SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

M

Master
Engenharia Informática

IIoT Data Ness: From Streaming to Added Value

Ricardo André Araújo Correia

10/2022

SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

M

MASTER
ENGENHARIA INFORMÁTICA

IIoT Data Ness: From Streaming to Added Value
Ricardo André Araújo Correia

Ricardo André Araújo Correia. IIoT Data Ness: From Streaming to Added Value

10/2022

SCHOOL OF
MANAGEMENT AND
TECNOLOGY
POLYTECHNIC OF
PORTO

M

MASTER
ENGENHARIA INFORMÁTICA

IIoT Data Ness: From Streaming to Added Value

Ricardo André Araújo Correia

Cristóvão Dinis Polido Sousa

Davide Rua Carneiro

Acknowledgments

I want to thank all the people that supported me during the development of this work, because without you I'm certain that it wouldn't be possible.

I'll start by thanking both Professor Cristovão Sousa and Professor Davide Carneiro for the countless hours that was spent during the orientation. Thank you for believing in my potential and giving me all the support to develop this project. You're both outstanding role models.

At the personal level, I would like to start by thanking my beloved wife, that stood tirelessly by my side while I spent the days on end in the computer working on this project. Thank you for your partnership, strength and comfort that you have provided me.

I would also like to thank my friends Carlos and Pedro that have played a crucial part in keeping me motivated throughout the development of this work.

Lastly, I would like to thank all of my colleagues that quickly became friends in the companies I worked at, and cheered my efforts as a worker student. Thank you for contributing to my professional growth. A special thanks to André Duarte, who gave me the final push in motivation to finish this project.

Thank you all!

Abstract

In the emerging Industry 4.0 paradigm, the internet of things has been an innovation driver, allowing for environment visibility and control through sensor data analysis. However the data is of such volume and velocity that data quality cannot be assured by conventional architectures. It has been argued that the quality and observability of data are key to a project's success, allowing users to interact with data more effectively and rapidly. In order for a project to become successful in this context, it is of imperative importance to incorporate data quality mechanisms in order to extract the most value out of data. If this goal is achieved one can expect enormous advantages that could lead to financial and innovation gains for the industry. To cope with this reality, this work presents a data mesh oriented methodology based on the state-of-the-art data management tools that exist to design a solution which leverages data quality in the Industrial Internet of Things (IIoT) space, through data contextualization. In order to achieve this goal, practices such as FAIR data principles and data observability concepts were incorporated into the solution. The result of this work allowed for the creation of an architecture that focuses on data and metadata management to elevate data context, ownership and quality.

Keywords— Data observability, Data quality, FAIR data, Data Mesh, Data Fabric, IIoT

Resumo

O conceito de *Internet of Things* (IoT) é um dos principais fatores de sucesso para a nova Indústria 4.0. Através de análise de dados sobre os valores que os sensores coletam no seu ambiente, é possível a construção uma plataforma capaz de identificar condições de sucesso e eventuais problemas antes que estes ocorram, resultando em ganho monetário relevante para as empresas. No entanto, este caso de uso não é de fácil implementação, devido à elevada quantidade e velocidade de dados proveniente de um ambiente de IIoT (*Industrial Internet of Things*). Outro fator que também contribui para este possível cenário de insucesso é o facto de os sensores implementados serem sensíveis a erros de dados. Exemplos incluem dados corrompidos, dados não enviados, *outliers*, duplicados, entre outros. Muitos estudos indicam que fatores como a qualidade de dados e observabilidade de dados são as principais causas de sucesso no projeto de uma arquitetura de dados. Por estas razões, convencionais arquiteturas de dados não são suficientes para o desenvolvimento destes casos, e o requisito de uma metodologia baseada nas mais recentes arquiteturas com um maior foco na gestão de qualidade de dados e capacidades de observabilidade é cada vez mais inerente. Neste trabalho exploramos as mais recentes arquiteturas de dados, como *Data Fabric* e *Data Mesh*, para a construção de um sistema que consiga a gestão de uma elevada quantidade de dados de forma escalável, tendo em conta arquiteturas de referência já existentes, que contam com características como design modular, adaptável, flexível e centrado em desempenho. A construção desta metodologia começou pela criação de um *Data Lake* que consumia todos os dados provenientes do ambiente e, posteriormente, estes seriam distribuídos e processados por zonas diferentes do *Data Lake* por pipelines de processamento. A próxima iteração resultou na criação de uma camada responsável pela gestão de metadados, com inspiração nos princípios de arquitetura do *Data Fabric*. Esta nova camada permitiu um melhor controlo sobre a qualidade dos dados, fornecendo contextualização dos dados recolhidos por processos semânticos. A última iteração concebida foi feita com inspiração no padrão de arquitetura do *Data Mesh*, colhendo de benefícios resultantes de tratar os dados como um produto. São muitos os estudos na vertente de qualidade de dados, incluindo em ambientes de IIoT. Para estudo deste problema é preciso definir o conceito de dados e como se interliga com o de qualidade de dados. A definição de dados engloba os valores que são recolhidos continuamente através dos sensores e de outros pontos de integração do ambiente, assim como o contexto à volta destes valores, como, por exemplo, a maquina em que o sensor está, há quanto tempo o sensor for calibrado, os limites superiores e inferiores, entre outros. Parte deste contexto é composto por algumas métricas referentes a características de observabilidade de dados, tais como volume, distribuição, esquema, *lineage* e *freshness*. Estas características fornece ao utilizador uma visibilidade sobre não só os dados, mas também, sobre os processos com interferência sobre esses dados, podendo potencialmente identificar pontos de falhar ou ineficácia. Para além da observabilidade de dados, um dos grandes pontos de discussão no que toca a arquitetura de dados é capacidade da arquitetura para reutilizar e integrar os dados que coleta. Por esta razão foi desenvolvido um conjunto de princípios, denominados FAIR data principles, cujo objetivo é fornecer uma diretriz para alcançar este propósito. O primeiro princípio apresentado é a encontrabilidade dos dados, que indica que os dados devem estar visíveis na plataforma com metadados de apoio a esta tarefa. O segundo princípio é a acessibilidade dos dados, que dita que depois dos dados serem descobertos pelo utilizador, estes têm de ser facilmente acedidos, com padrões estabelecidos de comunicação. Interoperabilidade dos dados é o terceiro ponto destes princípios, que visa a integração de diferente conjuntos de dados entre si, para elevar o valor obtido dos dados. Em último lugar,

a reutilização dos dados é um ponto fundamental, que permite aos utilizadores processarem os dados múltiplas vezes para diferentes objetivos sem terem de comprometer os dados originais. A arquitetura desenhada toma como inspiração a arquitetura de *Data Mesh*, e foca-se nos pontos centrais de qualidade de dados, nomeadamente os princípios FAIR e observabilidade de dados. Com esta visão foi desenhado um sistema que usa tecnologias atuais. O primeiro componente desenvolvido é denominado *Context Broker* e funciona como a camada de gestão de metadados do ambiente, contendo informação de contextualização sobre os dados. Este componente toma vantagem da tecnologia de NGSI-LD, e usa Smart Models para a modelagem de dados. Seguidamente, o *Data Gateway* foi desenhado para consumir os dados que chegam continuamente do ambiente, fornecendo-os a uma série de serviços para poderem ser processados. Este componente foi desenhado com a tecnologia de Apache Kafka. O componente responsável pelo processamento de dados são as *Processing Pipelines*, que recolhem os dados de uma subdivisão do *Data Gateway* e que fazem uma serie de transformações aos dados. Apache Beam foi a ferramenta escolhida para este componente e toma uso do motor de processamento de Apache Flink. As *Processing Pipelines* foram desenhadas para serem reutilizadas entre processos. Por último, Apache Druid tomou o papel de auxiliar na visualização de descobrimento de dados na plataforma.

Palavras-chave— Observabilidade de dados, Qualidade de dados, FAIR data, Data Mesh, Data Fabric, IIoT

Contents

List of Figures

List of Tables

Listings

Acronyms

API Application Programming Interface. *Glossary:*

AWS Amazon Web Services. 11, 12, 26, VIII

BI Business Intelligence. 11

CBKA Context Broker Kafka Adapter. 44

CEP Complex Event Processing. 32, 37

CPS Cyber-Physical Systems. 3

CRM Customer Relationship Management. 12

DAG Directed Acyclic Graph . 39, *Glossary:* DAG

DoD Definition of Data. 18

ETL Extract Transform Load. 10, 24, 28, 29, 32, 33

FAIR Findable Accessible Interoperable Reusable. 7, 17, 23, 25, 60, II

HDFS Hadoop Distributed File System . 42, *Glossary:* HDFS

HTTP Hypertext Transfer Protocol. 48, 61

ICS Industrial Automation & Control Systems. 3, 4, VII

IIoT Industrial Internet of Things. 1, 3, 4, 7, 9, 17–19, 21, 23, 27, 37, 40, 45, 60, II

IoT Internet of Things. 1–5, 11, 12, 19–22, 26, 45, 53, VII

ISO International Organization for Standardization . 21, *Glossary:* ISO

JSON JavaScript Object Notation. 5, 43, 51

JSON-LD JavaScript Object Notation Linked Data. 26

JVM Java Virtual Machine. 42

ML Machine Learning. 37

NGSI Next Generation Service Interfaces. 26, V

NGSI-LD Next Generation Service Interfaces Linked Data. 26

NoSQL Not Only SQL. 43

RDD Resilient Distributed Datasets. 38

S3 Simple Storage Service . 42, *Glossary:* S3

SAO Stream Annotaion Ontology. 47, 48, 60

SCADA Supervisory Control and Data Acquisition. 3

SDK Software Development Kit. 39

SQL Structured Query Language. 12, 36, 38

URL Uniform Resource Locator. 26

XML Extensible Markup Language. 5

Glossary

DAG A set of actions can be conceptually represented by a directed acyclic graph (DAG). A graph, which is visually represented as a collection of circles with some of them connected by lines to indicate the flow from one action to the next, is used to show the sequence of the activities. Every circle is referred to as a ”vertex,” and every line is referred to as a ”edge.” Each edge must necessarily reflect a single directional flow from one vertex to another since ”directed” implies that each edge has a clearly defined direction. The term ”acyclic” refers to a graph in which there are no loops, or ”cycles.”. IX

HDFS HDFS is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes. HDFS is one of the major components of Apache Hadoop, the others being MapReduce and YARN. [17]. IX

ISO ISO is a nongovernmental organization that comprises standards bodies from more than 160 countries, with one standards body representing each member country. ISO members collaborate to develop and promote international standards for technology, scientific testing processes, working conditions, societal issues, and more. Documents detailing these standards are then sold by ISO and its members.. IX

S3 Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements. [18]. X

1 Introduction

In the recent past, Internet of Things (IoT) has emerged as a revolutionary paradigm for connecting devices and sensors. This allows visibility and automation of an environment, opening the path to industrial process optimization which might lead to improved efficiency and increase flexibility [19]. When that paradigm was applied to the industry world it became the fourth industrial revolution [20], seeking to improve efficiency and provide visibility over not only the machines and products but also the whole value chain. The benefits of this new age of industrialization, also known as Industry 4.0, has been enabling small, medium, and large companies to improve their ways of working, thereby increasing quality and quantity of the product and services while reducing costs [21].

The adoption of IoT in the industry has been steadily increasing not only vertically, but also horizontally. Vertical growth is driven by adding all kinds of sensors, wearables, and actors, estimating the market to grow to 102.460 million USD by the year 2028 [22]. This is because more clients and business departments are interested in the data available. In contrast, horizontal growth has been stimulated by the integration of multiple companies, producing information to the same data repository [19, 23, 24]. With machine learning, heavy computation processes, and powerful visualization tools, the data collected is empowered to enhance process efficiency and predictability across workstations, resulting in a massive increase in productivity and lower costs [25, 26]. However, without a scalable architecture in place to extract and improve data quality, the data gathered within the environment becomes an asset difficult to convert into value. This leads to what data scientists describe as a Data Swamp [27]. The quality of data extracted is a crucial factor in the success of an Industrial Internet of Things (IIoT) environment since the data obtained will heavily contribute to key business decisions and even automated actions within the production floor [24, 28]. It is possible that such scenarios could result in monetary losses or even security risks if not handled correctly. For this reason, one cannot rely solely on the sensors to produce quality data, since many of them are pruned to failure [29, 30]. Instead, a resilient architecture capable of identifying faulty data, managing data quality metrics, and ensuring confidence in the englobing environment must be implemented. Adding quality restrictions to the gathered data allows users to promote a much more productive communication between machines, processes, people, and organisations.

One of the most significant aspects of data quality is the observability level that can be inferred from it. [31, 3]. This is especially relevant when the data is getting more and more complex due to transformations and relationships. For this reason, an architecture designed to cope with IIoT demands must include a feature to provide data observability at a large scale, thus providing much-needed insights into the data.

To understand the motivation behind this work, it is necessary to first examine the world of the fourth industrial revolution. This is how it is reshaping the manufacturing processes to become more efficient, flexible and robust. It is also critical to contextualize Big Data meaning and what constitutes a Big Data problem. Finally, a brief explanation about why Data Quality is an undinable key driver for successful data management tools must be provided.

1.1 IoT: A interconnected world

The term Internet of Things, commonly abbreviated as IoT, represents an emergent paradigm that envisions a global network of machines and devices fully interconnected with each other [32]. This advancing technology has

proven to have multiple beneficial results when integrated either at the personal or professional level [33].

In the 1980s, the vision of IoT was presented with the objective of integrating technology more deeply into everyday life [34], and now it is clear that this vision is coming to life on a daily basis. At the individual level, IoT has been playing a critical role in areas such as e-health, smart houses and smart learning. The applications at the professional and enterprise level are far more extensive, playing a key role in smart supply chain [35], transportation [36], remote monitoring [37], agriculture [23], smart cities [38], healthcare [39].

Within this context, both the amount of *things* as well has its types, raging from reading sensors, to entire buildings, have been growing immensely, estimating a more than 29 billion IoT devices in 2030 according to a recent study [40], almost tripling since 2020 with around 9.7 billion. In addition, this study shows that IoT devices used in the industry, such as electricity, gas, steam, and A/C, water supply, waste management, retail and wholesale, transportation and storage, and government, will see tremendous growth, projecting the addition of more than 8 billion devices. The reasons for this growth can be attributed to the improvements IoT applications have achieved within environments. This paradigm offers advantages such as real-time visibility [32], automated tasks [41], and predictive maintenance [42]. This concept has been elevated once again in light of advancements such as machine learning [43], blockchain [44], virtual reality [45], 5G [46], data security and privacy [47], cloud-native applications [48] and hyperautomation [49].

1.2 Industry 4.0: A emerging revolution

The industrial world has seen multiple revolutions that continuously increase production values and efficiency. The first revolution was marked by the invention of the steam engine, which allowed the transition to manufacturing processing, using coal has the main energy source, and trains as the main means of transportation. The second industrial revolution began with the invention of the internal combustion engine. This led to an era of rapid industrialization with mass production capabilities, with the help of electricity and oil. The third revolution was started with the integration of electronic components and information technology to automate production on a large scale [20].

The industry 4.0 was first proposed in 2011 as the next big step in developing the German economy [50], and is now known as the latest industrial revolution. This new revolution is building on the third evolution, elevating the digital environments previously constructed, and blurring the line between digital, physical and biological spheres [20]. Figure 1 illustrates the iterations between the different revolutions.

The further evolution of automation in factories is enabled by the advancement of miniaturization, digitalization and networking, enabling the creation of Cyber-physical systems where the material and digital levels merge. It is at this point that concepts such as Digital Twin emerge. Digital twin represents the process in which a real-world object is mirrored into a digital counter-part, including all the communications between them [51, 52]. With artificial intelligence, this method can be utilized in simulations in order to improve object behavior or prevent failure scenarios in near real-time. This technology can also be used to monitor and control the physical environment remotely [53]. Figure 2 represents a standart implementation of the digital twin concept and how it interacts with the physical space

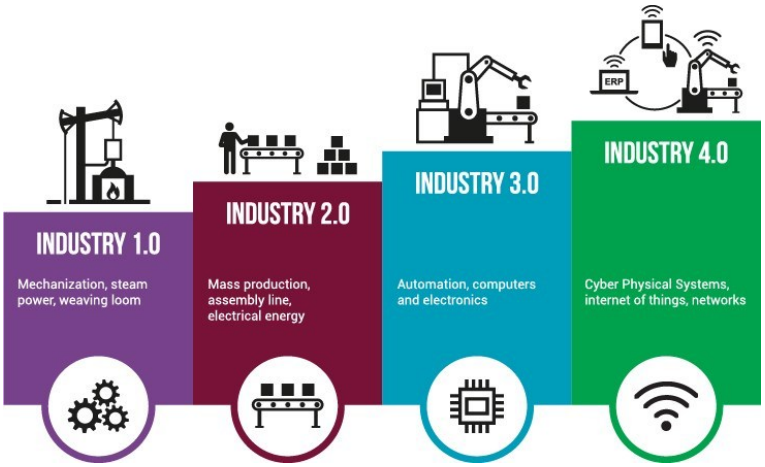

Figure 1: Industrial revolutions represented by Simio LLC, in Cision [1]

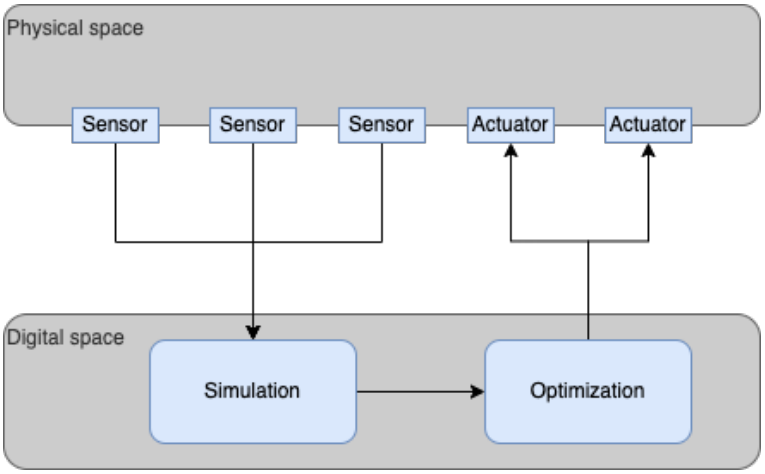

Figure 2: Digital Twin representation

1.3 The industrial internet of things

One of the main drivers of Industry 4.0 is the integration of the concept of IoT within the industrial space. This integration results in the industrial internet of things, commonly abbreviated to IIoT [21]. This integration allows for the creation of digital twins and eventually *Smart Factories* [54] through the creation of Cyber-physical systems (CPS). CPS can be defined as "a system comprising a set of interacting physical and digital components, which may be centralised or distributed, that provides a combination of sensing, control, computation and networking functions, to influence outcomes in the real world through physical processes" [21]. The implementation of sensors in the whole value chain, from the factory to the end customer, enables comprehensive visibility of the full environment, providing a general overview of the value-chain and identifying efficiency and probable failure points. Furthermore, such technology may also allow for automatic control, with artificial intelligence systems [49], which could alleviate the pressure of the hierarchical level of the industry [54], as well as manual remote control of the environment. The foremeantioned processes of automatic and manual control and visibility across the environment are supported by Industrial automation & control systems (IACS or ICS) and Supervisory control and data acquisition (SCADA) [21]. Figure 3 presents an example implementation of an ICS.

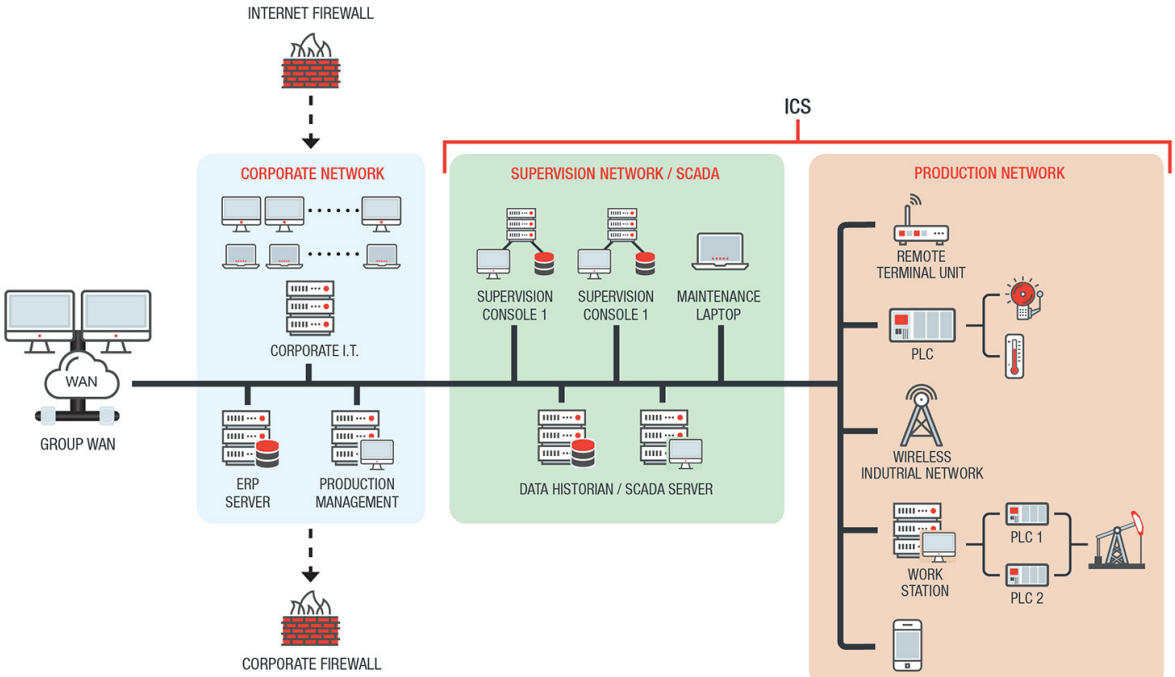

Figure 3: ICS example implementation overview, by Trend Micro [2]

The IIoT vision is one where smart connected assets operate as a part of a larger system that elevates manufacturing in smart factories [55] with or without human intervation [56], allowing for an network of *things* that improves visbility, automation and efficiency, functioning as one of the pillars for the fourth industrial revolution, knonw as Industry 4.0.

1.4 Big Data: myths and facts

Big Data has been a recurring subject between researchers and companies. While there are some references to this contested subject as early as the mid nineteens [57] with some contributions by NASA scientists [58], some researchers defend that this has been building up since 1944 [59], hence, this trend has become more and more popular over the years due to the requirements that emerged with the evolution of many environments like those of IoT [60]. Nonetheless small and medium enterprises have chased this new trend seeking to grasp all the advantages advertised [27]. There are many cases in which companies, after investing a large amount of money in a Big Data solution, were not successful in getting their investment back. This might be a result of a lack of understanding of what this concept represents and how it can be implemented. To mitigate such a possibility a clear definition of the term is required.

