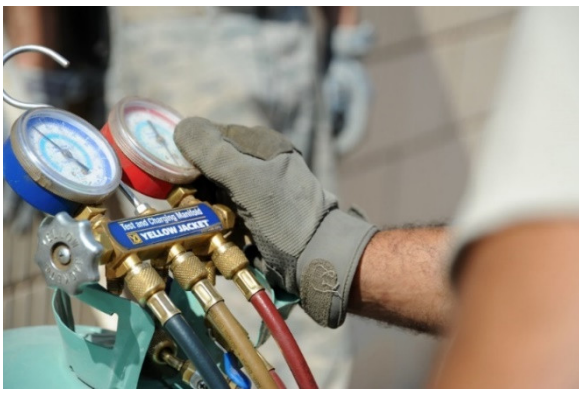

Figure 3 A technician controlling a Ventilation System
(snippet taken by Claudio Santo Longo from US air foces central command website, 2019, Official United States Air Force Website, https://www.afcent.af.mil/Units/321st-Air-Expeditionary-Wing/News/Article/934787/hvac-keeps-the-rock-cool/)

Moreover, a company can decide to use different types of maintenance strategies [3] [4] basing on the type of asset to be maintained: various assets might be totally different, thus they need precise maintenance strategies. Sometimes the used machines are very expensive and only specific components are designed to be programmatically changed over-time, sometimes machines need to be stopped and fully maintained.

Thus, maintenance is not only composed of corrective technical actions but comprehend an efficient coordination of these activities, necessary for:

- Reducing the number of stoppages;
- Reducing the number of maintenance actions to be taken over time;
- Ensuring the efficiency of the industrial plants;
- Reducing the number of programmed stoppages, due to maintenance;
- Reducing the time necessary to diagnose the faults.

Doing maintenance, taking into consideration these factors, might have a positive impact on the company's economy and competitivity.

In this chapter, subchapters 2.1 and 2.2 describe all the maintenance strategies [Figure 4] and highlight their economic impact [5] [6] [7] [8] [9] [10] [11]. Subchapter 2.3 provides with a brief description of the already used techniques for monitoring the asset health status [12] [13] [14].

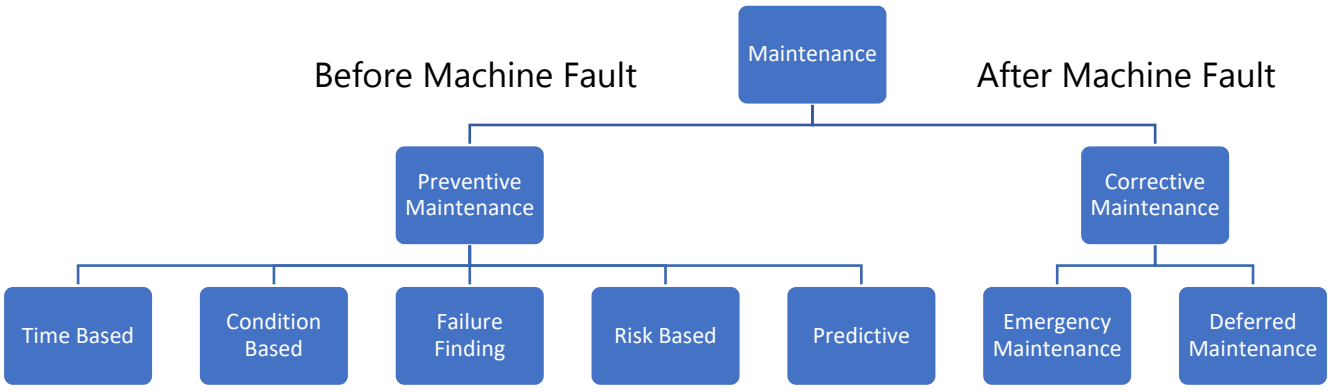

Figure 4 Types of Maintenance

2.1 Preventive Maintenance

Preventive maintenance is a process done before a failure occurs. In order to be implemented, the failure root causes to be eliminated must be known, potential failure locations must be found, and breakdowns caused by deteriorating equipment conditions must be avoided. As this information is well known, set of sensors and strategies (based on time and usage) are used to implement an optimal maintenance plan.

Sometimes, specific components require periodic servicing or replacement: this means that preventive maintenance is applied in terms of time intervals. In other scenarios, the usage of sets of sensors and other technologies (like image-recognition) allows the companies to keep the machinery under control and maintain them only when needed. Both strategies have pros and cons and must be chosen according to the company's needs and capabilities: the first, known as preventive maintenance, helps the company to bring back the asset like new, but it is invasive, requires specific know-how and implies downtime, letting the equipment be unusable for certain periods of time; the second strategy allows the company to implement a predictive maintenance strategy, stopping the machines only when the fault is predicted: in this case, the company might encounter a very positive effect on its financial and productive capabilities, as downtime is reduced as well as the number of maintenance operations.

In this subchapter, a list of preventive maintenance strategies is shown.

2.1.1 Time Based Maintenance

This is a maintenance strategy that requires maintenance tasks to be performed at already pre-defined time intervals, while the asset is still operational. Assets are designed to be durable in time: thus, this kind of maintenance is done periodically with the intent of checking the conditions of the machine and maximising their performances.

Thus, Time based Maintenance is planned in time and performed regardless of the fact that the assets require maintenance or not: that's because is assumed that incoming faults can be predicted in time. Given equipment MTBF, Time Based Maintenance is applied accordingly: companies use software solutions, like CMMS, in order to collect and elaborate data regarding asset faults, at

their best. Once all data is elaborated, assets maintenance plan is defined and applied.

This strategy is not very expensive to be performed but might lead to activities done too much frequently. Moreover, when this strategy is solely adopted, random breakdowns may anyway occur, causing economic and productivity losses.

2.1.2 Failure Finding Maintenance

This type of maintenance aims at detecting hidden failures associated with security devices or measures: this collides with all the safety equipment used in case a fault occurs. Thus, this asset is normally never used and is designed to be used only for safety reasons (an example can be provided by safety valves which must be periodically checked). This means that this asset is checked at fixed time interval, based on the country legislation or risk-based approaches.

2.1.3 Risk Based Maintenance

RBM is based on prioritising the maintenance, basing on which asset is most likely to fail over time: this allows the company to concentrate its maintenance effort on the most sensitive asset. Thus, assets that have a greater risk to fail are maintained and monitored more frequently than the others: this allows to minimise the risk of failure across the entire production plant, in the most economical way.

Risk Based Maintenance leads to the continuous optimisation of the maintenance processes as it is also based on testing and inspecting the asset and continuously define which machine is more due to fail.

2.1.4 Condition Based Maintenance

This kind of strategy is based on analysing the machine working status and on determining whether the asset is due to fail. The asset and its output are monitored over time: the product quality, as well as the machine health status, might determine the condition of the asset. Thus, CbM is based on the asset physical conditions: machines are due to degradation over time and this is

reflected on the output quality. Basing on this behaviour and on the technicians' experience, determining when maintenance is needed is made possible. In order to measure the machine working conditions, set of sensors measuring temperature, pressure, vibration and noise are being used and the acquired data is crossed with the maintenance technicians experience, providing information about when maintenance is needed.

Thus, CbM allows the companies to understand whether the asset is due to fail and when, basing on its physical evidence.

2.1.5 Predictive Maintenance

PM is a type of Condition Based Maintenance that is based on:

- the collection of machine data over time;
- on the prediction of possible machine malfunctions in time.

Unlike CbM, Predictive Maintenance relies on precise formula and combine them with sensor data (acquiring temperature, noise, pressure and vibration) in order to provide precise information about the needed maintenance.
This means that, this kind of maintenance is way more accurate than the Condition based Maintenance as predictive formulas are being used. PM is born with the advent of industry4.0 and the implementation of the IoT technologies: thus, the term IIoT (Industrial Internet of Things) is now broadly used as the market wants to implement the IoT in industrial scenarios, bringing the advantages of way more connected and controllable machines.

2.2 Corrective Maintenance

Corrective maintenance focuses on restoring equipment to its normal operational state after a fault has been detected, replacing or repairing the faulty parts/components. This type of maintenance can be successfully applied when it does interest parts which are inexpensive, simple to replace, and the failure doesn't affect any high-value asset.
Sometimes, machines and components might break down without any kind of control. Thus, even this kind of scenario is included in corrective maintenance and might lead to catastrophic events which must be treated as soon as possible: repair costs might be very high, and the production plant might face

very long downtimes, dwarfing what you would have spent on preventive maintenance.

In this subchapter, a list of corrective maintenance strategies is explained.

2.2.1 Emergency Maintenance

This is a type of maintenance that occurs when the asset requires immediate attention, in order to keep a production plant operational and safe. When needed, Emergency Maintenance gets top priority over the other, already scheduled, maintenance activities: it is designed to cope with machine faults which were not inadequately covered by preventive maintenance. This is the most expensive and least efficient type of maintenance: that's because, when faults occur, the company must stop the other maintenance activities and sometimes may lack in spare parts which must then be ordered and paid, overpriced, increasing the downtime. This leads to increased management costs and decreased performances.
This kind of maintenance can be avoided by applying efficient Preventive Maintenance plans.

2.2.2 Deferred Maintenance

Like the Emergency Maintenance, this type of maintenance occurs when a fault in the production plant happens.
The difference between both corrective maintenance types is the fact that Deferred Maintenance can be scheduled, due to limited company budged capabilities and available resources. Postponing maintenance may lead to save costs and meet the budget funding level but may also lead to the deterioration of the faulty machines: it's a matter of fact that, Deferred Maintenance must be applied with precise criteria and with timings which will not furtherly impact the faulty asset.

2.3 Monitoring Systems

In order to control the assets health status, many solutions may be adopted:

- Manually controlling the machines: this is a process used by companies which are only applying time-based maintenance. The usage of IT systems is minimum, and it is only related to scheduling and registering the inspection feedback, by using set of spreadsheets and notes. Sensors and advanced control systems are not considered, and the assets are manually regularly inspected (by checking the machines and their SCADA systems). This kind of strategy, when not integrated with IT systems, might not be very flexible as the maintenance managers must use paper and pen and the inconveniences might not be handled in an agile way;

- Controlling the machines health status by integrating them with advanced IT systems: in this case, a set of sensors is being used to control the machine health status over time. Data like temperature, vibrations as well as the products final quality are monitored over time: this data is being collected and used by software solutions in order to alert the technicians in case of incurring anomalies and to help the management in planning the maintenance activities. Thus, these software solutions help the companies in managing all the maintenance activities, preventing faults (when possible), improving the throughput of the machine and improving the performance of the asset. In this case, we talk about the usage of Computerized Maintenance Management Systems (CMMS) [Figure 5], which allows the companies to track all the faults, to plan maintenance activities, as well as to have a clear picture about the owned asset;

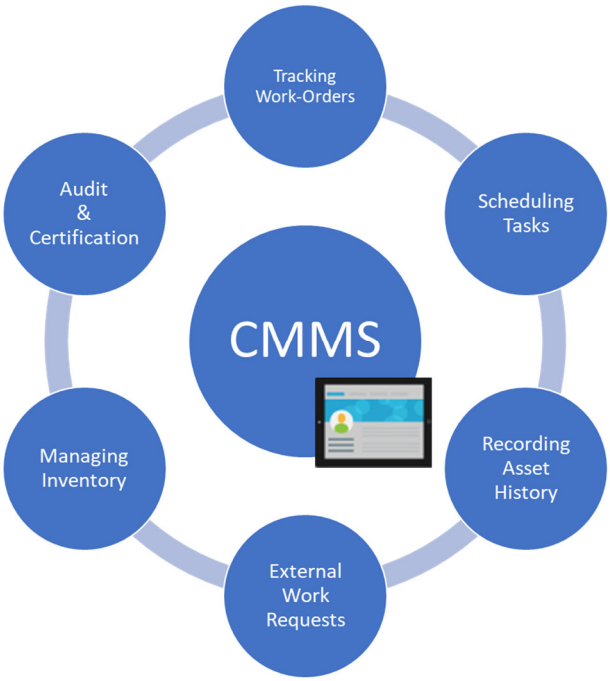

Figure 5 CMMS purposes

With the advent of Industry4.0, markets are now moving towards the application of IIoT technologies and the implementation of predictive maintenance systems: major IT companies are tackling the automation sector, providing solutions capable of integrating machine learning, Big Data and Cloud solutions with the sensor data acquisition technologies. The aim is to deliver systems capable of predicting faults in time, allowing the companies to implement smart maintenance processes (maintenance is done only when needed).

3 Related Work

The architecture of the machine monitoring solution, output of this work thesis, has been developed basing on the nowadays edge technologies: before the development of this complex system, a study over the most important technologies and concepts has been made.
The following chapter provides a survey of them.

3.1 Introduction to Big Data and difference among NoSQL databases

The nowadays interconnected devices can generate a huge amount of data day by day. Considering the nowadays privacy regulations, companies can store this growing amount of data, analyse it and generate insights: due to its exponential growth, information needs to be stored quickly and safely. Moreover, we need to consider that data is becoming very heterogeneous and unstructured because it is generated by several different devices: thus, in order to be used for generating value, data needs to be pre-processed or to be stored by means of software solutions capable of handling it in a non-structured format and of supporting parallel-computing [15] [16]. Due to the complexity that companies are facing, NoSQL databases are being used and generate the nowadays Big Data: with this term we intend a massive dataset that comprehends data coming from different sources. Many IT companies are using them to generate insights and thus, value: in fact, these technologies are being used in any kind of situation where data is being generated in a very fast way and insights are needed very quickly (or are being acquired from big data-sets) in order to provide the companies with important information that may vary from the social-media trends analysis to working environment alerts. Moreover, with the advent of IoT (Internet of Things) and IIoT (Industrial Internet of Things) technologies, the usage of NoSQL databases is becoming crucial because of their capabilities: in these use-cases sensor data is being acquired, analysed and insights are generated.

We can start from the definition of Big Data, by reading what the National Institute of Standards and Technology (NIST) states: *"Big Data consists of extensive datasets, primarily in the characteristics of volume, variety, velocity, and/or variability, that require a scalable architecture for efficient storage, manipulation, and analysis"* [17].

This definition highlights four characteristics (the four V's), which are the main attributes of these complex systems [18]:

- Volume: this is the main characteristic that leads to the definition of "Big Data". All the acquired data is being collected and stored in these systems, which tend to become huge day by day;

- Variety: collected data is heterogeneous as the data sources are different each to other, generating structured and unstructured data;

- Veracity: refers to the incompleteness and inconsistencies in data as it is being acquired by different sources and the communication channel or the devices might be corrupt or might send data in an inconsistent way;

- Velocity: not only the number of available communicating devices is huge but also the transfer speed provided by them.

The complexity brought by the four V's is addressed by the usage of NoSQL databases: despite the traditional RDBMS [19] which might only "scaling up" by using faster resources, NoSQL databases bring the advantage of "scaling out" by replicating and managing data in multiple clusters and offering load balancing capabilities (important when data grows up exponentially and very quickly). Moreover, NoSQL databases offer the possibility to handle unstructured and structured data, impossible task for classical RDBMS (a pre-processing is needed). On the other hand, RDBMS guarantees ACID (Atomicity, Consistency, Isolation, Durability) transactions and the powerfulness of the SQL query language.

Due to the heterogeneity of the NoSQL database type, classification of them is mandatory and it is shown below [20] [21] [22] [23] [24] [1]:

- Wide Column Store databases: are a distributed storage system that stores data in columns. In order to perform queries, they use a SQL-like language and they have a graphical representation that is similar to the RDBMS.
  Moreover, as said before, columns might be added afterwards without any sort of problem, giving the opportunity to handle heterogeneous data and store newly added features to the data sources.
  This type of database was created in order to store a huge amount of

data to be then distributed across servers and clusters.

- In-Memory Databases: also known as Key Value Store, is a type of databases that uses the hardware memory to store and serve information. They are similar to dictionaries where information is stored in form of keys and values. While values are opaque to the system, the only way to retrieve them is to refer to their keys which are uniquely stored. Since these databases store data in memory, the usage of NVRAM memory is also recommended as it can old the processed data even if the system is down;

- Document-Oriented Databases: are databases designed for storing, reading and managing document-oriented information, which is known as semi-structured data. In fact, they consist of versioned documents that are collections of key-values combinations. Documents are identified by an "ID", which is also unique. Values are not opaque to the system that can be queried, and they are encapsulated in JSON or JSON-like documents;

- Graph Databases: in contrast with the already explained NoSQL database types and SQL databases, this kind of database is used in case data is heavily linked. The discovery of the relationship between data is optimised and fast querying and lookups are made possible.
  In order to represent the data, graph databases consist of:
  - Nodes: entities which have to be tracked and are equivalent to a record stored in an RDBMS;
  - Edges: define the connection and thus, the relationship between Nodes. They have a direction, a type, a start and end Node. Properties quantify the relationships, defining information like costs and weights.

  Most of the use case in which the graph databases used are social-medias, location-based services and retail solutions.

3.1.1 A Comparison Between NoSQL and SQL databases

A comparison between the SQL and NoSQL features is shown below [25] [26] [20] [21] [22] [23] :

- Scalability: as said before, SQL databases can "scale-up" (vertically) and NoSQL databases can "scale-out" (horizontally). This means that,

meanwhile the traditional RDBMS can be handled by adding faster hardware resources to the server, the NoSQL databases can be handled by adding more nodes to the cluster. This kind of manageability has impacts on the costs because the upgrade of old systems might be expensive due to the lack of support and software limitation. On the other hand, the upgradability opportunities offered by NoSQL, allow to use new hardware and to make a transition of the data to newer hardware platforms. In fact, NoSQL databases rely on the BASE (Basically Available, Soft state, Eventually consistent) principle, coming from the CAP theorem [Figure 6] [27] [28]:

- Consistency: data replicated on a different server must be the same;
- Availability: data must be always accessible;
- Partition tolerance: data and databases must always be accessible despite machine or network failures.

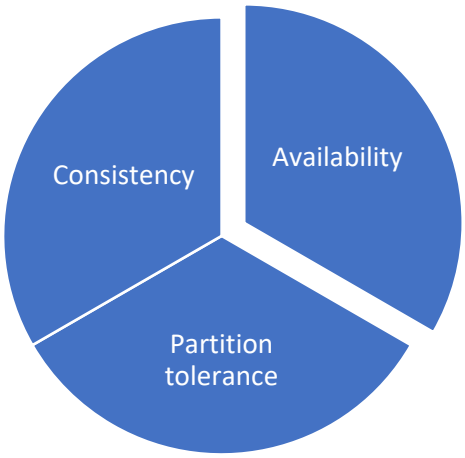

Figure 6 The CAP Theorem

- Query Language: meanwhile RDBMS use the Structured Query Language, NoSQL databases use a language that depends on the database itself.
  We can then highlight the consequences as follows:
  - Relational Database Management Systems use a standardized and yet very powerful language which allows using a very structured query. The design of an RDBMS depends only on the nature of the data;
  - NoSQL databases are very heterogeneous, and every database server solution has its own language. The provided query language is not so strong as the SQL, and data retrieval (for batch

processing) is sometimes only possible by using other frameworks (e.g. Apache Storm, Apache Flink..).
The database design depends on the application and on the nature of the data to be handled.

- Flexibility: RDBMSs have a fixed schema that must be pre-defined before data insertion and NoSQL databases have a dynamic schema that has not to be pre-defined. Another point is the data type to be managed: as RDBMS can only handle structured data, the NoSQL database can handle any kind of data (structured, semi-structured and unstructured);

- Data Management and Access: while in SQL databases, data is normalised and redundancy is avoided, in NoSQL databases "data redundancy" is inevitable due the lack of relationship between data. When performing replication across computer clusters, availability is always increased but performance is only decreased in RDBMS, with long time and storage consumption consequences;

- Security: most of the NoSQL databases do not provide with a lot of mechanisms capable of ensuring many security measures types. Meanwhile, cryptography in some NoSQL server is guaranteed, features like authentication, access control, secure configurations, and auditing are not guaranteed in NoSQL databases. Due to this situation, data security is way more impacted when stored data is stored in different computer cluster located in different locations. RDBMSs provide with all the security features described above.

Basing on the comparison shown above and the use case requirements, software solution might use SQL, NoSQL or both database types and achieve the final goals providing the end-user with the best performance and experience. These choices must be based on the project purposes and its constraints.

3.2 Message Brokers Systems

Sensors are installed on machines and are responsible for acquiring and sending a high volume of data over time. In order to cope with this scenario, Message Brokers Systems [29] [30] come in hand, as they provide with the possibility of:

- acquiring high volumes of data;
- storing it in a shape of logs (data retention);
- serving multiple publishers and subscribers.

These solutions are publishing-subscribing systems which allow multiple device and software solutions to intercommunicate each with other: moreover, frameworks like Kafka integrate a functionality which allows the developers to analyse data on the fly. This is an interesting solution when the hardware usage must be optimized, as the end-user does not need to use other frameworks in addition to the ones used for acquiring data. In literature, many articles talk about them and provides the reader with use-cases and bench-markings [30], [31], [32], [33], [34], [35] [36] [37]. The Message Brokers Systems allow the development of efficient pipelines, establishing interconnections between all the used IT solutions by means of the same software interface and thus, becoming the core of the final software solution. Moreover, frameworks like Apache Kafka provide with the possibility to retain data for certain amount of time, providing with the possibility to recover and analyse batches of data. Message Brokers use different communication models: the common consists in using queue and topic, creating different streams of data which can be transformed (e.g. transformation from XML to JSON) and lately consumed.

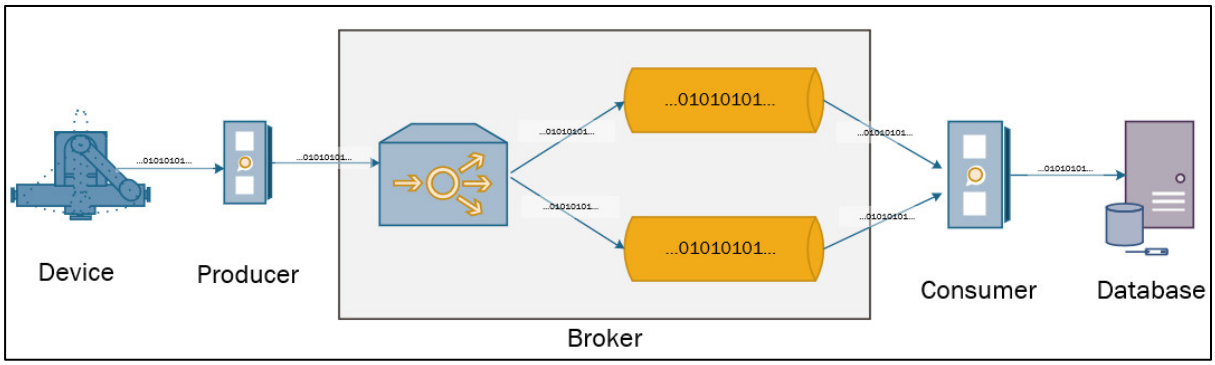

Figure 7 Example of a Message Broker System Architecture

The functioning mechanism of these systems is depicted in [Figure 7]:

1. Producers receive data from other devices and are responsible for sending it to the Broker System;
2. The Broker acquires data from the Producers and is responsible for routing it in different queues, according to the Producers' instructions and the system load;
3. Messages are then received by the Consumers: these are the connection point between the Broker and the end device or software solution that will read and use the streamed data. These messages can be also pre-processed by the consumers, providing with the possibility to serve the end-system with data in the desired way.

Message Broker Systems carry multiple properties:

- Scalability: data transfer speed can be always assured by splitting the data in different queues. Moreover, Brokers can be clustered providing load balancing and replication capabilities;

- Flexibility: basing on the use-case, communications between different endpoints (producers and consumers) can be established and interrupted without letting the system be faulty. Moreover, data can be filtered, providing the endpoints with the needed data;

- Extensibility: nowadays, Brokers Systems provide with the possibility to use different programming languages. This allows the company to continue using its already used technology without the need of using a new one, letting the integration of the chosen Broker System feasible;

- Fault tolerance: Brokers not only carry replication capabilities but provide with the possibility to store the managed data on the disk, preventing data loss in case of faults. Moreover, in case a server or a node responsible for transferring data is being lost, the Broker can automatically manage this situation, by stopping non-responsive services and dynamically managing the existing queues.

