frontiers in Chemical Engineering

TECHNOLOGY AND CODE
published: 16 June 2021
doi: 10.3389/fceng.2021.665545

AVUBDI: A Versatile Usable Big Data Infrastructure and Its Monitoring Approaches for Process Industry

Sabrina Luftensteiner, Michael Mayr, Georgios C. Chasparis and Mario Pichler*

Software Competence Center Hagenberg GmbH, Hagenberg, Austria

The amount of sensors in process industry is continuously increasing as they are getting faster, better and cheaper. Due to the rising amount of available data, the processing of generated data has to be automatized in a computationally efficient manner. Such a solution should also be easily implementable and reproducible independently of the details of the application domain. This paper provides a suitable and versatile usable infrastructure that deals with Big Data in the process industry on various platforms using efficient, fast and modern technologies for data gathering, processing, storing and visualization. Contrary to prior work, we provide an easy-to-use, easily reproducible, adaptable and configurable Big Data management solution with a detailed implementation description that does not require expert or domain-specific knowledge. In addition to the infrastructure implementation, we focus on monitoring both infrastructure inputs and outputs, including incoming data of processes and model predictions and performances, thus allowing for early interventions and actions if problems occur.

Keywords: big data infrastructure, process monitoring, sensor data processing, process industry, containerization

OPEN ACCESS

Edited by:
Davide Fissore,
Politecnico di Torino, Italy

Reviewed by:
Santiago Muiños Landin,
Northwestern Metallurgical Research Association, Spain
Hector Diego Estrada-Lugo,
University of Liverpool, United Kingdom

****Correspondence:***
Sabrina Luftensteiner
sabrina.luftensteiner@scch.at

Specialty section:
This article was submitted to Computational Methods in Chemical Engineering, a section of the journal Frontiers in Chemical Engineering

Received: *08 February 2021*
Accepted: *17 May 2021*
Published: *16 June 2021*

Citation:
Luftensteiner S, Mayr M, Chasparis GC and Pichler M (2021) AVUBDI: A Versatile Usable Big Data Infrastructure and Its Monitoring Approaches for Process Industry. Front. Chem. Eng. 3:665545. doi: 10.3389/fceng.2021.665545

1 INTRODUCTION

It has recently been recognized that machine learning and data analytics play a critical role in realizing long-term sustainability goals in the process industry. The European Union 2030's goals proposed rather ambitious key targets in the https://ec.europa.eu/clima/policies/strategies/2030_en which includes a 40% decrease in greenhouse gas emissions, 32% share for renewable energy and 32. 5% improvement in energy efficiency. Such goals require the active participation and involvement of the whole industrial sector, whose energy consumption reached the 24.6% of the total energy consumption in Europe in 2017, according to the European Environment Agency (2017). However, these goals cannot be achieved without large scale digitization of the process industry and a subsequent data management analysis for efficient decision making, which pose significant challenges for industrial manufacturers, as observed by Zeng and Yin (2017).

Indeed, industrial digitization is expanding with a relatively fast pace, given that sensors are getting faster, better and cheaper Reis and Gins (2017). This trend is rather evident in every aspect of our lives, given that the amount of produced and recorded data has reached 60% of annual growth, according to Srinivasan and Rajeev (2012). To stay competitive while following efficiency targets, an intelligently supervised and controlled manufacturing production line is a necessity. To enable Big Data processing in terms of Industry 4.0, further development of the required infrastructures, concerning loading, processing, storing, and visualizing the data, is required. Furthermore, such processing cycle should be performed automatically in a near real-time fashion in a versatile, scalable and easily deployable manner. Such a transformation requires substantial innovation in the context

of Big Data management and analytics. Furthermore, in order to reach a broader community, it should be deployable on various systems.

As it will be discussed in detail in the forthcoming related work section, several researchers have emphasized the need for Big Data analytics in the process industry and cyber-physical systems, cf., Ge et al. (2017); Sarnovsky et al. (2018); He and Wang (2018); Wang et al. (2018); Xu and Duan (2019); ur Rehman et al. (2019) and have described the induced requirements and research challenges, as in ur Rehman et al. (2019). There have been several research efforts proposing specific Big Data infrastructure and deployment solutions, by describing the necessary technology stack, as for example in Sarnovsky et al. (2018); Xu and Duan (2019), without however providing details on the orchestration/configuration of such technologies. In fact, there has been limited work on providing detailed guidelines on the deployment and orchestration of Big Data infrastructure, making its realization a rather difficult task for non-experts.

Indeed configuring and deploying a Big Data infrastructure requires prior expertize and familiarity with these technologies. For this reason, our goal in this paper is to present a *versatile*, *scalable* and *easily deployable* Big Data infrastructure that is capable of running data processing pipelines as a basis for advanced machine learning and process mining algorithms. We identified key necessities for data analytics in the process industry and developed an infrastructure that covers the full-stack of data analytics, i.e., data gathering, preprocessing, exploring, visualizing, persisting, model building, and model deploying in real-time and historical data. The overall structure is as modular as possible, enabling the integration of custom data processing and visualization. It is worth noting that the focus in this paper is on the infrastructure design and its deployment, that enables a rather wide range of processing possibilities and a user-friendly environment for both non-experts and experts. In addition, all of the employed technologies/services are open-source. Finally, given that this paper addresses the issue of designing a versatile and user-friendly Big Data infrastructure, the specifics of the data analysis methodologies and the overall computational complexity of the pipeline are not addressed in this paper.