Typically, big data consists of datasets whose size and structure exceed the capabilities of traditional computing tools (databases, software, etc.) for data collection, storage, and processing in a reasonable amount of time and effort. Additionally, data can be structured, semi-structured, and unstructured which makes it impractical to manage and process them effectively in traditional ways [61]

It was Laney who was one of the first to introduce the concept of Big Data ”Vs”[62], which are widely used among researchers[60, 61, 63].

Accordingly, the so-called Big Data Vs stands for:

- Volume - This represents the size or magnitude of the data. In Big Data's context, the numbers for this metric are around multiple terabytes and petabytes. As of today, reports show that 4 petabytes of data are created daily on Facebook. By 2025, it's estimated that 463 exabytes of data will be created each day globally [64]. A large slice of this number is coming from IoT environments, which were expected to have doubled since 2016, multiplying the generated data.
- Velocity - Representing the rate at which data is being generated, having a substantial impact on real-time analytics. In Big Data environments data is constantly coming in, especially from IoT scenarios in which sensors are recording values every millisecond.
- Variety - A also well-known characteristic of Big Data is the format in which data comes in. The produced data contain logs and sensor data (e.g., from the Internet of Things), low-level customer behavior (e.g., Website click streams), social media, document collections (e.g., email and customer files), geo-location trails, images, video and audio and other data useful for integrated analyses[61]. These data can come in three forms: structured, semi-structured, and unstructured. Structured data represents the most reliable sources of data like databases and CSV files, which have a strong structure that needs to be respected. Secondly, semi-structured data is data that is bound to a defined structure that can be understood at the human or machine level, but that structure is somewhat malleable. XML and JSON are some examples. Lastly, unstructured data is data that is not restricted to any structure and is highly malleable. Social media posts text, images, video, and audio are good examples of this, and reports reveal that it represents the larger slice of volume in Big Data [60].

After the community well accepted these characteristics some more "V"s were introduced: IBM introduced "Veracity" representing the unreliability inherent in some sources of data; "Variability" was introduced by SAS referring to the variations of data flow rates; Oracle introduced "Value", representing the "low-value density" of Big Data, which means that there is considerably low value in raw produced data compared to its volume, however, significant value can be created by analyzing a large number of data [60].

These characteristics are key to understanding what classifies a real Big Data use case. Without them, companies build inefficient solutions that require a monumental effort to extract some amount of useful information [27]. Many are real-life scenarios in which Big Data architectures have drained vast amounts of resources only to fail at the end [65].

1.5 The need for data quality

Multiple studies show that data quality has been a key factor in determining the success rate of data platforms [66, 67, 8, 68]. Assurance and visibility of data quality can provide many advantages in the project, allowing the business to understand their data and how well it fits their patterns.

A simplistic definition of data quality states that it is the level of "fit-for-use" of data. This means that data needs to be relevant to the use case that is required, which requires both accuracy and freshness. The accuracy dimension of the data defines how well the data consumed actually represents the real-life conter-part. If accuracy

is low, data is not reliable because the system is being fed data that do not correctly depitcs the actual situation, thus rendering the solution useless. Freshness is the other main dimension of data which depicts how recent is the data, representing the interval from which data was generated to the actual consumption. If the interval is within a defined tolerance then data is usable, otherwise, data cannot be relied on since it does not fit into the time-sensitive requirements of the business. In today's world, data processing tools are either fast or irrelevant.

The lack of data quality management in a platform can lead to revenue lost, company reputation and even legal risk [3].

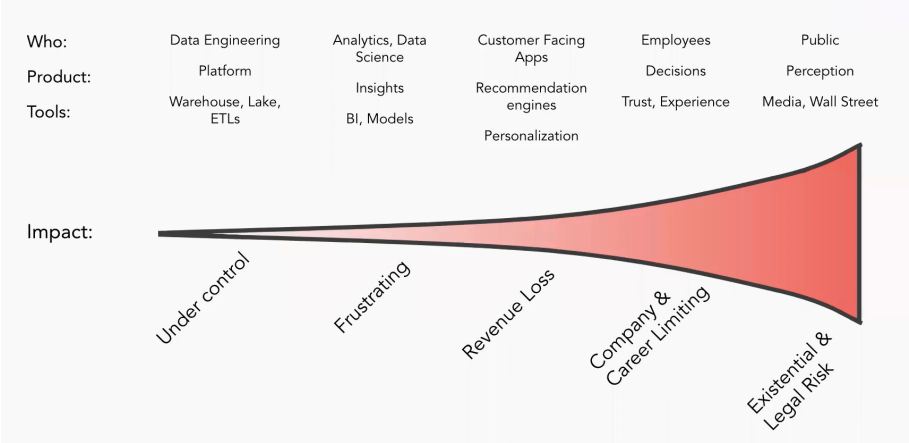

Figure 4: Data downtime consequences by Moses [3]

In Figure 4, a representation of data downtime over time is depicted, representing the risks that the company is exposed to when it doesn't have a data quality management tool. A data downtime occurs when data being fed into the company's system is not healthy and lacking in quality. This scenario is particularly relevant because often the data engineers who are responsible for the quality of the data are the last to know that the data that is being fed to the system is not in proper shape. This is where the new concept of data observability is introduced to aid the data quality management task in a company. In this concept, a set of monitoring categories for the data are proposed. These categories are so that it can be assessed over time as to its quality and usability, thereby providing visibility when data quality drops below a certain standard and is no longer trustworthy.

Data quality definition is not strictly confined to the health state of the data within the system, it also extends to data contextualization characteristics [69]. Data needs to be understood so it can be repurposed, integrated and accessible, or else the data does not live up to its full potential. Contextualization through semantics elevates the data, depicting a more accurate picture of the world. Spatial and temporal correlations of the data, periodicity, and data variations are all examples of this contextualization [8].

1.6 Motivation

The fourth industrial revolution has been proven to provide incredibly effective results for companies that are willing to restructure their manufacturing process. The industrial internet of things has played a key role in this transformation by providing real-time information to all interested parties, such as business owners, workers, and machine processes. However, on the one hand, this paradigm offers a voluminous amount of raw data from

sensors, machines, transportation, people and processes, but on the other hand such data has consistently proven that a scalable and quality-centric data management tool is needed to handle it. Most data management tools today, do not offer direct data quality capabilities, rendering implementations in these environments inefficient.

A methodology that is based on the state-of-the-art data management tools, with the main data quality principles defined at its core, and follows the existing reference architectures in the space could lead to an easier transition to the new industrial revolution for medium small and micro enterprises.

1.7 Objectives

The purpose of this work is the development of a reliable and scalable methodology, that can be used to face the highly complex data challenges that can be found in the Industrial Internet of Things applications, such as data quality issues, lack of data context visibility and understanding, and scalability. Data quality and context management through semantics needs to be the key focus of such a methodology, as well as observability patterns and FAIR data principles. Furthermore, decentralized components, centralized infrastructure, resilient, accessible, and observable data and metadata repositories, data ownership information, and scalable data transformation tools are also specific requirements that must be implemented to assure the required capabilities.

The reference architectures of today [70, 71], must also be taken into account, so as to create a design that is based on successful use cases in IIoT environments, incorporating features such as modular design, horizontal scalability, adaptability, flexibility, and performance efficiency.

This work also aims to explore the data and data quality definitions, in order to understand their role in the industry. In addition, it aims to understand how quality can be achieved through paradigms such as data observability.

Lastly, this paper must also explore the state-of-the-art data management tools, with the purpose of helping establish a robust methodology capable of dealing with the data requirements above mentioned.

1.8 Structure

Section 2 explores the state-of-the-art data management tools and the differences between them. They are presented in the order that they were studied and implemented over several iterations of the methodology. Each explains the reasons why they were considered for the implementation and how it translated to the final solution.

In section 3 the importance and challenges of Data Observability is discussed. The topic explains how different concepts relate to one another and how data quality is at the core of the discussion. There are also introductions to the FAIR data principles, the data definition, categories of data observability, and the data act.

The architecture of this project is described in section 4, presenting the main ideas behind the structure and the designed components. These components and respective technologies have their place justified in the overall architecture and what benefits they provide. The inner workings of the technology and its contextualization are presented accordingly.

Section 5 presents the use case that was chosen to test the methodology. Analysis of the use case and its data is performed in order to determine how well it matches the target environment. In this section are also provided examples for each section of the architecture as well as the overall flow of data inside the architecture.

Finally, in section 6 the proper conclusions are presented, along with the future work.

2 Architectures for data management

The interest in closing the gap between data and knowledge within IIoT has driven the design of multiple data management architectures and patterns from early stages. Such designs were iteratively improved along the way, splitting into very different architectures for very different use cases [72, 73]. These architectures have evolved and can even be hybridized in certain environments. The main architectures that are most popular today are to be presented and discussed in the following section.

The data collected from IIoT environments exhibits some Big Data characteristics that need significant effort to be effectively utilized [73]. For this reason, in order to be successful in designing an IIoT data management and gov- ernance solution it was necessary to analyse existing state-of-the-art architectures with focus on capabilities to close the gap between data and knowledge. Due to the significant volume and heterogeneity of data within these environments, the first strategy analyzed was the implementation of a Data Lake. Although the Data Lake architecture does offer some interesting features that can be used to construct an efficient data management tool, there are some requirements for data quality that this centralized data storage couldn't easily fulfill. In order to tackle these challenges such as metadata management, the Data Fabric architecture was examined. The final iteration of the data management system was an evolution into a Data Mesh platform. This step emerged from the data quality benefits that handling data as a product can offer. This path of iterations was driven by the search for a data management architecture that prioritized data quality aspects of the data. The first step was to consider a large centralized repository that handled all the data within the environment, allowing for data quality processes to be applied to the data within. Then a series of services were created to interact with the consumed data and context metadata. And finally, the last iteration allows for multiple features that improve data quality, interoperability, observability, reusability and visibility.

2.1 Data Warehouse

One of the first data management systems to appear within the industry was the proprietary enterprise data warehouse. This architecture was widely accepted in the past and it still can provide adequate results in the present [74, 75, 76], it has been losing traction as a data management structure for more complex use cases such as IIoT [77, 72, 78], since its focus on answering a priori questions, that force a delimited structure that must be followed, and changed if something within the environment evolves and it does not offer real-time capabilities needed in such cases [79]. ThoughtWorks has put implementations of this architecture on hold since 2014, stating that it has a failure rate higher than 50%, due to demanding up-front data modeling that takes years to develop [80].

This architecture was designed primarily for easy query and analysis of transaction data from one or multiple sources, allowing decision-makers to effectively support their choices about the business. For these reasons, the data warehouse contains historical data, that will be presented to very few users for data exploration, being almost exclusively a read-heavy architecture, not fit for real-time environment management.

The data warehouse architecture has a set of main characteristics that are to be noted, such as being subject oriented, having integrated data, having a time variant, and being non-volatile [81]. Subject-oriented design refers to the fact that data warehouses need to be designed around a specific topic. This improves the query efficiency of the data and user understanding. However, this attribute requires the majority of the data to be discarded,

hindering a more broad view of the data on-demand. The integrated feature of the architecture touches on the necessity of integrating multiple data sources in order to empower exploration. Nonetheless, this integration must be carefully considered before the data warehouse can be implemented effectively. The time-variant attribute of data warehouses refers to the ability to retain historical data to be queried. Lastly, the non-volatile aspect of this architecture refers to the fact that when data is imported into the Data Warehouse it will not be changed by operational and transactional data. The data inside the Data Warehouse can only be rewritten or added to through large computing jobs, commonly known as ETL.

The data warehouse typical architecture and usage is depicted in figure 5.

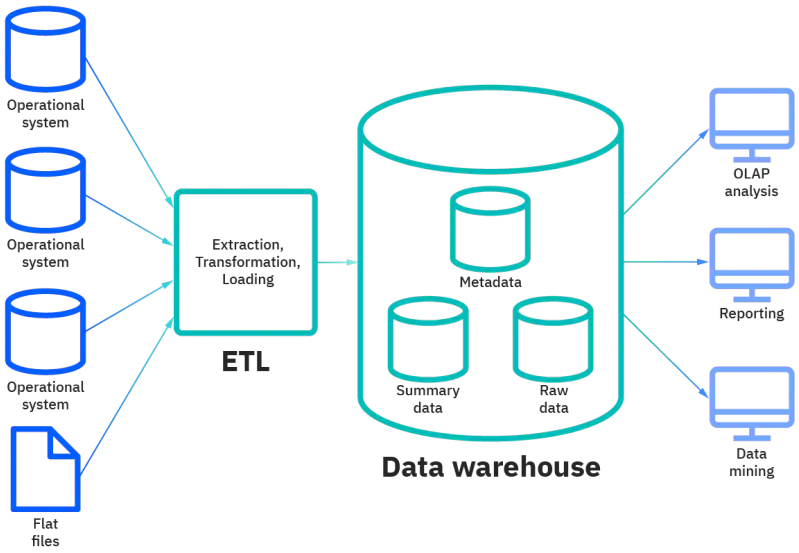

Figure 5: Data Warehouse architecture by IBM [4]

2.2 Data Lake

The Data Lake is a massive centralized and scalable repository that contains a vast amount of data in its original format and/or the result data from transformation to the native data [82] which needs to be analyzed by business experts and data scientists alike [83]. Data Lakes have the capability of processing voluminous and quickly generated unstructured data [61]. The Data Lake should be architected to be divided into sections, which Bill Inmon refers to as data ponds [27] and some other researchers refer to as zones [84]. As a result, data separations facilitate data lifecycle management and, therefore, data quality. Having the Data Lake separated into different sections allows for a more scalable solution since each section can grow separately. As an example, the raw data station and its process pipeline can be scaled to boost a fast data application, providing quick data insights and sacrificing the processed data level; this can be extremely useful depending on the context. Using this architecture, an archival data level can be deployed [27], so old generated data may be stored in a cheaper storage system, allowing the data to be kept as long as possible.

While this architecture was born as a result of the necessity left by data warehouses [3], it has been at the center

of attention in the Data Engineering community since. While it has been praised by many, there are many reports in which this architecture has failed miserably, creating monumental data silos. That allowed some products to grow, such as Delta Lake, to respond to such necessities. This most recent iteration of the Data Lake architecture aims to provide more functionality to the Data Lake, contributing features such as stream and batch processing unification, cloud-based storage, and real-time data availability.

Although this architecture is often seen as an alternative to the Data Warehouse approach, they are really designed to meet different types of business requirements. As mentioned in the previous section, data warehouses are highly specialized in one particular business domain, having a strong schema associated with it for fast data querying. In addition, data needs to be cleaned and enriched before it can be stored in the Data Warehouse. On the other hand, Data Lakes aim to retain as much information as possible in its raw format. As well as reducing the delivery time for data analytics and data staleness, since data does not require a definite structure to be able to be stored, a raw data format requirement for Data Lake is the ability to meet future requirements. By having all data available in one repository, users are able to query the past history to understand if such a requirement would be feasible. These cases are rarely taken into account by data warehouses since they need to filter and exclude data in order to provide a better performance based on their present requirements. Amazon Web Services (AWS) provides a table that highlights these characteristics [16].

Characteristics	**Data Warehouse**	**Data Lake**
Data	Relational from transactional systems, operational databases, and line of business applications	Non-relational and relational from IoT devices, web sites, mobile apps, social media, and corporate applications
Schema	Designed prior to the data warehouse implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
Price/Performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data Quality	Highly curated data that serves as the central version of the truth	Any data that may or may not be curated (ie. raw data)
Users	Business analysts	Data scientists, Data developers, and Business analysts (using curated data)
Analytics	Batch reporting, BI and visualizations	Machine Learning, Predictive analytics, data discovery and profiling

Table 1: Data Warehouse compared to Data Lake by AWS [16]

As it can be seen in Table 1, while Data Lake tend to fit more broad scenarios, Data Warehouses still may find a place in data analytics in terms of being capable of analysing very specific scenarios.

The use of Data Lakes has tremendous benefits. The data can be accessed faster, allowing for faster analytics. It is also a centralized architecture in which can facilitate the discovery of data across the organization, empowering

reusability and interpolation, enabling for high value use cases such as integrating CRM data with marketing, social media analytics, buying history, and website engagement. This allows the business to elevate the customer experience to a new level. Another result of these features is that scenarios such as IoT can produce an enormous amount of data to be analysed, and potentially find ways to reduce operational costs, and increase quality.

Having this large amount of data stored also allows for users to take advantage of it to test new features with real raw data, lowering the implementation time of business features.

However, these features often introduce difficult challenges. The main challenge of Data Lake implementation lies in its most distinctive characteristic: raw data ingestion. Since the Data Lake architecture is capable of high throughput ingestion and storage of data in its raw form, some companies have fallen into the mistake of ingesting every single piece of data on its newly implemented Data Lake [27]. Such practice may transform the Data Lake into a Data Swamp. If ingested data is not properly governed, labeled, and carefully integrated into the Data Lake, its use may come at a very high price. In the absence of data quality practices such as semantic context and data cataloging, acquired data quickly becomes difficult to discover and impossible to integrate due to a lack of trust and security in the acquired data.

With the increase interest in the Data Lake architecture, some companies now offer cloud solutions for this architecture, such as AWS, Microsoft Azure, Cloudera, Databricks, and Google Cloud. AWS is one of the main providers that has created some features to respond to the increasingly complex demands of such use cases. For example for data that is most frequently accessed (also known as hot data) a SSD can be used to store and cache it for better performance, and for other data that is not accessed as often features such as AWS Glacier can be used to create archival storage, lowering the storage costs of the implementation [85].

A new interesting iteration over the Data Lake architecture has been the Data Lakehouse [85], which proposes the to take advantage of the main benefits of Data Lake, as well has having the query efficiency of Data Warehouses. This new movement proposes to use the main components of a Data Lake such as low-cost object store for storage, but on top of that introduce an metadata layer, that allows for data specification and categorization, with techniques such as versioning, statistics and semantics. The other component that was introduced in this iteration is to use a caching system, so hot data is quickly available on demand so it can achieve a SQL-like performance, and at the same time keeping an large amount of raw data for it b used in more extensive processes such as machine learning mode training. This architecture presents some similarities to the Data Fabric architecture, by constructing a virtualization layer on top of its storage layer, to facilitate the access and visibility to data, and potentially increasing the overall performance.

2.3 Data Fabric

This novel architecture emerged from the need for ingestion, governance, and analytics of the uncontrolled growth of data, which provides features that can streamline the Big Data platform [6]. The solution proposed aims to assist in end-to-end data integrations and data discovery through data virtualization, with the assistance of APIs and data services data across the organization allowing for better discoverability and integration within the environments even if they are in old legacy systems [86]. Instead of proposing a completely revamped design for data management, the data fabric simply seeks to create a virtualization layer on top of each data stage, such as transformation, storage, and serving, creating a global uniformization to facilitate data access [87].

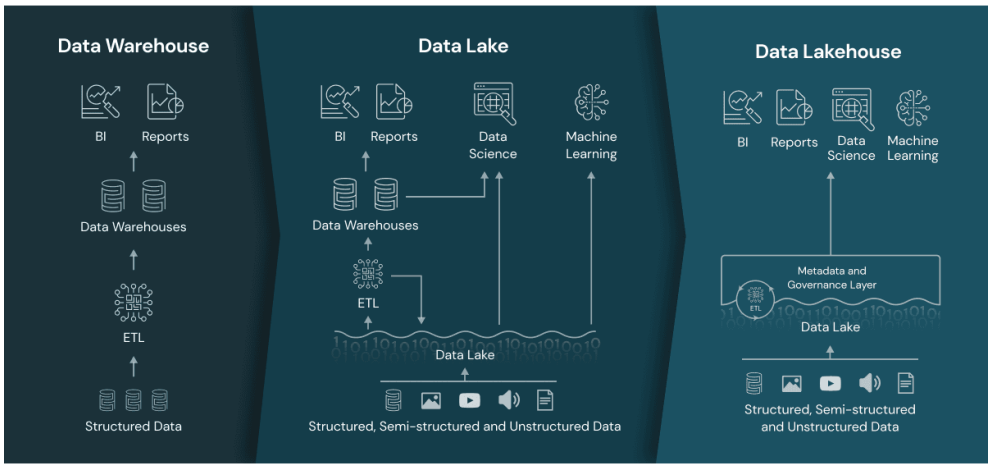

Figure 6: Data Lakehouse architecture compared to Data Lake and Data Warehouse by Databricks [5]

This movement focuses on automatizations in areas such as ingestion, curation, and integration of diverse and heterogeneous data sources across the organization, with the goal of augmenting the capabilities of data analytics [6].

The interaction with data in a Data Fabric platform is centralized and abstracted from the tools that store and process the data, which aids in processes such as:

- Security - Since data governance is done in a centralized space, it is much easier to manage each user's access to the data.
- Data discovery - The virtualization service enables the user to discover data from all available data sources, regardless of the tool or location.
- Self-service for business users - These users can explore the data in a single place instead of having to search within specific divisions or be limited by data warehouse implementations.

Researchers present the multiple virtualization layers of the architecture as data ingestion, processing and persistence, orchestration, data discovery, data management, intelligence, and data access as it can seen in Figure 7 [6].

2.4 Data Mesh

Data Mesh aims to restructure the organizational structure around Data utilization, following the concepts laid out by Domain Driven Design [88] to introduce more ownership over the data, thus creating less friction when trying to produce valued information and mitigating the problems encountered in Big Data Lakes and warehouses[72].

In order to face the challenges of extracting information and inter-divisional problems, this approach proposes a paradigm shift comparable to the microservices architecture in software architecture, focusing on treating data as a product, and creating data teams to handle the whole subset of data belonging to the business domain, rather than dividing them into teams for the different data processes, such as collection, transformation, and provisioning.