3.3 Delivering containerised applications with Docker and Orchestrations Tools

When delivering software solutions composed of a multitude of different code components and services, companies need to install every single used software and must make sure that its software solution is compliant with the customers' system environment. This means that, not only the software solution must be compatible with the hardware and operative system provided by the customer, but also dependencies need to be satisfied [38] [39] [40]. The usage of classical Virtual Machines running on top of the same Hypervisor provides isolation and scalability of the applications but can increase the used resources, the redundancy and the overhead because multiple deployed Operative Systems are being used at the same time. Moreover, use cases like big data processing and provisioning require the ordinary user to install and use complex big data analytics solution which might be a significant challenge.

The problems shown above can be handled by adopting the popular Docker containerisation solution and scale its containers by means of Kubernetes. This chapter provides the reader with a brief description of them, highlighting the advantages.

3.3.1 Docker

Docker is an open platform which allows companies in developing, shipping and running different applications very quickly [41] [42]. Unlike the traditional "hypervisor - virtual machines" [43] architecture where VMs are an abstraction of the physical hardware and all of them are placed on top of a hypervisor (which coordinates them), with docker the abstraction is obtained at the application layer [Figure 8]. Applications and their dependencies are packaged inside "Containers" and can run as isolated processes (by means of cgroups and kernel namespaces [44]) on the same host, sharing the OS kernel. By using Docker, the redundancy of multiple virtual machines is reduced, and Containers can run in multiple virtual machines as well: for example, an application may need to read data stored on a database that resides in a different virtual machine and both might be docker containers as well [45], maximising the utilisation of the server resources.

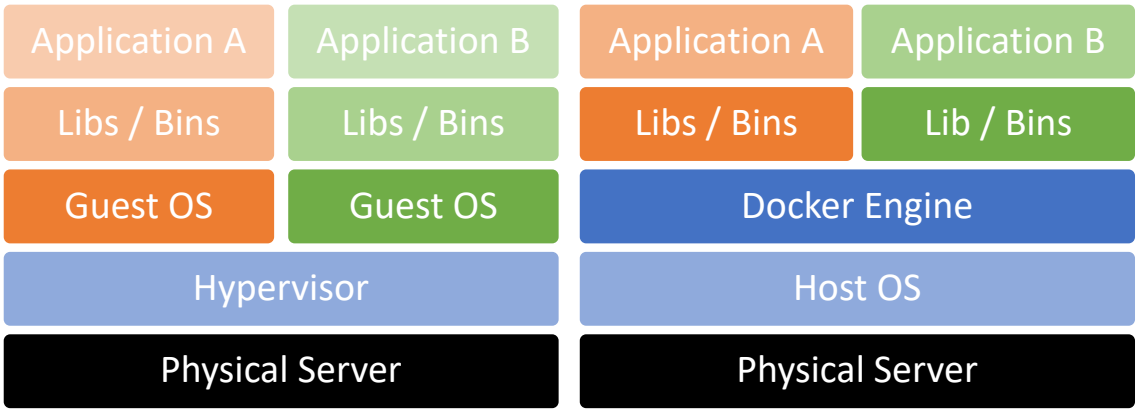

Figure 8 Difference between regular "Hypervisor-Virtual Machines" solutions and Docker

The advantages of docker are the following [39]:

- Portability: all the applications and dependencies are packaged in Containers. These can be easily moved to different platforms, allowing to use the same software solution in the same way and providing applications which can run with the same original behaviour;

- Lightweight: instead of using several virtual machines, which causes redundancy and an increase in resources utilisation, Docker containers are an abstraction of the needed application which does not need to run on a specific Host OS. Starting and stopping a Docker Container is a fast process if compared to managing multiple virtual machines;

- Optimised resource utilisation: we can run more containers on the same Host OS and define the amount of resources to be allocated. Moreover, containers can share the same data, reducing or removing redundancy;

- Fit for microservices architecture: Docker Containers provide with the possibility to deploy software solutions as microservices, reducing the complexity provided by monolithic architectures and improving their maintainability and performances.

Docker works by means of a client-server architecture: clients (Docker Containers) communicate with the server (Docker Daemon) which is hosting them. The Daemon is responsible for running the Containers: the communication among them is established by means of a network bridge (docker0).

The main components of Docker are [46]:

- Docker Engine: this is the layer on top of which Docker runs. It's a lightweight runtime responsible for running the containers;

- Docker Client: a command-line tool is used for communicating with the Docker Daemon;

- Docker Daemon: it is responsible for executing the commands sent with the Docker Client and runs on the host machine;

- Docker Image: it is a template used to create a Docker Container and is buildable by means of the Dockerfile;

- Dockerfile: it is a script used to store all the instructions necessary to build a Docker Image [Figure 9];

```
1   FROM java:8
2   LABEL maintainer="CLAUDIO SANTO LONGO & KHALED AL-GUMAEI"
3   ADD /target/kafka-streams-temperature.jar kafka-streams-temperature.jar
4   ENTRYPOINT ["java","-jar","kafka-streams-temperature.jar"]
5
```

Figure 9 A Dockerfile packaging a Kafka Streams Application

- Docker Registry [47]: consists of a server-side application which holds the docker images and might be public (Docker Hub) or private (used for internal releases). Developers and DevOps can easily use a remote registry to push and pull their applications and deploy them into production very quickly;

- Union File System (UFS) [48]: images are composed of different layers and when multiple images are using the same data, then this layer is being used by all of them without duplicating it;

- Docker Container: built off docker images, Containers contain all the necessary files in order to run the desired application;

- Volume: data part of a container. This data is not part of the UFS and consists of a directory, stored on the Host OS HDD, responsible for persisting container's data and share it between multiple containers.

3.3.2 Containers Orchestration with Kubernetes

In case a set of multiple containers is supposed to run on a computer cluster, a container orchestrator is a very important solution: it allows to manage multiple containers which reside on different servers. Containers management not only means starting and stopping containers but also their automatic deployment, scaling (creating a different number of replicas and providing with high availability), monitoring, scheduling, coordinating, securing and connecting them [49].

According to Forbes [50] and by querying the trends of the moment [Figure 10], Kubernetes is the most popular Orchestration Tool until now. Moreover, after a deep study about the Docker Orchestrators of the moment, [49] states: *"Kubernetes is one of the most complete orchestrators nowadays on the market".*

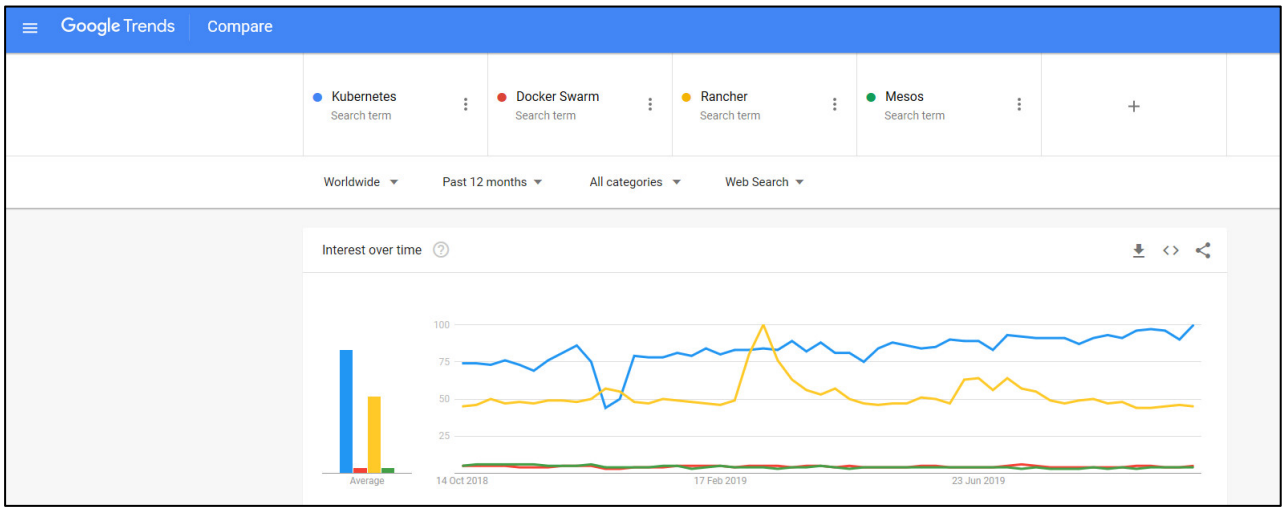

Figure 10 Most popular Orchestration Tools based on interest (snippet taken by Claudio Santo Longo from Google Trends, 2019, https://trends.google.it/trends/explore?q=kubernetes,Docker%20Swarm,Mesos)

Most popular means, more interest in the tool and more support by the community as well as continuous development and improvement of the software.

Kubernetes [51] [52] [53] [54], developed by Google in 2014, is an open-source platform for managing containerised applications and provides the end-user with:

- Storage Orchestration: possibility to mount on-premise or in-cloud storage systems;

- Service discovery: exposure of docker containers by means of DNS or IP addresses;

- Load balancing: network traffic is automatically balanced and distributed, maintaining the deployment stable;

- Automated rollouts and rollbacks: describe the desired state of the deployed containers, deploy and exchange the old with the new releases;

- Automatic bin packing: allocate resource usage to running containers;

- Self-healing: containers which suddenly fail are subtly restarted, always providing with the pre-defined number of up and running replicas;

- Secret and configuration management: sensitive information is stored and used in order to provide with more control and less exposure of the deployed applications.

The Kubernetes architecture is depicted in [Figure 11] and briefly explained as follows [55] [56]:

- Containers: they are running on a Worker Node (a Virtual Machine or Physical Server), they are responsible for running the application and they are inside a pre-defined Pod;

- Pod: is a management unit in Kubernetes which comprises one or more containers. Each of them has a unique IP-address, storage, namespace and option which define how the containers should run. Every container, inside of it, shares this networking and storage resources. We can use a .yaml file to define the Pod attributes like metadata and Apiversion. The number of Pods replica can be specified and managed by means of a “Deployment” .yaml file or instruction.

Finally, Pods can be accessed by means of a "Service" .yaml file or sets of instructions which define their access policies;

- Worker Node: a virtual machine or physical machine, responsible for running Pods and its managed by the Master Node;

- Master Node: is the entry point for every administrative task for managing the Kubernetes cluster. Has an Api Server, a scheduler and a controller manager that watches the desired state of the object it manages and its current state (matching them).

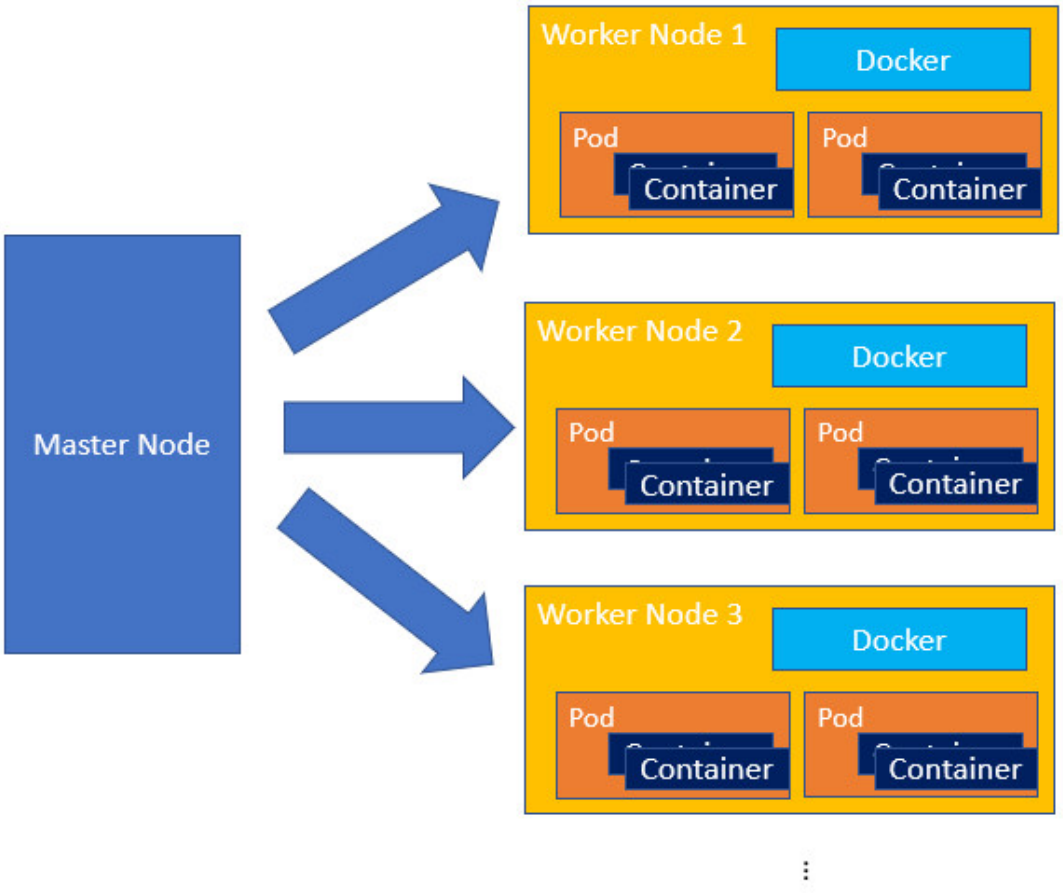

Figure 11 Kubernetes Architecture

Kubernetes can be managed through SSH and an installable Web-based Dashboard [Figure 12].

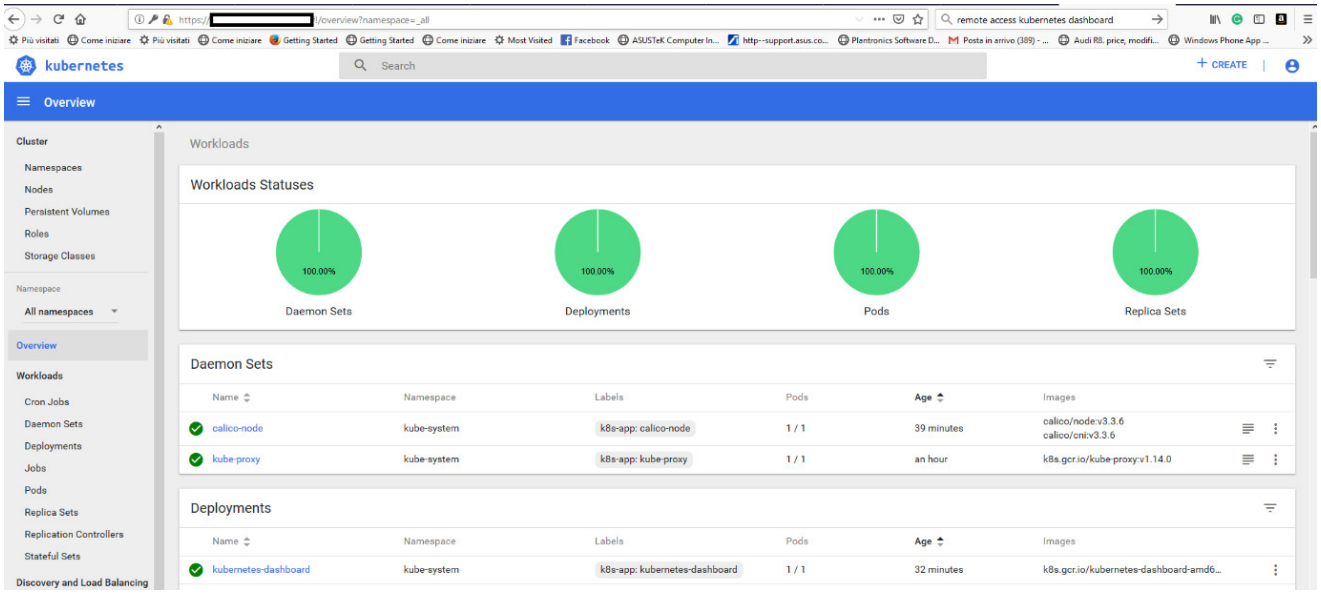

Figure 12 Kubernetes Web Dashboard

3.4 The hardware in the loop

When working with automated machines, the comprehension of their performance due to working conditions is fundamental. The execution of extensive tests on real machine might be a usual but very expensive procedure: during the preparation and execution of the tests, materials and energy are being extensively used, resulting in increasing costs and testing time.

In order to cope with this scenario, the Hardware In the loop (HIL) [57] [58] [1] [59] [60] technique is currently taken into consideration and applied for a fast Verification & Validation (V&V): it connects the real machine controllers with a simulated version of the machine. This technique can be applied to already developed machines, as well as to machines under development, because it allows the System Engineer (SE) to rapidly and safely design and test a machine before constructing it: the simulated machine interacts with the controllers to be tested and behaviour of the coupled machine-controller is obtained accordingly. HIL is also used for testing entire production lines.

On the other hand, the application of the Hardware in the Loop technique is made difficult by the development of a reliable real system model: a machine is a complex system which cannot be dominated if taken as a whole.
This situation is coped by applying the Modular Machine Development (MMD) approach: machines are reproduced by functional decomposing them into

mechatronics parts. Therefore, these complex systems are a result of the interconnection of sets of mechatronic subsystems which are simulated as single components. All the subsystems internal behaviour is hidden from the others thus, they communicate through interfaces. If the developed subsystems can be used in different machines, they can be deployed in form of libraries which can be applied multiple times and independently from the machine taken under consideration, if applicable.

The development steps of a machine system model, according to the Hardware In the Loop technique, is explained as follows:

1. Machine decomposition: according to the Modular Machine Development approach, the machine is decomposed in subsystems. This allows the modelling of them;

2. Identification of the discrete event (DE) behaviour: this step allows to model the machine according to its DE behaviour, defining whether its behaviour is a result of its sensors or internal PLC components;

3. Identification of the continuous event (CE) behaviour: in each state, the identification of CE phenomena is fundamental;

4. Loop control: this step defines the identification of CE behaviour used in closed and open control loops;

5. Development of a mathematical model of the system;

6. Testing sessions of the machine modules.

3.5 Industry 4.0

Technologies are rapidly evolving: information and automation systems are gaining benefits from this and an incredible evolution in such systems might be obtained by connecting both worlds.

Thus, starting from the Hannover fair of 2011, the term "Industry 4.0" or "Industrie 4.0" [61] [62] was born [63]: a German government initiative [64], part of the technology strategic plan for 2020. The aim is to create a "Smart Factory" of the future capable of shifting the industrial plant decision-making production from "centralised" to "decentralised".

The principles of Industry 4.0 are:

- Decentralised decision-making: the automated system must be capable of analysing the current machine working status and make decentralised decisions;

- Information transparency and interoperability: sensor data must be collected from the operating machines, providing with the possibility to analyse it and generate insights;

- Technical assistance: ability to predict failures in time, basing on the collected sensor data.

Basing on the principles shown above, Industry 4.0 comprises the following terms and technologies, which are:

- Internet of Things (IoT) or Industrial Internet of Things (IIoT) and Embedded Systems (ES): integration of sensors in an automated machine for monitoring purposes and remote controlling;

- Cloud Computing and Internet of Services: usage of internet-based services capable of providing with scalability, reliability, performances and allowing to operating leverage [65];

- Cyber-Physical Systems: monitoring physical processes by acquiring data from them, applying this data on a simulated reproduction of the production plant and provide with automatic decentralised decisions [66] (in accordance with "The Hardware in the loop" technique, [Chapter 3.4]).

4 The Methodology

With the SIRO Project, Elettric80 S.p.A. wants to constantly monitor its LGV machines, in order to provide a better after-sales service as well as enhancing the efficiency of its machines, by analysing the most occurring alarms over time and consequentially improve their machine quality and performances.

For this purpose, a system capable of acquiring sensor-data is mandatory: fleets of LGV machines were currently equipped with PLCs capable of obtaining this information and sending it to a pre-defined destination. Data must be then analysed in real-time or in batch and generate insights in form of alarms and trends: the accordingly developed software solution must help the maintenance technicians, in order to do maintenance in time, and the company engineers to improve the quality of these machines.

This project has been accomplished in collaboration with the University of Modena and Reggio Emilia, and its related software solution has been already delivered and installed in an existing industrial plant.

This chapter starts by showing the requirements acquired during the first meeting held with Elettric80 [Chapter 4.1], followed by the proposed conceptual architecture that could fit with them [Chapter 4.2]. Basing on the architecture and the requirements, the specific architectural components have been chosen accordingly and explained in this chapter [Chapters 4.3, 4.4, 4.5].

Figure 13 An operating LGV machine

4.1 Projects requirements acquisition

Several meetings have been organised by the University of Modena and Reggio Emilia (Unimore) with the intent of studying the machines, main topic of this project, and defining an architecture that had to be capable of coping with the project objectives which aim to remote monitoring and improve the LGV machines performances.

Managers and technicians have been widely engaged for understanding the LGV machines and the needs encountered by them on the field: technicians were doing maintenance basing on the alarms raised by the machines and mostly basing on their own experience. Sometimes, the raised alarms could lead to extensive maintenance which was causing downtimes and letting the relationship with their customers being harder time to time: these situations could be solved by introducing Elettric80 maintenance technicians in the customers' warehouse and by letting them assist the maintenance operations. The usage of proprietary technicians in the customers' warehouse is, of course, a cost that impacts the machines selling process: eliminating or reducing the failure causes is a way to reduce these costs and to improve the after-sales.

Thus, the participation of Elettric80 technicians was fundamental to understand how the LGV machines work and which were the most important needs the software solution had to meet: this allowed Elettric80 and Unimore to define the objectives of the project.

Moreover, during these meetings, the following information have been acquired:

- The LGV machines work by means of two types of equipment:
    - Incremental encoders: these are used to track the incremental movement of the LGV machines over time. Incremental encoders acquire information which are transmitted by a driving wheel. As the wheel is subjected to wear, this kind of system must be calibrated over time;
    - Laser-scanners: reflective bars are installed in the whole warehouse and the LGV machines can define their position by transmitting a laser to these bars. This system allows the LGV machines to freely drive inside the warehouses and accomplish pick and place operations.

- Maintenance is done basing on the alarms raised by the machines and it might be sometimes false due to other factors: this leads to an extensive inspection of the machines which could be speeded up by the Elettric80 technician experience;

- Technicians must manually monitor the machine's activities and recalibrate the navigation system when needed;

- Elettric80 had already deployed a communication system on their machines which is capable of transmitting sensor data.

One of the concerns of the technicians was regarding the navigation system recalibration: this must be monitored and accomplished by them manually. The LGV machines wheels wear over time and this leads to a wrong encoder calculation, thus to a wrong machine navigation: this situation is solved by directly recalibrating the encoder.

Thus, Unimore and Elettric80 agreed on developing a machine monitoring solution capable of:

- Acquiring and storing sensor data over time;
- Analysing the acquired data in real-time;
- Raising alarms in real-time and notify the in-plant technicians;
- Showing the acquired and analysed data in real-time on web-based dashboards;
- Being scalable and adaptable to any situation where thousands of LGV machine could operate.

For this project, they wanted to only raise alarms basing on the calibration of the encoder.

Moreover, another requirement, requested by Elettric80 during the last project month, was regarding the possibility to integrate in the architecture even some software which could be easily accessed, used and modified by the company even after the end of the project: sensor data trends and maintenance tickets, are vital for the company and must be accessed with already used in-house technologies and expertise. The company wanted to integrate the software solution inside its commercial offer as well as using it for monitoring its machines and improve them.

Moreover, the used technologies implemented in the machine monitoring solution had to be open source.

Elettric80 provided Unimore with a workbench (shown in Chapter 4.3) that is composed of a PLC unit and an encoder, for testing purposes: this allowed the research group of the university to check the PLC unit code and to communicate with it, reproducing the possible data an LGV machine could generate and acquiring it with the machine monitoring solution under development.

4.2 Conceptual Architecture

Basing on the acquired requirements, a conceptual architecture responsible for coping with such complexities, described in [Chapter 4.1], has been designed. It refers to the lambda architecture [67] [68] [69] [70] [71], which considers the following concepts:

- Data Source: streams of data coming from different sources must be acquired at the same time;
- Data can be analysed in batches and/or when acquired (stream);
- Data is stored on NoSQL or SQL databases;
- Data is then shown on a Dashboard or by means of Business Intelligence software solutions;

In this specific case, we must develop an architecture that has to cope with automated machines in movement: thus, this architecture communicates with IIoT devices which might be several (a warehouse, customer of Elettric80, manages more than 100 LGV machines at the same time).
The interconnection of the software solution with the machines is the first and important step for this project: if a proper and secure communication channel is not being established, the data acquisition might be the weakest point of the entire architecture.
Moreover, our architecture must be scalable and adaptable to any required workload: some warehouse might have 10 LGV machines and other customers more than 100 thus, this solution must be capable of handling such complexity.
Data must be analysed in real-time and then stored on databases which must be accessible in an easy way.