This research has been conducted under the European Union's Horizon 2020 project https://www.cogniplant-h2020.eu/. The overall goal of this project is the development of an innovative approach for advancing both the digitization as well as the intelligent management of the process industries. One of the main tasks of this project includes the development of a Big Data analytics platform that would allow for a near real-time data processing, predictive modeling, and process mining of the industrial processes. The goal is to improve the performance of those cognitive production plants through the derivation of corrective actions. The involved industrial processes include a chemical plant in Austria, an alumina refinery in Ireland, a cement production plant in Italy, and a metal manufacturer in Spain. Common Key Performance Indicators (KPIs) of such plants include energy and resource consumption, as well as CO_2 emissions.

In the remainder of this section, we present related work and the main contributions of this paper. *Methods* covers methods in connection to the existing Big Data infrastructures and technologies, including a brief insight into their advantages and disadvantages, especially with respect to their utility in the process industry. The presentation has been organized into four main topics: data management, storage, analysis and visualization. In this section, we also present our main contribution: a versatile usable big data infrastructure for process industry. *Results* presents various monitoring approaches that provide an insight into the processing results. Specifically, the input data and the performance predictions are presented. Finally, *Discussion* provides a discussion over the features of the proposed infrastructure and future development steps.

1.1 Related Work

As pointed out in Venkatram and Geetha (2017), "Big Data is something like a set of huge data sets which are complex and requires tedious work to capture, store, process and analyze them." The definition may differ in different application domains, depending on the size and rate of data, the methods/techniques used for data management, and data analysis. Big Data has recently attracted significant attention in academia and industry, and it is expected to play a significant role in Industry 4.0. It is indeed evident in many several domains that, in order for organizations to remain competitive, they have to invest in Big Data technologies with a large variety in the size and rate of data as well as large variety in data types, as mentioned in Ardagna et al. (2016).

The large variety of data volume and types, as well as the different needs of each application domain, may lead to different Big Data solution architectures. Even for the same application domain, there might be several alternative solution concepts and services that could be used. For example, we may use a mixture of technologies, going from NoSQL databases, like Cassandra or HBase, data preparation utilities like Paxata, and distributed parallel computing systems like Hadoop and Spark, as mentioned by Ardagna et al. (2016) and discussed more extensively in the forthcoming *Methods*. This variety in the availability of technologies and their selection for each application is one of the hindering factors in using Big Data. Simultaneously, this variety requires a large set of skills for implementing and managing a Big Data solution, which makes its implementation rather costly for many organizations.

Indeed, there has been a large literature on the design of Big Data management and analysis solutions, including, for example, the works of Ardagna et al. (2016); Venkatram and Geetha (2017); Sarnovsky et al. (2018); Xu and Duan (2019). In particular, Xu and Duan (2019) provides a generic overview of solutions for data-analytics and Big Data management, Venkatram and Geetha (2017) lay out an overview of technologies for an overall Big Data solution, Sarnovsky et al. (2018) provide the details of the Big Data technology stack, and Ardagna et al. (2016) which introduces Big-Data-Analytics-as-a-Service and a general description of the applications that could be used in the different parts of the Big Data pipeline.

Other relevant work, especially in the context of the process industry, includes the paper of ur Rehman et al. (2019) that

describes requirements and research challenges in Big Data processing and the paper of He and Wang (2018) that discusses recent developments and challenges in statistical process monitoring in manufacturing. Relevant work in other application domains include the paper of Zhou et al. (2016) that provides findings regarding various Big Data driven smart-grid management approaches, the paper of Poddubny et al. (2017) that developed an approach for a distributed Big Data infrastructure using smart agents and parallel computing based on heterogeneous sources, and the paper of Rios and Diguez, (2014) that proposed a Big Data infrastructure for analyzing data generated by Wireless Sensor Networks based upon Hadoop and Storm.

Summarizing, we may argue that one common characteristic of the aforementioned literature is the fact that implementation aspects and details of the proposed technologies and their orchestration are not discussed in detail. In a way, expert/domain knowledge is still required for implementing such Big Data solutions, rendering this problem an extremely challenging task for non-experts.

1.2 Contributions

As we discussed in the previous section, current literature offers a broad range of Big Data solutions, also within the context of the process industry. However, either they mostly focus on one part of the infrastructure without offering complete infrastructure compositions, or, even if they do, expert knowledge is still required for implementing the proposed solutions. For this reason, in this paper, our goal is to provide a unified and comprehensive Big Data Management Schema for tackling a broad range of machine learning tasks in industrial processes and with possibly heterogeneous data sources. A detailed presentation and discussion over the selection of the used technologies and their configuration/orchestration are also provided, thus significantly reducing the level of expert knowledge required. This way, the presented schema can easily be reproduced and adapted to various use-cases. In particular, our contributions can be summarized as follows:

1. We provide a detailed insight into state-of-the-art technologies for building a big data infrastructure;
2. We propose and describe the structure of *a versatile usable Big Data infrastructure*, or briefly AVUBDI, relevant for a wide range of application domains including process industry;
3. We use open-source tools for the structure of AVUBDI and enable a user-friendly environment for non-experts as well as experts;
4. We describe in detail the data flow and types within the proposed AVUBDI;
5. We describe in detail the monitoring approaches for industrial plants using AVUBDI.