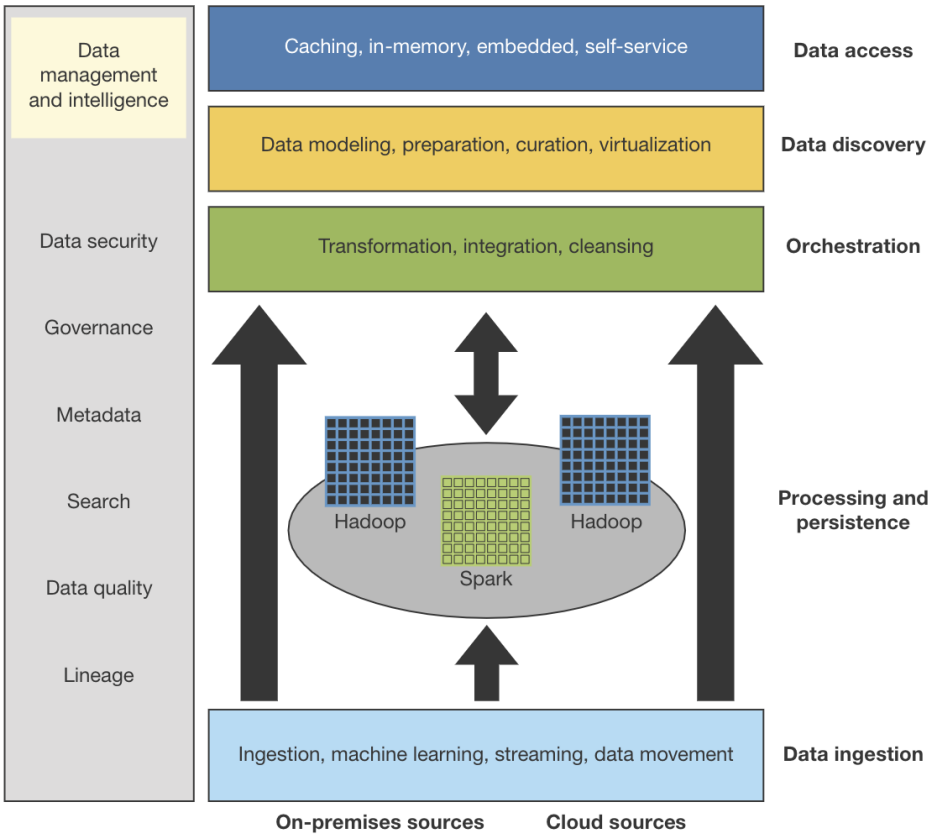

Figure 7: Data Fabric architecture by Noel Yuhanna [6]

This allows for increased ownership of the data itself by the teams, thus leading to more agility when it comes to producing knowledge [7].

This paradigm does not aim to replace any of the architectures for data management, it instead aims to re-structure the organization around it, creating bounded contexts around the data management. This will allow each data team to use the preferred data structure for the specific domain. With these changes, teams are also more incentivized to maintain data quality. This is because they are the owners of that domain of data that will be served to other data teams and customers [89].

Researchers propose a set of principles to guarantee correct Data Mesh function and thus improved data quality. The principles are domain-oriented decentralized data ownership and architecture, data as a product, self-serve data infrastructure as a platform, and federated computational governance.

The first principle focuses on the data ownership aspect of the Data Mesh, and follows the author description of ”decentralization and distribution of responsibility to people who are closest to the data in order to support continuous change and scalability” [7]. To create scalable tools around a Data Mesh infrastructure it is necessary to analyze the development process across the organization to fully identify the different bounded contexts.

The principle of data as a product focuses more on the data itself, striving for better data quality, accessability, discoverability and understanding. The core idea of the principle is to treat data as the product of a process, and its users as customers that need to be satisfied. The components that are a part of this principle are code, infrastructure, and data and metadata. The code includes data pipelines, to consume, produce and compute data,

as well as s to access data and metadata repositories. The infrastructure part of it aims to manage the deployment and management of such processes. Lastly, the data and metadata are the representation of the actual final product and what must be shown to the customers. A visual representation of this principle can be seen in Figure 8

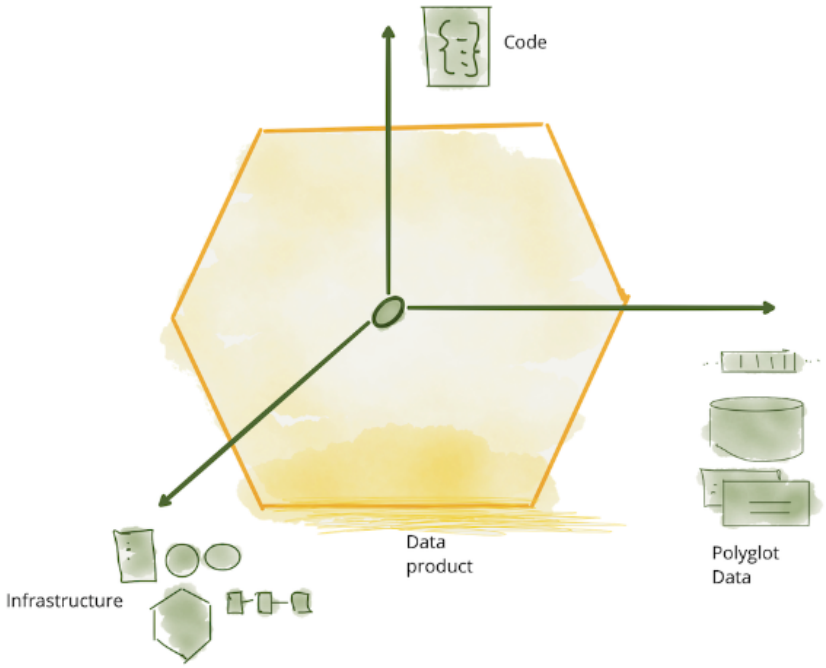

Figure 8: Data as a product representation by Dehghani [7]

The next principle of the Data Mesh is the self-serve data platform, and it focuses on ”scalable polyglot data storage, data products schema, data pipeline declaration and orchestration, data products lineage, compute and data locality” [7]. This principle aims to remove the challenges of having to replicate different data infrastructures to every context within an organization, appealing for a use of a decentralized architecture in a centralized platform.

Last but not least is federated computational governance. This emerged out of necessity for a grouping of distinct data products with distinct lifecycles that were developed and distributed by probably distinct teams. However, for the majority of use cases, these independent data products must work together in order to gain value in the form of higher-order datasets, insights, or machine intelligence. This requires the ability to correlate them, create unions, find intersections, or perform other graph or set operations on them at scale. A governance paradigm that includes decentralization and domain self-sovereignty, interoperability through global standardization, a dynamic topology, and most crucially automated decision-making by the platform are necessary for any of these processes to be feasible in a data mesh implementation. The author refers to this process as federated computational governance [7].

Domain-oriented decentralized data ownership and architecture	So that the ecosystem creating and consuming data can scale out as the number of sources of data, number of use cases, and diversity of access models to the data increases; simply increase the autonomous nodes on the mesh.
Data as a product	So that data users can easily discover, understand and securely use high quality data with a delightful experience; data that is distributed across many domains.
Self-serve data infrastructure as a platform	So that the domain teams can create and consume data products autonomously using the platform abstractions, hiding the complexity of building, executing and maintaining secure and interoperable data products.
Federated computational governance	So that data users can get value from aggregation and correlation of independent data products - the mesh is behaving as an ecosystem following global interoperability standards; standards that are baked computationally into the platform.

Table 2: Data mesh principles by Dehghani [7]

3 Data Observability Challenges within IIoT data management

There have been significant efforts to create and iterate data management architectures to close the gap in the level of knowledge that can be extracted from raw data. However, all of those architectures have faced similar problems when formulating their solution [90, 28]. The problems around data quality that were focused on with this approach were traceability, trust, fit for use, context and semantic value, interoperability and reusability. Different strategies can be used to formulate solutions to these problems, and the prime strategy was the implementation of strong data observability practices.

3.1 Data observability

Derived from the original concept of system observability, data observability has followed the same practices that made systems successful in their monitoring practices.

Instead of tracking logs, traces, and metrics as is usual for system observability, data observability practices aim to monitor other concepts such as freshness, distribution, volume, schema, and lineage to prevent downtime and ensure data quality [3]. This has become an essential requirement within data management applications to ensure availability and robustness.

Each dimension of data observability aims to enrich data quality in different ways. **Freshness** seeks to understand how up-to-date the ingested data is, as well as the frequency at which it is updated. It is imperative to consider this dimension when designing a data management system, so it can determined if the data consumed is stale and prevent becoming irrelevant in the IIoT world. **Distribution** is a dimension that establishes the limits of data values, defining the accepted values that reading can have, and defining outliers and possible flawed data and sources. **Volume** refers to the completeness of the data, identifying possible data shortages and sources that stopped sending data downstream. **Schema** monitoring keeps track of data structure definitions and changes. This is particularly relevant due to the importance of interpolating multiple datasets to produce more valuable knowledge, implementing one of the most important FAIR principles [91]. **Lineage** is the last dimension that observability focuses on but it's one of the most critical since it allows us to have full visibility of which transformations data has been through since its conception, allowing us to identify possible breaking points and which systems might be impacted.

The dimensions of data observability allowed for some of the challenges to be tackled in an efficient way. Freshness increased data quality in the fit for use and trust dimension. Distribution and volume improved context value and trust in data. In addition, schema monitoring and lineage allowed for better context value, traceability, and data interoperabilty.

When designing an architecture that aims to manage data produced from an IIoT system, it's crucial that these dimensions are considered in the design. This allows for major data quality enhancements and streamlines the placement within the FAIR framework.

3.2 Data (re)usability principles - FAIR principles

As mentioned before, many data management projects fail because they fall short of their ultimate goal of extracting knowledge. This has led to more advanced patterns and principles being a prerequisite when designing an

effective architecture. Such principles try to tackle the problem of ungoverned metadata that can lead to data that is challenging to read, discover, and process, requiring an immense effort to extract valuable information.

Recently, researchers have published the FAIR principles [91] to help design a data management platform. These principles emphasize the capacity of computation systems to find, access, interoperate, and reuse data with minimal human intervention. They enable scale with the increasing volume, complexity, and velocity of data [91].

A successful data management strategy is not a goal in itself, but rather a conduit for innovation and discovery within such structures, and the authors present four fundamental principles that guide their approach.

The first concept introduced in the FAIR principles is **Findability**. Data should be visible and easily discoverable within the platform. Metadata context for these purposes is essential since it can be used to allow users to easily locate related datasets. This will allow for more data contextualization and thus a higher potential for knowledge creation. The next concept is **Accessibility** which focuses on the stage after the data has been discovered. Data needs to be accessible and retrievable. Often paired with authentication and authorization, documented and uniformed through the platform. The third principle is referent to one of the most relevant requirements to use data. The **Interoperability** principle is centered on the necessity of integrating different datasets. The key conduit for this process is context metadata and data semantics, allowing us to connect different datasets through common properties and similar circumstances. The process of implementing this principle should be looked upon with special attention since the interpolation of data is crucial for information and knowledge creation [92, 78]. The last principle presented by the authors is described as the FAIR principle's ultimate goal, which is **Reusability**. And to achieve this, both metadata and data are required to be well described so they can be replicated and reused. Keeping a history of data and data changes is also a critical part of this principle, in order to be able to reprocess untouched data.

The authors propose a set of techniques to achieve these principles, such as having metadata indexed in a searchable resource, having it accessible via a standard communication protocol, and metadata representation using a standard language so it can be acted upon.

These principles led to the development of a service to host context and semantic metadata. The use of such a service enabled continuous data quality improvements through streaming data processes and data observability practices. This newly introduced service also allowed for easier data interoperability and reusability since it hosted all the metadata needed for such use cases.

In order to understand which metadata was kept in the service, first the concept of definition of data must be introduced.

3.3 Definition of data

The Definition of Data (DoD) is the first problem that arises when discussing IIoT data management platforms. Data and data quality needs to be defined so the designed solution can fit the needs of the environment.

The need for this step is paramount, not only for modeling and integrating data, but also for establishing common concepts to which the entire business chain can refer, thus ensuring resilient communication across the organization. When these concepts are implemented into the application contextualization system, a common understanding and development platform for the entire organization can be created.

The process of data definition must be done iteratively across all the business layers to ensure scalability. Some

of the data definition concepts may rely on technical terms that are very specific to the business as well as some more broad definitions that can be used across multiple projects. For this reason a definition of the concept of data and data quality in the IIoT environment must be provided.

The next two sections will be exploring the definitions of data and data quality respectively. Although this division is being made, there are characteristics that can be applied to both concepts and some researchers have used them in different ways [8, 93, 94].

It's imperative to understand the data within any data management use case, and IIoT is no exception. The particular case of the Internet of Things contains some characteristics that should be taken into consideration when designing a solution to cope with its challenges.

Laura Sebastian-Coleman [69] defined Data as "abstract representations of selected characteristics of real-world objects, events, and concepts, expressed and understood through explicitly definable conventions related to their meaning, collection, and storage". Lina Zhang [93] has proposed a derived interpretation applied in the IoT environment by stating that "from the perspective of data semantics, the data in the IoT can be the underlying raw data and the high-level generalized data". Sunho Kim [95] has identified types of data that can be found in an IoT environment, which include:

- Sensor data is the actual values that are recorded by the sensors in the environment. Other than the values this data must also contain a timestamp from when that reading was made in order to be properly analyzed.
- Observed metadata describes the behavior of the sensor data. This can include information about the rules for data value changes, and the range of data value changes across a time period. Such metadata is dynamic and should be continuously calculated through the data lifecycle.
- Device metadata refers to the information around the device that records real-life readings. This metadata often includes sensor installation data, location, model, precision, value range limits, and manufactories, among other information. This data is commonly defined when the sensor is installed and does not change often, in contrast to the observed metadata.
- Business data represents the category of data that is related to the business section of the environment. This can include information such as service history or warranty status from business systems. This data source can reveal some interesting data interpolation opportunities.
- External data is often described as the data that comes from external systems. Some examples include energy prices, traffic, weather, or social media.

  Technical data is the dimension that englobes the technical specification of the data. This can include information such as data standards and data structure.

These data categories allow for a better understanding of the IoT environment and the data that can be generated within.

These descriptions and data categorizations suggest that the definition of data in the IoT environment englobes the sensor readings and context metadata around the sensor reading, being complemented with representations of other business aspects.

This definition can be further understood by classifying the types of data into a hierarchy.

- Streaming or continuous data
  - Sensor
  - Business
  - External
- Context Metadata
  - Device
  - Observed
  - Technical
- Historical data

The streaming data includes sensor readings that are continuously being recorded within the environment, projecting a replicated portrayal of the real-life environment. The business data is another instance of streaming data, representing other events higher up in the value-chain. And finally, external data refers to all other data collected from external systems, such as weather, traffic or energy costs.

The second categorization in this hierarchy is the context metadata. This field contains the contextualization of data which allows for better data reutilization, accessibility, integration, visibility, and, ultimately, better quality. The context metadata is divided into device metadata, observation metadata, and technical metadata. The device metadata aggregates all the context around the device that did the reading. This can include things such as owner, installation date, room, or reliability. On the other hand, observation metadata refers to metadata that is associated with the reading itself, containing attributes that contribute to the distribution and lineage capability of data observability. Accuracy and outlier information are some of the example fields that can be found in this category. Lastly, technical metadata combines information about the technical aspect of the data, such as where it was stored or schema classification.

The final stage in the data hierarchy is the historical data. This category gathers previously collected data that can be used for future analyses, creating a repository that can be queried for various purposes, such as audit or model training.

Karkouch [8] has identified IoT data quality characteristics and challenges that must also be addressed.

- Uncertain, erroneous, and noisy: There are several factors that threaten data quality in this dimension, such as deployment scale, network, sensor calibrations, or fail-dirty. Karkouch has presented these factors and how they relate to architecture layers in a typical IoT solution in Figure 9
- Voluminous and distributed. Typically there is a multitude of sensors deployed at any time constantly sending data. This can overwhelm software systems.
- Smooth variation. A minor fluctuation (or none) occurs between two successive timestamps for many physically observed variables (such as temperature).
- Continuous. Related to the smooth variation and voluminous and distributed points, data is constantly being generated by the sensors reading real-life metrics.

- Correlation. Datasets generated from sensor values frequently contain underlying correlations. The correlations between the data might be either temporal, spatial, or both.
- Periodicity. Many phenomena-related datasets may have an underlying periodic pattern where the same values repeat at regular periods.

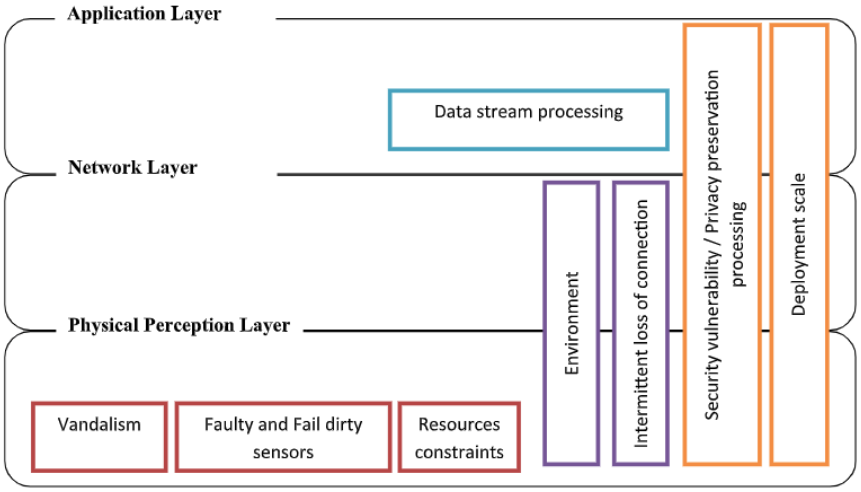

Figure 9: IoT data quality threats by Karkouch [8]

3.3.1 Data quality definition

There have been some definitions of data quality which are focused on the fact that it's "fit for use". The ISO 9000 standard [96] declares quality as "the degree to which a set of inherent characteristics fulfills a need or expectation that is stated, generally implied, or obligatory", Juran [97] defines it as "data are of high quality if they are fit for their intended uses in operations, decision making,and planning", and Wang [98] also supports these definitions by writing "data thatare fit for use by data consumers". Karkouch [8] extends the data quality definition to the IIoT domain, declaring that "for the IoT, data quality means essentially how suitable the gathered data (from the smart things) are for providing ubiquitous services for IoT users".

ISO 8000-8 expands the data quality definition into three dimensions. The first dimension is syntactic quality, and refers to the extent which data conforms to the syntax previously defined, and is according metadata specification. Semantic quality is the second category, and it refers to how well the data corresponds to the content that is representing. Lastly, pragramatic quality is referent to the measuring of usage of data to a specific objective.

Following these definitions, additional aspects of data quality can be identified:

- Data is fit for use: Data follows a specified schema that is defined in metadata.
- Data is correct: Data correctly represents the real-world counterpart.
- Data is completed: Data that is collected at a fixed time frame is present any of those collections.
- Data is relevant: The system has the right data at the right time. Data is only relevant if its content can be used for specific use-case and if its timing match its requirement.

- Data is findable and acessable: Data can easily be found and accessed to extract value.
- Data is visible and understandable: In order to extract the maximum value, data needs to be understood. Context over the data is needed in this point. This context must include how the data relates to other entities in the environment, the quality history of it, what is measuring, how reliable is the value, what kind of transformation has data gone through before getting at the current point, and much more. All of this information is relevant to interpolation, reusability, and overall better environment visibility.

In order to better understand the context around data, and how it can help to increase data quality, definition of the contecept was divided into two sections. Each section focuses on concepts inspired by separate dimensions of data observability. The first category contains statistical and computed metadata, which are automatically generated through computing processes along with data processing. This category presents fields that relate to dimensions such as freshness, distribution, and volume. These fields include metrics such as medians, minimums and maximums, outlier percentages, time evolution, missing data percentages, throughput, and much more. The second category centers on the semantic value of the data. The information in this category focuses not only on schema monitoring and lineage dimensions of data observability but also on interoperability capabilities by constructing a semantic net relating entities from the environment to each other. This perspective of data quality is one of the most influential and impactful categorizations within data quality. This is because it strongly contributes to the contextualization of the data within the whole environment of data, improving visibility, interoperability, and discoverability [99, 100]. During the construction of this metadata, questions such as what, when, who, why, and how should be asked, and then, additional values should be continually added to enrich the data context that can be valuable for data utilization. Some examples of semantic metadata in IoT include sensor creation date, sensor brand, sensor expiration date, IP address, battery level, owner, location, sensors in the same room, and much more.

3.4 Data act

There's also been some development in the political spectrum to compel companies to improve their data management so it can be used more ethically [101]. This includes the requirement for users that generate the data to be able to access their data. In addition, the public sector must have access to data generated by it held by the private sector.

Overall, this requires better data management with proper semantics and a platform that allows for high availability and accessibility for multiple clients.

4 An intelligible Data Mesh based architecture for IIoT environments

Considering the challenges that were identified, such as the need for decentralization, data quality metrics evolution, data context as a central component and low latency, an architecture was designed that focuses on metadata management and standardization to enhance data value.

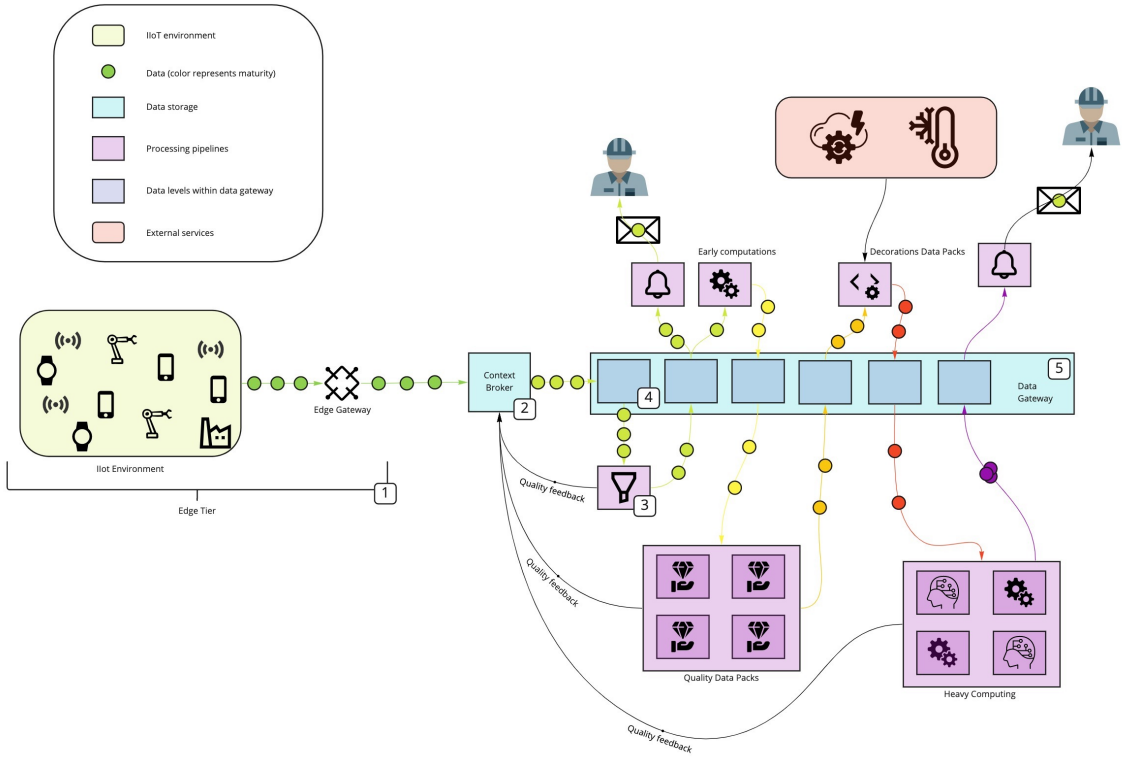

Figure 10: Data Ness Architecture

The entry point of the architecture is the Context Broker. Besides being responsible for receiving all the raw data from the IIoT environment, this component is also responsible for the key aspect of metadata management. When raw data comes from the IIoT environment it is ingested by the context broker and is automatically enriched with the context metadata. The base semantics and context management should be managed beforehand, so the semantic graph can have a wider reach.