In order to analyse data streamed on the fly, connect all the components and ensure scalability, a broker message is mandatory.
Basing on the requirements, data must be then browsable in terms of temporal sequences as well as in real-time: Elettric80 is interested in watching the sensor data trends over time in terms of batches and as the user is watching a Scada system. If the software solutions used for this solution are not properly chosen, they might be a bottleneck for this last layer.

Figure 14 shows the Conceptual Architecture on which the final software solution is based, and it is explained below:

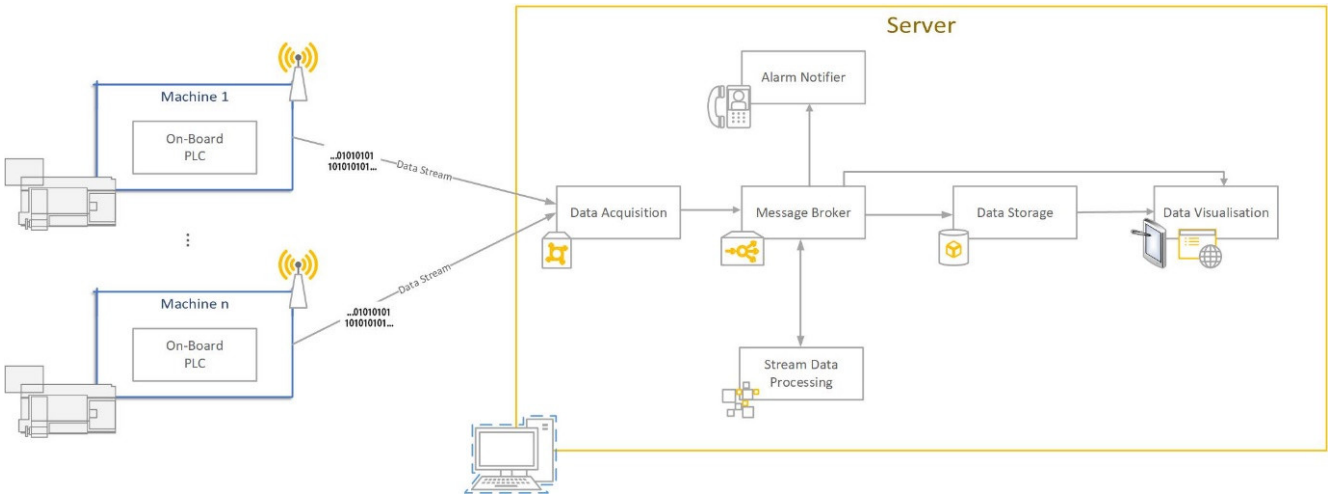

Figure 14 Conceptual Architecture acquiring data from 1-n machines

- Machines: every automated machine must be capable of packaging the information coming from its sensors and of sending it to the machine monitoring solution, which resides on a Server.
  The definition of the package structure is fundamental for the other architectural components, as it allows them to interpret the sent messages and transform them into a more readable structure. A communication channel between machines and the Server must be enabled and reliable over time, otherwise information might not be feasible for the afterwards analyses and visualisation;

- Data Acquisition: this architectural component is responsible for acquiring data from the Machines, parsing and sending it to the Message Broker component. When this component receives the message, checks its integrity and if it is compliant.

Once these steps are being accomplished, the Data Acquisition component parses the received message and it sends it to the Message Broker;

- Message Broker: this is the core part of the structure as it is responsible for receiving the data coming from the sources and then letting it be available for the other components as a unified interface. This means that, the other architectural components can access the Broker in the same way and use the same libraries, letting them be usable multiple time. The Message Broker is also scalable and reliable, letting the messages being divided into multiple topics, partitions and brokers. Moreover, some Message Brokers provide with the possibility to retain data for a certain amount of time (or number of messages per Partition), letting the stream processing to read many of them at once;

- Stream Data Processing: this architectural component is responsible for reading data on the fly and do analyses. It is composed of a set of scripts which carries algorithms into them: they are responsible for doing different analyses types and publishing their results on the Message Broker;

- Alarm Notifier: this architectural component is responsible for reading data that is being published by the Stream Data Processing or the Data Acquisition on the Message Broker(s) and alert the technicians in case an alarm is raised;

- Data Storage: this architectural component is responsible for receiving data from the Message Broker(s) and store it. The databases must be designed carefully, as a transition to another database or solution might be very time and resource consuming. Basing on the system requirements, this component might consist of one or different database types, allowing the data management and analysis being easier: different SQL and NoSQL databases can be mixed basing on the requirements and analyses to be done afterwards;

- Data Visualisation: this architectural component, consisting of a web-based dashboard, is responsible for showing the machines data and the output of the data analytics algorithms. Basing on the requirements, data can be shown in a batch or in a real-time fashion:
    - Batches of data: data is being read and shown on a dashboard. The objective is to show trends in a chosen time window (by the user) thus, batches of data are retrieved from the database and plotted;
    - Real-time data: the dashboard continuously read data from the Message Broker, providing the end-user with the current machine working status, in a real-time fashion.

Starting from this conceptual architecture and basing on the requirements, the software solutions to be used have been chosen accordingly.
In this chapter, the most prominent and chosen technologies are explained.

4.3 Chosen equipment for data acquisition

PLCs belonging to the working LGVs machines are not only responsible for controlling them, but also for packaging and sending the collected machine sensor data.

Elettric80 had already implemented a communication channel between the machines and the warehouses in which are located: thus, it provided us with a workbench which allowed us to test and debug their PLC code as well as building the machine monitoring software solution on top of it.

This solution allowed us to deep understand the structure of the code, to troubleshoot the PLC coded sensor data acquisition technique, to save time and resources by directly working with it.

The provided workbench [Figure 15] [1] was consisting of:

- PLC: Beckhoff CX5020-0111;
- Power Supplier: Siemens SITOP modular 5A;
- 1/2phasig 6EP1 333-3BA;
- Incremental Encoder used for analysing the data acquirable from it;
- Modem-Router: Netgear N600.

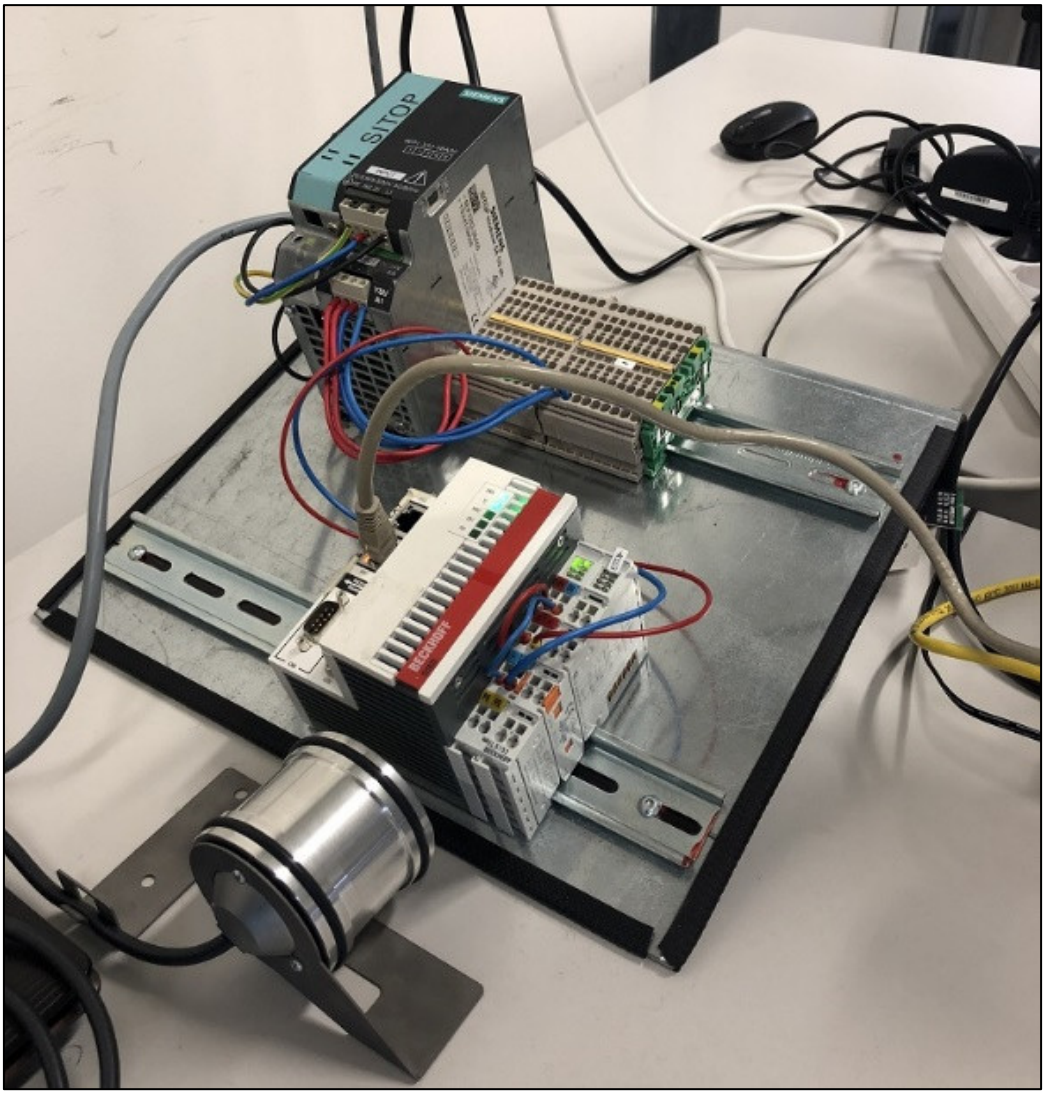

Figure 15 The workbench provided by Elettric80

Like in the real LGV machines, this workbench is capable of transmitting data by means of a UDP protocol.

4.4 Chosen database

Basing on the system requirements, two types of data must be acquired over time: machine sensor data and maintenance tickets. Data must be collected as it arrives to the software solution and the system must be scalable.

Moreover, Elettric80 needs a system which allows it to browse the maintenance tickets and trends in an easy way, even after the end of the project: a database which can be easily used by the company and which is compliant with its already used technologies, is mandatory.

Thus, two databases have been designed for coping with the project complexities:

- Apache Cassandra: a NoSQL database, to be used for storing sensor data, with fault tolerance and scalability properties;

- MySQL: an RDBMS to be used for collecting the machines maintenance tickets, define alerting rules and aggregated sensor data over time.

In this chapter, a brief overview of both databases is done.

4.4.1 Apache Cassandra

Apache Cassandra is an open-source [72], distributed NoSQL wide columnar data store [Chapter 3.1] that can ingest and process massive amounts of data very fast [73] [74] [75]. Deployable as On-Premise, in Cloud and in a Hybrid Data Environment, Cassandra was originally developed by Facebook for its inbox search system and later become an open-source project.

It provides the following features:

- Fast writes: it provides fast writing capabilities [73], letting it be an interesting database for storing sensor data storage [76].

- Elastic Scalability: horizontal scale-up or scale-down of the cluster is made possible in Cassandra without the need of restarting it;

- Fast linear-scale performance: throughput is increased as the number of nodes becomes bigger;

- Flexible data storage: structured, semi-structured, and unstructured data can be stored. It can accommodate changes in the database structure, allowing the end-user to add more column to the existing tables as well as adding more tables to the same keyspace (database);

- Easy data distribution: Cassandra distributes data across multiple datacentres by means of replication functionalities;

- Always on architecture: Cassandra is a database with no single point of failure.

Cassandra is designed to handle heavy data workloads, distributing data across multiple nodes (servers) [Figure 16] by means of a peer-to-peer distribution model. All the nodes are interconnected: when a node is down, the other nodes can anyway handle read/write requests. Moreover, as data is distributed, Cassandra is capable of understanding if a node does not contain the most recent data and of performing, consequentially, an automatic repair. Requests are handled by a node, called Coordinator, which is responsible for the managing, reading and writing operations.

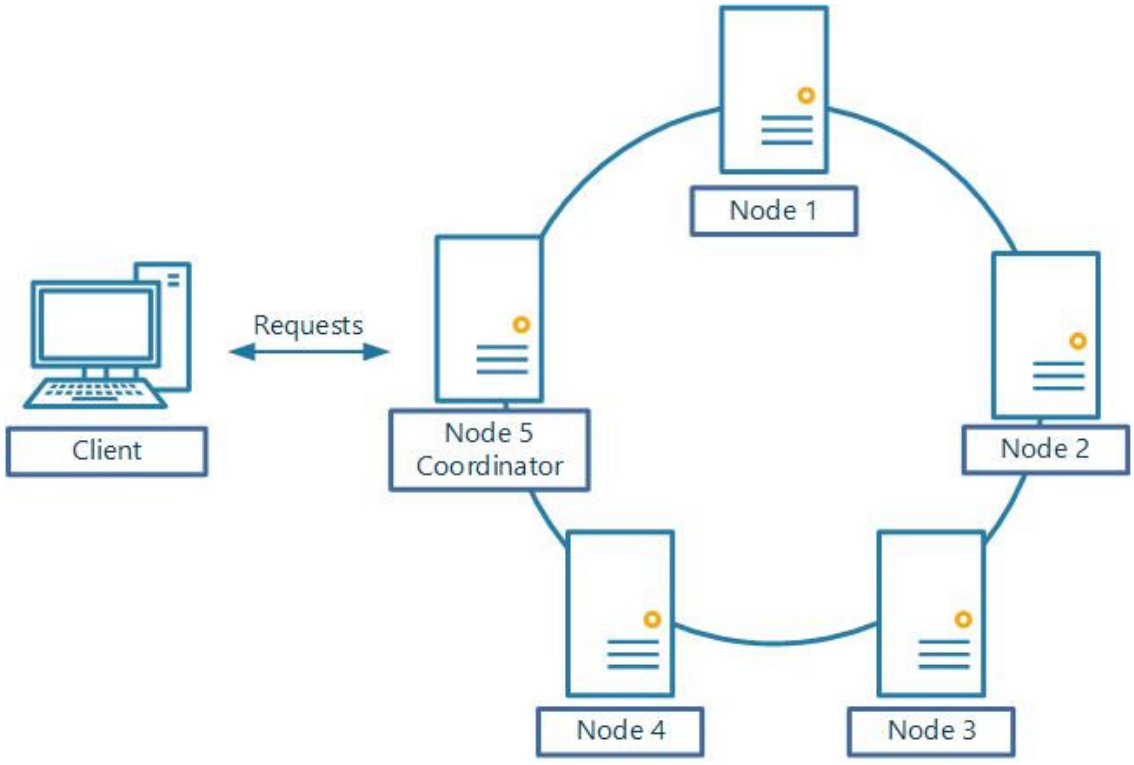

Figure 16 Cassandra Architecture

Cassandra can be queried by using its own language, that is like SQL, and is called CQL: unlike SQL, CQL has several limitations like the join operations which are not supported.
As a wide column data store, Cassandra stores data in terms of tables of rows and columns.
The primary key is composed of:

- Row Key: it uniquely identifies rows or groups of rows if used in combination with "Cluster Columns";
- Clustering Key: it organises the rows inside each Row Key.

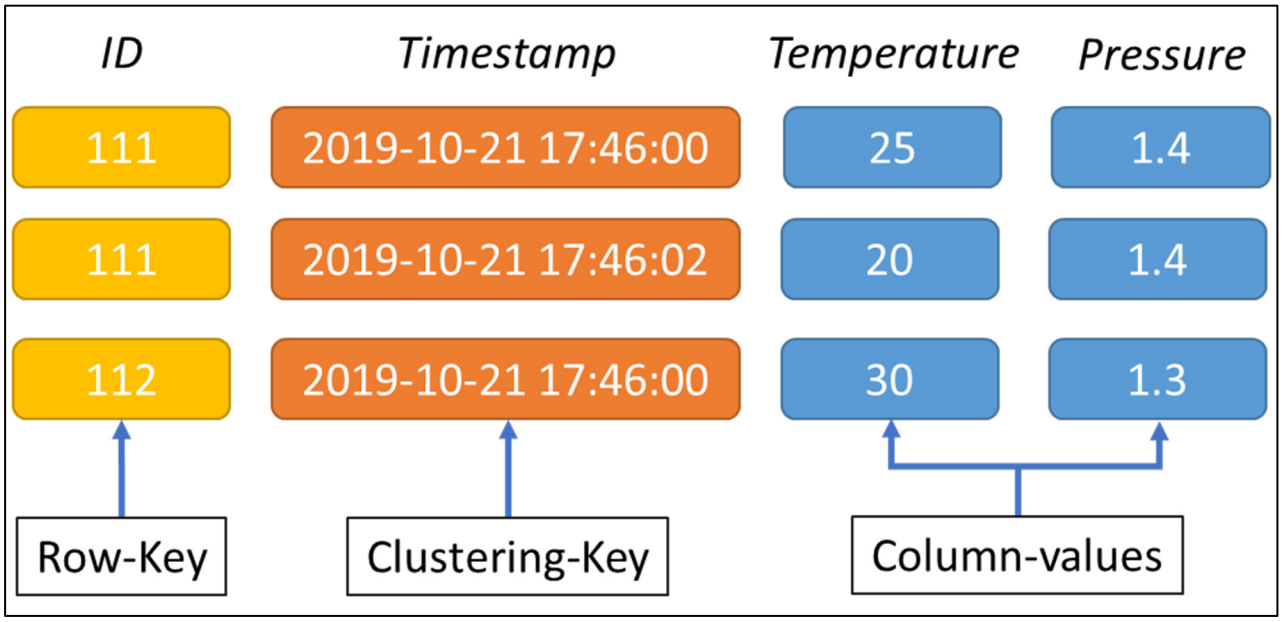

Figure 17 A Cassandra Table Sample

In [Figure 17], an example of a Cassandra table is depicted: Temperature and Pressure (Column-Values) are browsable by using the Row-Key and Clustering-Key.

Used by many companies, like Facebook and Spotify, this database type is recognized as one of the most powerful databases for operations/second performance [73].

Thus, Cassandra has been chosen because of its throughput and scalability properties, needed in a complex environment where data must be collected very quickly and must be always available. Moreover, it can be installed on Windows and Unix OS', letting it usable on many different projects.

4.4.2 MySQL

MySQL is an open-source RDBMS that uses the SQL language [77]: it is provided by Oracle and widely used worldwide [78] [79] [Figure 18].

The RDBMSs are database management systems that operate in compliance with the relational theory, formulated by the British computer scientist Edgar F. Codd: a system must operate on the data through relationships between the different tables in which these are divided and sorted.
In the relational model, data within a database is organised in different tables which are related each to other. All the data that an RDBMS processes is saved in tables that can be related through keys.
In an RDBMS, all the tables are composed of columns and rows.
Each column of a table represents a specific attribute and are organised in terms of records.
A record is usually uniquely identified, or numbered, using a primary key which allows a unique assignment.

An RDBMS has been introduced in the machine monitoring software solution because of the powerfulness of the SQL language and because of the requirement of Elettric80 to analyse and manage data on its own, by using its already known technologies: it allows to do join and better search data without using other frameworks. For this purpose, alarms data and aggregated data are being stored on it.

MySQL has been chosen as RDBMS as it is the most used open-source RDBMS [78] [79] [Figure 18] and for its performances [80].

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.

trend chart

355 systems in ranking, October 2019

Rank Oct 2019	Rank Sep 2019	Rank Oct 2018	DBMS	Database Model	Score Oct 2019	Score Sep 2019	Score Oct 2018
1.	1.	1.	Oracle	Relational, Multi-model	1355.88	+9.22	+36.61
2.	2.	2.	MySQL	Relational, Multi-model	1283.06	+3.99	+104.94
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1094.72	+9.66	+36.39
4.	4.	4.	PostgreSQL	Relational, Multi-model	483.91	+1.66	+64.52
5.	5.	5.	MongoDB	Document, Multi-model	412.09	+2.03	+48.90
6.	6.	6.	IBM Db2	Relational, Multi-model	170.77	-0.79	-8.91
7.	7.	8.	Elasticsearch	Search engine, Multi-model	150.17	+0.90	+7.85
8.	8.	7.	Redis	Key-value, Multi-model	142.91	+1.01	-2.38
9.	9.	9.	Microsoft Access	Relational	131.18	-1.53	-5.62
10.	10.	10.	Cassandra	Wide column	123.22	-0.18	-0.17

Figure 18 Database Ranking (snippet taken by Claudio Santo Longo from DB-Engines, 2019 solid IT gmbh, db-engines.com/en/ranking)

Moreover, it provides the end-user with a GUI which allows it to fast design and deploy a database schema [Figure 19].

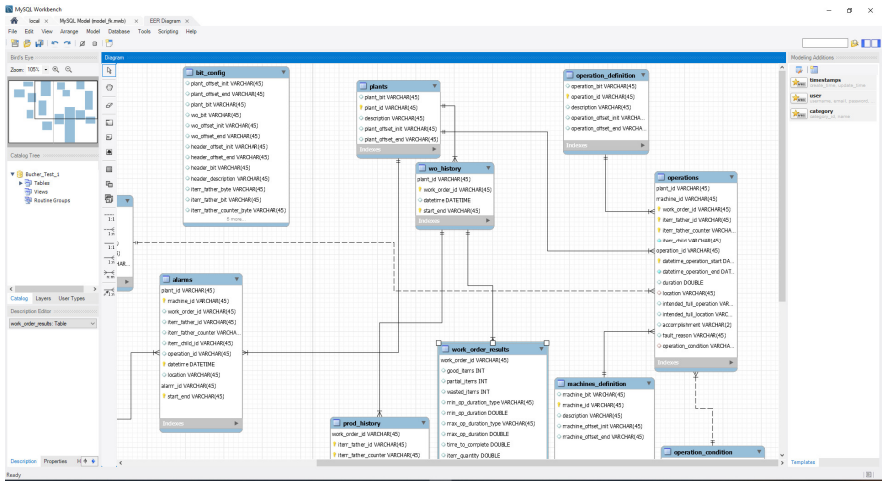

Figure 19 A MySQL Workbench schema sample

It can be installed on Windows and Unix OS' and it is well supported by Oracle and by the community.

4.5 Chosen streaming solution

Broker message and stream analytic frameworks have been considered for developing the final software solution.

For the presented use-case, Apache Kafka [81] [68] has been chosen because of its capabilities and usability: with its publishing-subscribing system, this framework provides with high throughput capabilities, handling large volume of data coming from many sources and providing results to many other devices.

It is becoming very popular and supports SCALA, JAVA and Python.

Kafka has three main capabilities, which are:

- Establishment of a Publishing-Subscribing system connection;
- Storage of stream of records for data retention purposes;
- Real-Time data processing, by means of Kafka Streams.

Kafka allows the IT-Architects to develop real-time streaming pipelines, connecting several devices in a reliable and secure way [82].
Its architecture is shown in a High-Level way in Figure 20.