2 METHODS

In this section, we first discuss already existing approaches and state-of-the-art technologies for Big Data management. We also provide a comparative analysis of these methodologies with respect to their versatile usability. This presentation is organized into four topics, namely data management, storage, analysis and visualization. In the second part of this section, we present the structure and implementation of the proposed versatile usable Big Data infrastructure.

2.1 State-Of-The-Art Technologies

This subsection covers various state-of-the-art technologies needed for the implementation of a Big Data infrastructure, ranging from data management to visualization tools. As multiple technologies exist for various parts of the infrastructure, the advantages and disadvantages are discussed regarding our setting for a versatile usable Big Data infrastructure in process industry.

2.1.1 Raw-Data Storage

As it is difficult to deal with high amounts of data in various and varying formats in traditional database systems, data lakes are likely used instead. Data lakes used for data storage are more complex to handle as they may store unprocessed data in its raw format or unstructured data according to Miloslavskaya and Tolstoy (2016). At first glance, unprocessed and raw data may be seen as a problem as it contains a lot of possibly unnecessary information, which is currently not needed and takes up space. In industrial settings, it may still be an advantage as the complete data is stored without trimming, containing all of the available information and therefore enabling various approaches for further processing. Low-level sensor data, for example, may be used in different ways to gain a higher level of abstraction for machine learning. Usually, data entries also include meta-information about machines or specific processes. Using higher-level data in combination with available meta-information allows the generation of event logs, which may be used in process mining to better understand processes, machine behavior and their deviations.

2.1.2 Data Management and Routing

The management of data is crucial in Big Data infrastructures as it is not feasible to process Big Data in one single application due to timely or even computational restrictions. Instead, it is reasonable to create various processing steps, which exchange messages with the needed data to avoid bottlenecks, to increase performance and to provide the possibility for extensions and flexible adaptations. We refer to data management in our setting as 1) the gathering of data using specific sources and 2) the message routing within the infrastructure as well as 3) data or message format specifications for further processing steps. Especially for process industry, it is important to create a flexible data management as it is likely that process paths and/or information differ over time due to changing requirements.

Apache Kafka.[1], ActiveMQ.[2] and RabbitMQ.[3] are popular technologies for handling the asynchronous message routing part of an infrastructure. Apache Kafka is an open-source publish-subscribe messaging system, which has good horizontal scaling capability. Message producers send their messages to a topic in a

[1]https://kafka.apache.org/
[2]http://activemq.apache.org/
[3]https://www.rabbitmq.com/

broker, which are then pulled by consumers. Apache Kafka is able to persist messages for a pre-defined time, has a high throughput, may be distributed and acts in real-time (Garg (2013)). ActiveMQ is a general-purpose message broker that supports several messaging protocols, e.g., AMQP, STOMP or MQTT. In contrast to Apache Kafka, ActiveMQ is push oriented, which leads to problems with slow brokers as messages are kept in RAM until they are processed. Furthermore, ActiveMQ is a traditional messaging system in contrast to Apache Kafka's distributed processing approach (Christudas (2019)). RabbitMQ implements a broker architecture, which queues messages on a central node before they are sent to clients. This approach enables easy usage and deployment of RabbitMQ, but also leads to a less scalable and slower system compared to ActiveMQ and Apache Kafka according to Dobbelaere and Esmaili (2017). Comparing throughput and latency over those three technologies, Apache Kafka leads with its ability to handle huge amounts of data with very low latency. This characteristic is important in an industrial setting as the amount of data to process may vary heavily between use-cases.

The next important step, after the definition of a message routing tool, is the assurance of an appropriate data format that is passed between different processing steps. It is necessary to check the message content as changing environments in industrial settings may lead to changing content, e.g., new features are added, and existing processing steps may have problems handling the new data format. For this reason, Confluent developed a tool called Schema Registry, which manages message schemas and is directly connected with Apache Kafka for the surveillance of schema evolution. It provides a serving layer for metadata, covers a RESTful interface for storing and retrieving Avro®, JSON Schema, and Protobuf schemas and is able to hold the whole infrastructure in a consistent state according to https://docs.confluent.io/current/schema-registry/index.html. Using this tool, it is possible to define schemas, validate messages and follow the evolution of schemas, e.g., integration of a new feature. The coordination of processes in distributed applications requires high effort. As Hunt et al. (2010) describes, Zookeeper provides a centralized service for the maintenance of configuration information, the naming and the provisioning of distributed synchronization and group services.

2.1.3 Data Storage

In most use-cases data has to be stored either temporarily or permanently into a database. The stored data represents a broad variety, including data that is later used for further processing, historical data, and results from prediction models. Various database approaches also exist for different use-case requirements. Especially in the field of Big Data, it is necessary to select the most appropriate technology, given that several database technologies are excluded due to their poor performance with respect to capacity and access speed. Cassandra.[4], InfluxDB.[5] and CrateDB.[6] are capable of dealing with Big Data, especially data gathered in industrial environments.

Lakshman and Malik (2010) define Cassandra as a distributed storage system for managing very large amounts of structured data, which is spread across multiple servers. It provides high availability, but lacks some features of a relational database. Cassandra uses replication to achieve high availability and durability, where each data sample is replicated at a predefined number of hosts. InfluxDB is an open-source schemaless time-series database and enables a high throughput regarding write and read actions, according to Naqvi et al. (2017). It incorporates so called measurements instead of tables, and it focuses on timestamps and their corresponding keys and features. In contrast to Cassandra, InfluxDB does not need a pre-defined schema and measurement tables are flexible regarding the integration of new features. This is especially interesting in changing environments, e.g., industrial plants. CrateDB is a relatively new database and represents itself as taking the most advantageous features of SQL and NoSQL databases for industrial usage. It has a high scalability potential and provides dynamic schema adaptations, similar to InfluxDB. CrateDB supports distribution across multiple servers and also covers relational operations, such as joins.