The context broker can also function as documentation for all the sensors and relations within the environment, and when a newly added component is integrated, the semantics graph should be updated with the revised values, keeping a realistic view of the monitored environment. This environemnt documentation is used with the aid of shared smart data models.

The component that is responsible for the data storage and delivery is the data gateway. This piece is designed according to the event sourcing pattern, to retain the full history of data, and maximize the interoperability and reusability aspects of the FAIR data principles. Ideally, the data gateway should be decentralized and meant to hold all the data across all the domains and stages of the data lifecycle, providing enough flexibility to satisfy specific business needs, and facilitating discoverability, access, and findability, thus enhancing the other two principles of FAIR data. This design enables the data gateway to be the central point of data access, allowing for processing

pipelines to move the data around, third party projects to use the stored data, and data visualization tools to empower environment observability.

The final model implemented in the architecture is the plug-and-play pipeline design. These pipelines are meant to connect to the data gateway and move data around it, performing all necessary computations in between, so data can be iteratively converted into knowledge. The computations that can take place within a pipeline include, but are not limited to, filtration systems, machine learning model building, automated actions, ETL processes, alerting, and data quality metrics. Among the most significant pipeline types are the data context enrichment pipelines, which will take data from a data stage, add context information in the form of *data packs*, and output the newly computed data back to the data gateway to be used in a more mature data stage.

To ensure data lineage capabilities, all pipelines should mark data with some type of metadata. Such metadata should include information such as pipeline identifier, timestamp that data was consumed, timestamp that data was produced, initial value, and output value. Such data lineage metadata should belong to a shared and common pipeline model that needs to be maintained. This is so the pipelines can be more easily understood so they can be applied across multiple business divisions and data stages. Said model should include values such as pipeline name, description, data input requirements, data output format, input model, and ownership.

The pipelines may output results to a diverse range of destinations, such as:

- Data gateway, in the form of the next iteration of enhanced data, passes data to the next data stage.
- Context broker, with updated data context and freshly calculated data quality metrics. This path is specially significant because it allows for context iterations, allowing for continuous progress in data quality reflecting changes within the environment.
- External services, such as alerts, environment updates, or monitoring systems.

The flexibility of pipeline development allows for abstractions in the form of parametrizable variables, which empowers reusability. Also, pipeline development should aim to create simple and business-focused code, following the single responsibility principle which allows for a lower development cycle, high cohesion, and also increasing reusability.

The presented architecture assures the most significant components of the reference architectures of today [70, 71], such as: context management with device management and defined ontology, data management with ingestion and provisioning capabilities, analytics processes, visualization support, and decentralization.

The presented architecture in Figure 11 is the implementation version of the conceptualization presented before. The role of context broker is played by Fiware Orion Context Broker (1). The data gateway responsibility is attributed to Apache Kafka (2). The processing of data and metadata within the architecture will be done by Apache Beam with Apache Flink (3). As a visualization tool, the choice was Apache Druid (4). For last but not least, it was decided to take advantage of the plug-in capabilities of the architecture and use MongoDB (5) to function as an archival system to store all data that flows through the architecture.

A closer look at these components will be done in the next sections.

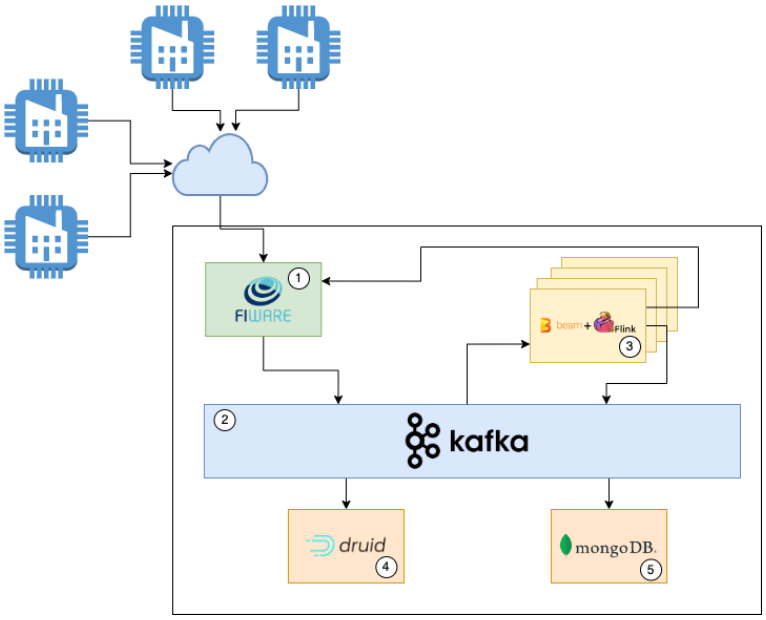

Figure 11: Data Ness technologies

4.1 Context Broker

The first component of the architecture is the context broker, and plays the main part in data quality management. This component is identified by the number 1 in the Figure 11.

The context broker serves as a data management layer on top of the architecture, and receives all the data that comes through the architecture. When this component receives the data, it is its responsibility to enhance it with context metadata. For this reason the context broker will always have the most recent version of the collected data and its context in form of metadata. The static metadata needs to be provided by users when the entity is created in the system. For example, when a brand-new sensor is installed, it needs to be also added to the context broker along with its context, which may include installation date, building room installed, owner, reading type, among others. In contrast, dynamic metadata, which is stored in the context broker, will be continuously calculated by underlying quality enrichment and assessment processes that will provide information back to the context broker. Examples of this metadata are average values, outlier detection, room-aggregation, deviations, and evolution over time.

The design of this component was inspired by a virtualization layer design described in the Data Fabric architecture [6]. This design can also be found in the Data Lakehouse architecture, which proposes a metadata and governance layer to assure data quality [85]. This new layer proposed in the Data Lakehouse was the core evolution from its predecessor, the Data Lake. Last, but not least, FAIR data principles also proposes to have the metadata accessible and searchable indexed resource via a standard communication protocol [91].

This component allows for the findability, accessibility, and interoperability of the data and the context metadata around it to users and processes alike, containing a range of multiple metadata types, from dynamic metadata that is continuosly calculated, static metadata that does not change as often, and tecnhical metadata that may contain information such as schema and locations. This allows for efficient knowledge sharing and data quality management, ultimately leading to a high success factor for the architecture.

The technology of choice for this role was Fiware Orion context broker [102], which may also be referred to as Orion. The reason for this choice was that it's an open-source tool that is at the forefront of context management technologies, being developed by one of the key organisations in IoT and Big Data solutions in the market. The FIWARE Foundation has continually provided open-sourced platforms to enable the development of smart solutions [103], backed by major organizations such as AWS, RedHat, and Telefónica.

4.1.1 NGSI and Orion

Orion is a C++ implementation of NGSIv2 REST API binding developed as a part of the FIWARE platform. First released as an open specification by ETSI, NGSI-LD is an information model and an API for easy use and standard management of context information. Ontology-based NGSI-LD information models use JSON-LD as serialization format for context information. [104]. The NGSI standard is part of FIWARE organization, which is a foundation that provides a set of tools for supporting the development of Smart City, Smart Agri-Food, and Smart Industry applications and is currently being promoted in Europe to be used in large research and innovation projects [105].

NGSI-LD defines three levels of data abstraction: core meta-model, cross domain model and domain specific model [104].

The core meta-model describes the core information that can be published to the context broker. This includes entities, properties and relationships. An entity in Orion represents a physical counterpart in the real world and can include concepts such as a sensor, machine, room, etc. These entities have many properties associated with them, representing the actual values of some concepts. A few examples of these properties would be air quality, temperature, the number of sensors in a room, location, and owner. Relationships are the last type of information that is managed in Orion's context broker. Within the context broker, entities can have zero or more relationships. This defines the contextualization of the entity and its values across the full environment, providing an excellent semantic reference for data quality metrics. One distinguishing feature of the most recent version of Orion is that it uses NGSI-LD, which is modeled using the JSON-LD standard, which means the relationships defined are linked by URL, which enables quick access to information. To maximize the contextualization of data and to be compliant with the Linked Data world, every entity should specify a context binding the data representation to a vocabulary, which can be published to Orion.

Alternatively, the cross domain model focuses more on concepts like time and space. Geolocation, creation time, and time intervals are among its concepts.

Last but not least, the domain-specific model should be defined in accordance with the domain-specific vision of the use case, because it is closely related to the application.

4.1.2 Smart Models

Based on actual use cases and open standards, the Smart Data Models program offers multisector agile, standardized data models that are freely available and openly licensed [106]. The data models available are based on the NGSI standard, and they will accommodate most smart use cases, including smart cities, smart agrifood, smart energy, and smart sensoring.

The available smart data models are currently hosted in Github [107] and are open-sourced and free to use.

The data models are currently supported by initiatives such as Fiware, IUDX, TMForum, and OASC, along with more than 70 companies that have already contributed.

The smart data models project is another initiative led by Fiware. These models allow for a specification of a baseline context-based ontology that fits well in the IIoT space, with terms and properties that correctly identify the environment. Besides, the smart data models were designed to be used alongside Orion, integrating seamlessly with it. Another great feature of this project is that it presents a level of flexibility that allows for extension with another smart data model or a different ontology.

4.2 Data gateway

The data gateway, identified by the number 2 in figure 11, is the conduit where data flows through and reaches all the other components. Designing this component must meet certain requirements, such as being decentralized storage, scalable, and persistent. This is the glue that binds all of the architecture together, as shown in Figure 10 and 11.

The decentralized requirement is the baseline requirement for all components of this architecture, and this component is responsible for storage decentralization, as it needs to distribute data out across different nodes, to ensure robustness [19]. This distribution also needs to assure scalability, in order to handle a sudden increase in traffic. The persistent requirement refers to the necessity of keeping the data for a defined period of time, so it can be accessed after the actual event has occurred.

With these requirements, it is of great importance that this component provides decentralization in a centralized infrastructure, so the data that flows through it can be easily discoverable and accessible.

This component was designed with data accessability and findability in mind, keeping the data in a decentralized component in a centralized infrastructure. This capability allows for multiple actors to have access to the information in a scalable and performant manner. The data gateway was also designed to keep an recent history of data, in order to ensure data replayability and reusability. The scalable feature is particulary important because this component is the conduit of data inside the architecture, in which all the data actors will connect to, such as data transformation processes, visualization tools, and even the context broker.

The technology that was chosen for this component was Apache Kafka. This tecnhology is the state of the art in event streaming platforms, used by thousand of companies to build high-performant data pipelines, streaming analytics, data integration, and mission-critical applications [108]. Another critical characteristic of this project is that is open-source. The choice of this technology was clear and no great comparisons were made due to either the lack of usability of other technologies or lack of scalability required for the use case [109]. Besides Apache Kafka has other features that aided the development, such as Kafka connectors.

4.2.1 Apache Kafka

Apache Kafka is an open-source publish-subscribe messaging system. This technology is designed to have persistent messaging, high throughput, real-time processing, and multiple client support [108]. One of the main characteristics of Kafka is that it is also distributed across multiple nodes.

The Kafka nodes, commonly referred to as Kafka brokers, are servers that run a Kafka instance that can be

connected to, in order to publish or subscribe to a particular segment of data.

Kafka has a specialized set of terminology, which will be defined as it's proven relevant moving forward.

- Kafka broker - Server that runs an Apache Kafka instance, which can be connected to a client, allowing for publishing or subscription actions.
- Kafka cluster - Aggregation of Kafka brokers that connect with each other, providing resiliency and robustness to the service.
- Topic - Aggregated segment of data to which producers or consumers can connect. As a rule, topics hold information in the same format and belong to the same context.
- Partition - The messages sent to a Kafka topic will be mainly a key-value pair, and messages that have the same key will be published to the same partition. The partition is a sub-segment of data inside a topic. Since messages with the same key are persisted in the same partition, the order of those events will be ensured inside Kafka.
- Retention policy - Rule that defines how long Kafka needs to retain messages or events. This rule is defaulted to a time-based retention of seven days.
- Consumer group - Using consumer groups, distributed clients can connect to Kafka without the possibility of duplicate events. Clients that have multiple nodes can share the same consumer group to connect to Kafka, which will result in a single consumer from the Kafka cluster, avoiding data duplication
- Replication factor - Configuration that defines how many brokers the message will be replicated to.

This component follows a leader-follower approach for distributed data. Inside a cluster, a particular Kafka broker will be the leader of topic partition. This broker will be the main source of truth for that data. However, depending on the replication factor of the cluster, the message will be propagated to other brokers. Zookeeper, a node-administrating system, orchestrates this process by keeping meta-information about nodes in a cluster, and determining whether they are leaders or followers, and if replication is possible [110]. A Kafka cluster representation can be seen in figure 12.

One of the main differences between Apache Kafka and other publisher-subscriber systems, such as RabbitMQ or ActiveMQ, is that it's designed to persist the events that flow through it, providing a historical view of past activity. In a Big Data context this could be problematic, since a large amount of data needs to be kept for a specified amount of time, especially if the data is replicated across nodes. With Kafka, however, messages are highly compressed, allowing for space optimization in the cluster.

Another aspect that has elevated Apache Kafka to be the system of choice when designing performance sensitive systems, is that it provides real-time capabilities and one of the highest throughputs when comparing to other tools in a Big Data context [109].

4.3 Processing pipelines

Data processing pipelines, identified by the number 3 in figure 11, are responsible for all types of data modification. This can include but not limited to decorations, filtrations, alerting, machine learning model training, integration with external services, data quality calculations, and ETL processes.

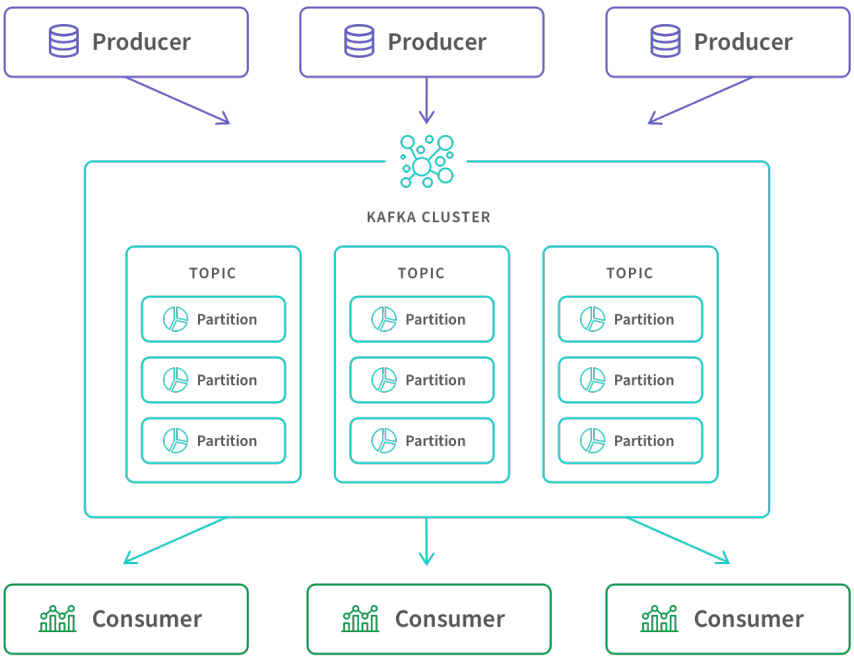

Figure 12: Apache Kafka cluster illustration, designed by Qlik [9]

The necessity of having data iteratively processed according to different requirements led to the creation of this component. This component is the driver that moves the data across the data gateway and context broker. The design of the component was made with the ”plug-and-play” model, to facilitate the development of new pipelines in the architecture. Using this model, the ability to build a new pipeline and connect it to any part of the data gateway without interfering with the already existing pipeline that moves the data allows for a high degree of reusability and accessibility. A further capability of this component is the reusability of the pipelines themselves. This allows users to reuse created pipelines for a different use case simply by changing a set of parameters. Lastly, these pipelines must offer both streaming and batch processing capabilities to fit the different requirements of the environment.

The main input for this component will be the data gateway, but it is not limited to it, as it can connect to other sources to enrich the data. The typical use cases for this are to connect to the context broker in order to get full semantic context of the data or to access external services to get third-party contextualization.

The output of these processes is not as streamlined as the input, since there are considerable increases in use cases, such as quality calculations feedback to the context broker, data decoration results to the data gateway, ETL processes that move the data to a data warehouse, and connection to external services for alert porpuses.

The pipelines are divided into different types, so they can be referenced more easily. The types will be distinguished by their inputs and outputs, data transformation types, and processing classification (stream or batch processing). Examples are going to be provided for each category, to better explore the capabilities of each type.

Context decoration pipelines

These pipelines add information to the data, either from an external service or from another entity within the context broker. The input is typically the data gateway, and then it will connect to the context broker or an external

service to add further contextualization of the data. Ultimately, the output will be the data gateway, either for a different data stage or for the next pipeline. The processing time of these pipelines tends to be stream processing, as it can be calculated easily at the rate that events are consumed.

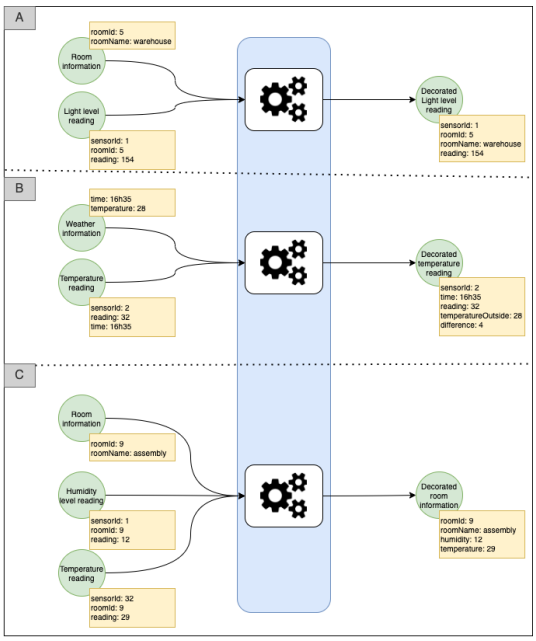

Figure 13: Decoration pipelines examples

In Figure 13 it is presented three use cases for decoration pipelines. In the first scenario (A) it's making a simple decoration, by adding the roomName to a light level reading. Using 2 data sources, the reading and the room information, the pipeline combines them to create a more detailed event reading.

The second use case (B) uses a similiar approach to the first, but instead of using a data source that is typically inside the context broker, such as room information, uses an external service to add relevant information to the consumed event. In this case, the pipeline is taking measurements from the environment and combining them with meteorological data at the same time.

The third and final example for these categories of pipelines (C), is using the data consumed from the environment to decorate the room information. This is done by aggregating readings from the sensors into a single common entity.

Quality assessment pipelines

These pipelines are responsible for generating data quality metadata, which will be referred to as quality data packs. This metadata contains attributes and properties to be added so it can be accessed the data quality level of ingested data. Pipelines of this type tend to have similar inputs and outputs, as it consumes both the data dateway and the context broker, and then produces the results to both of these components for further processing. Simpler versions of these pipelines can be done via stream processing. However, more complex implementations, such as the use of machine learning processes to evaluate data quality, might be done via batch processing due to the complexity.

The four example use cases presented for this category in Figure 14 will be explored further.

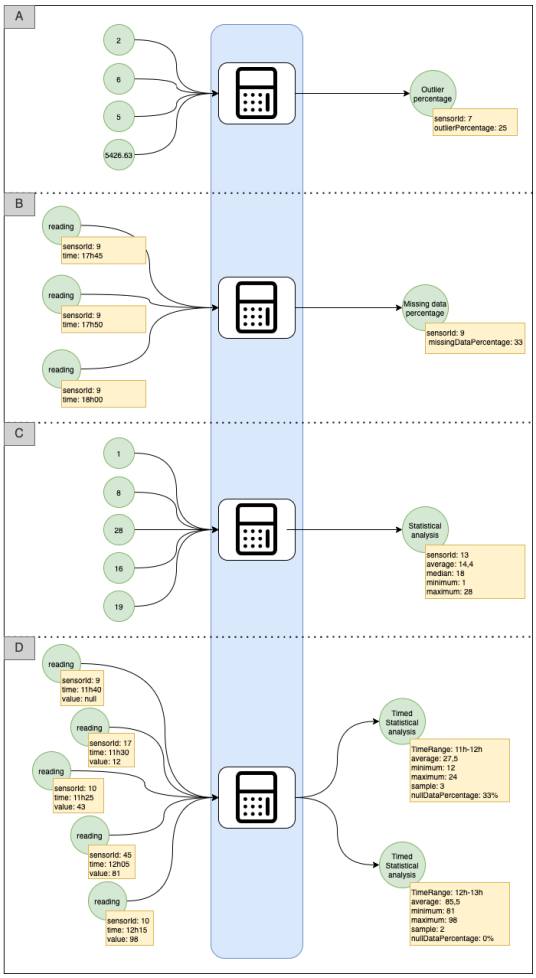

Figure 14: Quality assessment pipelines examples

The first two scenarios present pipelines that calculate statistical metadata about the data, at the rate that is generated. The first scenario (A) is calculating the percentage of outliers, based on recent and acceptable values for a given measurement. In the second use case (B), it's being observed that an instance where is expected to have data for a given measurement every five minutes. When the data that is expected to arrive is missing (in this example, data at 17h55) characteristics such as missing data percentage can be computed.