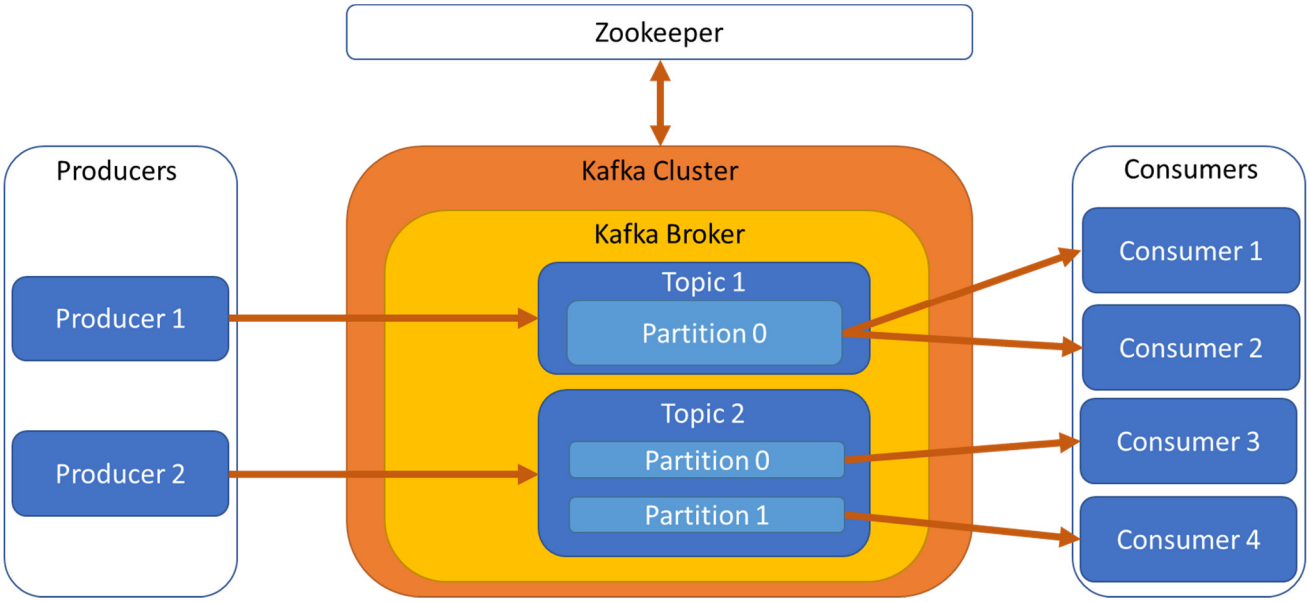

Figure 20 High-Level Apache Kafka Architecture

Producers send data to a defined topic and partition: data is being retained for a predefined time or volume and can be used by other consumers. Multiple consumers can have access to the same data retained in the same topic at the same time, for different purposes:

- On-line Analysis;
- Connection with stream analytic frameworks;
- Connection with databases;
- Connection with other software solutions.

Each message retained in a topic consists of a key, value and timestamp. As said, a topic is composed of one or more partitions to support:

- Parallel Processing;
- Scalability.

The order with which messages are stored in the topic is guaranteed inside partitions. Immutability property is provided: once the data has been stored, it cannot be modified anymore.
Topics are retained in one or more brokers: this technique allows the data replication and so, the data loss prevention.

A perfect starting point is having 3 brokers: one of them is the leader and the others are its replica. In fact, in case the leader stops working, another leader is being elected. With a N replication factor, producers and consumers can tolerate up to N-1 brokers being down.
Thus, a replication factor of 3 is a recommended configuration:

- Allows one broker to be taken down for maintenance;
- Allows another broker to be taken down unexpectedly, permitting the system to continuously work.

Finally, a collection of Brokers is a Cluster. Kafka can run by means of Zookeeper: a centralized service for distributed systems, which is used for providing a distributed configuration service, synchronization service, and naming registry. It is the service responsible for managing the brokers, electing a new leader in case the previous dies.

5 The Developed Architecture

The following chapter describes the implemented architecture [1] [Figure 21] [Figure 23]: starting from the original concept shown in Chapter 4.2 and basing on the most prominent technologies explained in Chapters 4.3, 4.4, 4.5, the final architecture has been designed and developed accordingly.

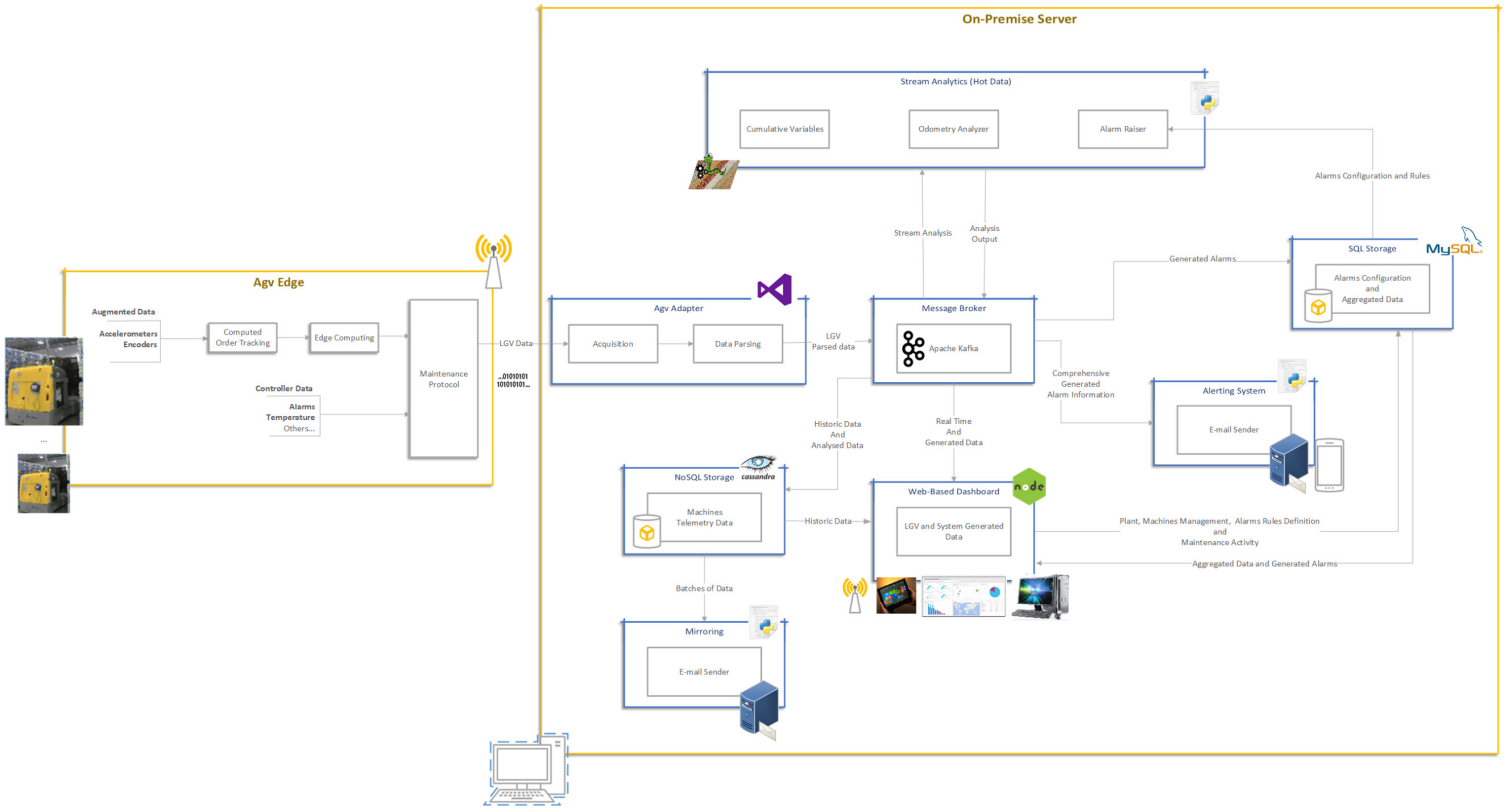

Figure 21 The developed architecture

The architecture [Figure 21] is explained as follows:

- Working Environment: environment where the software is installed and running. In Figure 21, two environments are depicted: the LGV machine (AGV Edge) and the server on which the system monitoring solution has been installed (On-Premise Server);

- Macro components: refer to equipment or set of functionalities that the software solution accomplishes (E.g. Stream Analytics algorithms);

- Micro components: regard specific functions that the software solution accomplishes.

Every component copes with the project requirements and the architecture has been designed in order to be scalable, independently from the complex environment in which its consequent software solution must be installed: as said in Chapters 4.1, 4.2, Elettric80 has many different customers which might have a little (e.g. 7 machines) or a large number (e.g. 100 or even more) of machines which are working in their warehouses and this software solution must be capable of acquiring data from all the machine.

According to the Hardware In the Loop technique [Chapter 3.4], the workbench provided by Elettric80 has been extensively used [Figure 22]: this instrumentation, with the needed software already installed and an encoder, allowed to simulate the machines sensor data.
This allowed a faster development of the software solution, without the need of going to the company headquarter for any kind of experiment.

Moreover, while developing the architecture, other possible extensions have been taken under consideration: Elettric80 expressed its interest to continue working with the research group in order to add more functionalities to the machine monitoring solution in possible future projects.

Finally, measures which ensure the right functioning of the system have been taken under consideration as well (described in Chapter 5.3), providing with a software solution that is always up and running.

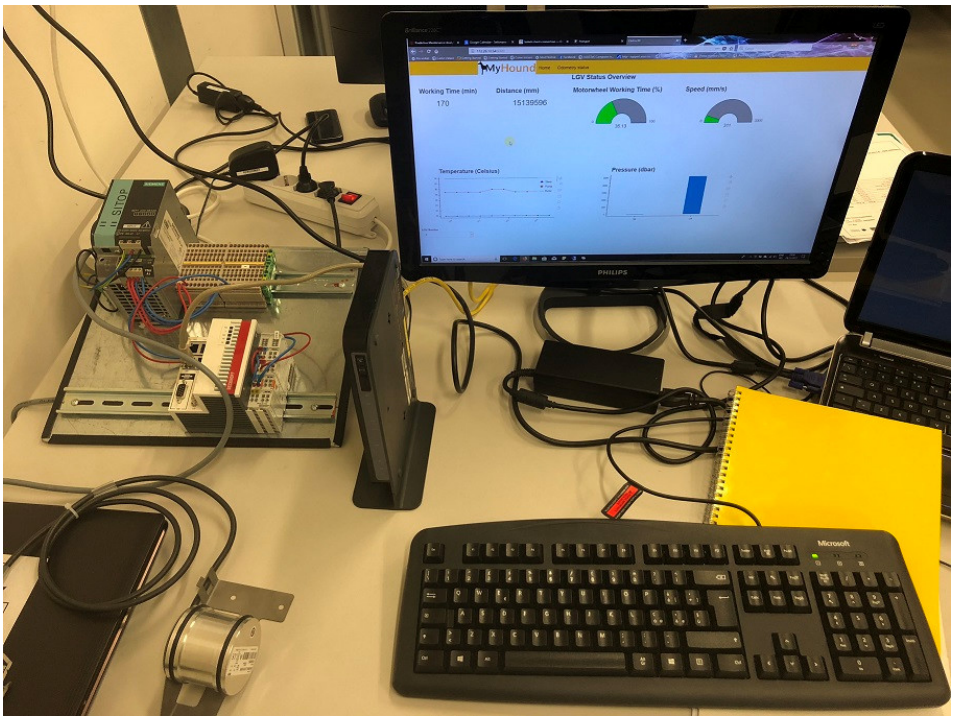

Figure 22 The workbench and a pc used for developing the first release of the software solution

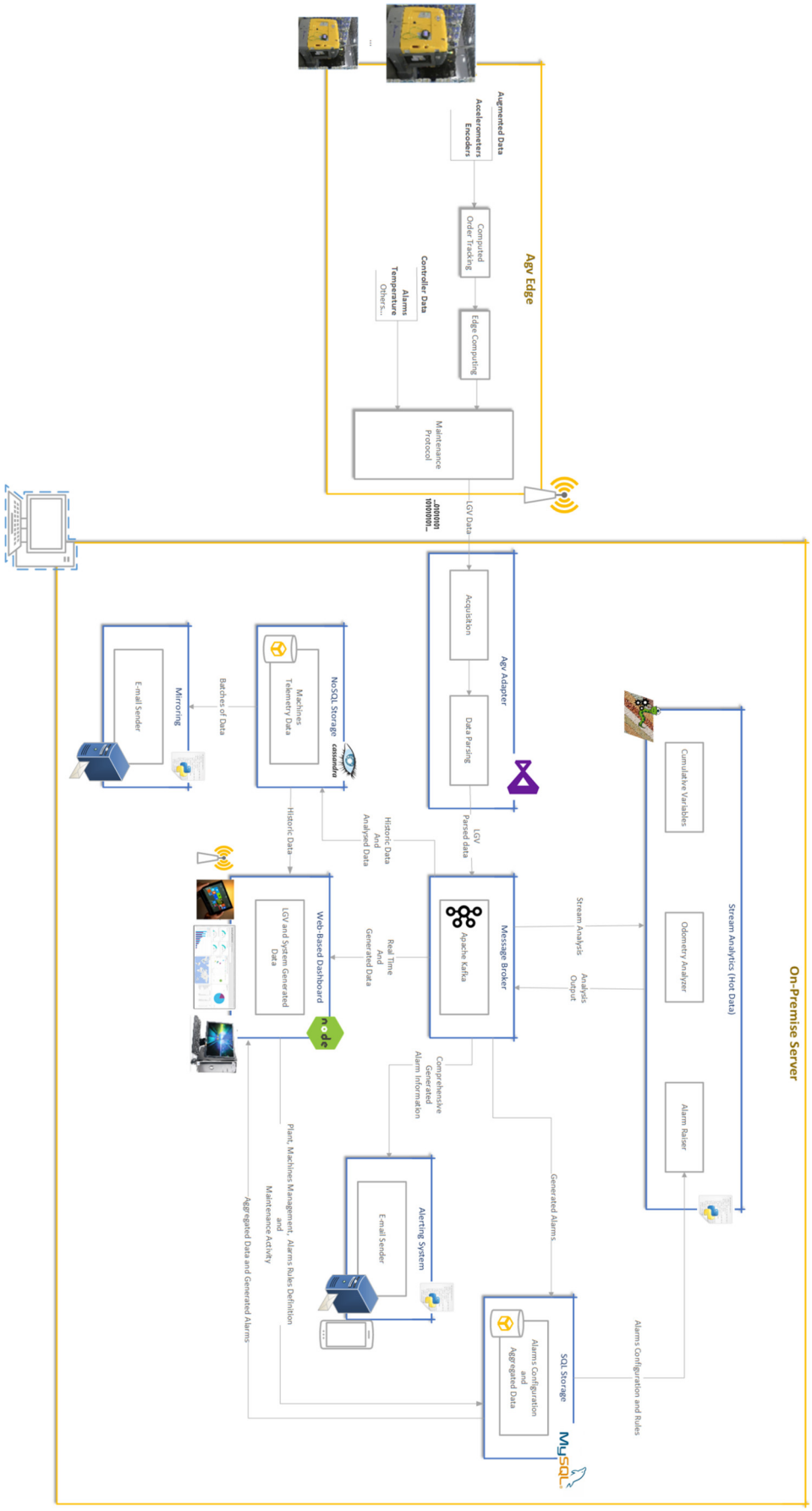

Figure 23 The developed architecture (Enlarged)

5.1 AGV Edge

The purpose of this component [Figure 24] is to provide an edge computing solution for acquiring data coming from the machine sensors, packaging and sending it to the machine monitoring solution.

Data of interest is produced by the LGV machine, used by the software solution for delivering predictive maintenance functionalities, can be grouped into the two following categories:

- Controller data: comes from sensors which were already installed for controlling the LGV machines and is sent at a low frequency (e.g laser scanner data, temperature etc.) every 2 seconds;

- Augmented data: comes from sensors which have been installed during the project phases (accelerometers, encoders).

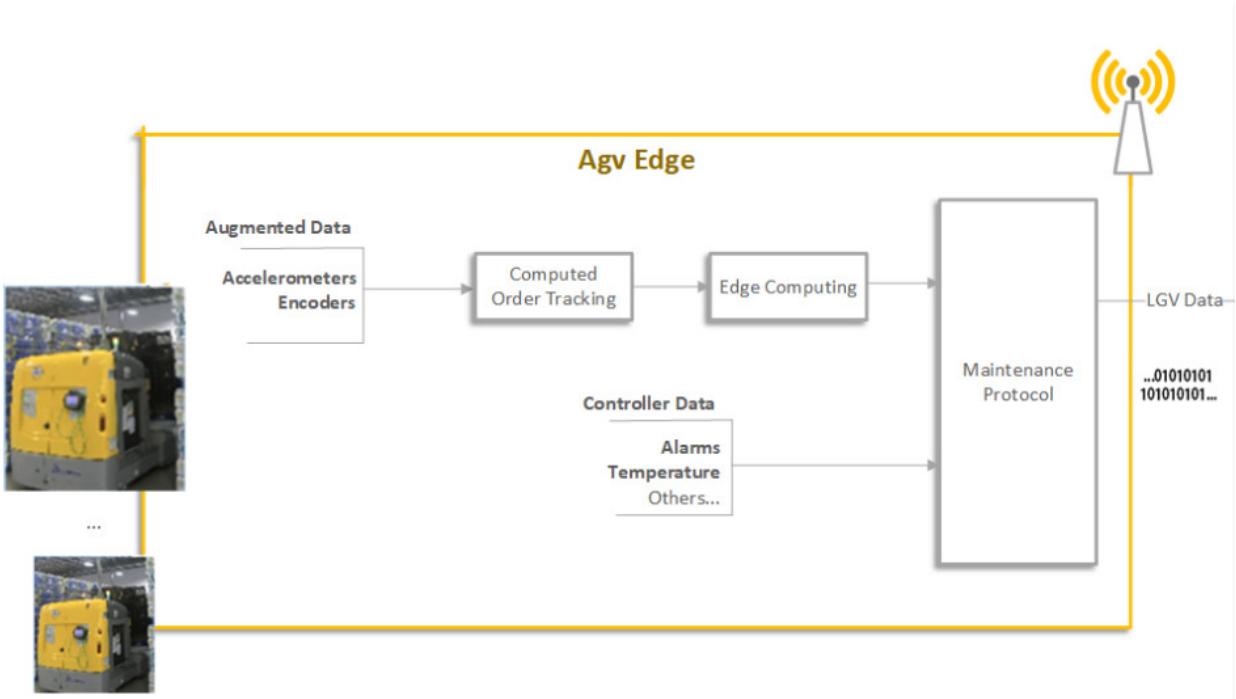

Figure 24 The AGV Edge architecture

Meanwhile, the acquisition of controller data is done by the already installed PLC, the acquisition of augmented data is done by a dedicated additional PLC: this allowed the company to not change the original code of the already installed PLC. Once the augmented data is acquired by the additional PLC, it is transmitted to the main PLC (by means of Beckhoff ADS protocol) of the LGV machine.

Controller data and augmented data are exploited to evaluate low frequency "synthetic parameters", summarising the behaviour of the system.
Finally, the main Programmable Logic Controller is responsible for packaging both types of data and of sending it to the "AGV Adapter" for further pre-data preparation and then sent to the Message Broker.
In the following chapter, the "AGV Edge" structure is explained.

5.1.1 Computed Order Tracking

Computed order tracking is a technique by which data, obtained from different sensors (each of them with its own sample time), is resampled and a unique time scale obtained. In this case, the following instrumentation has been used:

- Sensors: accelerometers, used to obtain high-frequency vibration signals;
- Encoders: used to obtain the angular positions at a lower frequency and detect the rotation cycles.

Thus, by means of the encoders data, shaft revolutions are identified: in presence of a constant velocity, the corresponding signal is a sawtooth signal with the same frequency as the rotating shaft. The encoder signal is resampled by means of a linear interpolation implemented, as a task running in the PLC: by doing this, the total number of sampling points in the accelerometric signal is made equal to the number of the sampling points of the angular position. Thus, it is possible to directly relate the accelerometric signal with the angular position, so detecting the exact beginning and the end of a rotation cycle. Each cycle is considered as a single time window in which the values of the synthetic parameters are evaluated.

5.1.2 Edge Computing

Edge computing is made possible by exploiting both high frequency and low-frequency data together with iterative algorithms: this operation does not affect the real-time behaviour.

The following synthetic parameters are calculated:

- RMS;
- Kurtosis;
- Peak Value of the accelerometers signal;

Augmented data, synthetic parameters and controller data related to a 10-cycle time window are collected task by task in a data structure and then stored in the storage memory of the PLC as a binary log file. The interval between different acquisitions should be compliant with the time required for emptying the buffer and the available storage memory provided by the PLC: the time required for emptying the buffer depends on the dimension of the data structure, while the available storage memory depends on the specific used PLC.

Calculated data is then packaged and integrated into the "Maintenance Protocol" component: this data is used for generating alarms regarding the quality of the surface on which the LGV machines is driving.

5.1.3 Maintenance Protocol

This component, based on a library created by Elettric80, is responsible for collecting both sensor data and the output of the "Edge Computing" component. Data is packaged in a binary format and sent to the warehouse access point by means of a UDP protocol (already established by Elettric80 itself and depicted in Figure 25).

4.2 "Data"(d) message

The counters stars when the LGV is switched on.
For the Indicators, refer to the chapter

Byte	Byte Length	Description	Unit of Measure	Var Type	Sampling	Ref Chapter	Min Value	Max Value
[redacted]	[redacted]	[redacted]					0	65535
[redacted]	[redacted]	Timestamp	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	LGV working time	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	Motorwheel working time	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	Steer working time	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	Pump working time	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	Conveyors working time	s	Counter	1/message		0	4294967295
[redacted]	[redacted]	Telescopic pole working time	s	Counter	1/message		0	4294967295

Figure 25 Sample of the documentation provided by Elettric80 Maintenance Protocol structure

Depending on the LGV machine type and on the warehouse in which it is supposed to work, the Order Tracking and Edge Computing features are enabled or disabled accordingly: when enabled, the sent synthetic parameters are:

- RMS for each wheel;
- Kurtosis for each wheel;
- Peak Values for each wheel;
- A quality parameter based on the mean of the above ones.

In order to let the data transfer work, the LGV machine makes a request for sending data to the "AGV Adapter" component and when the communication has been established, by means of an Acknowledge message exchange, the data transfer is started.

5.2 On-Premise Server

The system monitoring solution has been installed on an On-Premise Server, provided by Elettric80, that has then been installed in a server rack of one of its customers.

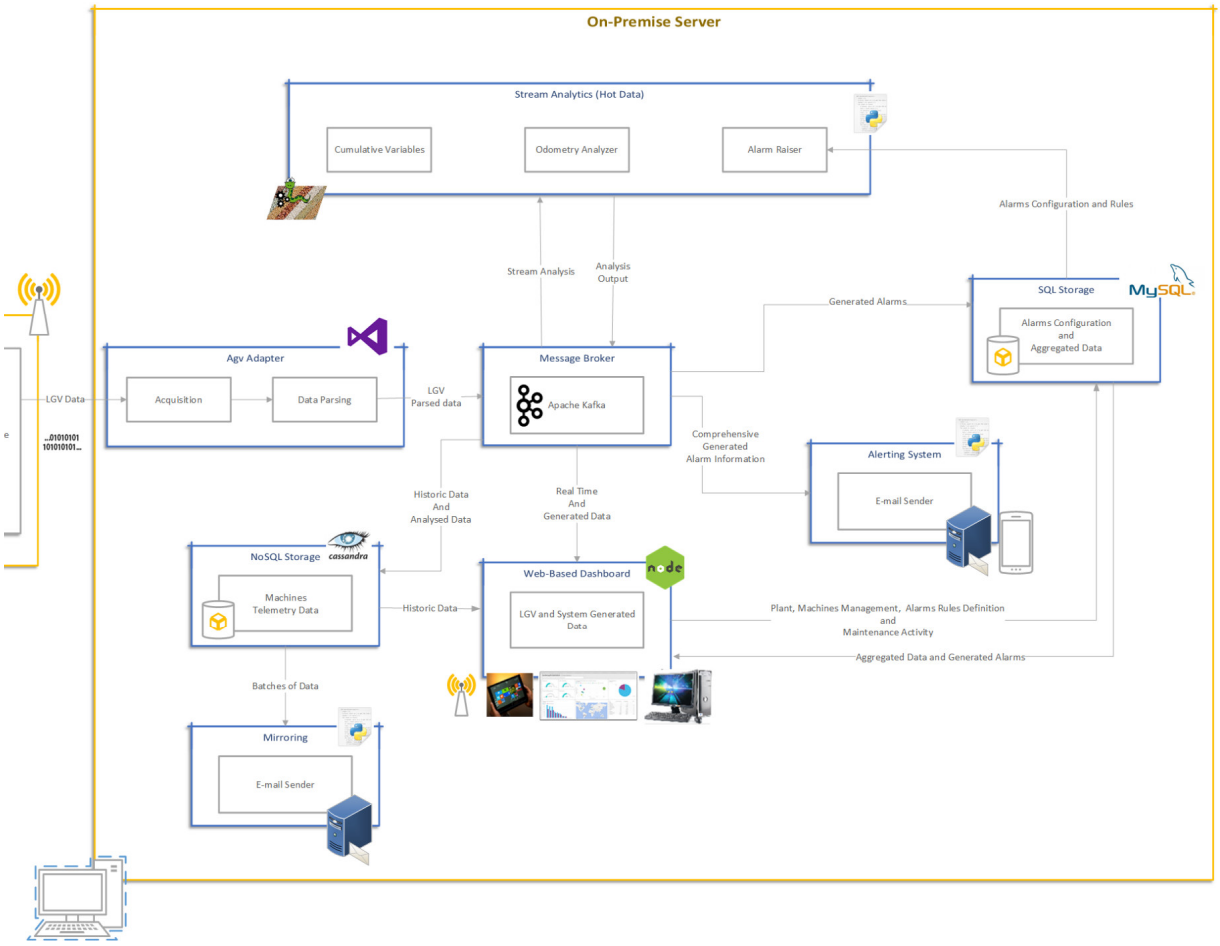

Figure 26 The On-Premise Server architecture

The provided server was carrying the following characteristics:

- CPU: Intel Xeon E5-2620;
- Memory: 16 GB;
- Hard-Drive: 4 HDD;
- Network Adapter: 1000 Mbps.