Those three databases have overall very strong advantages that address rather well the challenges met in Big Data applications. In order to choose the best one for an industrial application may not be a simple task. On the one hand, Cassandra is easily scaleable and can be distributed across multiple servers, but it lacks the ability of flexible schemas, which InfluxDB offers. CrateDB incorporates both advantages of the distributed approach and the flexible schemas, but it still has several issues regarding its implementation and its throughput.

2.1.4 Data Processing and Analysis

Another important topic during the development of a Big Data infrastructure is the selection of the most appropriate data processing and analysis tool. The integrated tool has to be able to deal with large amounts of data in a near real-time manner and provide fast answers regarding analytical evaluations or predictions. This is especially important in industrial settings so downtimes of machines, poor quality of products or emissions during production are minimized. Fulfilling such goals will enable industries to improve their performance indicators.

Popular tools for data processing and analytical evaluations are Apache Storm.[7], Apache Spark.[8], Apache Flink.[9], Apache Samza.[10] and Apache Drill.[11].

Apache Storm is a real-time distributed processing system, which can process the streams of data fast, while it still provides easy usage. It is highly scalable, offers low latency with guaranteed data processing and allows developers to develop their logic virtually in any programming language according to Iqbal and Soomro (2015).

[4]https://github.com/apache/cassandra
[5]https://github.com/influxdata/influxdb
[6]https://github.com/crate/crate

[7]https://storm.apache.org/
[8]https://spark.apache.org/
[9]https://flink.apache.org/
[10]http://samza.apache.org/
[11]https://drill.apache.org/

TABLE 1 | Comparison of analytical tools for Big Data. The categories are chosen upon influence within a Big Data system and range from " ++ " (very good fulfillment) to "–" (not fulfilled). Additional information is given according to the requirements.

Features	Storm	Flink	Spark	Samza	Drill	Hadoop
Real-Time	+	++	++	~	+	~
Distributed Processing	+	+	+	+	+	++
Running Analytics	+	++ MLFlink	++ MLLib	+	+	+
Streaming Type	+ Micro batches	++ Event streaming	+ Micro batches	+ Micro batches	Mini batches	~ Batch and mini batches
Latency	++	++	+	+	+	Component
Throughput	++	++	+	++	+	Component
Fault Tolerance	++ Auto-restart	~ Checkpoint	~ Checkpoint	~ Checkpoint	~	¯ Replication
Message Delivery	++ Exactly once	++ Exactly once	++ Exactly once	¯ At-least once	n.A	Component
Documentation Community	+	¯ Small	++ Big	~ Medium	+	++ Big
Data sources	+	+	++ HDFS, kafka	++ kafka, kinesis, ...	++ schema-free	Component
Scalability	++ auto-scaling	++ auto-scaling	+	++	+	Component

Apache Spark is a next generation engine for Big Data analytics and alleviates key challenges of data preprocessing, iterative algorithms and interactive analytics among others. The data can be processed through a general directed acyclic graph of operators using rich sets of transformations and actions. Apache Spark supports a variety of transformations and eases the data preprocessing especially for Big Data. Furthermore, Apacha Spark provides an adapted library of machine learning algorithms for faster performance, the so called MLLib, Salloum et al. (2016).

Apache Flink is an open-source system for processing streaming and batch data, where real-time analytics are also supported. It includes continuous data pipelines, historic data processing, a. k.a. batch processing, and fault-tolerant dataflow pipelines. Similar to Apache Spark, Apache Flink also provides its own high-performance machine learning library called MLFlink, Carbone et al. (2015).

Apache Samza is a distributed system for stateful and fault-tolerant stream processing. It is able to scale to massive state sizes, e.g., hundreds of TB, due to the use of partitioned local state in combination with a low-overhead background change-log mechanism. Next to the processing of infinite data streams, it also allows finite datasets as a stream, e.g., *via* Kafka or file system, without having to change the code, Noghabi et al. (2017).

Apache Drill is a distributed system for interactive analysis of large datasets, which is designed to handle Petabytes of data spread across numerous servers. Its goal is to respond to queries in a low-latency manner and it is designed for scalability, including well-defined APIs and interfaces, Hausenblas and Nadeau (2013).

Table 1 contains a comparison of the aforementioned analytical Big Data tools. The categories were chosen upon their influence in a Big Data system for successful and fast processing of data, where "++" represents very good fulfillment and "¯" represents no fulfillment. All of the aforementioned tools support (near) real-time processing and evaluations, among which Apache Flink and Apache Spark are more qualified due to their implementation design. Regarding running analytics and machine learning approaches, all tools are able to support the analytical tasks involved, e.g., some support to windowing or joining. In addition, Apache Flink and Apache Spark may offer their adapted machine learning libraries for faster performance. The streaming types vary, e.g., Apache Flink is the only tool among the aforementioned ones that supports real event streaming. Apache Drill, on the other hand, only supports mini batches, which may have a stronger impact on the performance. Apache Spark was originally developed for batch processing only and recently adapted to also support streaming, which leads to slightly worse performance compared to Apache Flink, which was originally developed for stream processing. Apache Spark has the most detailed documentation and biggest community as well as a high support for various data sources such as Kafka.