In the third scenario (C) its presented a pipeline that is creating a statistical report for the data that is consumed. This generated report can then be used to enrich the context metadata for data, allowing for value estimation and continuously improving processes such as outlier detection.

The fourth scenario (D) presents us with a similiar premisse as the third scenario, but it's creating a more detailed report that is subdivided by time frame. This allows for a better understanding of data through time, and even greater certainty in processes of identifying inaccurate data.

Filtration and alerting pipelines

The pipelines of these types generally refer to filtration of events, either for data cleaning activities or alerting systems. These filtrations can be accomplished by simple data classification, analysing the values according to a defined set of rules that the data must obey to. This can be done through the analysis of patterns within the

environment. The input of this type of pipeline is typically the data gateway, and the output can be the data gateway in the cases of data filtration, or external services in the case of the alerting scenario. Due to the necessity of seeing data near real-time, these pipelines typically employ stream processing.

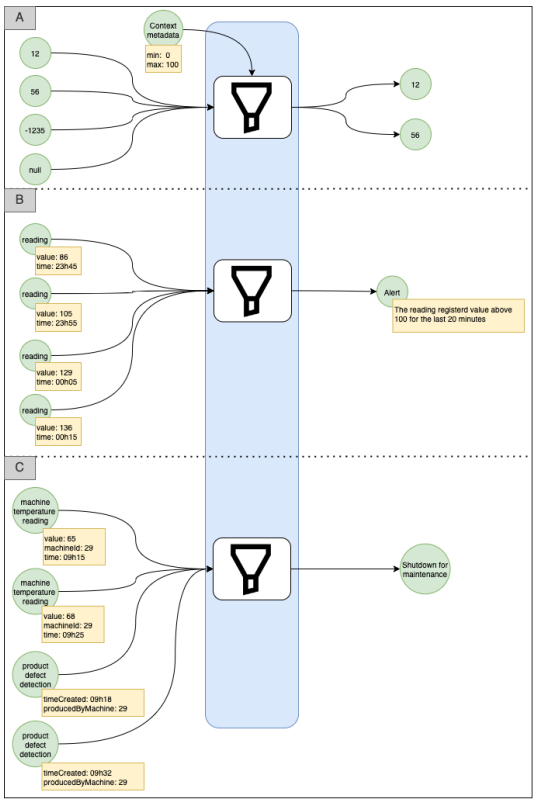

Figure 15: Filtration and alerting pipelines examples

Figure 15 illustrates three pipelines for filtration and alerting.

The first scenario presents a pipeline that filters events out according to a set of rules. In this case it filters events that have a value between 0 and 100.

The second scenario contains a pipeline that is creating an alert based on the condition of having 3 events with a value bigger than 100 in the last 20 minutes.

These first two scenarios depict simple, yet common processes in the data ingestion environment. This makes the reutilization of such pipelines with parameterized values a key feature of this architecture.

The last use case illustrates a more complex scenario of processing multiple event sources, and triggering an action depending on a certain predetermined pattern. This is based on Complex Event Processing (CEP) [111] and it is used to filter related events, according to a pattern. In this particular instance its painted a simple scenario where it's predicting if a given machine needs to be stopped for maintainence based on recorded temperature and amount of product defect generated.

Heavy batch processing

This category englobes processes that take a batch processing approach to the computation of data. This umbrella term includes processes such as machine learning model training, ETL processes, pattern processing and

aggregations.

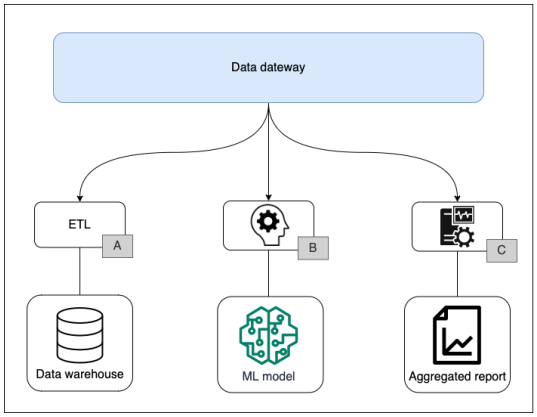

Figure 16: Batch processing pipelines processing

For this last category, it was identified three common processes in Figure 16 that fit in the description.

The first process (A) is an ETL process. These processes are commonly used to extract data from a given data source. They then transform it and mold it to a certain required criteria and then store the result in a separate data storage. These data storage systems are typically data warehouses and are very useful for speciallized requirements within the business.

The second process (B) presents the usage of machine learning algorithms to create a machine learning model that can then be tranformed into pipelines to enrich, classify, aggregate and filter the data depending on the use case. Other machine learning methods such as Deep Learning can be utilized in this field to identify patterns within the enviroment [112]. The patterns identified with these techniques are very useful as context information, since they provide a visual representation of the correlation of the data. Besides being of extreme importance to the user, this extracted information can also be shipped back to components such as the context broker, which can then be used for filtration pipelines that use complex event processing.

The last identified scenario in this category (C) represents an aggreation process. The result of this process includes a broad overview of the total or subsets of the environment. This can then be used for analysis by the user, and sent context broker to update semantic and context information. These practices are very relevant from a data quality perspective, since it provides information across the integrated systems and how they are behaving.

Classifications such as these enable us to better understand typical pipelines that can be developed for this architecture, but they should not be taken as an absolute rule. Although these classifications will be suitable to use in most scenarios, they can be expanded and hybridized to better address highly specialized requirements.

It's imperative for the architecture to offer options for both streaming and batch processing, taking inspiration from the Lambda Architecture. This will allow for a more flexible and scalable architecture [113]

The pipelines for this architecture also have a very distinct characteristic. This is that they can be assembled in a hierarchical way, so that an output of a pipeline can be the input of another, creating a pipeline system. Pipelines can be reused in multiple *pipeline systems* with different parameters. For this process to be implemented successfully, pipelines need to define a set of requirements. They can be related to the data consumption stage,

required data fields, previous pipeline results or data model versions.

Example of these *pipeline systems*, and the dependency of the pipelines can be seen in Figures 17 and 18.

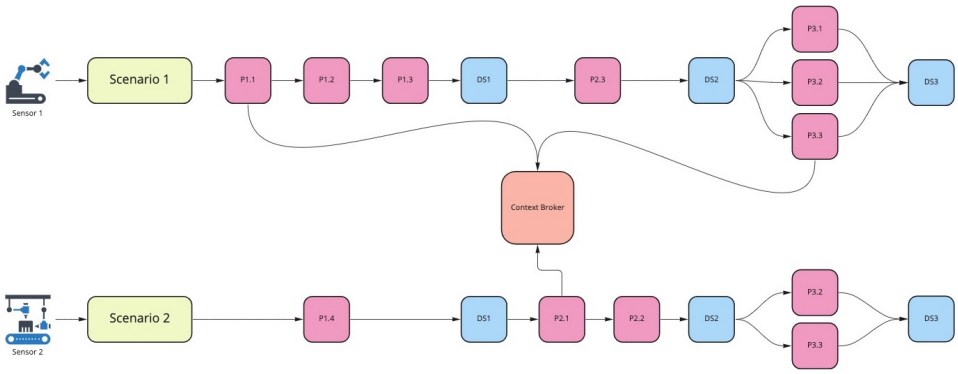

Figure 17: Pipeline systems example

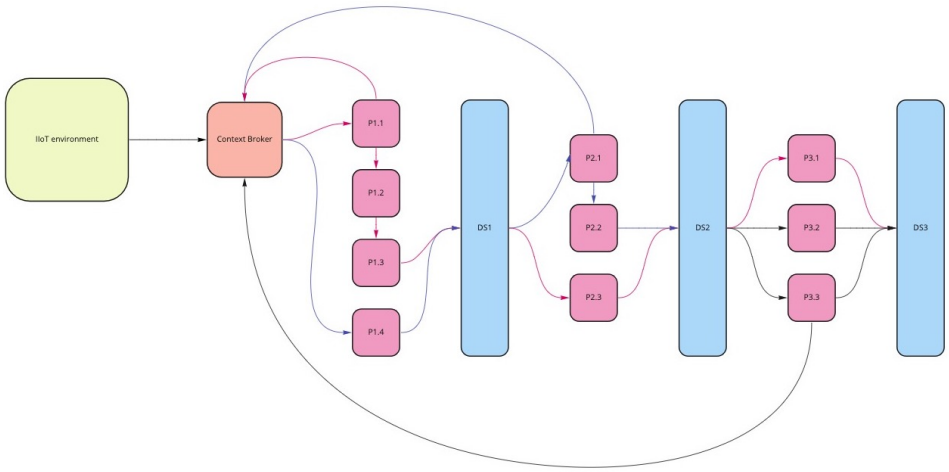

Figure 18: Pipeline systems role in the data flow

The figures illustrate how pipelines are integrated within the architecture, their hierarchy and how they are shared between *pipeline systems*.

Analysing the first scenario in depicted in figures 17 and 18 it's clear that pipeline *P1.1* does not have any requirements and consumes every data that comes in from the environment. Then the next pipeline *P1.2* that depends on *P1.1* and *P1.3* that depends on *P1.2*, resulting in a direct dependency in this first stage. After data is processed by *P1.3*, its moved to the next data stage, then being picked up by another pipeline *P2.3*. In addition to the data being processed by *P1.3*. This pipeline was the additional requirement of data that lives in the second data stage, adding another dependency level.

It can be seen in the example pipelines *P3.2* and *P3.3* being shared across *pipeline systems*. It also can be examined how these pipelines can have quality feedback functionality, sharing computed data with the context broker, so that it can subsequently be used.

The technologies examined for these components were Apache Flink, Apache Spark, and Apache Beam.

Apache Flink and Apache Spark are both open-source projects in the Apache Software Foundation, and are the state-of-the-art tools for stream-processing and batch-processing. There is a high public contribution and interest in both of these technologies, and they can be used in similar ways. Although studies have shown that there's no one-size-fits-all solution [114, 115, 116, 117, 118], both of the tools are capable of handling a high velocity and significant volume of data. Ultimately, the choice was Apache Flink for stream processing, since its on the core design of the technology, offering features such as checkpoints, watermarks, and in-memory state processing. For batch processing, Apache Spark was the chosen candidate, offering high performance processing along with machine learning and advanced analytics features.

Apache Beam was the chosen technology to create the pipelines that process the data. It allows for the creation of code that is agnostic to the engine that is being run beneath it. This allows users to construct pipelines that can run in both Apache Flink and Apache Spark engines without modifying them. With these features users can take advantage of both engines and compare the performance of both for different use cases, finding the most appropriate tool that fits their requirements.

4.3.1 Apache Flink

Apache Flink [10] is an open-source platform for streaming and batch processing of data. Data processing applications such as real-time analytics, continuous data pipelines, historical data processing (batch), and iterative algorithms (machine learning, graph analysis) can all be expressed and executed as pipelined fault-tolerant dataflows in Flink [119].

To express complex datapipelines, Flink's dataflow execution encapsulates distributed, record-centric operator logic, following the guidelines of the Dataflow model [120]. Data processing pipelines written in Flink are committed to preserving consistent application state using a modular state backend. Furthermore, the system manages operations state and orchestrates failure recovery and reconfiguration(scale-out/in) whenever necessary without imposing heavy impact on execution or violating consistency [121].

The Flink architecture consists of two main components: JobManager and TaskManager. In addition to determining when the next job (or collection of tasks) should be scheduled, the JobManager orchestrates the distributed execution of Flink Applications. This includes coordinating checkpoints, coordinating failure recovery, and responding to completed tasks or execution errors. On the other hand, Task managers, also known as workers, execute the tasks of the dataflow. They are composed of task slots that indicate how many concurrent processing tasks are able to run inside the pipeline. This architecture is illustrated in figure 19.

Apache Flink is capable of processing data in both unbounded and bounded data. Unbounded streams have a start, but no defined end. This is why events that come in this kind of processing mode need to be continuously processed, and they must be processed in the order that those events occur. Unlike unbounded streams, bonded streams, or batch processing, have a clear start and end. For this type of processing, all the events can be ingested and order is not relevant because the stream can be sorted in any way.

Figure 20 illustrates how these different streams can be modeled.

When it comes to aggregation functions, Flink has a efficient way to process unbounded streams. While this issue is easily resolved in batch processing, since the events are stored in the same place at once. However, at unbounded streams Flink resorts to windows to make these aggregations.

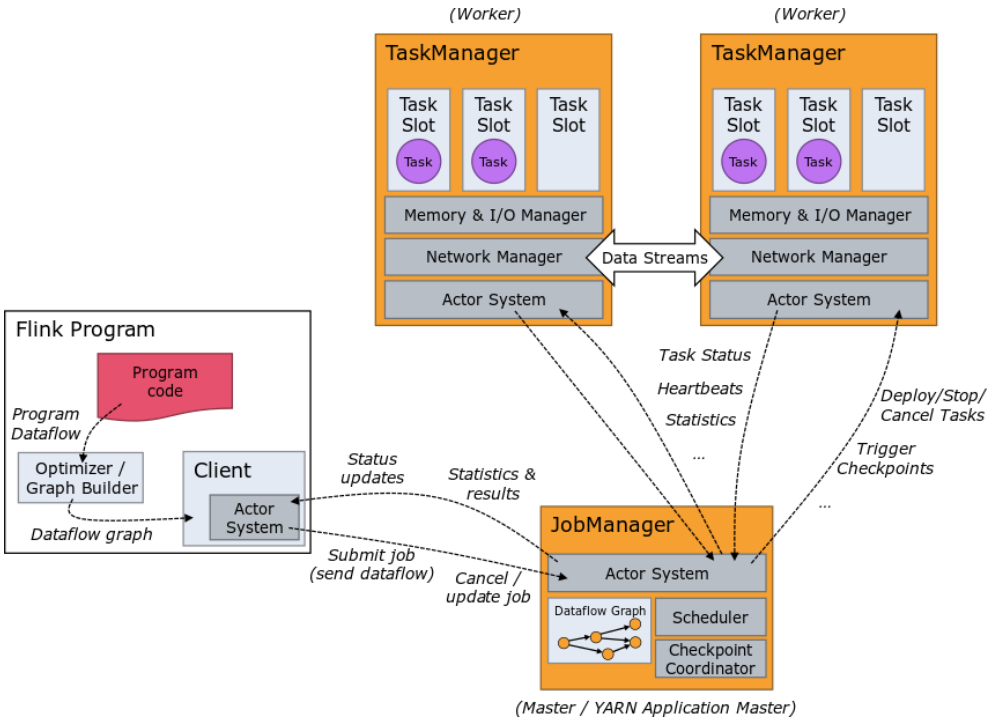

Figure 19: Flink architecture [10]

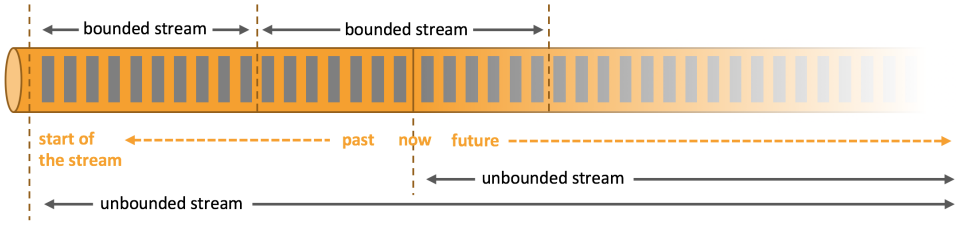

Figure 20: Bounded and unbounded streams [10]

Event-time processing in Flink is based on specific timestamped items known as watermarks, which are injected into the stream by either the data sources or a watermark generator. A watermark with a timestamp t can be interpreted as a claim that all events with timestamps t have already arrived (with acceptable probability). Figure 21 illustrates this scenario.

Another significant feature of Apache Flink is the low latency processing that it offers, due to its in-memory state feature. For some operations Flink will need to retain state in order to make calculations. Some examples include aggregations, decoration data and filtration rules. This state is stored in memory, which results in very low latency to access the state. The state will periodically be stored in a persistent data store, to guarantee failure recovery.

Flink also provides a set of packages to serve different requirements. Some examples include:

- SQL support - Flink SQL allows for simple development of streaming applications using standard SQL. This allows users with SQL skills to develop complex applications without knowing other programming languages.

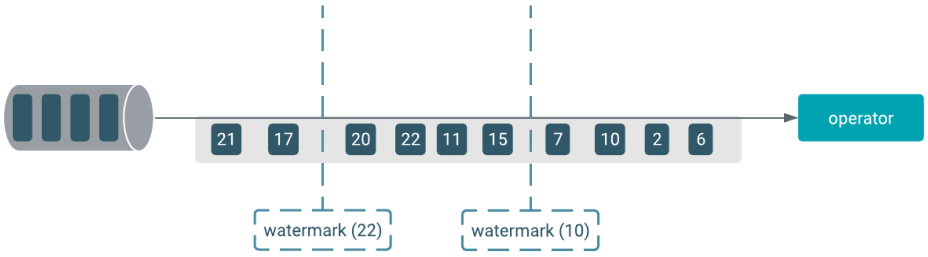

Figure 21: Watermark visualization by Cloudera [11]

- Flink ML - This library provides machine learning APIs and infrastructure that streamline the building of machine learning pipelines. Users can utilize the standard ML APIs to develop ML algorithms and then leverage these infrastructures to build ML pipelines for both training and inference jobs.
- FlinkCEP - this library allows for Complex Event Processing (CEP) operations inside a Flink pipeline. This empowers the ability to detect event patterns in a continuous event stream.

The choice of Apache Flink to implement the architecture pipelines is justified by its scalability, consistency, and real-time capabilities. When dealing with IIoT environments, it's of the utmost importance to process real-time streaming data to create relevant information, since the more time the data takes to be processed, the less relevant is its result.

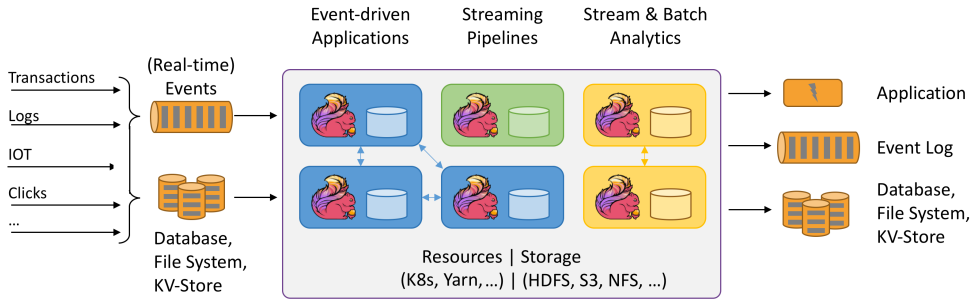

Figure 22: Stateful Computations over Data Streams by Flink [10]

4.3.2 Apache Spark

The Apache Spark [122] engine offers a unified solution for large-scale data analysis on a variety of workloads. Due to its multipurpose processing engine and general-purpose languages, data science and engineering have been revolutionized. As a result of its advanced programming model, Apache Spark has been adopted by both academia and industry as a fast and scalable framework. It has become the most active big data opensource project and one of the most active projects in the Apache Software Foundation [123].

Apache Spark has been the tool of choice for data analytics for many years, but in recent years Apache Flink has emerged as an alternative. The main reason for this is that Apache Flink filled a gap that Spark was lacking, which was stream processing. While Flink is tuned specifically for stream processing, with the possibility of

applying those capabilities to batch processing as well, Spark was first designed for batch processing and then applied those concepts to processed streaming data, resulting in higher latency in some cases. [117].

Spark has a fundamental data structure known as Resilient Distributed Datasets (RDD). An RDD is formally defined as a read-only, partitioned collection of records. RDDs can be generated using deterministic operations on either stable storage data or other RDDs. RDD is a fault-tolerant collection of items that may be used concurrently. This concept functions as a programmatic abstraction over the distributed data of a dataset.

Much like Flink, Spark takes advantage of in-memory computing which reduces latencies. This allows to run iterative algorithms, as programs can checkpoint data and refer back to it without reloading it from disk [124].

Spark has another relevant characteristic in that it has a wide range of libraries available. This ecosystem includes:

- Spark core - This library functions as the main engine for distributed data processing. Subsequent libraries are built on top of this engine, taking advantage of its memory management, fault recovery, scheduling, distributing and monitoring capabilities.
- Structured streaming - This library adapts the Spark engine to process streaming data, allowing batch capabilities to be adapted to run in streaming jobs. By default the input stream data is divided into small chunks of data, also known as micro-batches. Alternatively, this can be configured to run in a smaller latency mode called continuous processing.
- Spark SQL - This library adds native SQL capability to Spark and simplifies the process of accessing data stored in RDDs as well as external sources.
- MLib - This is a machine learning library, that has a series of features that simplify the development of machine learning pipelines.

The Spark Cluster architecture is presented in figure 23.

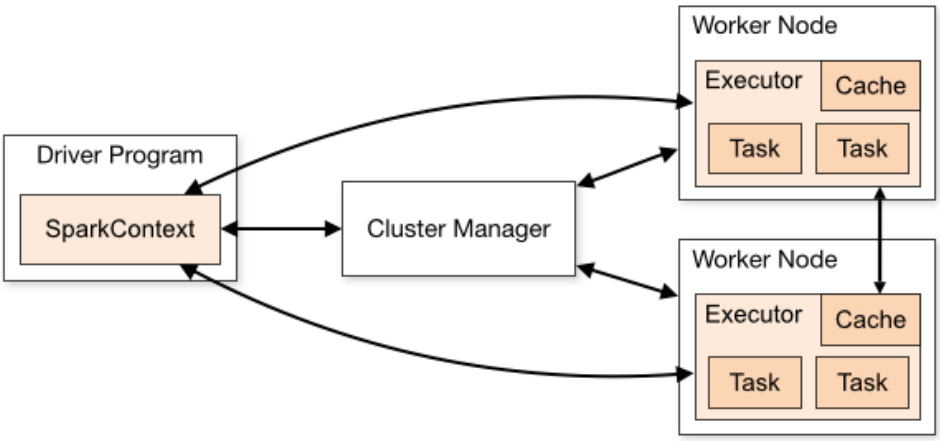

Figure 23: Spark cluster architecture [12]

The first components of the architecture are the *SparkContext* and the Driver program. The *SparkContext* contains the basic functions of the application, while the Driver Program is responsible for components such as

DAG scheduler, task scheduler, backend scheduler, and block manager. These components are responsible for translating user-written code into jobs that can be executed inside the cluster.

The next component in the architecture is the Cluster Manager, and it is responsible for controlling the execution of vaious jobs, with the help of the Driver Program. Once the job has been broken down into smaller jobs, the Cluster Manager distributes them to worker nodes.