In this chapter, all the macro and micro components are explained, highlighting the challenges and showing the found solutions.

5.2.1 AGV Adapter

This component [Figure 27] is a .Net application responsible for receiving, processing and sending the machine data to the "Message Broker" component.
Once it has been started, this software is set on listening mode on a pre-defined port, ready to acquire and handle the incoming data.
The used connection protocol is UDP.

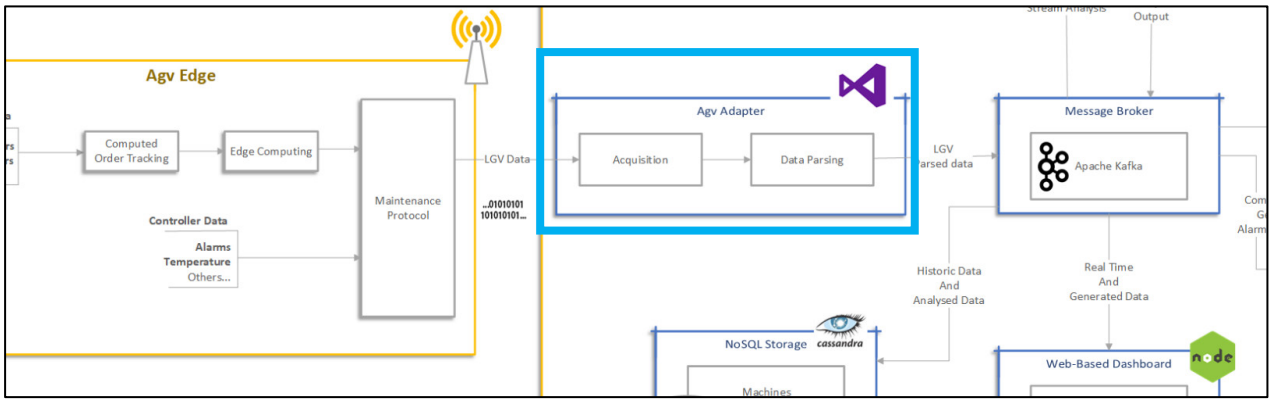

Figure 27 The AGV Adapter macro Component

This is a custom component created for the Elettric80 use-case, which must use proprietary libraries made by the company, for establishing the communication with the LGV machines.

All its micro-components are explained as follows.

5.2.1.1 Acquisition

This component is responsible for acquiring the messages from the LGV machines [Figure 28].
The communication channel and the data acquisition steps are described as follow:

1. Each LGV machine continuously sends a synchronization request;

2. When the software is started, it is set on listening mode on a predefined port;

3. The software answers to the LGV machine synchronization request, enabling a communication channel with it;

4. The LGV machine starts sending messages:
    a. whether the “Acquisition” component receives the message, it checks it as follows:
        i. interprets the message;
        ii. checks its length (the 15th message byte contains this information);
        iii. calculates its CRC (it calculates the XOR for each bit) and compares it with the CRC contained in the last byte of the message (they must be equal);
    b. if the LGV does not receive any ACK message, it retries sending the message: if it doesn’t receive any ACK anymore, it restarts asking for the synchronization;

5. if the LGV does not receive any ACK message, it retries sending the message: if it doesn’t receive any ACK anymore, it restarts asking for the synchronization;

6. The message is stored into a buffer and it is then processed by a set of threads;

7. Finally, the message is ready to be fully interpreted;

```
using Elettric80.Ems.Channel;
using Elettric80.Ems.Domain;
using Elettric80.Interprocess;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net;
using System.Threading;
using System.Threading.Tasks;
using WindowsFormsApp1;

namespace Elettric80.Ems.UdpManager.Domain
{
    public class ChannelEquipment //: EquipmentManager
    {
        Dictionary<ushort, ushort> ids = new Dictionary<ushort, ushort>();
        UDPChannel channel;
        public static List<LgvData> list = new List<LgvData>();
        public virtual Endianness Endianness { get; set; }
        private string _listeningIp;
        private int _listeningPort;
        private int _replyPort;
        public byte[] standingby;
        public int x = 0;
        private bool _saveMessageArray;
        public static int i = 0;
        public static bool th = false;
        public byte[] multiworker;
        Cluster cluster = Cluster.Builder().AddContactPoint("127.0.0.1").Build();
        Thread Worker = new Thread(bufferize);



        static ISession session; //I initialise the session as global (ndars)

        public ChannelEquipment()
        // : base(ems)
        {
            _listeningIp = System.Configuration.ConfigurationManager.AppSettings["ListeningAddress"].ToString();
            _listeningPort = Convert.ToInt32(System.Configuration.ConfigurationManager.AppSettings["ListeningPort"]);
            _replyPort = Convert.ToInt32(System.Configuration.ConfigurationManager.AppSettings["ReplyPort"]);
            _saveMessageArray = Convert.ToBoolean(System.Configuration.ConfigurationManager.AppSettings["SaveMessageArray"]);
            session = cluster.Connect("florim");  //I connect to the keyspace only once! (ndars)

```

Figure 28 A snippet of the code used for initialising the communication with the LGV machines

5.2.1.2 Data Parsing

Once the messages are stored in an array, they are ready to be interpreted by this architectural component, which works as a separate thread.
Each message is a string of bytes that must be divided and converted in the proper format.

The message processing considers:

- The observation name;
- The conversion type;
- Its relative offset.

Given this information, the message (in form of bytes) is split according to the observation offset. Then, each part is converted and assigned to a proper variable.
Finally, the message is ready to be sent to the Message Broker: a topic is created for each machine, for each data type and receives the data coming

from the AGV Adapter.

5.2.2 Message Broker

Kafka has been chosen as Message Broker [Figure 29] and it is the core part of this architecture: it allows the interconnection between all the shown components in Figure 21. Every software component acquires data from Kafka and uses it for different purposes:

- On-line analysis;
- Storage on the databases;
- Real-time view.

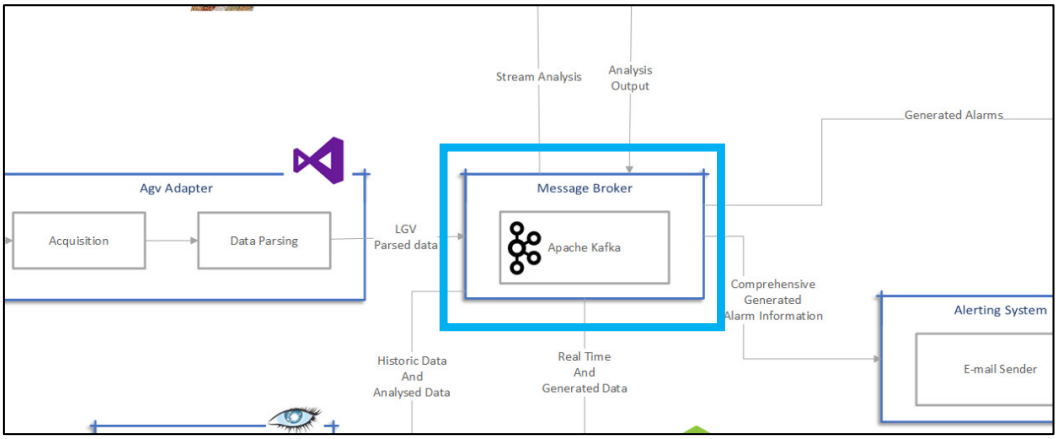

Figure 29 The Message Broker component

A naming convention for the Kafka topics has been based on the machine type, on its identification number and on the data type (e.g. telemetry data, analysed data, etc.): in case, for example, the LGV machine 1 wants to stream its telemetry data, it will send it to the "LGV_1_Telemetry" topic. This naming convention allows the developers and data analysts to clearly understand the data type, machine number and machine type in a clear way. Kafka has been configured for retaining the incoming data: this allows the software solution to correctly restart in case the server is being restarted, without losing data. Moreover, as said in Chapter 4.5, Kafka provides with very high scalability properties, allowing the software solution to cope even with hundreds of machines.

5.2.3 Stream Analytics

This component, depicted in Figure 30, consists of a set of Python scripts which are responsible for on-line analysing the streamed data: data, once is stored in Kafka, is being analysed and the results streamed back to the Message Broker.

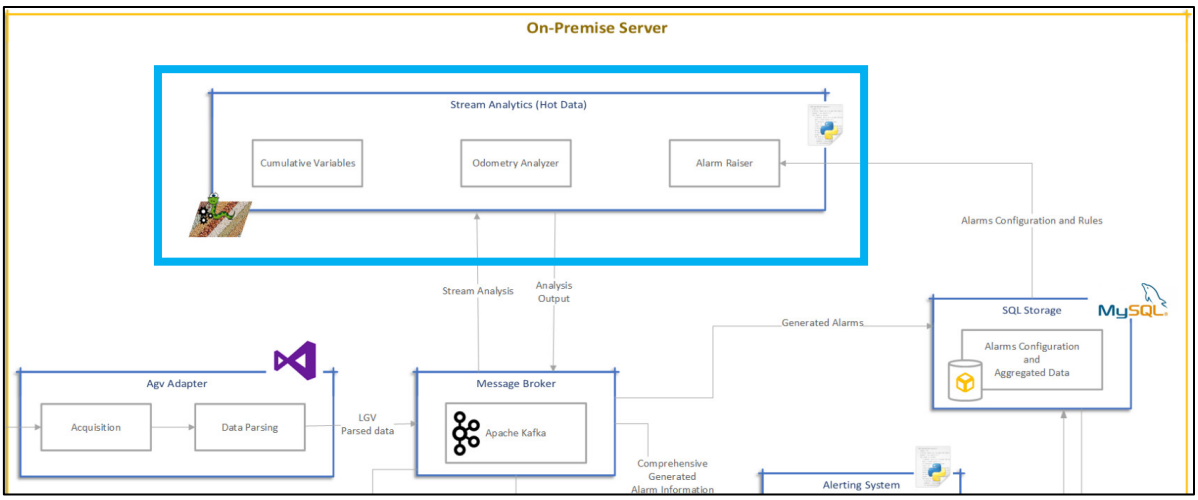

Figure 30 The Steam Analytics scripts

Three micro-components are part of this macro-component and are shown is the following chapter.

5.2.3.1 Odometry Analyser

This micro-component calculates the LGV machines speed and the distance they have travelled, using the encoder and laser-scanner data.

The two types of distance are compared every time the LGV machines have continuously travelled along a straight path for at least 10 meters.
The result is then streamed back to Kafka;
For proofing the reliability of the data coming from encoder and laser-scanner, the distance travelled by the LGV machine is calculated by using its position provided by its laser-scanner (in terms of coordinates) and compared with the encoder data (that is an incremental value). Both are incremented in-software over time, providing a reliable algorithm even if the LGV machine has been reset and consequentially its internal PLC variables values restart from "0": we call the incremental laser scanner position as

"`incremental_laser_scanner_distance`" and the incremental encoder distance as "`incremental_encoder_distance`".

The algorithm is explained as follows:

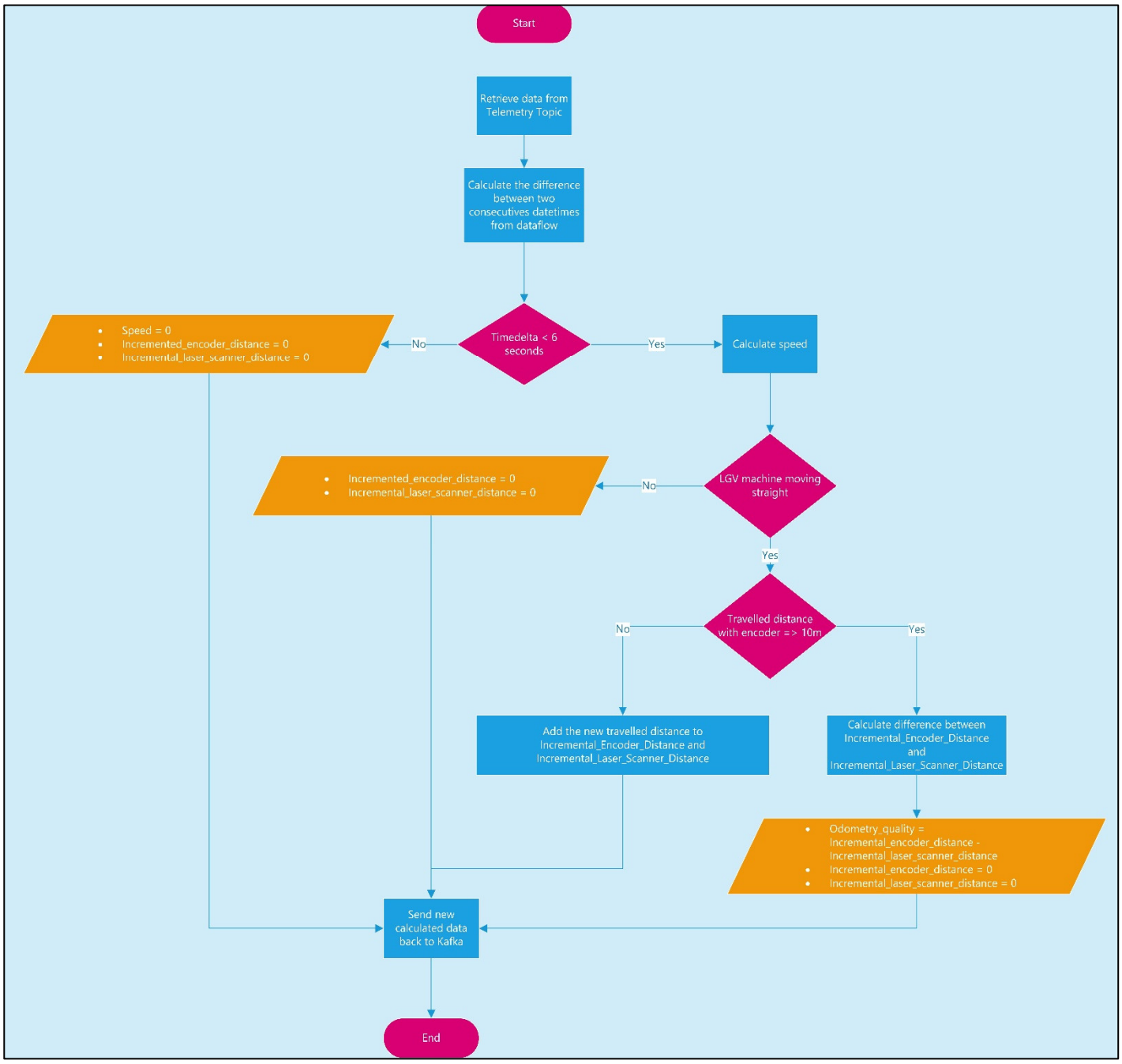

Figure 31 Flowchart explaining the algorithm logic

5.2.3.2 Cumulative Variables Calculator

PLC data is sent to the On-Premise Server in real-time. Among the streamed variables, some of them regard values which are cumulated over time (e.g. encoder data, machine usage time, machine activity time, etc.). Whether the LGV machine is being restarted, the values of these variables which are stored in the machine PLC, restart from the '0' value.
In order to cope with this problem, this micro-component is responsible for acquiring the variables which values are accumulated over time and accumulate their values basing on the past ones: this technique prevents to lose data in case the working LGV machines are being shut down for maintenance purposes and thus, always provide reliable data over time.

5.2.3.3 Alarm Raiser

This component is responsible for analysing the streamed data and raise alarms basing on already defined rules.
In case a threshold is overlapped, an alarm is raised: this consists in a message that is then sent back to Kafka and handled by the "Alerting System" component for notifying the technicians, responsible for maintaining the operating machines.

5.2.4 SQL Storage

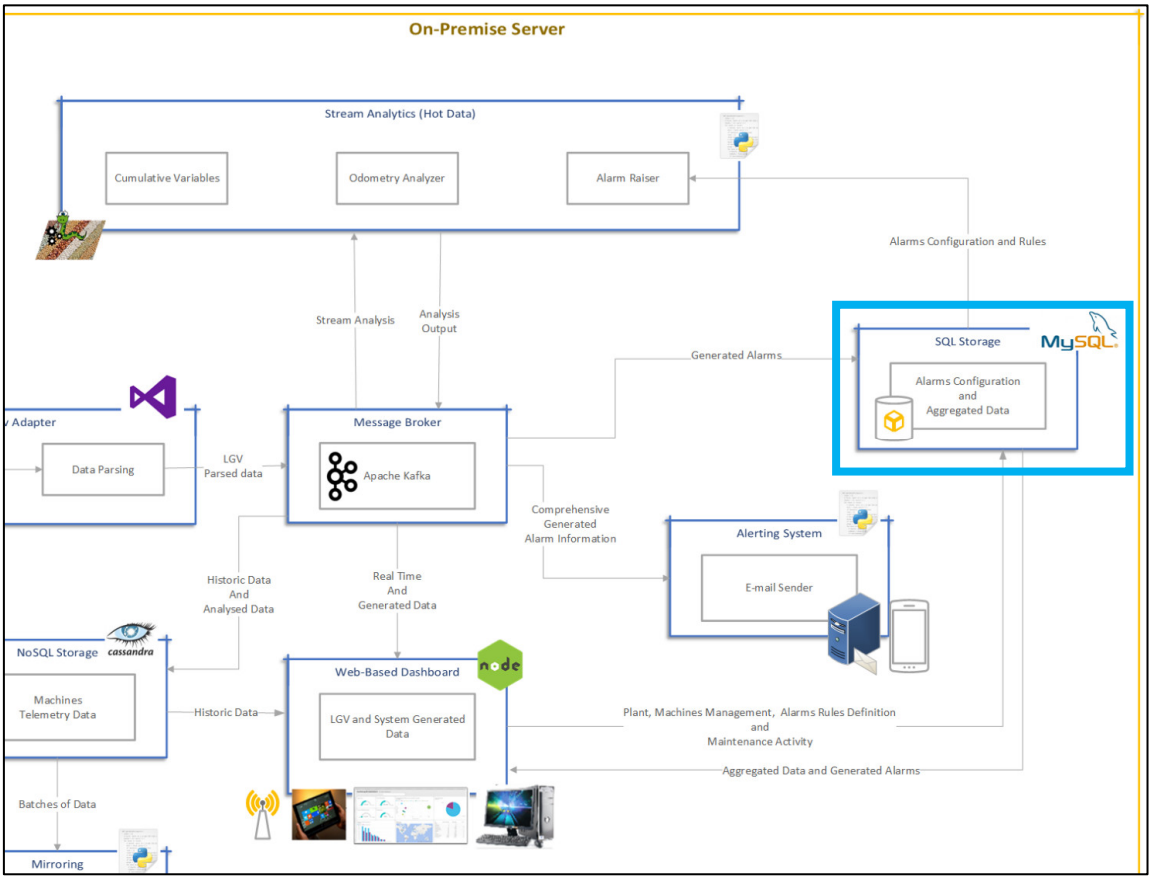

Figure 32 The SQL Storage component inside the architecture

This component consists of a MySQL database that is responsible for storing data regarding:

- Plant, Machines Management, Alarms Rules Definition and Maintenance Activity: starting from the customers' information, data regarding contacts, working technicians, machines and their installed components are taken under consideration.
  Every machine is composed of a list of components and every component type can trigger different alarms.
  Alarms are defined and assigned to the combination machine-component. The assigned alarm comprehends variables like the type of threshold (above or below a defined value), the threshold value and the personnel that is supposed to receive an alerting message. The triggered alarm creates a ticket which is stored in a predefined table, and allows the end-user to track its resolution status;

- Aggregated Data: LGV machines telemetry data is being aggregated per hour, days, weeks, months and years. This allows the end-user to

visualize this data on the dashboard by using a set of pre-defined queries. By means of the power of the SQL language, the end-user can watch trends of data on the web-dashboard by selecting the desired variables, time frame and thus, their trend over time. The calculation of these aggregated values is done by means of Python scripts that periodically query the Cassandra database and aggregate the acquired values.

As said in Chapter 4.1, an SQL database has been chosen for providing Elettric80 with a structured open-source solution which allows them to easily browse and manage the stored data, even after the project.

5.2.5 Alerting System

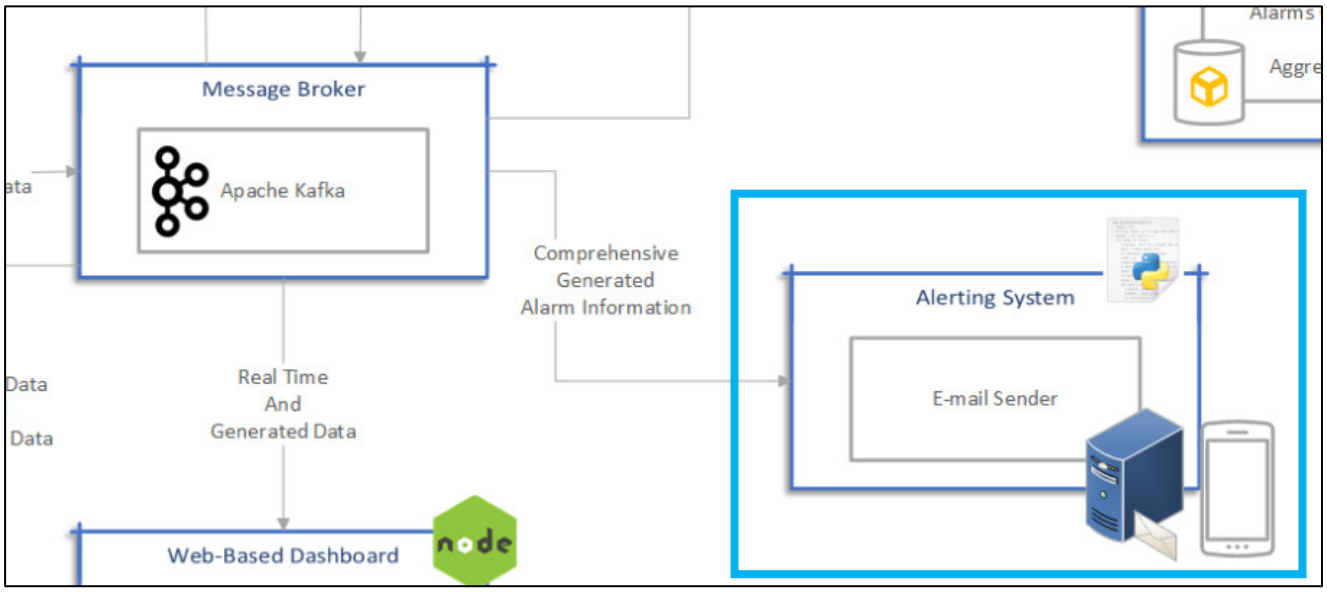

Figure 33 The Alerting System placed inside the architecture

This component [Figure 33] consists of Python scripts responsible for collecting, from Kafka, the generated alarms raised by the "Alarm Raiser" component and send their content to the specified users via e-mail.