The selection of the right tool is crucial. When focusing on industrial plants with high amount of equipped sensors that are producing a continuous data stream, it is reasonable to look for tools supporting Event-Streaming or Micro-Batches. Furthermore, to train prediction models based on historical data, it is also recommended that the analytical tool supports both machine learning and also batch processing. Using the comparison in **Table 1**, it is reasonable to use either Apache Flink or Apache Spark for the creation of a versatile Big Data infrastructure, as they cover most requirements, they provide an integrated high-performance machine learning library and they are considered state-of-the-art analytical tools. Depending on whether batch processing is required, e.g., for the training of prediction models, it is better to use Apache Spark over Apache Flink.

2.1.5 Visualization

To get a good overview on the running process, it is useful to integrate a visualization tool into the Big Data infrastructure. Various approaches are possible to implement visualizations, e.g., using R Shiny apps or self-built Python apps. Nevertheless, there are also some ready-to-use tools for basic and advanced visualizations. Four popular tools in this area are PowerBI.[12], Tableau.[13], Grafana.[14] and Chronograf.[15].

PowerBI is a versatile platform for analyzing and visualizing data within live dashboards and reports, aiming at non-technological users. It supports a variety of sources, e.g.,

[12]https://powerbi.microsoft.com/

[13]https://www.tableau.com/

[14]https://grafana.com/

[15]https://www.influxdata.com/time-series-platform/chronograf/

databases or files, and is based upon business analytics. PowerBI has a restricted free usage that is expandable by purchasing licenses. Similar to PowerBI, Tableau is also not free, can be used without code and is also used broadly in the business analytics field. It is very fast at processing Excel files and data groupings, but lacks the ability to process complex needs, various sources of data and a powerful query builder. Grafana and Chronograf both are free tools for visualizing data, focusing on time-series data stored in databases. They are simple to use for developers with SQL skills since queries are created directly by the developer himself. This enables the opportunity for more elaborate and joined queries to visualize complex connections within data. Chronograf supports as source database InfluxDB and provides multiple visualization types, e.g., line plots or gauges, and enables the development of unique dashboards. Grafana is similar to Chronograf, but it supports more databases and also enables the integration of additional visualization plug-ins.

All of the mentioned tools work on top of the infrastructure and use data stored either in databases or files. They are not directly connected with message routing and only represent information previously stored, even though the tools support live updates. Live updates denote the immediate forwarding of newly inserted information to the dashboards to be processed and visualized. The final selection of the visualization tool should be in coordination with the dashboard developer. If non-technical people are creating or adapting dashboards, it is reasonable to use PowerBI or Tableau. With these tools, it is possible to create insights into data with the downside of less advanced visualizations. The usage of Chronograf and Grafana enables further elaborate visualizations and queries on databases to provide more insight into existing data and is therefore a good starting point. For more complicated or not supported visualizations, it is recommended to develop own applications using, e.g., R Shiny.

2.1.6 Service Orchestration

Orchestration and life-cycle management of the different used services is crucial but often pose a central problem. In this context, containerization gained a lot of research interest over the last few years, Pahl et al. (2019). Efficient development time, scalability and portability are key enablers for this technology. The main advantage of wrapping services in containers is the ability to create predictable software environments that are isolated from other applications and can run everywhere, Khan (2017). The two big players in container orchestration are Docker Swarm.[16] (Swarm.[17] + Compose.[18]) and Kubernetes.[19] Shah and Dubaria (2019). Both tackle similar problems, i.e., orchestrating a set of containerized applications; however, the technologies target different application domains. In short, Docker Swarm Mode is more appropriate when orchestrating simpler service stacks where high-availability, fault-tolerancy and automatic scalability are not of highest priority and Kubernetes in case of more complex situations Pan et al. (2019). While Docker provides an open standard for encapsulating and distributing containerized applications, the complexity can increase quite fast when hundreds or thousands of containers need to be managed. The higher the number of containers, the trickier the coordination, scheduling and communication of the containers. In that case, Kubernetes can help to mitigate these problems. It is important to note that these technologies are not mutually exclusive, and a combination enables developers to use the best of both worlds.

TABLE 2 | Comparison of orchestration technologies for Big Data infrastructure development and deployment. The feature categories are chosen with respect to the identified necessities in the infrastructure stack.

Features	Docker swarm	Kubernetes
User interface	3rd party	Yes
Scalability	Highly (manual)	Highly (automatic)
Complexity	Low	High
Logging	Yes	Yes
Environment	Development	Production
# Containers	Small (<100)	Big (>100)

In **Table 2**, we identified important features for developing, deploying and managing Big Data infrastructure stacks and compared the two technologies side by side.

2.2 Technology Stack Components of AVUBDI

This sub-section covers used technologies for the development and structure of a versatile Big Data infrastructure. The technologies selected, which were presented earlier, are further described with a special focus on why they are fitting best to process industry Big Data infrastructures. In addition, tools for the support of monitoring the complete infrastructure during and after the development stage are presented to gain further insight into the dataflow.