The worker nodes, along with the executor, are responsible for running the actual jobs.

4.3.3 Apache Beam

With Apache Beam [125], both streaming and batch data processing are handled in a unified way. This is done with a common programming interface that can be used across various engines without modifying applications. This reduces switching overhead, which has drawn increasing attention from academia and industry. A pipeline is constructed using *PCollections* and *PTransforms* from the Beam SDK. There are two types of *PCollections*: bounded datasets for batch processing and unbounded data streams for stream processing. *PTransforms* take one or more *PCollections* as inputs, apply user-defined functions, and output *PCollections*. [126].

Beam SDK allows for developed pipelines to run in different computing engines such as Apache Apex [127], Google Cloud Dataflow [128], Apache Flink [10], Apache Gearpump [129], Apache Hadoop [130], Apache JStorm [131], Apache Spark [122], and IBM Streams [132].

With this tool, pipelines can be easily developed that will be used for both streaming and batch processing without the switching overhead. It also can utilize specialized engines for the use case at hand, taking advantage of the strongest features of each engine. For example, Spark can be used for jobs that require batch processing and Flink for streaming use cases. The Figure 24 illustrates how Beam abstracts both the input mode of the data as well as the underlying engine

Apache Beam provides multiple features that can deal with the data pipeline at a very flexible level. These features include transformations, time windowing, grouping functions and viewing functions. It also provides other features that are critical when interacting with streaming data, such as watermarking. The watermarking process allows for the pipeline to identify late events and decide how to deal with them. For this feature to work properly Apache Beam distinguishes event time from processing time, in which event time is the time when the event occurs, while processing time is when the event is being processed.

The pipelines that were used for the architecture are implemented with Apache Beam, with the possibility of choosing between Apache Flink and Apache Spark processing engines for different scenarios. This provides extreme flexibility when designing pipelines and pipeline systems, promoting reusability and scalability.

Apache Beam, Apache Flink and Apache Spark can all be deployed to work on the cloud and use distributed processing.

4.4 Storage tools

Data storage systems are the component of the architecture that allows users to browse and explore data. These practices enable discovery and reuse within the architecture. Also, it provides the framework to enhance the quality

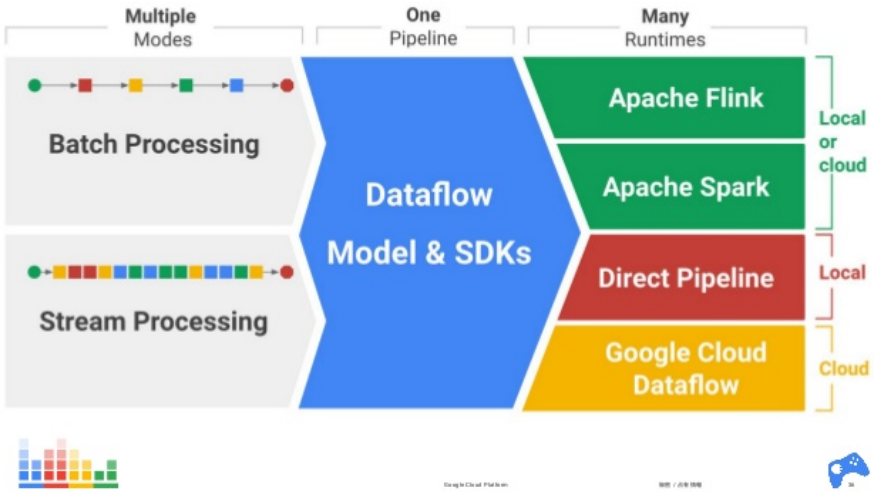

Figure 24: Apache Beam overview by Data Science Central [13]

of data by visualizing how data can be interpolated. It's also necessary to keep history records of the data so it can be used for auditing purposes or simply to keep track of processes.

The IIoT environment requires a scalable and decentralized component to accommodate fast data ingestion. It also needs to be quick when serving data to the user, so its use can be as impactful as possible.

The technologies of choice for this component were Apache Druid and MongoDB.

MongoDB plays a simple but significant role in storing all of the data that flows through the data gateway. Despite Apache Kafka having the ability to keep data for a set period of time, old data is deleted according to a time or size-based schedule. In order to maintain the archival capabilities of the architecture, MongoDB was introduced. This source-available tool was chosen because of its flexibility and data search capabilities.

Apache Druid was introduced to function as an analytical data source, able to perform low-latency queries on real-time data that is ingested from Kafka. Additionally, this open-source tool serves as a column-oriented time-series database that is able to ingest enormous amounts of data. The analytical capabilities of this tool allowed for data discovery and interoperability.

4.4.1 Apache Druid

Apache Druid is an open source, distributed, column-oriented, real-time analytical data store to power modern analytics applications. This technology was conceived after facing problems with Hadoop implementations. When it comes to processing data in batch, Hadoop is a very powerful tool. It also serves as a data store for the results of such processings. The problem with this technology is that it struggles to meet query performance and data availability standards when data begins to scale. Druid was designed to fulfill these requirements [133]

This tool is an excellent candidate for the main data visualization and discovery tool, since it provides high query performance, low-latency aggregations [134].

Druid provides visualization of data in form of datasources, which are collections of events that are partioned into a set of segments with million of rows. These segments are the basic unit of storage where queries will retrieve the data.

Being a time-series database, Druid always require that the events ingested have an associated timestamp, which allows for datasources to be divided into well defined time intervals, and simplifying data distribution policies, retention policies and first level query prunning.

Each data source is divided by a time range and optionally by one or more data attributes. Each time range that partitions the data is called a chunk, which is for example a day, if the data is partitioned by days. Within each of those chunks, the data is divided into segments, as seen in figure 25.

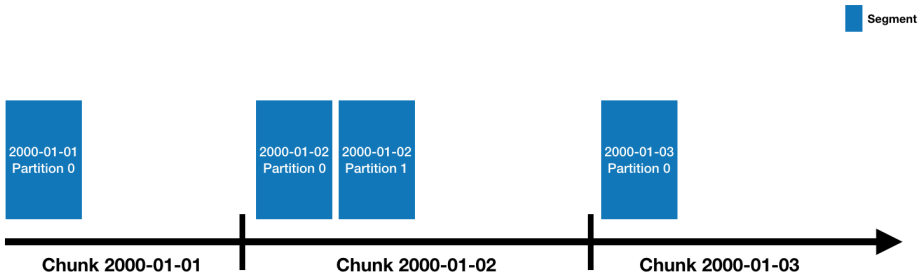

Figure 25: Druid chunk and segment architecture [14]

Each segment is identified by a data source identifier, the time interval of the data, the version of the segment, and the partition number. The version represents the freshness of the contained data, and is increased every time a new segment is created and its typically an timestamp corresponding to when the segment was first started. Later versions represent newer views of data than segments with older versions. Metadata such as this is used for concurrency control, which dictates that read operations always access data in a particular time range from the segments with the latest version identifiers for that time range. These segments are stored in a column orientation, since one of its main features is the aggregation of data event streams. This is because column orientation allows for more efficient CPU usage since some columns can be immediately discarded before processing, loaded and scanned, while row-processing requires the whole row need to be scanned for aggregation porpuses [134]. Another characteristic of Druid segments is that they are immutable. This offers benefits such as read consistency, and multiple threads can scan the same segment at the same time. This helps enable higher read throughput.

The Druid cluster architecture is composed of different types of servers, each with different capabilities and purposes. The intercommunication between these nodes is kept to a minimum, making them independent of each other, which assures data availability when intra-cluster communication fails [14, 133]. A diagram of the Druid cluster is presented in Figure 26.

Druid servers are divided into three categories: Master, Query and Data. Master servers manage data availability and ingestion, query servers address queries, and lastly, data servers execute ingestion workloads and store all queryable data.

Coordinator This component is part of the master servers of Druid and is responsible for segment management and distribution. Nodes of this type notify historical processes to load or drop segments, based on the configuration given. They are also responsible for ensuring that segments created are replicated across nodes to ensure availability.

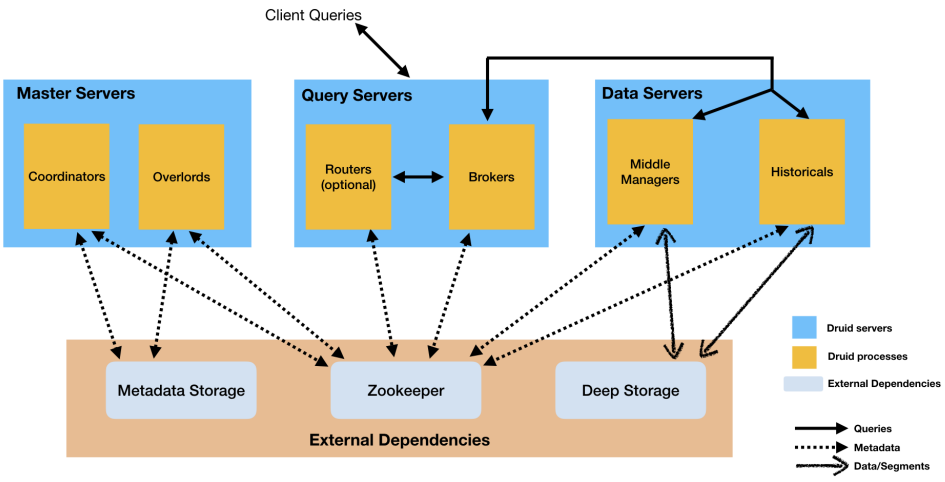

Figure 26: Druid architecture [14]

The coordinator runs periodically, and connects to the zookeeper cluster to acquire cluster information.

Overlord This process is in charge of accepting tasks, organizing their distribution, putting locks around them, and updating callers on their status.

Broker This component is responsible for routing queries received to different nodes, based on segment metadata stored in the Zookeeper cluster.

Router This component is used to route different queries to Broker processes. This allows for the isolation of hot zones, which contain the majority of the queries received. This is done with the objective of keeping less important data from interfering with more critical data.

Historical This process is responsible for serving segments and queries. The segments are assigned by the Coordinator. This component downloads segments from deep storage and responds to queries about these segments. The data is stored in memory.

Middle manager This is the process that is responsible for task execution and data ingestion. These tasks are distributed by Peons, which are small processes that run in the JVM.

Deep storage This component is used to store any data that has been ingested into Druid. This is not directly part of the Druid system, it is an file storage system that Druid have access to. In a cluster environment, this is generally a distributed object store such as S3 or HDFS. Druid does not access this component for data serving, since that would take more latency, it instead uses this component to restore the data if this ever gets lost, making this an important component of the architecture since it allows for a fault-tolerant design. Segments are periodically

committed and published to this component.

Metadata storage This component is responsible for storing metadata about the system, such as segment information, rule records, configurations, task-related tables and audit records.

Zookeeper This component is used for internal service discovery, coordination, and leader election. Zookeeper is responsible for processes such as coordinator leader election, segment publishing protocol, segment load and drop protocol, overlord leader election, and middle manager task management.

The purpose of Druid in the architecture is to provide a platform for data exploration and accessability. Allowing for multiple actions such as data aggregation, interpolation and distribution, it is identified by the number 4 in Figure 11,.

4.4.2 MongoDB

MongoDB is one of the most popular NoSQL databases. It is an open-source, high performance, high availability database that provides automatic scaling [135]. Data is stored in the form of documents inside MongoDB, which have flexible schema modeled by JSON objects.

This non-relational database also provides high performance query language processing, that empowers the analysis of data. The architecture of this tool is represented in figure 27.

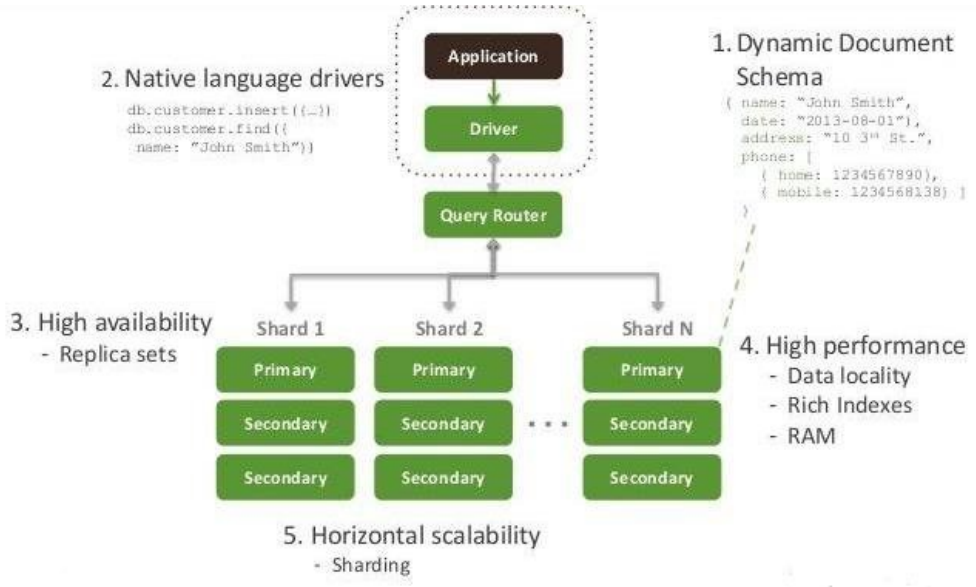

Figure 27: MongoDB architecture by Mungekar [15]

The data is distributed per shards in MongoDB, so scalability can be assured. Inside these shards, the data is replicated into replica sets, to ensure high availability. The query router distributes queries to the shards to process the data.

MongoDB will play a small but significant part in the architecture. It will be used as a history data store, storing all events that pass through the architecture and is identified by the number 5 in figure 11,

4.5 Other components

There are a few components that were designed specifically for this architecture to assure the proper kilter of this architecture.

The first component designed is an adapter that subscribes to all newly created events in the context broker and publishes them to Kafka. This component is called Context Broker Kafka Adapter, shorted to CBKA, and is a Java application that can be scaled up or down depending on the throughput needs.

The other component is the MongoDB Kafka Connector [136], and it was configured to store all information that flows through the topics in collections.

5 Use case

The use case used to test this methodology was a dataset that contains the of different sensors in multiple houses in New York with an IoT environment. This dataset was chosen for its sensor and sensor reading data. Although it does not come directly from an IIoT environment, it offers strong parallels, making it a valid first use case.

The dataset presents its data in a column oriented form. The first two columns are dataid and localminute, which represent a house identifier and the time that the values are being recorded respectively. The remaining 77 columns represent different devices that record values within the environment. The devices are identified by the type of metric that is recording followed by a number. Some examples of these devices are air1, bedroom2, car2, dishwasher1.

In order for this dataset to serve the purpose that is needed for the use case, it needs to be transformed so the columns that represent the devices can be turned into rows, containing the sensor, house identifier, timestamp that the value is recorded and type of recording, fitting into the standard sensor reading format found in most IIoT devices. A Apache Spark job was developed for this porpuse, which transformed the data in the format seen in Table 3.

After the original dataset is transformed, an Python application was created to read from the newly transformed dataset. In this application, the values are sent to the entrypoint of the architecture, which is the context broker. The application sends dataset rows at a configurable frequency, acting as an gateway in a real-life IIoT scenario.

The dataset transformation flow is illustrated in Figure 28.

Figure 28: Dataset transformation flow

5.1 Applied definition of data

The data model that was used for this project was heavily based on the smart models that are hosted by Fiware [106]. The model that was shown to be most suitable to fit the current use case was the Smart Sensoring Model. This model is extremely focused on the Sensor interconnection within an IoT environment, with extensive modulation of sensor and sensor reading information. In this model, the entities most commonly used in the context broker were Device and DeviceMeasurement, seen in figure 29. The Device entity is a combination of hardware, software and firmware intended to represent the sensor as a whole. It is responsible for sensing a particular part of the environment. By contrast, the DeviceMeasurement indicates the values that the Device has measured inside the environment, creating a direct correlation between them.

The *Device* entity has numerous fields to represent the sensor itself. The following list will enumerate some of them.

- ID - This is the main form of device identification and reference. Using this field, all entities can be connected to each other. If some device is related to another device or even another entity, such as a room,

house id	localminute	sensor name	reading value	type
5997	2019-06-24 15:55:00-05	bathroom1	0.001	bathroom
5997	2019-06-24 15:55:00-05	clotheswasher1	0.0	clotheswasher
5997	2019-06-24 15:55:00-05	dishwasher1	0.0	dishwasher
5997	2019-06-24 15:55:00-05	drye1	0.0	drye
5997	2019-06-24 15:55:00-05	garage1	0.003	garage
5997	2019-06-24 15:55:00-05	grid	-1.036	grid
5997	2019-06-24 15:55:00-05	kitchenapp1	0.002	kitchenapp
5997	2019-06-24 15:55:00-05	lights_plugs1	0.011	lights plugs
5997	2019-06-24 15:55:00-05	microwave1	0.001	microwave
5997	2019-06-24 15:55:00-05	solar	1.686	solar
5997	2019-06-24 15:55:00-05	utilityroom1	0.503	utilityroom
5997	2019-06-24 15:55:00-05	waterheater1	0.0	waterheater
5997	2019-06-24 15:55:00-05	leg1v	122.046	legv
5997	2019-06-24 15:55:00-05	leg2v	122.846	legv
3700	2019-06-24 15:59:00-05	grid	0.469	grid
3700	2019-06-24 15:59:00-05	heater1	0.001	heater
3700	2019-06-24 15:59:00-05	kitchen1	0.002	kitchen
3700	2019-06-24 15:59:00-05	livingroom1	0.012	livingroom
3700	2019-06-24 15:59:00-05	microwave1	0.003	microwave
3700	2019-06-24 15:59:00-05	oven1	0.006	oven

Table 3: Sample of the result dataset from the Spark job

it will reference that external entity by its ID.

- name - This is also a form of device identifaction, but it tends to be more human-readable for better contextualisation.
- description - This field is entirely for contextualization of the device, in order to better understand its place in the environment.
- owner - This field allows for the specification of an organization or people that are responsible for the device.
- location - This allows for spatial contextualization of the device inside the environment.
- category - This field describes the type of device. It can be a multitude of values such as Sensor, Actor, Meter, Network, etc.
- controlledProperty - This describes which environment values are being monitored. This represents anything that can be measured. Some examples are temperature, air quality, conductance, humidity, and light.

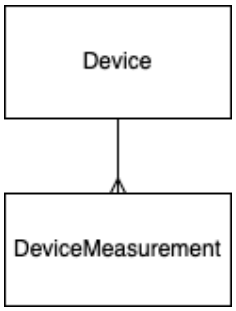

Figure 29: Device to device measurement relation

- controlledAsset - Represents which real-world object is controlling. This can include buildings, rooms, objects, machines, or cars. This field can also be used to reference an external entity. For example, if a device is monitoring a building or a car, it can refer to them by their ID.
- dateInstalled - A time field that shows when the device was installed.
- dateFirstUsed - Another time field, but this one represents when the device was first used.
- dateLastCalibration - This field shows when the device was last calibrated.
- provider - Provider of the device. It can be considered to be the company that manufactured it.
- deviceState - Represents the operational state of the device.

The next list will follow the same strategy, but will examine some of the properties of *DeviceMeasurement*.

- id - Unique identifier of the measurement
- numValue - This field contains the numeric value of the measurement. When a reading is parsable to a number this field will contain it.
- textValue - This field contains the text value of the measurement. It will always contain the value regardless of the format.
- controlledProperty - In the same way that the *Device* has the same field.
- refDevice - This field contains the reference information to the *Device* that the measurement is tight to. This is what allows us to create a semantic net around the devices and the actual values that it records.
- measurementType - The type of measurement that is being done.
- dateObserved - The timestamp when the measurement was made.
- outlier - Defines if a particular measurement is an outlier.
- unit - The unit that in which the value is expressed.

The Smart Sensoring data model is the basis for data contextualization, but it's not limited by it. Smart models are meant to extended with other properties that can bring even more value to the solution. In this case, the original data model was enhanced the smart model with properties originated from other smart models and properties coming from the SAO ontology [137].

The smart manufactoring model was used to model manufactoring machines that sensors can be attached to, interconnecting the machines to the sensors and even other machines. The smart city smart model was used to enhance the data with building and room information, connecting them to the other existing entities in the semantic net.

The SAO ontology [137] is an defined stream annotation ontology that served as inspiration to various fields in the model. Many of those fields are directly connected to data quality metrics. This project also provides temporal concepts such as *StreamData*, *Segment*, *StreamAnalysis*. This last temporal concept is deeply tight to data observability, since it allows to model a series of transformation that were made to the data in a specific time frame, allowing for the understanding of the data and eventual reconstruction if needed. This ontology is also heavily connected data quality classifications, through a field that directy refers to the Quality Ontolgy [138]. The Quality Ontology is extremely important, as it provides fields that are used to model the main quality issues that were already mentioned, such as *Accuracy*, *Age*, *Completeness*, *Correctness*, *Deviation*, *EnergyConsumption*, *Frequency*, *Latency*, *MonetaryConsumption*, *Precision*, among others. These two ontologies allows us to enhance the established smart model with already accepted fields in the data quality and stream analysis spectrum.

Most of the *Device* context will be provided by a user, since it contains almost entirely static metadata, which can only be given initially by a manual insert. The devices inserted will be attributed an ID which the *Device-Measurement* will then reference, so the measurements can be associated with the ID and then with the whole environment. As of now, these devices can be inserted via a HTTP request to the Orion Context Broker. The created entities will then be sent to the context broker and then to a Kafka topic, to be processed by the pipelines. *DeviceMeasurement* entities which represent the sensor readings are also sent via HTTP request to the context broker, but these ones are sent through the gateway instead of manually. When the measurement is loaded into the system, the context broker will automatically add the context to it via the Device reference, reaching the data gateway with already context associated for proper processing.

The context broker also serves as an explorable and searchable tool to find data. This component contains the most recent data that is found in the system. This is because it is the entrypoint and any pipelines that do data decoration feedback to the context broker. Orion Context Broker contains an that allows for searching entities with specific parameters [139]. Additionally, due to the lineage capabilities incorporated into the pipelines for observability purposes, the context data should include both the topics and pipelines that the data is flowing through. This will enable users to connect to those topics and retrieve the data.

5.2 Pipeline ontology

This section will describe the pipeline model inside the architecture. This will serve as documentation to understand the function of the pipelines as well as allowing for creating pipeline templates. The pipeline template allows for pipeline reusability by only changing specific parameters to conform to separate but similar use cases. A simple example that fits into this description is the usage of a pipeline that filters sensor readings that have a value above a certain limit. In this described scenario a template pipeline could be created that takes in parameters such as input information, value limit, and output information. The template pipeline instantiates a specific pipeline that conforms to the defined use case. This process allows for reutilization of created processes and improves the scalability and maintenanceability of the architecture.