5.2.6 NoSQL Storage

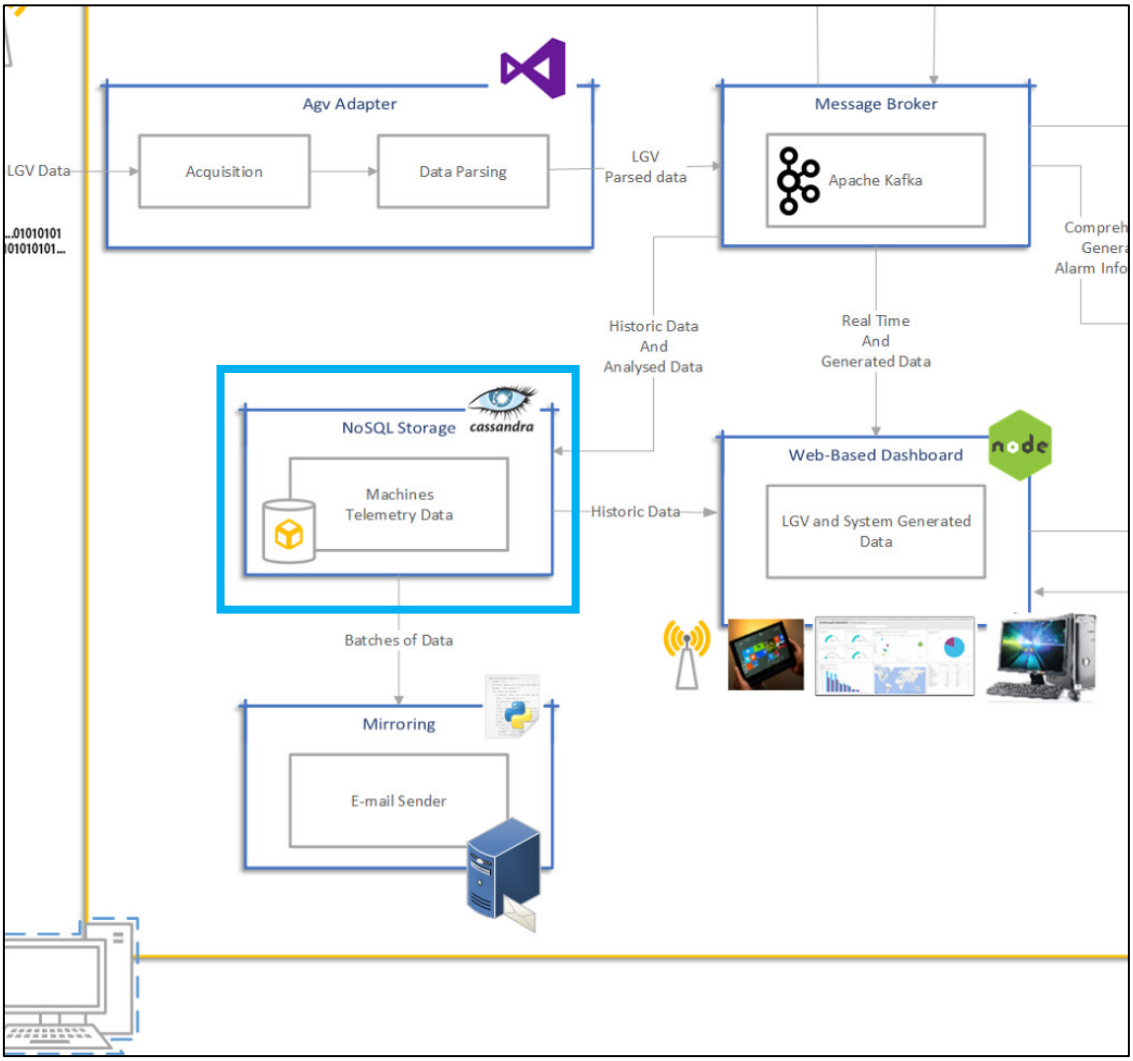

Figure 34 The NoSQL Database placed in the architecture

Unlike the RDBMS, the column-oriented database design starts from the application point of view: as it is not possible to join the single tables, the database should be designed in order to store and provide the desired data with a simple query.
Moreover, a single table can contain up to 2 billion of columns: in case new sensors have been added, their data can be anyway stored by adding its related column to the database table.
As said in Chapter 4.4.1, Apache Cassandra has been chosen for acquiring data that is generated very fast and in large volume over time. Basing on its characteristics and on the fact that the project is a research project (thus, destinated to evolve quickly over time), Cassandra resulted to fit better in the complexities the project brings: sensors and other data generable from the machine PLC can be easily collected by adding rows to already existing tables.

Eight tables have been identified:

- LGV List: register of the LGV machines which are working inside the customer warehouse, comprehending their identification number and type;

- Raw Messages: a table designed to store the LGV machine identification number, PLC DateTime, server DateTime and the received message in a hexadecimal format;

- Telemetry: composed of LGV number, PLC DateTime, server DateTime and all the variables the LGV machines can send. In this table, the high-frequency data is collected as an average evaluated in 2 seconds. An example is shown at the end of this chapter [Figure 35][Figure 36];

- High Frequency: a table designed to store the LGV machines sensor data collected and packaged by the machine every 200ms and then sent every 2 seconds. This data is stored in terms of lists, in order to easily read and understand how and which data has been collected over time;

- Odometer Data: this table collects the data calculated by the ”Odometry Analyzer” Python script;

- Cumulative Data: this table collects the output of the ”Cumulative Variables Calculator” component;

- Machine Status: this table is populated by a Python script that periodically checks whether machines are sending data. It is composed of LGV number, server DateTime and machine status (on-off).

Cassandra CQL Shell

```
CREATE TABLE [illegible].telemetry (
    lgv_number int,
    datetime timestamp,
    auxiliary_battery_working_time int,
    battery_conveyor_working_time int,
    battery_pusher_working_cycles int,
    battery_pusher_working_time int,
    conveyors_working_time int,
    energy_consumption int,
    general_pressure int,
    h_orientation int,
    insert_time timestamp,
    lgv_working_time int,
    lift_pressure int,
    lift_quote_recovery_number int,
    lms_motor_working_time int,
    mast_pressure int,
    motorwheel_covered_distance int,
    motorwheel_current int,
    motorwheel_motor_temperature int,
    motorwheel_working_time int,
    operation_phase int,
    plc_status int,
    plc_status_2 int,
    pls_alarms_number int,
    point_number int,
    presstop_working_time int,
    pump_current int,
    pump_motor_temperature int,
    pump_working_time int,
    quality_level int,
    "screwLift_(CB8)_covered_distance" int,
    segment_number int,
    steer_angle int,
    steer_current int,
    steer_motor_temperature int,
    steer_working_time int,
    straight_steering int,
    system_status int,
    system_status_2 int,
    telescopic_pole_working_cycles int,
    telescopic_pole_working_time int,
    tilt_pressure int,
    vertical_pls_motor_working_cycles int,
    vertical_pls_motor_working_time int,
    "wagon_(reach)_working_cycles" int,
    x_coordinate int,
    y_coordinate int,
    PRIMARY KEY (lgv_number, datetime)
) WITH CLUSTERING ORDER BY (datetime DESC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
```

Figure 35 Description of the telemetry table

lgv_number	datetime	quality_level	x_coordinate	y_coordina
1	2017-05-23 10:57:01+0000	100	38267	157
1	2017-05-23 10:57:03+0000	100	38267	157
1	2017-05-23 10:57:05+0000	100	38267	157
1	2017-05-23 10:57:07+0000	100	38266	157
2	2017-05-23 10:57:01+0000	95	31939	121
2	2017-05-23 10:57:03+0000	95	31939	121
2	2017-05-23 10:57:05+0000	95	31938	121
2	2017-05-23 10:57:07+0000	95	31937	121

Figure 36 Telemetry sample table containing acquired data

5.2.7 Mirroring

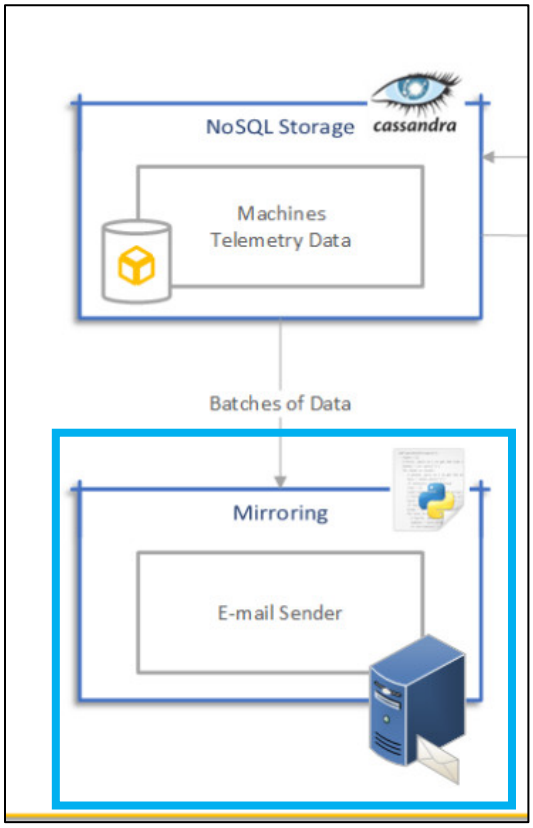

Figure 37 The Mirroring system placed inside the architecture

This component is responsible for periodically collecting data from the Cassandra database and sending it to other servers via e-mail in the form of .csv files. The receiver server collects then the .csv files from the mailbox and stores their content in proper tables.
This technique is useful when the system is working under strict security measures and the SMTP port is opened.

5.2.8 Web-Based Dashboard

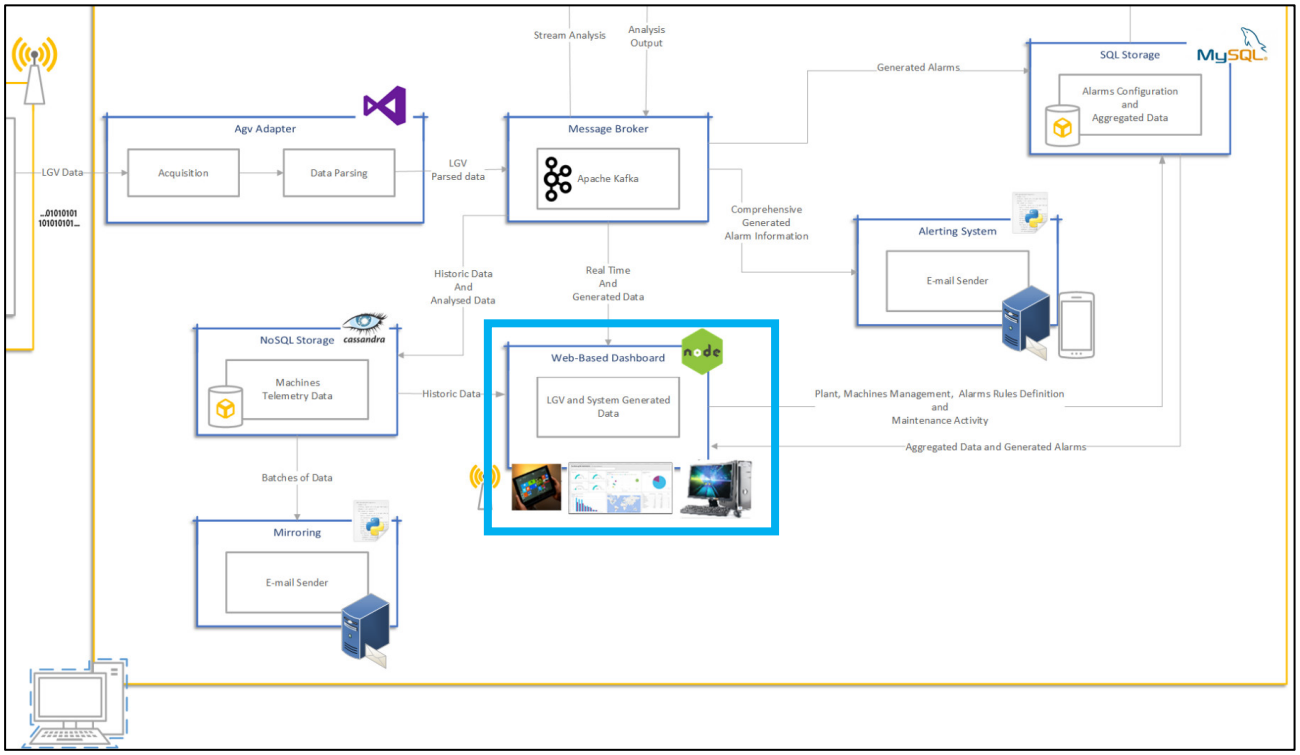

Figure 38 The Web-Based Dashboard placed inside the architecture

The development of the web-based dashboard has been preceded by a series of meeting with Elettric80, during which the requirements of the company have been acquired.

This interface consists of a real-time dashboard, developed by using Node.js, a JavaScript runtime, to locally host the application [83] [84].
The dashboard comprehends a set of different fully customizable charts and objects (like tables, buttons and drop-down menus), created by using the D3.js library [85].

The dashboard is composed of six pages on which LGV data filtering is made possible by using a drop-down menu.

The pages are described as follows:

- LGV Maintenance: this page allows the end-user to create and view the maintenance activities to be done or already done [Figure 39];

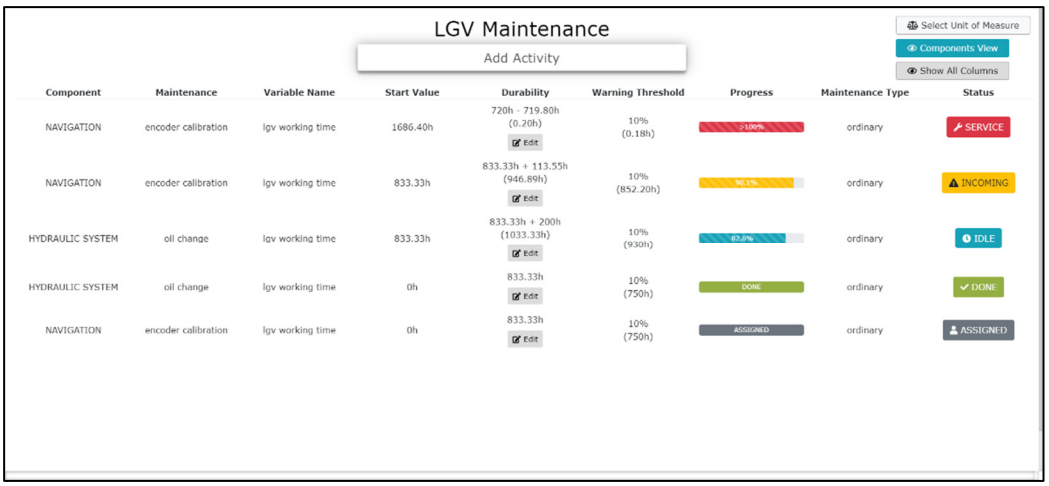

Figure 39 LGV Maintenance page

- Remaining Use Before Maintenance: this page displays several gauges that show the remaining useful life of several LGV machines component [Figure 40];

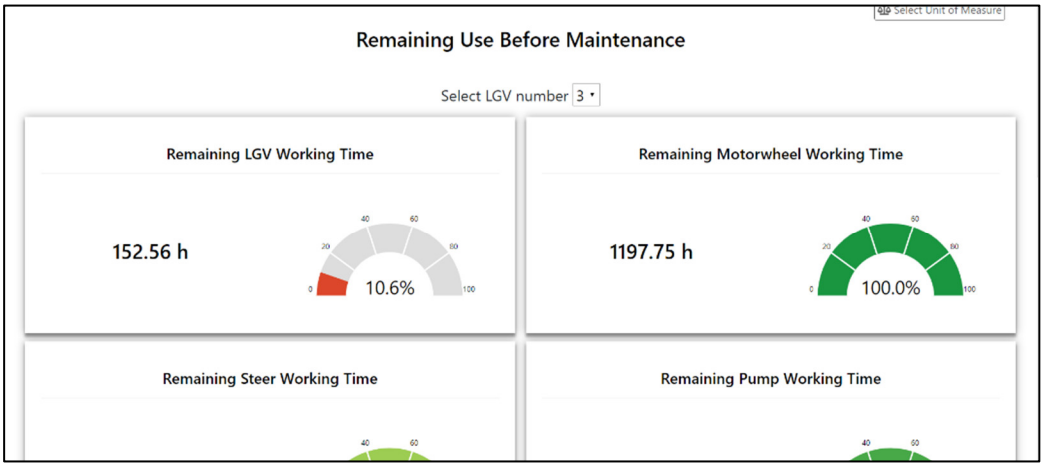

Figure 40 Remaining Use Before Maintenance page

- Condition Based Monitoring: in this page, the user can monitor two parameters [Figure 41]:
    - Odometry quality: this part shows the calculated difference between the distance covered by the LGV machines, according to the encoder data and the laser scanner data;

    - Quality Factor Wheels Surface: the wheel surface quality due to accelerometers, output of the maintenance library.

  The page is divided into two sections and the user can change the thresholds rule parameters which trigger an alarm when the quality falls below its defined value;

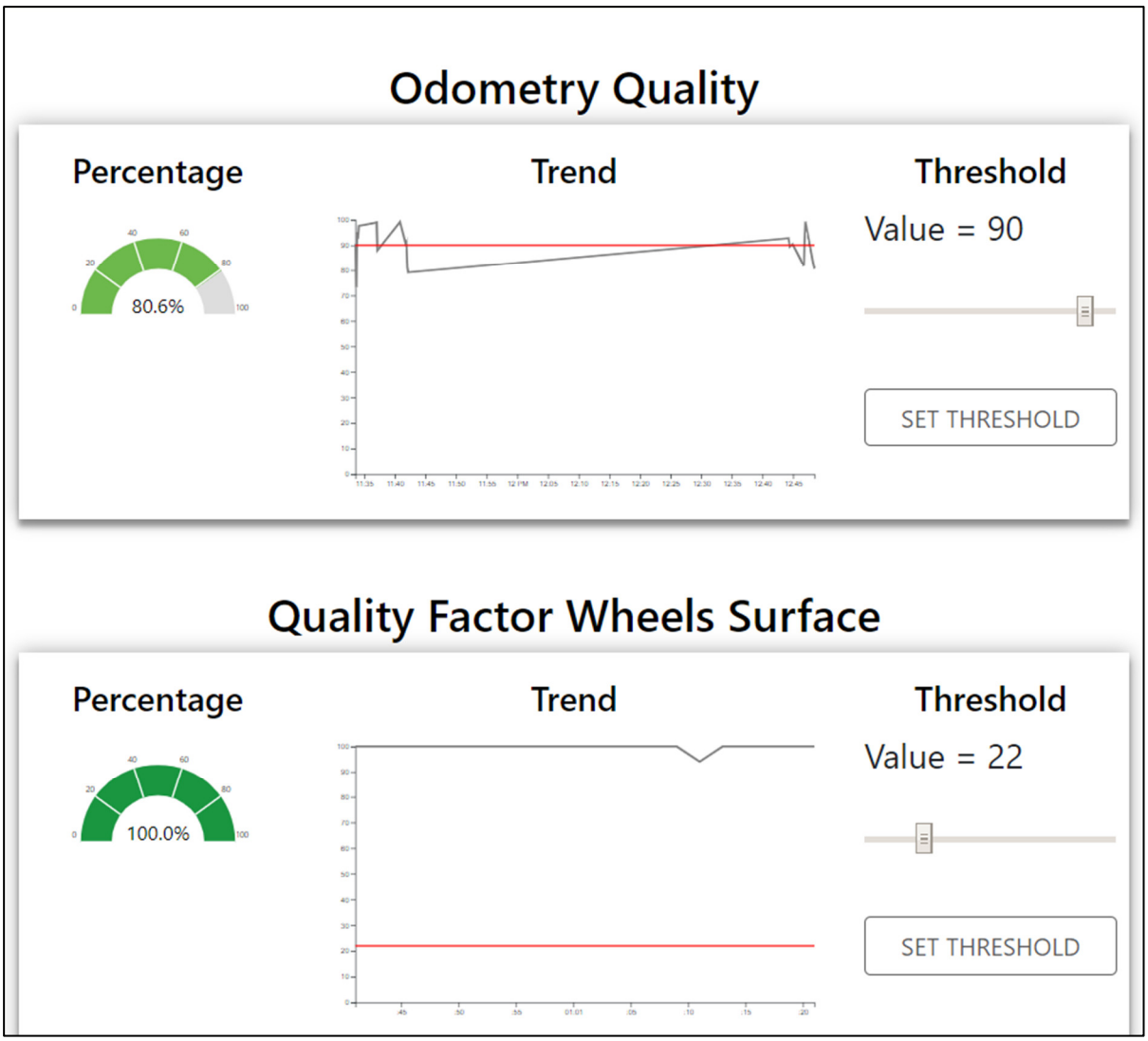

Figure 41 Condition Based Monitoring page

- LGV Monitor: on this page, the user can see the LGV machines working parametres about which Elettric80 was more interested [Figure 42];

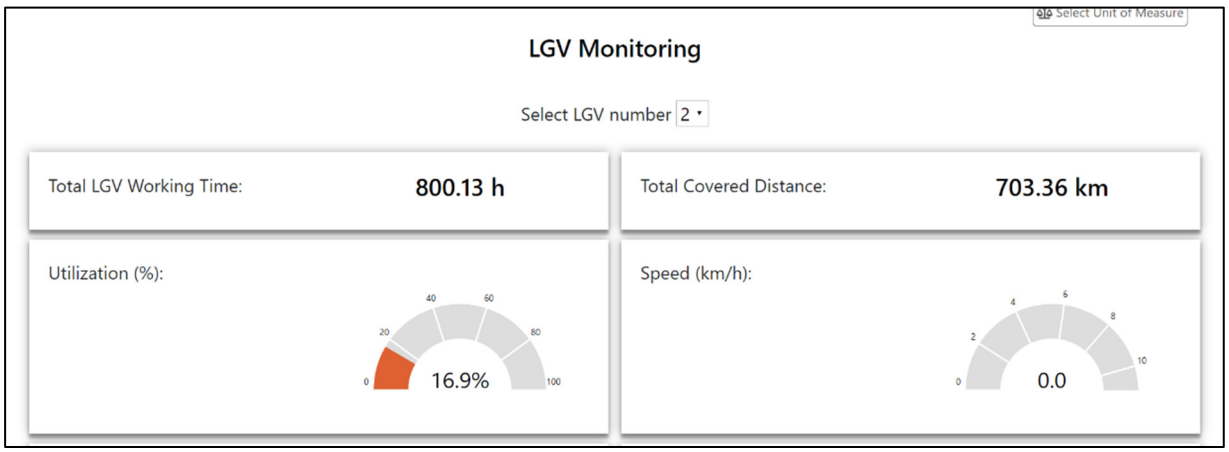

Figure 42 LGV Monitoring page

- Trends: in this page, the user can create, through a provided menu, custom charts which are then used for observing the main LGV machines parameters trends [Figure 43]. Depending on the needed resolution (in terms of seconds, hours, days, weeks, years), data is acquired from Cassandra or MySQL accordingly;

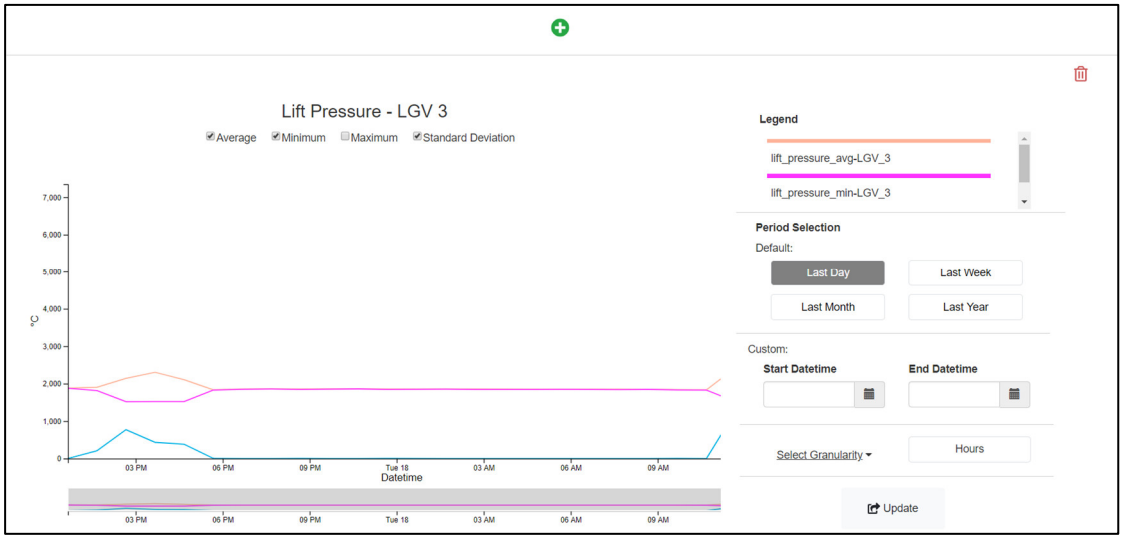

Figure 43 Interactive Graph page

- Alarms page: in this page, the user can check the alarms that the system has raised and stored into the database over time [Figure 44].