2.2.1 Data Management and Routing

Apache Kafka is the core for routing batch and streaming data as messages between different (processing) steps across our infrastructure. Among the aforementioned technologies in *Data Management and Routing*, it has the highest throughput, lowest latency and supports batch as well as stream processing. For these reasons, it fits rather well to process industry environments. In process industry, it is important to receive information, ranging from their raw format to predictions, in a timely manner to avoid machine downtimes, scrap material and poor quality. Furthermore, Kafka has the advantage of being equipped with 1) a well integrated scalable deployment, which is useful regarding variations in data volumes, and 2) a long term message storage, which in combination with message replay improves fault-tolerance.

Another advantage of using Apache Kafka is provided by pre-configured and customizable Kafka connectors, which are used for the gathering and storing of messages. Those connectors

[16]https://github.com/docker
[17]https://github.com/docker/swarmkit
[18]https://github.com/docker/compose
[19]https://github.com/kubernetes

enable a quick development of source and sink connections between Apache Kafka messages and other systems, e.g., databases such as InfluxDB or Cassandra, and reduce development time and potential errors. To handle the format of messages between (processing) steps in the infrastructure, the Confluent Schema Registry is used. It is able to interact with Apache Kafka for monitoring the schema and its evolution over time. Next to its communication with Kafka, it is also able to handle schemas provided to Kafka connectors for storage issues. The importance for the integration of such a tool is given by the likely changes in process industry, e.g., new sensors are added or a process path changes. For the general coordination of processes in our distributed applications, Zookeeper is used. It provides a simple integration with Apache Kafka and other services and therefore reduces integration problems. Zookeeper is used for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

2.2.2 Data Storage

One to enable further usage of data, it is advisable to store it in a persistent analytical storage. The data stored ranges from raw data gathered by machines to (pre-) processed data within the infrastructure and predictions or decisions. A data storage provides reusability of data for further evaluations, visualizations and the training of new models offline using historical data. A versatile infrastructure has to deal with varying amounts of data and therefore it is the best choice to focus on data storage technologies, which are developed for the usage in Big Data scenarios. As it is likely that access to recently added data is more important, e.g., for analytical purposes, than older data, we selected InfluxDB as our main storage. InfluxDB provides, in contrast to Cassandra, a schemaless database and is developed on the basis of time-series data. The schemaless aspect is rather interesting as it allows to store evolving data, e.g., if new sensors are added to machines.

Apart from storing the data, it is also required to enable the storage of models within the infrastructure. This topic was not mentioned in *Data Storage* as it is a special form of storage type and requires adapted technologies. The storage and also versioning of models is important due to steadily changing data, either in its composition of features or its value, and the resulting demand of model adaptations. A popular tool in this area is called MLFlow, which is an open-source platform to manage the machine learning lifecycle. It covers a central model registry, its deployment and the recording of experiments and results (Zaharia et al. (2018)). To work properly, it uses a database of choice in its background, here Postgres.[20]. Postgres is a powerful, open-source object-relational database system that has earned a strong reputation for reliability, feature robustness, and performance, Momjian (2001). It fits well to MLFlow as it does not need to handle Big Data and it is simple to use.

2.2.3 Data Analysis and Visualization

For data processing, analysis and predictions, the versatile usable Big Data infrastructure is using Apache Spark. Apache Spark is the best fitting tool out of the one's presented in *Data Processing and Analysis* as it supports batch as well as streaming analysis, provides a built-in machine learning library with in-memory processing and a well documented users-guideline. These features are important as different requirements may be necessary in different use-cases. The support of batch and stream processing of data is required so that models can either be trained on historical data or continuously adapt to streaming data. The analysis part of the infrastructure consists of a Spark master, which handles the workload and the overall management, and its worker nodes, which are used for processing the data. So called Spark Jobs can be written in, e.g., Scala or Python and incorporate the analysis and processing part of the infrastructure. The input data of Spark Jobs is, in our case, received *via* Kafka topics and further processed inside the node, e.g., for preprocessing of data or predictions.

For the visualizations at the top layer of our infrastructure, we decided to use Grafana. As all parts of the infrastructure are in general open-source, it is a good idea to also include an open-source visualization tool as Grafana or Chronograph. They also provide advanced possibilities compared to PowerBI and Tableau, e.g., due to their customizability and ability to present complex relations using queries. Furthermore, as process industry is likely to work with time-series data, Grafana and InfluxDB are the most reasonable to include in a versatile usable Big Data infrastructure. Both tools are very similar to use as they receive their data directly from databases and visualize information using an SQL statement for data selection and aggregations. The visualization tool can be easily exchanged as it is atop of the infrastructure and is only connected directly to the database(s).

2.2.4 Infrastructure Monitoring

For the monitoring during the development of the infrastructure, and also for later checks, various tools are used for the structure's components. On the level of message routing, Kafdrop is used to monitor messages passing through the Kafka cluster. It is able to display information regarding brokers, topics, partitions, topic consumers and enables us to take a view on messages and their content. The Kafka Connect UI is used to set up and manage Kafka Connector instances, which represent Kafka sink and source connectors as well as transformation connectors. It already offers a broad variety of sink and source connectors, e.g., for InfluxDB or Cassandra, but also accepts customized implementations of connectors. For the monitoring of message schemas, the Schema Registry UI is a good choice. It is a fully featured tool for the underlying Schema Registry that allows visualization and exploration of registered schemas and their evolution. For the monitoring of databases, Adminer is used. This database management tool supports various databases and allows insights into the database structure, including tables and their contents, and provides a possibility for the execution of queries.