Ultimately, these pipeline templates can be stored in a repository that users can access, providing visibility and shareability.

id	String	Id of the pipeline. This will be the main method of identifying pipelines. It will also be used for watermarking the data.
name	String	Name of the pipeline. This aims to get a small description of the functionality of the pipeline.
description	String	This field contains the full description of the pipeline, so it can be easier to understand its requirements and purpose.
tags	Array of String	Set of words or phrases that can be used to quickly search for pipelines inside the repository. It can be the technology itself or the way data is transformed.
dependencies	Array of Object	This object contains all the requirements that are needed for the pipeline to ingest the data.
dependencies.dataStage	Array of String	Definition of the data stages that it aims to consume from. Although most cases will require data pipelines to be connected to only DataStage, the architecture allows for multiple data stage connections.
dependencies.stagingTopic	Array of String	Definition of the data stages that it aims to consume from. Although most cases will require data pipelines to be connected to only DataStage, the architecture allows for multiple data stage connections.
dependencies.dataFilter	Object	This field allows for the specification of data filtration. The value of this field is a set of key-value pairs, where the key identifies the target property name that will be filtrated, and the value is a string that defines a regex on how the data should be filtrated.
dependencies.pipeline	Array of String	Set of pipeline IDs that are required to run before this pipeline can take action.
dependencies.model	Array of objects	Specification of the models that the pipeline will consume. When multiple models are defined, the pipeline will consume different data sources and types.
dependencies.model.id	String	The ID of the model in which data will be consumed.
dependencies.model.minVersion	String	Minimum version of the model that the pipeline is able to process.
dependencies.model.maxVersion	String	Maximum version of the model that pipeline is able to process.
output	Object	This object defines the output specification of the pipeline process
output.dataGateway	Object	Definition of the output that is sent to the data gateway.
output.dataGateway.model	Object	Model specification of the data that is sent to the data gateway.
output.dataGateway.model.id	String	The ID of the model outputted.
output.dataGateway.model.name	String	Name of the model outputted.
output.dataGateway.model.version	String	Version of the model outputted.
output.dataGateway.model.link	String	Link of the model outputted.
output.dataGateway.dataStage	String	Name of the datastage in which data will be output to.
output.dataGateway.stagingTopic	String	Name of the staging topic in which data will be output to.
output.dataGateway.documentation	Object	Documentation of the transformation that was made to the data.
output.dataGateway.documentation.changeType	String	Description of the data transformation process, such as aggregation, filtration, decoration, alerting, etc...
output.dataGateway.documentation.changedFields	Array of String	An array of fields that were added, deleted or edited during the processing.
output.contextBroker	Object	Definition of the quality feedback that was sent from the pipeline to the context broker.
output.contextBroker.entityName	String	The entity that was modified in the quality feedback.
output.contextBroker.valueChanged	Array of String	Names of the properties of the model that were added or edited in the quality feedback iteration.
output.externalService	Object	An explanation of the integration of external services with the pipeline.
output.externalService.name	String	Name of the external service.
output.externalService.description	String	Description of the interaction and why it is needed.

Table 4: Pipeline model specification

The pipeline model presented in Table 4 is divided into three dimensions: Base information, input or dependency information and output information.

The base information of the pipeline consists of fields such as ID, name and description, that allows for pipeline identification and understanding. Another critical field that is included in the base information about the pipeline is the tags. The tags are a set of words or phrases that can be used to search for the pipeline inside the given pipeline repository. This should include information such as the technology used, the processing type, the business context, and any other characteristic that is pertinent for a search of the pipeline.

The second dimension pertains to the information associated with data that is consumed by the pipeline. This includes the specification of each section the data is coming from in the data gateway, as well as the data model of the data. Another key field to be noted is the filter field, which allows for data filtration, so data only relevant to the pipeline is consumed. Lastly, it can be further expanded the dependency information by specifying a set of pipelines that are needed to run before the specified pipeline can run. This allows for a pipeline hierarchy when needed.

The final set of fields are directed to the output information, which can be a combination of data gateway, context broker, and an external service. The data gateway output can be to a central data stage or a staging topic. If both are specified, then it means that it will produce for both outputs. The context broker specification refers to an entity that will be modified with updated information. Finally, the external service is anything that does not fit into the first two categories. For readability and reusability purposes, the output information also contains the modified fields for the data gateway and context broker.

An example implementation of this model in JSON format can be found in Listing 1

```
{
   "id":"984",
   "name":"Example pipeline",
   "description":"This is an example pipeline. This field contains a description
     of the processing",
   "tags":[
      "example",
      "beam",
      "flink"
   ],
   "dependencies":{
      "dataStage":[
         "DS1"
      ],
      "stagingTopic":[
         "staging-topic-6"
      ],
      "dataFilter":{
         "data.type":"AirQuality"
      },
```

```
      "pipeline":[
         982,
         983
      ],
      "model":[
         {
            "id":"DeviceMeasurement",
            "minVersion":"0.0.1",
            "maxVersion":"9.9.9"
         }
      ]
   },
   "output":{
      "dataGateway":{
         "model":{
            "id":"DeviceMeasurement",
            "name":"Device Measurement",
            "version":"1.0.0",
            "link":"https://github.com/smart-data-models/dataModel.Device/tree/
    master/DeviceMeasurement"
         },
         "dataStage":"DS2",
         "stagingTopic":"staging-topic-8",
         "documentation":{
            "changeType":"Decoration",
            "changedFields":[
               "data.quality.field.example"
            ]
         }
      },
      "contextBroker":{
         "entityName":"Sensor",
         "valueChanged":[
            "sensor.quality.field.example",
            "sensor.air-quality.lastValue"
         ]
      },
      "externalService":{
         "name":"Alerting service",
         "description":"This example pipeline will call an alerting service if
    the air quality is too bad."
      }
```

```
60     }
61 }
```

Listing 1: Example pipeline model

5.3 The flow of data

Data pipelines are one of the most dynamic components in this architecture. This is because they are responsible for moving the data across data stages and providing feedback to the context broker so that it can be correctly contextualized.

Let's assume that it exists two defined data stages, Data Stage 1 (DS1) which contains mostly raw data that just arrived from an IoT environment, and another, called Data Stage 2 (DS2), which contains an enriched version of the data that already has passed through some kind of filtration, enrichment or augmentation. In this particular scenario, it's possible to have multiple pipelines and pipeline systems that move the data from DS1 to DS2.

The simplest scenario of this transition is to declare a single pipeline that does some kind of transformation to the data. This pipeline is sufficient to move the data from the first data stage to the second data stage. Let's take for example a pipeline that aggregates the energy consumption data every hour.

```
1 (...)
2 "dependencies": {
3   "dataStage": "DS1",
4   "filter": {
5     "type": "EnergyReading"
6   }
7 }
8 (...)
9 "output": {
10   "dataGateway": {
11     "model": (...),
12     "datastage": "DS2",
13     "documentation": {
14       "changeType": "Aggregation",
15       "changedFields": [
16         "sum",
17         "timeRange"
18       ]
19     }
20   },
21   "contextBroker": {
22     "entityName": "EnergyReading",
23     "valueChanged": "aggregations.consumedPerHour"
24   }
```

```
}
```

Listing 2: Energy reading aggregation pipeline

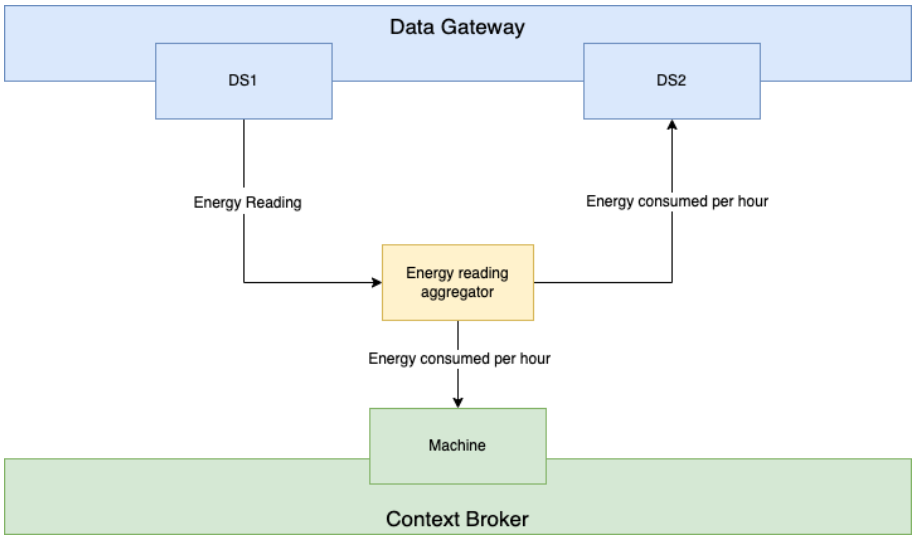

Figure 30: Energy reading accumulator data flow

The requirements of this pipeline would be that it needs to ingest data from the first data stage that contains raw data. The data that it consumes needs to be of the type *EnergyReading*. Based on these requirements, data will be filtrated by datastage and type of data. After the pipeline consumes the data it needs to keep a record of the state of the *EnergyReading* of the hour. Once this processing is complete, the data is finally pushed to the next stage, in the form of aggregation. This data flow is illustrated in Figure 30

Now taking on a more realistic and complex use case. As an example, let's consider that a form of predictive maintenance is being put in place to improve efficiency and lower downtime. It's necessary the creation of a pipeline system that consumes energy readings, air quality readings, temperature readings, and input and output times of the machine. After this the pipeline system will assess if an alert must be given so it can notify if the machine needs to be looked at.

First a pipeline that takes the input and output times of the machine and calculates the time that the machine takes to produce an output will be created. After that a second pipeline needs to be constructed, that ingests the previously calculated time, the energy readings, air quality readings around the machine, and temperature readings. After consuming all of this data the pipeline will have to assess if predictive maintenance is required and generate an event that specifies that. Finally, another pipeline will be put into place, that consumes the events produced by the previous pipeline and sends an alert for each event that it receives.

The following pipelines are designed as follows:

```
(...)
"dependencies": [
    {
      "dataStage": "DS1",
      "filter": {
```

```
        "type": "InputTime"
      }
    },
    {
      "datastage": "DS1",
      "filter": {
        "type": "OutputTime"
      }
    }
]
(...)
"output": {
    "dataGateway": {
      "model": (...),
      "stagingTopic": "predictiveMaintenance1-staging1",
      "documentation": {
        "changeType": "Calculation",
        "changedFields": [
          "timeToProduceOutput.avg",
          "timeToProduceOutput.median",
          "timeToProduceOutput.lastValue",
          "timeToProduceOutput.max",
          "timeToProduceOutput.min"
        ]
      }
    },
    "contextBroker": {
      "entityName": "Machine",
      "valueChanged": [
        "timeToProduceOutput.avg",
        "timeToProduceOutput.median",
        "timeToProduceOutput.lastValue",
        "timeToProduceOutput.max",
        "timeToProduceOutput.min"
      ]
    }
}
```

Listing 3: Time to produce output pipeline

```
(...)
"dependencies": [
    {
```

```
      "stagingTopic": "predictiveMaintenance1-staging1"
    },
    {
      "dataStage": "DS1",
      "filter": {
        "type": "EnergyReading"
      }
    },
    {
      "dataStage": "DS1",
      "filter": {
        "type": "TemperatureReading"
      }
    },
    {
      "dataStage": "DS1",
      "filter": {
        "type": "AirQualityReading"
      }
    }
]
(...)
"output": {
  "dataGateway": {
    "model": "(...)",
    "stagingTopic": "predictiveMaintenance1-staging2",
    "dataStage": "DS2"
    "documentation": {
      "changeType": "Filtration",
      "changedFields": [
        "maintenance.isNeeded",
        "maintenance.lastCalculated"
      ]
    }
  },
  "contextBroker": {
    "entityName": "Machine",
    "valueChanged": [
      "maintenance.isNeeded",
      "maintenance.lastCalculated"
    ]
  }
```

```
46 }
```

Listing 4: Predictive maintenance pipeline

```
1 (...)
2 "dependencies": [
3     {
4       "stagingTopic": "predictiveMaintenance1-staging2"
5     }
6 ]
7 (...)
8 "output": {
9     "externalService": {
10      "name": "email",
11      "description": "Sends an email to a distribution list stating that a
    machine needs to fixed. The email will contain the analysis made."
12    }
13 }
```

Listing 5: Email alert pipeline

```
1 (...)
2 "dependencies": [
3     {
4       "stagingTopic": "predictiveMaintenance1-staging2"
5     }
6 ]
7 (...)
8 "output": {
9     "externalService": {
10      "name": "push-notification",
11      "description": "Sends an notification to a set of devices stating that a
    machine needs to fixed."
12    }
13 }
```

Listing 6: Push notification alert pipeline

In the model presented in Listing 3 represents the pipeline that is responsible for calculating the production time of each machine. Basically, it will receive the input and output events of the machines and calculate the difference between them for each one. This will deliver that information to the data gateway as well as the context broker. For this reason the pipeline needs to consume two data sources and join them, as illustrated in the *dependencies* block, starting in line 2. One of the datasources is the datastage *DS1* that refers to the first data stage inside the data gateway and filters the events only by *InputTime*. It comes from the first data stage *DS1* but is filtered instead by *OutputTime*.

The output section has two different specifications. Firstly, there is the data gateway, to which the calculations are sent. The most peculiar difference in this pipeline as opposed to the previous ones is that the output is going to the data gateway but not to a particular data stage. As the processing is being done by the pipeline system and not by a single pipeline, this means that a single pipeline in a pipeline system is not capable of enhancing data sufficiently to be promoted to the next data stage. Due to this requirement, pipelines within a pipeline system output their results into a staging topic, as seen in Line 20. This staging topic will act as a buffer between pipelines inside the same system, so that data can remain outside the data stage while it is being enhanced. This process is inspired by the source-control branching model known as trunk-based development [140], in which developers have working software in the main branch, called ”trunk”, and when something needs to be changed, a new branch is created to host those changes, and finally when everything is working properly the created branch is merged back into the trunk, keeping a single source of truth. Pipeline systems act much like developers in this scenario, picking up data from the central source of truth, which is a data stage. They will iteratively make changes to the data until it is merged back into the main data stage.

The same concept is applied to the next pipeline in the dataflow. The pipeline defined in Listing 4 will consume from the previously mentioned staging topic with the calculated production time, as well as different data from the first data stage, such as *EnergyReadings*, *TemperatureReadings*, and *AirQualityReadings*.

The pipeline will relate all of this data through the Machine reference that it contains. Then, it will determine if the machine requires maintenance in the near future based on a set of criteria. Processing will mainly consist of a complex filtration process to filter machines that require attention. After this filtration, the pipeline will output its results to a newly created staging topic, so it can be picked up by upcoming pipelines, as well as the results of the junction of all the machine properties to the next data stage *DS2*. A very critical note must be added in this scenario, which is that this use case is intentionally taking on extra complexity to show how it can be handled by the proposed architecture. In a real-life scenario, the predictive maintenance pipeline shown in Listing 4 would be divided into at least two different pipelines: the first would join all the required machine properties and send them forward into the next data stage, and the second pipeline would consume from that enriched state so it would only need to be concerned with the filtering part of the process. This single responsibility model allows better pipeline maintenance and readability, and thus sustainability and scalability.

The next two pipelines depicted in Listings 5 and 6 illustrate the usage of alerts in the data flow, that take in the data that flows to the staging topic and send an email or a push notification to a set of actors.

The full data flow is illustrated in Figure 31

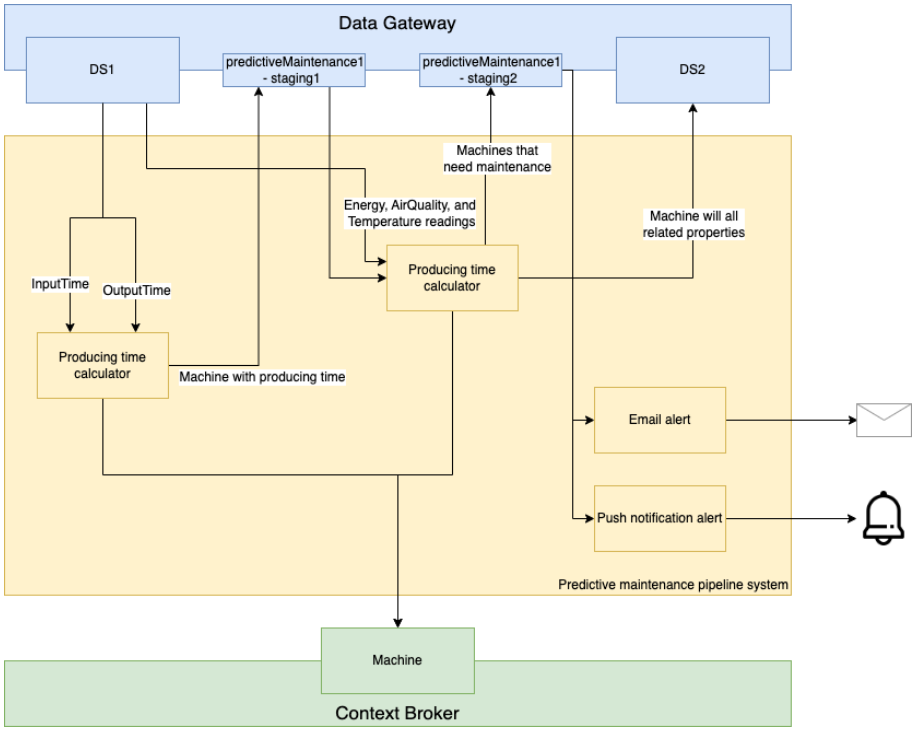

Figure 31: Predicitive maintenance pipeline system data flow

6 Conclusion

IIoT data management environments enclose many different challenges today. New patterns and technologies emerge, bringing security concerns about the data held, rising the need to understand where the data came from and how it affects the business. All these problems can be boiled down to an understanding of data and, more specifically, its ever-evolving context.

It was presented and discussed an methodology that addresses these problems. Design of this system focuses on iteratively enhancing data quality with decentralized components and centralized infrastructure. This provides a data management reference system to contribute to the reliability of the data quality level within the Industry 4.0 paradigm. The presented architecture was born out of an iterative process of redesign based on state-of-the-art architectures. Firstly the architecture was designed as a Data Lake divided by data sections to distribute the data across maturity levels, being processed by multiple pipelines. The second iteration resulted in a metadata management layer design on top of the existing Data Lake so data contextualization through semantics could be achieved. During this iteration, pipelines were also extended to take advantage of this metadata layer. This was so it could access the metadata defined and create a feedback with calculated metadata about data that was being processed. The final iteration of the architecture resulted in the incorporation of the Data Mesh design pattern. This pattern takes advantage of treating data as a product, allowing for increased focus on data quality, reusability, and interoperability.

FAIR data design principles are used to address observability challenges to achieve value-added data governance within IIoT real-time environments. In a data observability context, smart data models along with SAO ontologies are essential aspects of schema categories.

This methodology has proven that higher data quality can be achieved if the data management tool is built with data observability and context through semantics at the core of the design. Each component in the methodology plays a critical part in the overall system. The tecnhologies chosen for these components present state-of-the-art capabilities and are at the forefront of the respective fields.

The results of this research work are to be incorporated into a reference methodology for the development of data quality oriented big data architectures in industry.

6.1 Future work

Considering the development made in the work there are aspects that could be extended to improve the relavancy of this work.

The data ontolgy used can be further extended with other smart data models. Smart Energy can be used to extend the energy aspect of the environment, creating further device and manufactoring contextualization. Smart Environment can also be added to the ontolgy to the extend the data with waste management categories, as well as weather and water quality. Lastly, Smart Robotics might serve the purpose of describing robotics capabilities in some manufacturers.

Pipeline templates also can be extended to have a required schema, so they can be commonly shared across the platform. After that, those pipelines can be hosted in a repository that the users have access to, to search for existing pipelines in order to reuse them in new use cases.

The further developments for this project will also involve better interface and system interaction. The first suggestion should be the implementation of a form that would improve the way that metadata is entered for context broker entities, instead of the current HTTP requests that are required. Second, an important interface improvement should be a better data discoverability tool on top of the context broker that uses its , creating a middle component between the user and the context broker, facilitating communication between them. Lastly, this work aims to create an interface that takes in defined pipeline templates from the aforementioned repository and creates a parameterizable pipeline system. This task could be performed by users that do not have knowledge of pipeline creation, improving the way that data can be used and its delivery time.

References

[1] Simio, "Simio's 8 reasons to adopt industry 4.0." [Online]. Available: https://www.prnewswire.com/news-releases/simios-8-reasons-to-adopt-industry-4-0--300629039.html

[2] I. Sainz, "Diseñar para divergencias y convergencias. enfoques del dcg para los procesos de lectura por placer en la red," *Exploraciones, intercambios y relaciones entre el diseño y la tecnología*, pp. 57–79, 2019. [Online]. Available: http://zaloamati.azc.uam.mx/handle/11191/6874

[3] B. Moses, "The rise of data observability: Architecting the future of data trust," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1657. [Online]. Available: https://doi.org/10.1145/3488560.3510007

[4] "What is a data warehouse? — ibm." [Online]. Available: https://www.ibm.com/cloud/learn/data-warehouse

[5] "What is a lakehouse? - the databricks blog." [Online]. Available: https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

[6] N. Yuhanna, "Big data fabric drives innovation and growth — forrester," 2016. [Online]. Available: https://www.forrester.com/report/Big-Data-Fabric-Drives-Innovation-And-Growth/RES129473

[7] Z. Dehghani, "Data mesh principles and logical architecture," 2020. [Online]. Available: https://martinfowler.com/articles/data-mesh-principles.html

[8] A. Karkouch, H. Mousannif, H. A. Moatassime, and T. Noel, "Data quality in internet of things: A state-of-the-art survey," *Journal of Network and Computer Applications*, vol. 73, pp. 57–81, 9 2016.

[9] "What is apache kafka? how it works, benefits challenges." [Online]. Available: https://www.qlik.com/us/streaming-data/apache-kafka

[10] "Apache flink: Stateful computations over data streams." [Online]. Available: https://flink.apache.org/

[11] Cloudera, "Using watermark in flink — cdp private cloud." [Online]. Available: https://docs.cloudera.com/csa/1.2.0/flink-overview/topics/csa-watermark.html?