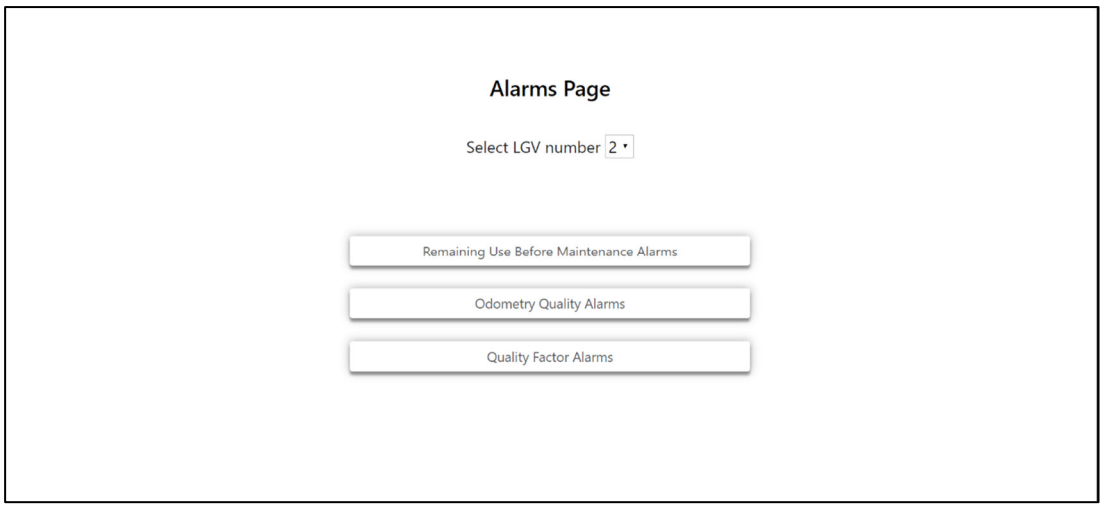

Figure 44 Alarms page

5.3 Architecture Stress Conditions and Recovery Operations

In this chapter, a list of stress conditions and a set of recovery operations are shown: the purpose of the document is to define all the fault events that may occur and generate a bad behaviour of the system.

The fault causes can be divided into two macro-areas:

- Environmental fault causes: these regard hardware, OS and network malfunctions;

- Software fault causes: these regard all the software solution malfunctions.

The developed architecture is composed of interconnected but independent macro component. This means that if one macro component stops working, the other ones are affected by this malfunction in terms of not receiving data anymore.

A set of automatic recovery operations have then been defined and applied to the system, guaranteeing its reliability: all the macro components are constantly monitored and recovered in case they are no longer working properly.

5.3.1 Software fault causes

The software solution is up and running 24/7 in the Customer Server Room. As we know, many factors might cause the occurrence of blocking faults that might stop the system from working properly.

All the components are constantly monitored and recovered in case a fault occurs.

In the following chapters, a list example of symptoms, their root causes and their solutions are shown.

We must note that, the name of the customer of Elettric80 has been hidden by its request.

5.3.1.1 The system is no longer storing data in the database

During the nominal working state, the system might stop storing the telemetry data.

In this case, Cassandra has been taken as an example: this process applies to the other databases included in the infrastructure.

This could be due to several causes:

- Cassandra does not work anymore;
- Kafka subscribers and consumers are no longer working;
- Kafka is no longer working;
- The data acquisition solution is no longer working;
- All the LGV machines are not responding.

Figure 45 shows a flowchart regarding the Cassandra troubleshooting, in case it is not storing data anymore.

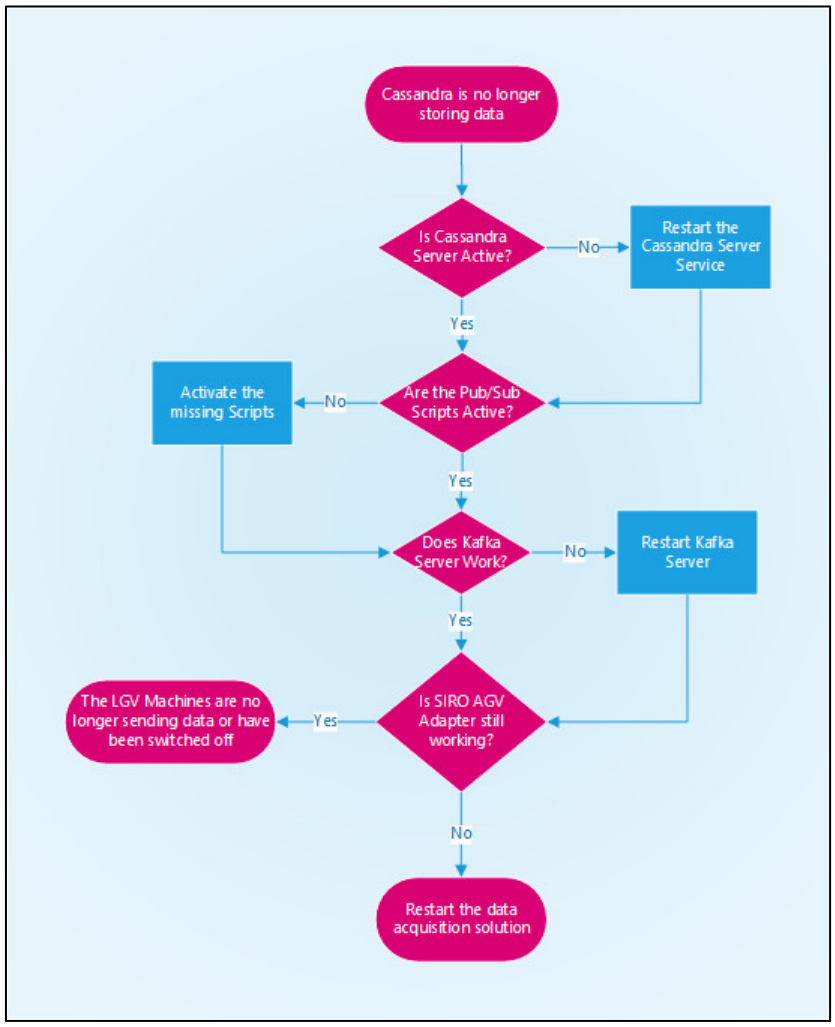

Figure 45 Cassandra fault root cause analysis flowchart

Moreover, in Figure 45, the Stream Analytics scripts and all the interconnections between Kafka and the other modules are defined as "Pub/Sub Scripts".

All the "Decisions" (red diamond) consist of scripts which are periodically executed by the Operative System: this means that an OS Task is responsible for triggering the execution of a script that checks if services are running and if the machines telemetry is being constantly acquired.

To manage the high system complexity, the activity of each macro component can be separately verified: this ensures the possibility to check their working status, independently each to the others.
We have to note that if one macro component stops working, the data flow is stopped accordingly.

5.3.1.2 The dashboard is no longer showing data

The dashboard is interconnected with Kafka and the databases: it shows data if the other modules are running.

Figure 46 shows all the verifications done in order to re-establish the correct functioning of the system, in case it stops working correctly.

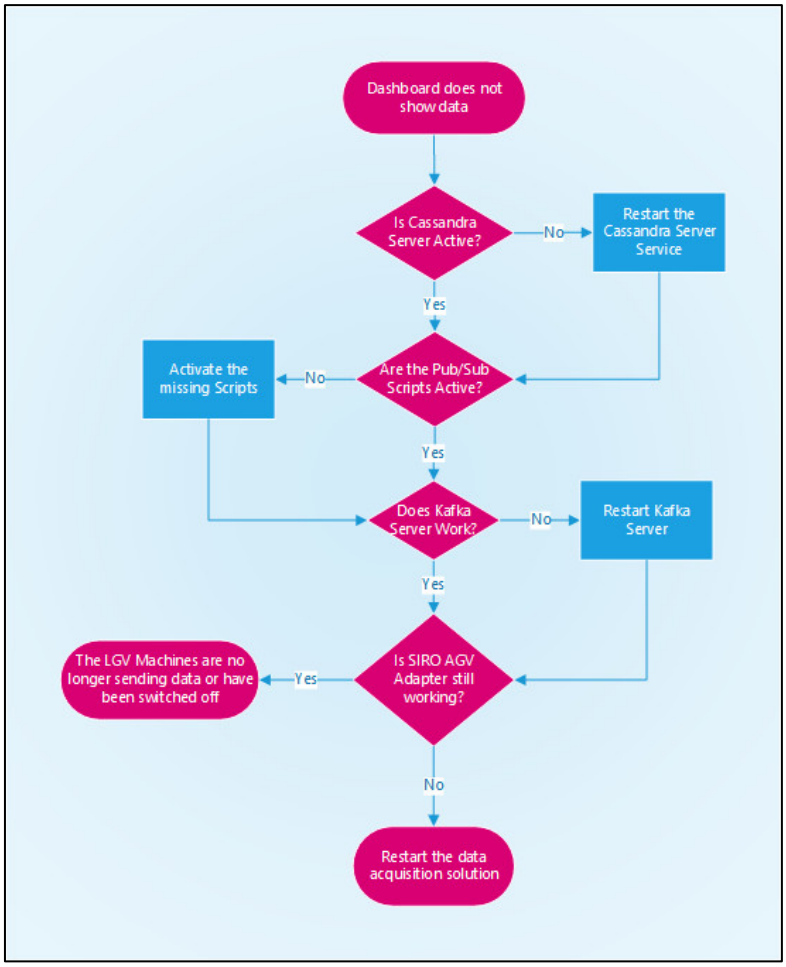

Figure 46 No data shown by dashboard" troubleshooting

Figure 46 is similar to Figure 45 and the components are constantly kept under control.

5.3.1.3 The data acquisition solution is no longer working

A script is responsible for defining if the data acquisition solution, SIRO AGV Adapter, is no longer working: it connects to the Cassandra database and checks if the data is being constantly collected.

If there is no new data, three events may have occurred:

- The data acquisition solution might be no longer active, or a deadlock might have occurred;
- One of the interconnected components might be no longer active;
- The LGV machines might no longer communicate with the server;

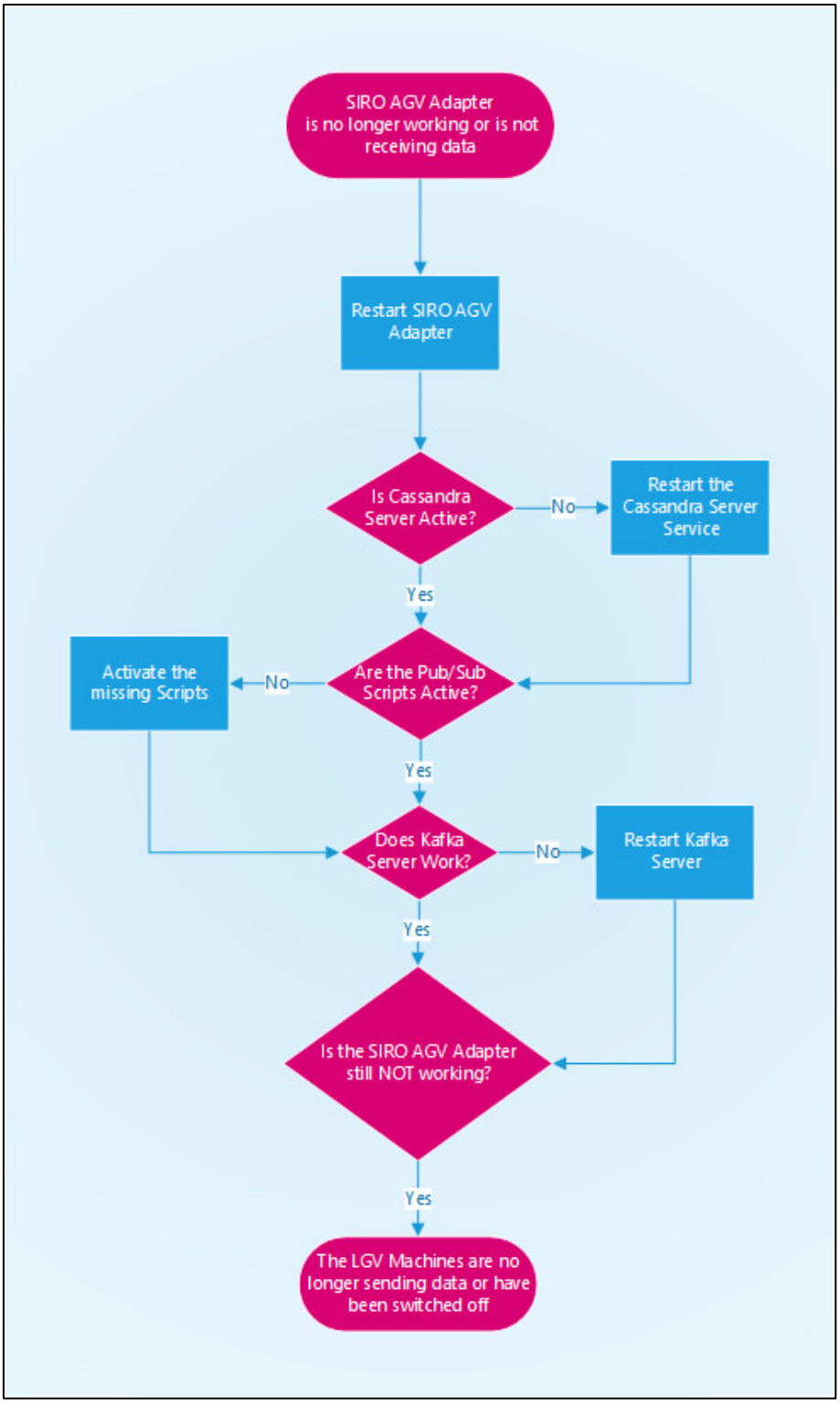

Figure 47 "SIRO AGV Adapter" troubleshooting

A script is responsible for checking if new data is being stored in Cassandra over time. If not, the SIRO AGV Adapter might have incurred in a deadlock and thus, it is restarted. This operation can be accomplished only if Cassandra is active.

5.3.2 Environmental fault causes

The software solution resides in a server located in a customer server room and it is running 24/7.
During its nominal working state, there might be environmental faults, which could be:

- Loss of internet connection;
- Operative System faults (BSOD, etc.);
- LGV machines no longer communicating with the server.

It is possible to automatically monitor the server machine activity status by using specific software, capable of acquiring the machine resource usage and of sending alerting messages in case particular events occur (if CPU and Memory usage exceed a predefined threshold if the machine is capable of sending its telemetry data over time, etc.).

The used software is DataDog [86] (https://www.datadoghq.com/): this is a software solution capable of acquiring software and hardware metrics and alert specific users basing on predefined rules.

Figure 48 shows a Datadog dashboard that constantly monitors and report Hardware usage and Cassandra activities over time.

Figure 48 Datadog dashboard

Figure 49 shows the Datadog alerting rules: if Cassandra or the server stop responding, predefined of users are alerted via email.

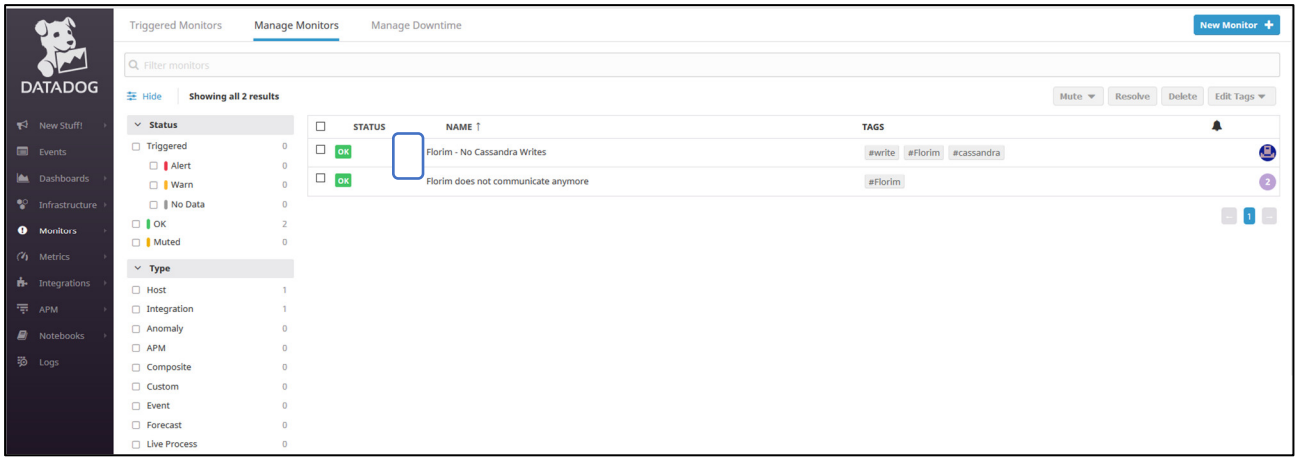

Figure 49 Alerting Rules

Figure 50 shows a set of procedures to be followed in case the server is no longer reachable.

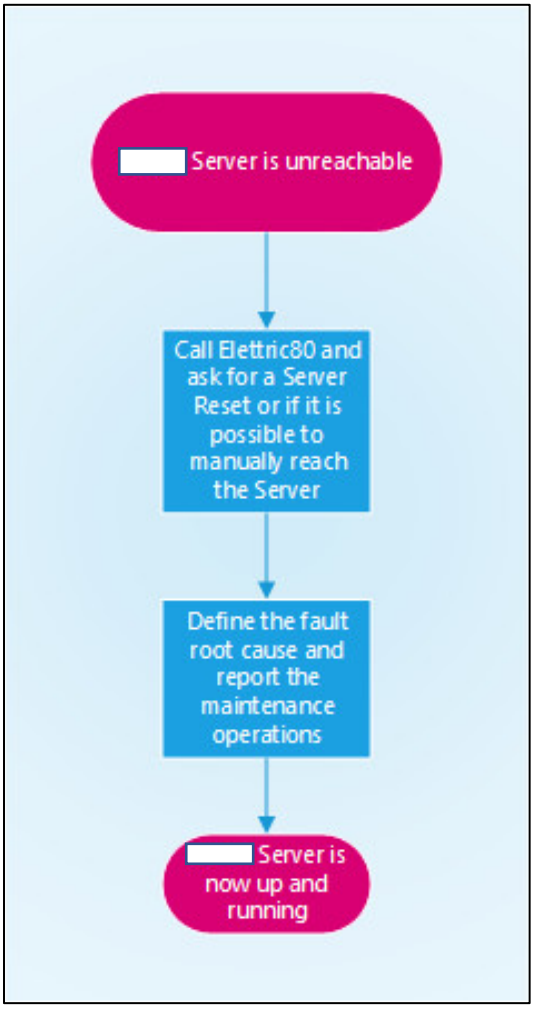

Figure 50 Server Fault Troubleshooting

6 Results

The result of this project is, firstly, the development of a conceptual architecture which is based on the lambda architecture.
Secondly, the machine monitoring solution requested by Elettric80.

The conceptual architecture was the first step to be taken in order to define the roadmap of the entire developed system.
The requirements acquisition allowed to define the main points to be addressed, which were:

- How to acquire data from automated machines;
- How to interpret data and how to structure it for analytical purposes;
- How to efficiently stream the collected data, in a distributed environment, letting it be available for all the required software components;
- How to efficiently store the collected data in a distributed environment, in which many actors are involved;
- How to analyse data in an efficient way: this point is connected with the defined way of streaming and storing data.
  If data is inefficiently streamed and stored, then it cannot be analysed in an efficient way;
- How the results are shown.

Basing on these technical requirements, a state of the art review has been done [Chapters 2, 3] and a conceptual pipeline has been developed and is re-shown in Figure 51:

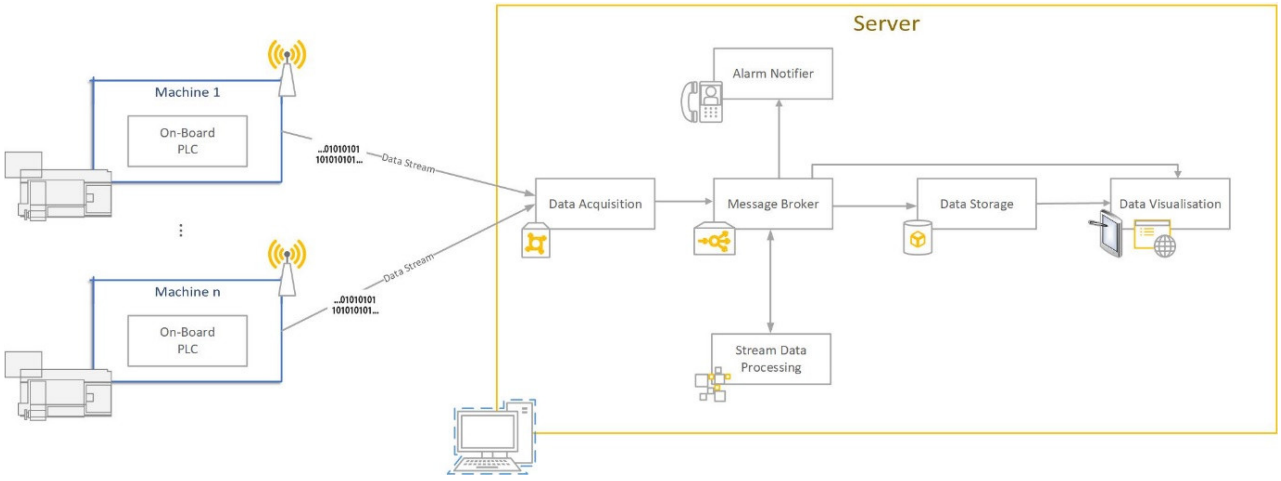

Figure 51 The conceptual architecture

The architecture has been developed in terms of micro-services: other services can be connected with others, ensuring total modularity. Moreover, as it is not monolithic, services and scripts can be replaced by others or updated without impacting the capabilities of the entire system.

Finally, basing on the requirements, the final architecture has been developed accordingly [Figure 52].

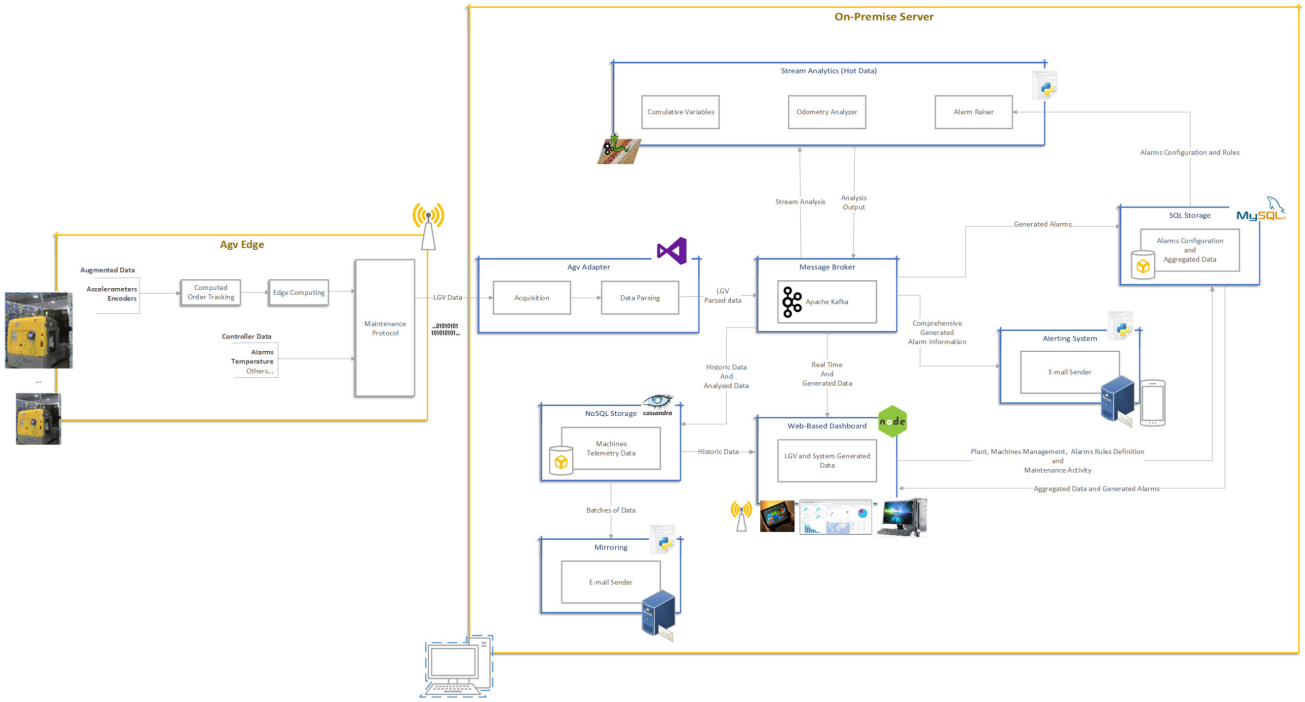

Figure 52 The developed architecture

In the developed software solution, the following features have been implemented:

- Machine data was pre-processed and was sent to specific software, developed for handling the communication with the machines, acquiring data and sending it to the Message Broker;
- The Message Broker was responsible for letting the data be transparent to all the other components;
- Scripts used the data contained in the message broker for analysing it on the fly;

- Databases were used for storing the machines data and the manually defined data (e.g. alarms);

- Data was sent to other servers, in a way which ensures an easy and secure data transferring methodology;

- Data could be overseen by means of a web-based dashboard, which allows the end-user to use it independently from the used device.