2.2.5 Dockerization

Docker, as covered in Merkel (2014), consists of Platform-as-a-Service (PaaS) products, which are using OS-level virtualization to deliver software in packages and therefore

[20]https://www.postgresql.org/

avoid the "dependency hell". The packages are called containers and isolate their software, libraries, and configuration files from other containers, but can still communicate with each other. Docker Compose, further covered in Smith (2017), enables the configuration of multi-container applications in YAML files. Docker Compose, combined with Docker Swarm, a container orchestration technology, allows the deployment of multi-container workloads on a cluster of machines, resulting in a decentralized, highly scalable and easy-to-deploy Big Data infrastructure according to Naik (2016). Furthermore, the micro-service structure, which is provided by Docker through the usage of containers, enables a more fault-tolerant and independent infrastructure compared to a monolith structure. Single services can be restarted faster and independently from other services. During the development, and later for monitoring and maintenance purposes, it is advisable to use a graphical tool, e.g., Portainer.[21]. Portainer is a popular open-source lightweight management UI for Docker, Docker Swarm, and Kubernetes, enabling simple management and monitoring of those environments.

2.2.6 Scaling

Depending on the use-case, infrastructure scaling might be a necessity. In combination with Docker Swarm, the containerized architecture allows easy scaling of infrastructure services by adding host systems. It is important to note that the used services bind to specific ports of the host system, leading to an issue when trying to scale up services like Kafka, Zookeeper or Spark automatically without increasing the number of separate host systems. It is necessary to manually configure the cluster members of the mentioned technologies as a separate service definition in the Docker Compose file (see *Container Configuration*).

2.3 Deployment and Configuration of Technology-Stack Pipeline

This subsection covers the implementation of AVUBDI, including a general overview of the composition of the infrastructure and an insight into the data processing pipeline.

2.3.1 Deployment Overview

Figure 1 provides an insight into the composition of the Big Data infrastructure, including the primary services, and the information flow. The general infrastructure consists of two central nodes, that can be deployed in a single or multiple virtual machines (cluster). These two nodes serve two main functionalities, namely: 1) *Data Ingestion and Preprocessing*, and 2) *Data Analytics*. The communication between nodes is carried out by Apache Kafka using its publisher and subscriber mechanism for message routing.

The *Data Ingestion and Preprocessing Cluster* node constitutes the basis of the overall Big Data infrastructure. It contains the central data management components, which correspond to the Kafka cluster, the Schema Registry and the Zookeeper. It is responsible for retrieving data from a data source, e.g., using Kafka source connectors, according to the use-case requirements. Kafka connectors can retrieve batch data or micro-batches of data for near real-time use-cases in streaming scenarios, which are then transformed into data streams for further processing in the analytics node. Data of batch scenarios, e.g., for training new models or analyzing historical data, are delivered as batches to further processing steps. Apart from the messaging aspects, the node is also responsible for performing essential data quality and preprocessing tasks. In particular, raw data are transformed into the appropriate format for model training or inference in the analytics node.

The *Data Analytics Cluster* node covers the processing of streaming data within the infrastructure. It receives the data streams generated by the *Data Ingestion and Preprocessing Cluster* node and processes it in the assigned Spark Jobs, which gather their needed data *via* subscriptions to a Kafka topic. The Spark Job's behavior is defined using Scala or Python code and could range from preprocessing steps to analytical tasks to predictions. The node is supposed to deal with data quality tasks and real-time or semi-real-time machine learning and process mining tasks. It is therefore responsible for providing indicators and predictions about the status of ongoing industrial processes. The resulting indicators, predictions and analytical findings are used for live adaptations in the process and are stored in InfluxDB using Kafka Connectors. Models used in this node are loaded using MLFlow.

The *Data Analytics Cluster* node also provides the possibility of training models in batches of historical data (i.e., *batch processing*), and the trained models can be stored in MLFlow. *Spark jobs* are used in combination with MLFlow models for processing the received batch data to train models for, e.g., predictions. The results are stored in InfluxDB using Kafka sink connectors to have additional evaluations of the model performance. The models and experiment setups are stored using MLFlow and further include the versioning of the models so evolutions can be followed. The models generated in this node are used to detect anomalies and for prediction purposes as part of the *streaming processing* for live predictions.

The *Data Analytics Cluster* provides also the possibility for visualization tasks. It accommodates Grafana and its customized dashboards, which are directly connected to the InfluxDB to enable live or timely defined visualizations of data stored either by the *streaming processing* or *batch processing*. The dashboards enable insights into predictions, allowing early intervention in case of behavior changes. Such visualization tasks are independent of the infrastructure as they only need access to the persistent storage and can easily be exchanged if other visualization tools fit better to the use-case.

2.3.2 Containerized Technology-Stack

Figure 2 provides insight into the concrete containerized technology-stack and its container communication linkages.