[12] "Cluster mode overview - spark 3.3.0 documentation." [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html

[13] M. Walker, "Apache beam - create data processing pipelines - datasciencecentral.com," 2016. [Online]. Available: https://www.datasciencecentral.com/apache-beam-create-data-processing-pipelines/

[14] "Druid — database for modern analytics applications." [Online]. Available: https://druid.apache.org/

[15] A. Mungekar, "Data storage and management project," 2 2019. [Online]. Available: https://www.researchgate.net/publication/330841309_Data_Storage_and_Management_Project

[16] "What is a data lake?" [Online]. Available: https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

[17] IBM, "What is hdfs? apache hadoop distributed file system." [Online]. Available: https://www.ibm.com/topics/hdfs

[18] A. Web Services. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[19] Y. E. Oktian, E. N. Witanto, and S.-G. Lee, "A conceptual architecture in decentralizing computing, storage, and networking aspect of iot infrastructure," *IoT*, vol. 2, pp. 205–221, 3 2021.

[20] M. Xu, J. M. David, and S. H. Kim, "The fourth industrial revolution: Opportunities and challenges," *International Journal of Financial Research*, vol. 9, 2018. [Online]. Available: http://ijfr.sciedupress.comURL:https://doi.org/10.5430/ijfr.v9n2p90

[21] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 10 2018.

[22] V. Reports, "Industrial internet of things (iiot) market is projected to reach usd 102460 million by 2028 at a cagr of 5.3% - valuates reports," 2022. [Online]. Available: https://www.prnewswire.com/in/news-releases/industrial-internet-of-things-iiot-market-is-projected-to-reach-usd-102460-million-by-2028-at-a-cagr-of-5-3-valuates-repo
html

[23] N. N. Misra, Y. Dixit, A. Al-Mallahi, M. S. Bhullar, R. Upadhyay, and A. Martynenko, "Iot, big data and artificial intelligence in agriculture and food industry," *IEEE Internet of Things Journal*, pp. 1–1, 5 2020.

[24] C. Liu, P. Nitschke, S. P. Williams, and D. Zowghi, "Data quality and the internet of things," *Computing*, vol. 102, pp. 573–599, 2 2020.

[25] P. Ambika, "Machine learning and deep learning algorithms on the industrial internet of things (iiot)," *Advances in Computers*, vol. 117, pp. 321–338, 1 2020.

[26] E. Adi, A. Anwar, Z. Baig, S. Zeadally, E. Adi, A. Anwar, Z. Baig, and S. Zeadally, "Machine learning and data analytics for the iot," 2020.

[27] B. Inmon, *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*, 1st ed. Denville, NJ, USA: Technics Publications, LLC, 2016.

[28] J. Byabazaire, G. O'hare, and D. Delaney, "Data quality and trust: Review of challenges and opportunities for data sharing in iot," *Electronics (Switzerland)*, vol. 9, pp. 1–22, 12 2020.

[29] Y. B. Lin, Y. W. Lin, J. Y. Lin, and H. N. Hung, "Sensortalk: An iot device failure detection and calibration mechanism for smart farming," *Sensors (Switzerland)*, vol. 19, 11 2019.

[30] P. Kodeswaran, R. Kokku, S. Sen, and M. Srivatsa, "Idea: A system for efficient failure management in smart iot environments *," 2016. [Online]. Available: http://dx.doi.org/10.1145/2906388.2906406

[31] S. Shankar and A. G. Parameswaran, "Towards observability for production machine learning pipelines," 2021.

[32] I. Lee, "The internet of things for enterprises: An ecosystem, architecture, and iot service business model," *Internet of Things*, vol. 7, p. 100078, 2019.

[33] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios," *IEEE Access*, vol. 8, pp. 23 022–23 040, 2020.

[34] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, 10 2010.

[35] M. Abdel-Basset, G. Manogaran, and M. Mohamed, "Internet of things (iot) and its impact on supply chain: A framework for building smart, secure and efficient systems," *Future Generation Computer Systems*, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2018.04.051.

[36] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, "A review of machine learning and iot in smart transportation," *Future Internet 2019, Vol. 11, Page 94*, vol. 11, p. 94, 4 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/4/94/htmhttps://www.mdpi.com/1999-5903/11/4/94

[37] S. Adhya, D. Saha, A. Das, J. Jana, and H. Saha, "An iot based smart solar photovoltaic remote monitoring and control unit," *2016 2nd International Conference on Control, Instrumentation, Energy and Communication, CIEC 2016*, pp. 432–436, 7 2016.

[38] V. Loia, A. Tommasetti, H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, and P. Siano, "Iot-based smart cities: a survey," 2016. [Online]. Available: https://www.researchgate.net/publication/301790173

[39] S. Selvaraj and S. Sundaravaradhan, "Challenges and opportunities in iot healthcare systems: a systematic review," *SN Applied Sciences*, vol. 2, pp. 1–8, 1 2020. [Online]. Available: https://link.springer.com/article/10.1007/s42452-019-1925-y

[40] L. S. Vailshery, "Iot connected devices worldwide 2019-2030 — statista," 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[41] J. Delsing, J. Eliasson, J. V. Deventer, H. Derhamy, and P. Varga, "Enabling iot automation using local clouds," *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016*, pp. 502–507, 2 2017.

[42] A. Kanawaday and A. Sane, "Machine learning for predictive maintenance of industrial machines using iot sensor data," *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, vol. 2017-November, pp. 87–90, 4 2018.

[43] M. Mohammadi, G. S. Member, A. Al-Fuqaha, S. Member, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," 2018.

[44] M. Samaniego, U. Jamsrandorj, and R. Deters, "Blockchain as a service for iot," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2016, pp. 433–436.

[45] M. Hu, X. Luo, J. Chen, Y. C. Lee, Y. Zhou, and D. Wu, "Virtual reality: A survey of enabling technologies and its applications in iot," *Journal of Network and Computer Applications*, vol. 178, p. 102970, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520304215

[46] S. Li, L. D. Xu, and S. Zhao, "5g internet of things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1–9, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X18300037

[47] A. Dorri, R. Jurdak, P. Gauravaram, and S. S. Kanhere, "Blockchain for iot security and privacy: The case study of a smart home cryptanalysis of hash functions view project hybrid ensemble learning for triggering of gps in long-term tracking applications view project see profile blockchain for iot security and privacy: The case study of a smart home," 2017. [Online]. Available: https://www.researchgate.net/publication/312218574

[48] P. C. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin, "A cloud native solution for dynamic auto scaling of mme in lte," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–7.

[49] A. Haleem, M. Javaid, R. P. Singh, S. Rab, and R. Suman, "Hyperautomation for the enhancement of automation in industries," *Sensors International*, vol. 2, p. 100124, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666351121000450

[50] BMBF, "Industrie 4.0 - bmbf." [Online]. Available: https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0

[51] C. Wagner, J. Grothoff, U. Epple, R. Drath, S. Malakuti, S. Grüner, M. Hoffmeister, and P. Zimermann, "The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pp. 1–8, 6 2017.

[52] Y. Tan, S. Takakuwa, W. . Yang, Y. . Tan, K. . Yoshida, and S. Takakuwa, "Digital twin-driven simulation for a cyber-physical system in industry 4.0 business process management project view project manufacturing simulation project view project digital twin-driven simulation for a cyber-physical system in industry 4.0," 2018. [Online]. Available: https://www.researchgate.net/publication/329043686

[53] C. Semeraro, M. Lezoche, H. Panetto, and M. Dassisti, "Digital twin paradigm: A systematic literature review," *Computers in Industry*, vol. 130, p. 103469, 9 2021.

[54] H. Lasi, P. Fettke, H. G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business Information Systems Engineering 2014 6:4*, vol. 6, pp. 239–242, 6 2014. [Online]. Available: https://link.springer.com/article/10.1007/s12599-014-0334-4

[55] J. Conway, “The industrial internet of things: An evolution to a smart manufacturing enterprise,” 2015.

[56] R.-T. Innovations, “Frequently asked questions (faq) industrial internet of things what is the industrial internet of things? how is it different from the internet of things?” 2015. [Online]. Available: http://www.iiconsortium.org/.

[57] F. X. Diebold, “Pier working paper 13-003 by a personal perspective on the origin (s) and development of “ big data ”: The phenomenon , the term , and the discipline ,” 2012.

[58] M. Cox and D. Ellsworth, “Application-controled demand paging for out of core visualization,” *Cox CONFERENCE*, 1997.

[59] G. Press. (2013) A very short history of big data. [Online]. Available: https://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/#7294b9265a18

[60] A. Gandomi and M. Haider, “International journal of information management beyond the hype : Big data concepts , methods , and analytics,” *International Journal of Information Management*, vol. 35, pp. 137–144, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ijinfomgt.2014.10.007

[61] N. Miloslavskaya and A. Tolstoy, “Big data , fast data and data lake concepts 2 big data concept,” vol. 88, pp. 300–305, 2016.

[62] D. Laney, “3d data management: Controlling data volume, velocity, and variety.” 2001.

[63] N. Golov and L. Rönnbäck, “Computer standards & interfaces big data normalization for massively parallel processing databases,” *Computer Standards & Interfaces*, vol. 54, pp. 86–93, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.csi.2017.01.009

[64] J. Desjardins, “How much data is generated each day? — world economic forum,” p. 1, 2019. [Online]. Available: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/

[65] S. Axryd. (2019) Why 85% of big data projects fail — digital news asia. [Online]. Available: https://www.digitalnewsasia.com/insights/why-85-big-data-projects-fail

[66] J. Hipp, U. Güntzer, and U. Grimmer, “Data quality mining - making a virtue of necessity,” 01 2001.

[67] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. S. Mittal, and V. Munigala, “Overview and importance of data quality for machine learning tasks,” 2020. [Online]. Available: https://doi.org/10.1145/3394486.3406477

[68] L. Berti-Equille, “Measuring and modelling data quality for quality-awareness in data mining urban data analytics view project data mining view project,” 2007. [Online]. Available: https://www.researchgate.net/publication/226642874

[69] L. Sebastian-Coleman, *Measuring data quality for ongoing improvement: A data quality assessment framework*, L. Sebastian-Coleman, Ed. Elsevier, 1 2013.

[70] IBM, “Internet of things architecture: Reference diagram - ibm cloud architecture center,” 2022. [Online]. Available: https://www.ibm.com/cloud/architecture/architectures/iotArchitecture/reference-architecture/

[71] M. Cosner, “Azure iot reference architecture - azure reference architectures — microsoft docs,” 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot

[72] Z. Dehghani, “How to move beyond a monolithic data lake to a distributed data mesh,” 5 2019. [Online]. Available: https://martinfowler.com/articles/data-monolith-to-mesh.html

[73] B. Diène, J. J. P. C. Rodrigues, O. Diallo, E. L. Hadji, M. Ndoye, and V. V. Korotaev, “Data management techniques for internet of things,” 2019. [Online]. Available: https://doi.org/10.1016/j.ymssp.2019.106564

[74] P. Edastama, A. Dudhat, and G. Maulani, “Use of data warehouse and data mining for academic data; a case study at a national university,” *International Journal of Cyber and IT Service Management (IJCITSM)*, vol. 1, pp. 206–215, 2021. [Online]. Available: https://iiast-journal.org/ijcitsm/index.php/IJCITSM/article/view/55

[75] A. Konikov, E. Kulikova, and O. Stifeeva, “Research of the possibilities of application of the data warehouse in the construction area,” 2018. [Online]. Available: https://doi.org/10.1051/matecconf/201825103062

[76] N. Garcelon, A. Neuraz, R. Salomon, H. Faour, V. Benoit, A. Delapalme, A. Munnich, A. Burgun, and B. Rance, “A clinician friendly data warehouse oriented toward narrative reports: Dr. warehouse,” *Journal of Biomedical Informatics*, vol. 80, pp. 52–63, 4 2018.

[77] M. Derakhshannia, C. Gervet, H. Hajj-Hassan, A. Laurent, and A. Martin, “Life and death of data in data lakes: Preserving data usability and responsible governance,” vol. 11938 Lncs. Springer, 2019, pp. 302–309.

[78] P. Sawadogo and J. Darmont, “On data lake architectures and metadata management,” *Journal of Intelligent Information Systems*, vol. 56, pp. 97–120, 2 2021. [Online]. Available: https://arxiv.org/pdf/2107.11152.pdf

[79] E. A. Evans, E. Delorme, K. D. Cyr, and K. H. Geissler, “The massachusetts public health data warehouse and the opioid epidemic: A qualitative study of perceived strengths and limitations for advancing research,” *Preventive Medicine Reports*, vol. 28, p. 101847, 8 2022.

[80] thoughtworks, “Enterprise data warehouse — technology radar — thoughtworks,” 2014. [Online]. Available: https://www.thoughtworks.com/radar/platforms/enterprise-data-warehouse

[81] J. S. R. No, “Warehouse data system analysis pt. kanaan global indonesia 1 st tino feri efendi, 2 nd mutiya krisanty 12 institut teknologi bisnis aas indonesia surakarta,” *International Journal of Computer and Information System (IJCIS) Peer Reviewed-International Journal*, vol. 01, pp. 2745–9659, 2020. [Online]. Available: https://ijcis.net/index.php/ijcis/index

[82] I. Terrizzano, P. Schwarz, M. Roth, J. E. Colino, and S. Jose, “Data wrangling : The challenging journey from the wild to the lake,” 2015.

[83] A. Alserafi and A. Abell, "Towards information profiling : Data lake content metadata management," 2016.

[84] B. Sharma, *Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases Ben*, 2018.

[85] M. Armbrust, A. Ghodsi, R. Xin, M. Zaharia, and U. Berkeley, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," 2021.

[86] N. Yuhanna and B. Szekely, "Ty — forrester surfacing insights in a data fabric with knowledge graph," 2021.

[87] "What is a data fabric? — ibm," 2022. [Online]. Available: https://www.ibm.com/topics/data-fabric

[88] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[89] I. A. Machado, C. Costa, and M. Y. Santos, "Data mesh: Concepts and principles of a paradigm shift in data architectures," *Procedia Computer Science*, vol. 196, pp. 263–271, 2021.

[90] R. V. Zicari, "Big data: Challenges and opportunities," 2014. [Online]. Available: http://odbms.org/wp-content/uploads/2013/07/Big-Data.Zicari.pdf

[91] M. D. Wilkinson, "Comment: The fair guiding principles for scientific data management and stewardship," 2016. [Online]. Available: http://figshare.com

[92] A. Alserafi and A. Abell, "Towards information profiling : Data lake content metadata management."

[93] L. Zhang, D. Jeong, S. Lee, E. Al-Masri, C.-H. Chen, A. Souri, and O. Kotevska, "Data quality management in the internet of things," *Sensors 2021, Vol. 21, Page 5834*, vol. 21, p. 5834, 8 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/17/5834/htmhttps://www.mdpi.com/1424-8220/21/17/5834

[94] C. Liu, P. Nitschke, S. P. Williams, and D. Zowghi, "Data quality and the internet of things," *Computing*, vol. 102, pp. 573–599, 2 2020.

[95] S. Kim, R. P. D. Castillo, I. Caballero, J. Lee, C. Lee, D. Lee, S. Lee, and A. Mate, "Extending data quality management for smart connected product operations," *IEEE Access*, vol. 7, pp. 144 663–144 678, 2019.

[96] D. Hoyle, *ISO 9000 Quality Systems Handbook-updated for the ISO 9001: 2015 standard: Increasing the Quality of an Organization's Outputs*. Taylor & Francis, 2017. [Online]. Available: https://books.google.ie/books?id=vkwrDwAAQBAJ

[97] J. M. Juran and A. B. Godfrey, *Juran's quality handbook*, ser. Juran's quality handbook, 5e. McGraw Hill, 1999. [Online]. Available: http://books.google.de/books?id=beVTAAAAMAAJ

[98] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Source: Journal of Management Information Systems*, vol. 12, pp. 5–33, 1996.

[99] L. Cai and Y. Zhu, "The challenges of data quality and data quality assessment in the big data era," *Data Science Journal*, vol. 14, 5 2015. [Online]. Available: http://datascience.codata.org/articles/10.5334/dsj-2015-002/

[100] P. Ceravolo, A. Azzini, M. Angelini, T. Catarci, P. Cudré-Mauroux, E. Damiani, M. V. Keulen, A. Mazak, M. Keulen, J. Mustafa, G. Santucci, K.-U. Sattler, M. Scannapieco, M. Wimmer, R. Wrembel, and F. Zaraket, "Big data semantics," *Journal on Data Semantics*, 2018. [Online]. Available: https://doi.org/10.1007/s13740-018-0086-2

[101] J. Bahrke and C. Manoury, "Data act: measures for a fair and innovative data economy," 2 2022. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/ip_22_1113

[102] Fiware, "Fiware-orion." [Online]. Available: https://fiware-orion.readthedocs.io/en/master/

[103] "About fiware — fiware." [Online]. Available: https://www.fiware.org/about-us/

[104] F. Viola, F. Antoniazzi, C. Aguzzi, C. Kamienski, and L. Roffia, "Mapping the ngsi-ld context model on top of a sparql event processing architecture: Implementation guideliness," *Conference of Open Innovation Association, FRUCT*, vol. 2019-April, pp. 493–501, 5 2019.

[105] Y. Tolcha, A. Kassahun, T. Montanaro, D. Conzon, G. Schwering, J. Maselyne, and D. Kim, "Towards interoperability of entity-based and event-based iot platforms: The case of ngsi and epcis standards," *IEEE Access*, vol. 9, pp. 49 868–49 880, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9388690

[106] "Faqs – smart data models." [Online]. Available: https://smartdatamodels.org/index.php/faqs/

[107] "Smart data models." [Online]. Available: https://github.com/smart-data-models

[108] N. Garg, *Apache Kafka*, T. Gaitonde, S. Mukherjee, A. Nair, K. Pai, A. Paiva, and A. Shetty, Eds., 2013.

[109] H. Wu, Z. Shang, and K. Wolter, "Performance prediction for the apache kafka messaging system," *Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019*, pp. 154–161, 8 2019.

[110] J. Kreps, L. Corp, N. Narkhede, and J. Rao, "Kafka: a distributed messaging system for log processing," 2011.

[111] G. Cugola and A. Margara, "Processing flows of information," *ACM Computing Surveys (CSUR)*, vol. 44, 6 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2187671.2187677

[112] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 3 2019.

[113] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 2785–2792, 12 2015.

[114] O.-C. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernández, "Spark versus flink: Understanding performance in big data analytics frameworks spark versus flink: Understanding performance in big data analytics frameworks spark versus flink: Understanding performance in big data analytics frameworks," 2016. [Online]. Available: https://hal.inria.fr/hal-01347638v2

[115] E. Nazari, M. H. Shahriari, and H. Tabesh, "Bigdata analysis in healthcare: Apache hadoop , apache spark and apache flink," *Frontiers in Health Informatics*, vol. 8, p. 14, 7 2019. [Online]. Available: www.ijmi.ir

[116] A. Sharma, D. Puri, M. Kumar, and G. Soni, "Implementation and comparison of big data analysis on large dataset using spark and flink," *Lecture Notes in Electrical Engineering*, vol. 828, pp. 385–394, 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-16-7985-8_40

[117] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "A comparison on scalability for batch big data processing on apache spark and apache flink," *Big Data Analytics*, vol. 2, 12 2017.

[118] I. S. Bongo, "Universidade da beira interior engenharia avaliação de desempenho das plataformas apache hadoop, apache spark e apache flink usando o benchmark hibench-master 7," 2019.

[119] P. Carbone, A. Katsifodimos, Kth, S. Sweden, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink™: Stream and batch processing in a single engine," 2015. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-198940

[120] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández, F. Fernández-Moctezuma, R. Lax, S. Mcveety, D. Mills, F. Perry, E. Schmidt, and S. W. Google, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," 2015.

[121] P. Carbone, A. Katsifodimos, Kth, S. Sweden, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "State management in apache flink," *Proceedings of the VLDB Endowment*, vol. 10, pp. 1718–1729, 8 2017. [Online]. Available: https://dl.acm.org/doi/10.14778/3137765.3137777

[122] "Apache spark™ - unified engine for large-scale data analytics." [Online]. Available: https://spark.apache.org/

[123] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, pp. 145–164, 11 2016. [Online]. Available: https://link.springer.com/article/10.1007/s41060-016-0027-9

[124] B. Bengfort and J. Kim, *Data Analytics with Hadoop*. O'Reilly Media, Inc., 6 2016. [Online]. Available: https://www.oreilly.com/library/view/data-analytics-with/9781491913734/

[125] "Apache beam." [Online]. Available: https://beam.apache.org/

[126] S. Li, P. Gerver, J. Macmillan, D. Debrunner, W. Marshall, and K.-L. Wu, "Challenges and experiences in building an efficient apache beam runner for ibm streams," vol. 11, pp. 1742–1754, 2018. [Online]. Available: https://doi.org/10.14778/3229863.3229864

[127] “Apache apex.” [Online]. Available: https://apex.apache.org/

[128] “Dataflow — google cloud.” [Online]. Available: https://cloud.google.com/dataflow

[129] “Apache gearpump (incubating): Overview.” [Online]. Available: http://gearpump.github.io/overview.html

[130] “Apache hadoop.” [Online]. Available: https://hadoop.apache.org/

[131] “alibaba/jstorm: Enterprise stream process engine.” [Online]. Available: https://github.com/alibaba/jstorm

[132] “Ibm streams - overview - portugal — ibm.” [Online]. Available: https://www.ibm.com/pt-en/cloud/streaming-analytics

[133] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, “Druid: A real-time analytical data store,” *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 157–168, 2014.

[134] F. Yang, G. Merlino, and X. Léauté, “The radstack: Open source lambda architecture for interactive analytics.”

[135] A. Chauhan, “Ijert-a review on various aspects of mongodb databases a review on various aspects of mongodb databases,” *IJERT Journal International Journal of Engineering Research and Technology*, 2019. [Online]. Available: www.ijert.org

[136] “Mongodb kafka connector — mongodb kafka connector.” [Online]. Available: https://www.mongodb.com/docs/kafka-connector/current/

[137] S. Kolozali, M. Bermudez, and P. Barnaghi, “Stream annotation ontology,” 5 2016. [Online]. Available: http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao

[138] M. Fischer, T. Iggena, and D. Kümper, “Quality ontology,” 2016. [Online]. Available: https://mobcom.ecs.hs-osnabrueck.de/cp_quality/#StreamAnalysis

[139] “Api walkthrough (v2) - fiware-orion.” [Online]. Available: https://fiware-orion.readthedocs.io/en/1.15.0/user/walkthrough_apiv2/index.html

[140] P. Hammant, “Introduction,” 2017. [Online]. Available: https://trunkbaseddevelopment.com/#one-line-summary