The architecture was designed in order to let it cope with any different Elettric80 customers scenarios: as said in Chapter 4.1, customers had thousands of machines and the system had to cope with such complexity.

Solutions like Apache Kafka and Apache Cassandra have been chosen for their scalability and reliability properties, allowing the system to even work with several machines.

Finally, the software solution received a name: MyHound [Figure 53].

Figure 53 The MyHound Logo

This provided us with the possibility to easily and efficiently identify the software solution, that has been developed for this use-case, with the intent of extending it to other similar use-cases and thus, projects.

7 Discussion

As said, starting from the requirements acquisition, a literature review and a methodology definition have been done accordingly: this allowed the development of a conceptual architecture which can be replicated in these project types.

In fact, industrial projects regarding machine monitoring solutions, including automated machines, always required the following practices and components:

- A deep study of the automated machine involved in the project as well as the environment in which has to work and its constraints. When developing the software architecture, analysis of the required hardware that must be used is fundamental;

- Development of a data structure to be sent from the machine and that has to be compliant with the objectives of the project: this means that, sometimes we might deal with projects in which sensor data is vital for the development of the machine monitoring solution. Other use-cases might only require the machine working state and the generable alarms. Thus, machine PLCs must be programmed in order to package all its sensor data and send it to the server responsible for acquiring this information;

- Establishment of a communication protocol with a device responsible for collecting data: this depends on the controller that has been installed on the automated machine as well as on the capabilities of the device for that has been designed for collecting data;

- A system responsible for letting the data be transparent to all the component is needed, but might vary depending on the number of machines which must be monitored: when several machines and their sensor data must be handled, a message broker is a useful solution which provides with scalability. Otherwise, data can be directly stored on databases which can then be browsed afterwards;

- A system responsible for storing data is needed when afterwards analyses are mandatory. Depending on the project and system constraints, NoSQL or SQL databases or together might be tied,

obtaining the best from the combination of both;

- A system responsible for showing data is necessary: we cannot pretend from a company to interface itself with the provided databases, because they might not have the necessary technical knowledge as well as the system might not appear complete at the end;

The work thesis regards the development of an architecture that had to be scalable and adaptable to any required workload as, during the requirements acquisition, Elettric80 expressed its interest in using the developed architecture in warehouses where many machines were working: if these constraints would have not been taken under consideration, further revisions of the entire architecture would have been needed and we could have lost the customer because of the time we would have spent.

Open-Source and edge technologies have been used for this architecture: these had to be innovative and usable at the same time. When choosing the components, researches about their capabilities and how much they were supported by the community and developers (as well as they roadmap) has been done: this ensures the usability of these technologies even after the accomplishment of the project.

Before their implementation, tests about their real capabilities and analyses about the effort needed to use them have been done: small exhibitors have been developed and used to prove them.
Finally, the chosen software components have been implemented.

As it happens with large projects, requirements might vary and change time to time: this requires the teams in charge of the development to be agile and work according to the newly acquired requirements.

Elettric80 resulted from the developed software solution enthusiast and together, with the research group, learnt a lot from this experience:

- The research group became agile and capable of coping with the new requirements as well as explaining complex structures and architecture in a simple and effective way.
  In order to do so, a platform for fast agile and development as well as project management tools have been used: Atlassian Jira [Figure 54] [87] Bitbucket [88].

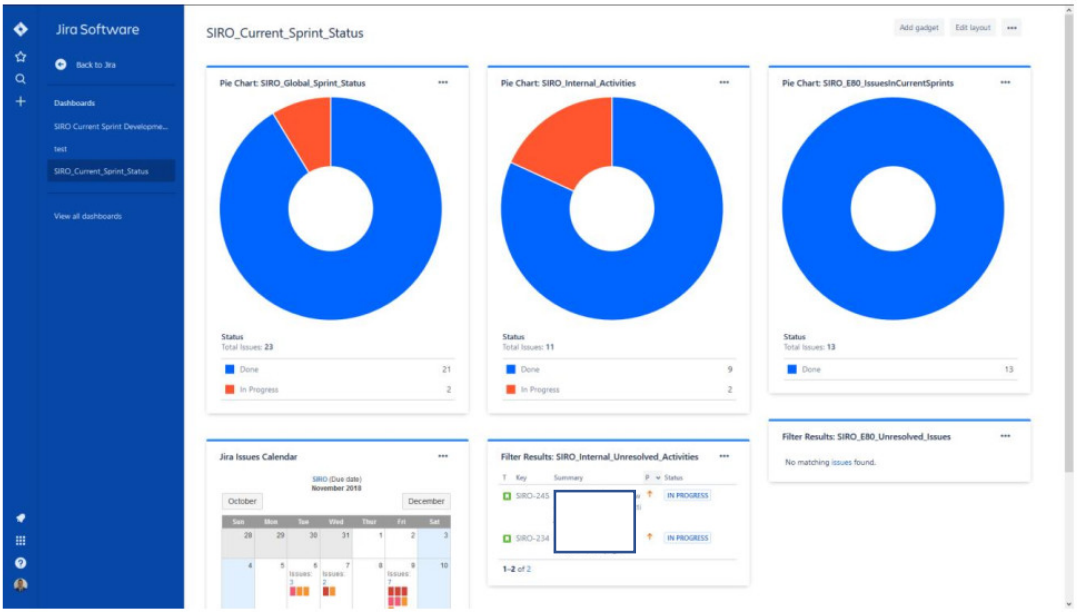

Figure 54 A Jira Dashboard used for the Siro Project

With Atlassian Jira, management of the project and team was possible: tasks could be easily assignable and connectable with the git repository project, stored in Bitbucket. Moreover, by means of an editable dashboard, the team could easily track and show the status of the work;

- The company learnt the used new edge technologies (like Cassandra and Kafka) and how to implement them in complex environments.

8 Conclusions and Future Works

This work thesis is focussed on the development of a software solution responsible for acquiring machine data and transform it into useful information for its end-users. The platform has been developed for acquiring signals from many machines and handle volumes of data in a scalable way.
A dashboard is responsible for monitoring the machine and provide results about their performances.

As it is a pipeline made of different solutions tied together, its deployment might be complicated when a certain performance grade must be reached.

Basing on this experience and on other industrial projects, software deployment has been always critical when its complexity was increasing: this problem can be solved by shipping Docker containers.
As said in Chapter 3.3, all the components can be Containerised and shipped to production: this allows the deployment of an entire pipeline without the need of installing every single package manually. In fact, the deployment of all the components might be automatic and easy at the same time by using the Docker Compose tool [89].

Chapter 3.3.1 suggests Docker as containerisation system, allowing to have the same experience from the same software on different systems.

When scalability and a system capable of administering it automatically is also needed, all the containers can be also managed by means of Kubernetes, as described in Chapter 3.3.2.

Docker and Kubernetes allow to manage complex environments in an easier way, handling availability and scalability by means of a unique system:

- Docker allows to deploy a micro services-oriented software solution in an easy way;

- Kubernetes allows to manage the docker containers (and thus the used software), scaling and monitoring them.

Moreover, both solutions can then be remotely administered, making the development and shipping to production even faster and easier.

9 References

[1] C. S. Longo, C. Fantuzzi, F. Monica, L. Manfredotti and M. Sorge, "Big Data for advanced monitoring system: an approach to manage system complexity," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, 20-24 Aug. 2018.

[2] J. D. Campbell and J. V. Reyes-Picknell, Uptime Strategies for Excellence in Maintenance Management, CRC Press Taylor and Francis Group, 2016.

[3] M. Sherif, L. Sang-Heon , D. Jantanee and C. Nicholas , Lean thinking for a maintenance process, Production & Manufacturing Research, Taylor & Francis Group, 2015, pp. 236-272.

[4] D. Mungani and J. Visser, "Maintenance approaches for different production methods," *South African Journal of Industrial Engineering,* vol. 24, no. 3, pp. 1-13, November 2013.

[5] British Standards Institution, Maintenance Terminology, BS EN 13306, 2001.

[6] R. Keith Mobley, Maintenance Engineering Handbook, McGraw-Hill Education, 2014.

[7] M. Ben-Daya, S. O. Duffuaa, A. Raouf, J. Knezevic and D. Ait-Kadi, Handbook of Maintenance Management and Engineering, Springer-Verlag London, 2009.

[8] UpKeep, "Types of Maintenance," UpKeep Maintenance Management, 2019. [Online]. Available: https://www.onupkeep.com/learning/maintenance-types/.

[9] G. Arbour, "Maintenance strategies: 4 approaches to asset management," Fiix, 2019. [Online]. Available: https://www.fiixsoftware.com/blog/evaluating-maintenance-strategies-select-model-asset-management/.

[10] E. Hupjé, "9 Types of Maintenance: How to Choose The Right Maintenance Strategy," roadtoreliability, 2018. [Online]. Available: https://www.roadtoreliability.com/types-of-maintenance/.

[11] Z. Huo, Z. Zhang, Y. Wang and G. Yan, "CMMS Based Reliability Centered Maintenance," *IEEE/PES Transmission & Distribution Conference & Exposition: Asia and Pacific,* 2005.

[12] I. Lopes, P. Senra, S. Vilarinho, C. Teixeira, J. Lopes, A. Alves, J. A. Oliveira and M. Figueiredo, "Requirements specification of a computerized maintenance management system – a case study," *Procedia CIRP,* vol. 52, pp. 268-273, 2016.

[13] R. Y. Zhong, X. Xu, E. Klotz and S. T. Newman, "Intelligent Manufacturing in the Context of Industry 4.0: A Review," *Engineering,* vol. 3, no. 5, pp. 616-630, 2017.

[14] F. Arévalo, M. R. Diprasetya and A. Schwung, "A Cloud-based Architecture for Condition Monitoring based on Machine Learning," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 2018.

[15] K. Al-Gumaei, K. Schuba, A. Friesen, S. Heymann, C. Pieper, F. Pethig and S. Schriegel, "A Survey of Internet of Things and Big Data Integrated Solutions for Industrie 4.0," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, 2018.

[16] R. Zafar, E. Yafi, M. F. Zuhairi and H. Dao, "Big Data: The NoSQL and RDBMS review," in *2016 International Conference on Information and Communication Technology (ICICTM)*, Kuala Lumpur, Malaysia, 2016.

[17] NIST Big Data Public Working Group (NBD-PWG), "NIST Big Data Interoperability Framework," 2015. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.1500-1. [Accessed 10 10 2019].

[18] J. Camacho, G. Macià-Fenandez, J. Diaz-Verdejo and P. Garcia-Teodoro, "Tackling the Big Data 4 Vs for Anomaly Detection," in *2014 IEEE INFOCOM Workshops: 2014 IEEE INFOCOM Workshop on Security and Privacy in Big Data*, Toronto, ON, Canada, 2014.

[19] A. Gueld, H. Gharsellaour and S. B. Ahmed, "A NoSQL-based Approach for Real-Time Managing of Embedded Data Bases," in *2016 World Symposium on Computer Applications & Research (WSCAR)*, Cairo, Egypt , 2016.

[20] T. Partel and T. Eltaieb, "Relational Database vs NoSQL," *Journal of Multidisciplinary Engineering Science and Technology (JMEST),* vol. 2, pp. 691-695, April 2015.

[21] A. Nayak, A. Poriya and D. Poojary, "Type of NoSQL Databases and its Comparison with Relational Databases," *International Journal of Applied Information Systems (IJAIS),* vol. 5, no. 4, pp. 16-19, March 2013.

[22] M. A. Mohammed, O. G. Altrafi and M. O. Ismail, "Relational vs. NoSQL databases: A survey," *International Journal of Computer and Information Technology,* vol. 3, no. 3, pp. 598-601, May 2014.

[23] A. Zahid, R. Masood and M. A. Shibli, "Security of Sharded NoSQL Databases: A Comparative Analysis," in *IEEE Conference on Information Assurance and Cyber Security*, Rawalpindi, Pakistan , 2014.

[24] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *2011 International Conference on Cloud and Service Computing*, Hong Kong, China , 12-14 Dec. 2011.

[25] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva and U. Saxena, "NoSQL databases: Critical analysis and comparison," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, Gurgaon, India , 2017.

[26] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia , 2018.

[27] S. Gilbert and N. Lynch, "Perspectives on the CAP Theorem," *Computer,* vol. 45, no. 2, pp. 30 - 36, Feb. 2012.

[28] K. Birman, D. Freedman, Q. Huang and P. Dowell, "Overcoming CAP with Consistent Soft-State Replicatio," *Computer,* vol. 45, no. 2, pp. 50 - 58, Feb. 2012.

[29] B. Yadranjiaghdam, N. Pool and N. Tabrizi, "A Survey on Real-time Big Data Analytics:," in *2016 International Conference on Computational Science and Computational Intelligence*, 2016.

[30] S. Kamburugamuve, L. Christiansen and G. Fox, "A Framework for Real Time Processing of Sensor Data in the Cloud," *Journal of Sensors,* vol. 2015, p. 11, 2018.

[31] B. R. Hiraman, C. Viresh M and C. K. Abhijeet, "A Study of Apache Kafka in Big Data Stream Processing," *International Conference on Information , Communication, Engineering and Technology (ICICET), 29-31 Aug. 2018,* 2018.

[32] R. Shree, U. Pradesh, T. Choudhury, S. C. Gupta and P. Kumar, "KAFKA: The modern platform for data management and analysis in big data domain," *2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 10-11 Aug. 2017.*

[33] A. Ara and A. Ara, "Case study: Integrating IoT, streaming analytics and machine learning to improve intelligent diabetes management system," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 1-2 Aug. 2017.*

[34] V.-D. Ta, C.-M. Liu and G. W. Nkabinde, "Big Data Stream Computing in Healthcare Real-Time Analytics," *IEEE International Conference on Cloud Computing and Big Data Analysis,* pp. 37-42, 2016.

[35] A. Sotsenko , M. Jansen, M. Milrad and J. Rana, "Using a Rich Context Model for Real-Time Big Data Analytics in Twitter," *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW),* 2016.

[36] V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 2015.

[37] L. Magnoni, "Modern Messaging for Distributed Sytem," *Journal of Physics: Conference Series 608 (2015) 012038,* 2015.

[38] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Vienna, Austria , 11-13 Oct. 2017.

[39] S. Singh and N. Singh, "Containers & Docker: Emerging roles & future of Cloud technology," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Bangalore, India , 27 April 2017 .

[40] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing ,* vol. 1, no. 3, pp. 81 - 84, Sept. 2014.

[41] P. E N, F. Jaison Paul Mulerickal, B. Paul and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," in *2015 International Conference on Control Communication & Computing India (ICCC)*, Trivandrum, India , 19-21 Nov. 2015 .

[42] J. Mahn, M. Schumacher and P. Siering, "Warum Container?," *c't wissen DOCKER: Komplexe Software eifach einrichten,* pp. 6-10, 2019.

[43] Docker Inc., "What is a Container? A standardized unit of software," Docker Inc., 2019. [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 14 10 2019].

[44] A. Lingayat, R. R. Badre and A. K. Gupta, "Performance Evaluation for Deploying Docker Containers On Baremetal and Virtual Machine," in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, Coimbatore, India, India , 15-16 Oct. 2018.

[45] M. Coleman, "Containers and VMs Together," Docker Inc., 08 04 2016. [Online]. Available: https://www.docker.com/blog/containers-and-vms-together/. [Accessed 14 10 2019].

[46] P. Kasireddy, "A Beginner-Friendly Introduction to Containers, VMs and Docker," freeCodeCamp, 04 03 2016. [Online]. Available:

https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/. [Accessed 14 10 2019].

[47] Y. Li and Y. Xia, "Auto-scaling web applications in hybrid cloud based on docker," in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, Changchun, China , 10-11 Dec. 2016 .

[48] E. Mavungu, "Docker Storage: An Introduction," Codeship by CloudBees, 05 05 2017. [Online]. Available: https://blog.codeship.com/docker-storage-introduction/. [Accessed 14 10 2019].

[49] I. M. Al Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari and A. Palopoli, "Container Orchestration Engines: A Thorough Functional and Performance Comparison," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, China , 20-24 May 2019 .

[50] A. Suleman, "Docker And Kubernetes: Furthering The Goals Of DevOps Automation," 10 10 2018. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/10/10/docker-and-kubernetes-furthering-the-goals-of-devops-automation/#508763af6506. [Accessed 14 10 2019].

[51] Kubernetes, "What is Kubernetes," 06 10 2019. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Accessed 14 10 2019].

[52] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin and J.-Y. Jeng, "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, Singapore , 4-8 Dec. 2017 .

[53] L. A. Vayghan, M. A. Saied, M. Toeroe and F. Khendek, "Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, Sofia, Bulgaria, Bulgaria , 22-26 July 2019 .

[54] M. Gawel and K. Zielinski, "Analysis and Evaluation of Kubernetes Based NFV Management and Orchestration," in *2019 IEEE 12th International*

Conference on Cloud Computing (CLOUD), Milan, Italy, Italy , 8-13 July 2019 .

[55] G. Rattihalli, M. Govindaraju and H. Lu, "Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy, Italy, 8-13 July 2019.

[56] Kubernetes, "https://kubernetes.io/docs/concepts/," Kubernetes, 2019. [Online]. Available: https://kubernetes.io/docs/concepts/. [Accessed 16 10 2019].

[57] C. Fantuzzi, R. Panciroli and M. Gargiulo, "Hardware in the loop simulation for distributed automation systems," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, Krakow, Poland , 17-21 Sept. 2012 .

[58] L. Racchetti and C. Fantuzzi, "Hardware in the loop simulation and Machine Modular Development: Concepts and application," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, Cagliari, Italy, 10-13 Sept. 2013 .

[59] C. S. Longo and C. Fantuzzi, "Simulation and optimisation of production lines in the framework of the IMPROVE project," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Cyprus , 12-15 Sept. 2017.

[60] C. S. Longo and C. Fantuzzi, "Simulation and optimization of industrial production lines," *at - Automatisierungstechnik,* vol. 66, no. 4, pp. 320-330, 2018.

[61] . R. Petrasch and . R. Hentschke , "Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method," in *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Khon Kaen, Thailand , 13-15 July 2016 .

[62] L. Bassi, "Industry 4.0: Hope, hype or revolution?," in *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, Modena, Italy , 11-13 Sept. 2017 .

[63] P. F. S. de Melo and E. P. Godoy, "Controller Interface for Industry 4.0 based on RAMI 4.0 and OPC UA," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT)*, Naples, Italy, Italy, 4-6 June 2019 .

[64] K. H, W. W and H. J, "Recommendations for implementing the strategic initiative Industrie 4.0 – Final report ot the Industrie 4.0 Working Group," Acatech National Academy of Science and Engineering, 2013.

[65] I. Fedoseenko, "Business benefits from leveraging the Cloud," BCS, The Chartered Institute for IT, 2018. [Online]. Available: https://www.bcs.org/content-hub/business-benefits-from-leveraging-the-cloud/. [Accessed 17 10 2019].

[66] S. Karnouskos, L. Ribeiro, P. Leitão, A. Lüder and B. Vogel-Heuser, "Key Directions for Industrial Agent Based Cyber-Physical Production Systems," in *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, Taipei, Taiwan, Taiwan, 6-9 May 2019.

[67] N. Marz and J. Warren, Big Data: Principles and best practices of scalable realtime data systems, Shelter Island: Manning, 2013.

[68] K. Al-Gumaei, A. Müller, J. N. Weskamp, C. S. Longo, F. Pethig and S. Windmann, "Scalable Analytics Platform for Machine Learning in Smart Production Systems," in *The 24th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2019)*, Zaragoza, Spain, September 10th - 13th, 2019.

[69] . Y. Yamato, . H. Kumazaki and Y. Fukumoto, "Proposal of Lambda Architecture Adoption for Real Time Predictive Maintenance," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, Hiroshima, Japan, 22-25 Nov. 2016.

[70] M. Kiran, P. Murphy, I. Monga, J. Dugan and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE International Conference on Big Data (Big Data)*, Santa Clara, CA, USA, 29 Oct.-1 Nov. 2015.

[71] S.-H. Han and Y.-K. Kim, "An Architecture of Real-Time, Historical Database System for Industrial Process Control and Monitoring," in *2011*

First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering, Jeju Island, South Korea, 23-25 May 2011.

[72] The Apache Software Foundation, "Apache Cassandra," Apache Cassandra, 2016. [Online]. Available: http://cassandra.apache.org/. [Accessed 21 10 2019].

[73] V. D. Jogi and A. Sinha, "Performance evaluation of MySQL, Cassandra and HBase for heavy write operation," in *3rd Intl Conf. on Recent Advances in Information Technology (RAIT) 2016*, Dhanbad, India, 3-5 March 2016.

[74] J. Hammink , "An Introduction to Apache Cassandra," DZone, 19 07 2019. [Online]. Available: https://dzone.com/articles/an-introduction-to-apache-cassandra. [Accessed 21 10 2019].

[75] Tutorials Point, "Learn Cassandra," Tutorialspoint, 2015. [Online]. Available: https://www.tutorialspoint.com/cassandra/. [Accessed 21 10 2019].

[76] K. Ferencz and J. Domokos, "IoT Sensor Data Acquisition and Storage System Using Raspberry Pi and Apache Cassandra," in *2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE)*, Budapest, Hungary , 20-21 Nov. 2018 .

[77] Oracle Corporation, "MySQL," Oracle Corporation, 2019. [Online]. Available: https://www.mysql.com/. [Accessed 21 10 2019].

[78] ScaleGrid, "2019 Database Trends – SQL vs. NoSQL, Top Databases, Single vs. Multiple Database Use," ScaleGrid, 4 3 2019. [Online]. Available: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/. [Accessed 21 10 2019].

[79] solid IT gmbh, "DB-Engines Ranking," solid IT gmbh, 2019. [Online]. Available: https://db-engines.com/en/ranking. [Accessed 21 10 2019].

[80] . S. Tongkaw and A. Tongkaw, "A comparison of database performance of MariaDB and MySQL with OLTP workload," in *2016 IEEE Conference on Open Systems (ICOS)*, Langkawi, Malaysia , 10-12 Oct. 2016 .

[81] The Apache Software Foundation, “Apache Kafka,” The Apache Software Foundation, 2017. [Online]. Available: https://kafka.apache.org/. [Accessed 22 10 2019].

[82] Apache Software Foundation, “Apache Kafka,” Apache Software Foundation, 2017. [Online]. Available: https://kafka.apache.org/10/documentation/streams/developer-guide/security.html. [Accessed 30 10 2019].

[83] Node.js Foundation. , “NodeJs,” Node.js Foundation. , 2019. [Online]. Available: https://nodejs.org/en/. [Accessed 24 10 2019].

[84] D. Laksono, “Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App,” in *2018 4th International Conference on Science and Technology (ICST)*, Yogyakarta, Indonesia , 7-8 Aug. 2018 .

[85] M. Bostock, “D3.js,” 2019. [Online]. Available: https://d3js.org/. [Accessed 24 10 2019].

[86] Datadog, “DataDog,” DataDog, 2019. [Online]. Available: https://www.datadoghq.com/. [Accessed 23 10 2019].

[87] Atlassian , “Atlassian Jira,” Atlassian , 2019. [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 28 10 2019].

[88] Atlassian, “Atlassian Bitbucket,” Atlassian, 2019. [Online]. Available: https://www.atlassian.com/software/bitbucket. [Accessed 28 10 2019].

[89] Docker Inc., “Docker Compose,” Docker Inc., 2019. [Online]. Available: https://docs.docker.com/compose/. [Accessed 31 10 2019].

10 Table of Figures