[21]https://www.portainer.io/

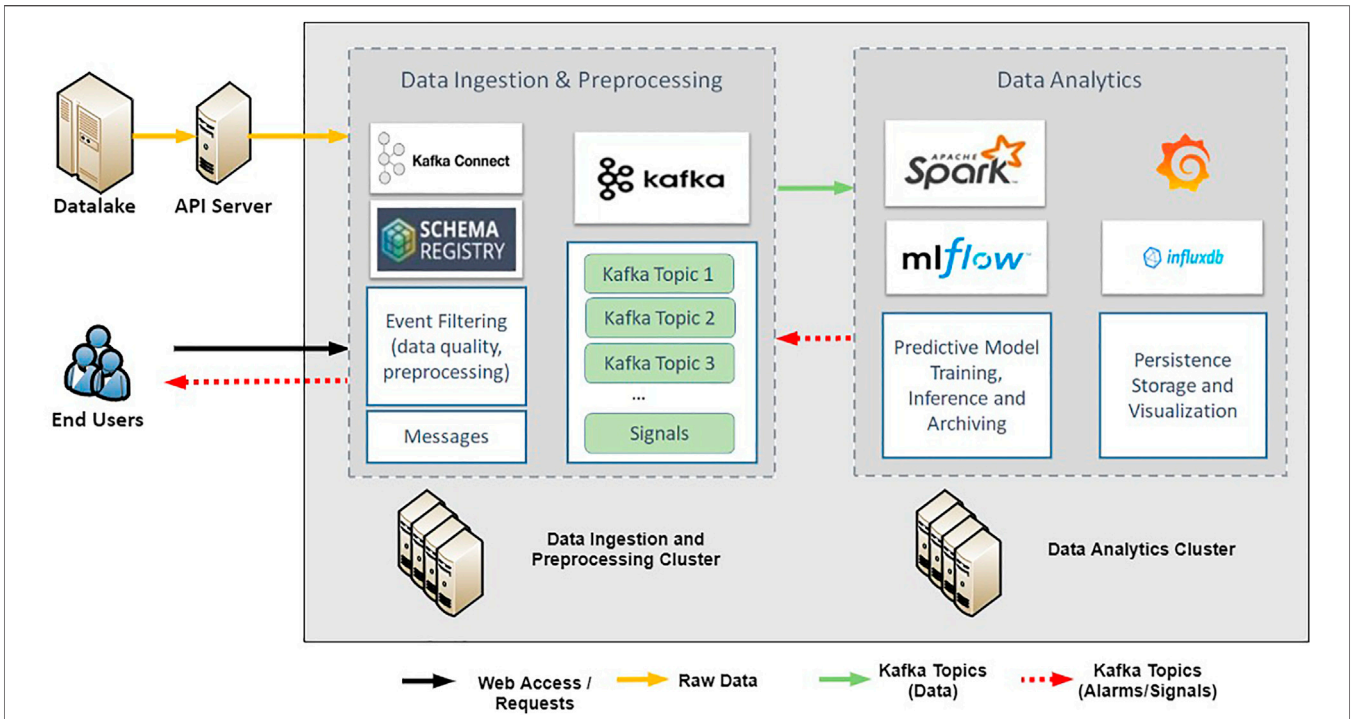

FIGURE 1 | Big Data Management Schema–Infrastructure layout containing main services and information flow.

All in all, the stack consists of 12 different services all containerized using *Docker* and each of them responsible for other parts of the stack. A service container's sample configuration can be seen in **Figure 3**. Core components like *Kafka*, *Kafka-Connect*, *Schema-Registry* and *Zookeeper* are interdependent and running at least one instance of every service at all times is crucial. *Kafka* is capable of temporarily persisting data for a certain time, meaning that a crash of *InfluxDB* only results in lost data when *InfluxDB* is unavailable longer than the configured temporary persistence period. Data from various external and internal data sources (e.g., database, *Kafka* topic, filesystem) are sourced and sinked using *Kafka-Connect*. The data schema of every data-flow is tracked using the *Schema-Registry*. Given the schema-defined *Kafka* topic, one can easily develop a *Spark* job in various languages that reads and writes to and from *Kafka* topics. *Schema-Registry UI*, *Kafka-Connect UI* and *Kafdrop* are third party tools used to monitor and configure the infrastructure. Data visualization is done by *Grafana*, however topics of *Kafka* may also be consumed and visualized outside of the infrastructure using tools like *R Shiny*, *Spring Boot + Angular* or *JavaScript*.

2.3.3 Container Configuration

All docker containers are configured using YAML files, which allows for establishing the overall definition and orchestration of these services. YAML files contain configuration definitions for the dockerized services, the containers and the connections to other containers. In **Figure 3**, we provide a simple configuration snapshot of the YAML container configuration concerning the Kafka standalone container service. A container is based on a specific *image*. Those images can easily be created or adapted manually, however in this paper we use community and industrial proven open-source container images from a centralized image repository.[22]. Containers run on top of the host system, meaning that in case of container communication, a port mapping of the host ports to container ports is a necessity. In this particular example, *Kafka* is linked to a *Zookeeper* container, as this *Zookeeper* is a prerequisite for the used *Kafka* version. The configuration variables of the services are injected into the container by using environmental variables. A list of supported variables and their configurations can be found on most image vendors' sites, e.g., Confluent.[23]. In order to persist data gathered in the container one has to map host VM volumes to container volumes using simple path mappings. Also, container placement constraints, resource restrictions, network configuration and restarting policies can easily be defined in the YAML.

2.3.4 Technology-Stack Deployment

In this subsection, we would like to describe the necessary steps for deploying the previously described technology stack in our test environment. It is worth mentioning that the selected Docker platform simplifies deployment of the required technologies and orchestration of their operation. In particular, **Table 3** provides detailed guidelines for setting up

[22]https://hub.docker.com/

[23]https://docs.confluent.io/platform/current/installation/docker/config-reference.html

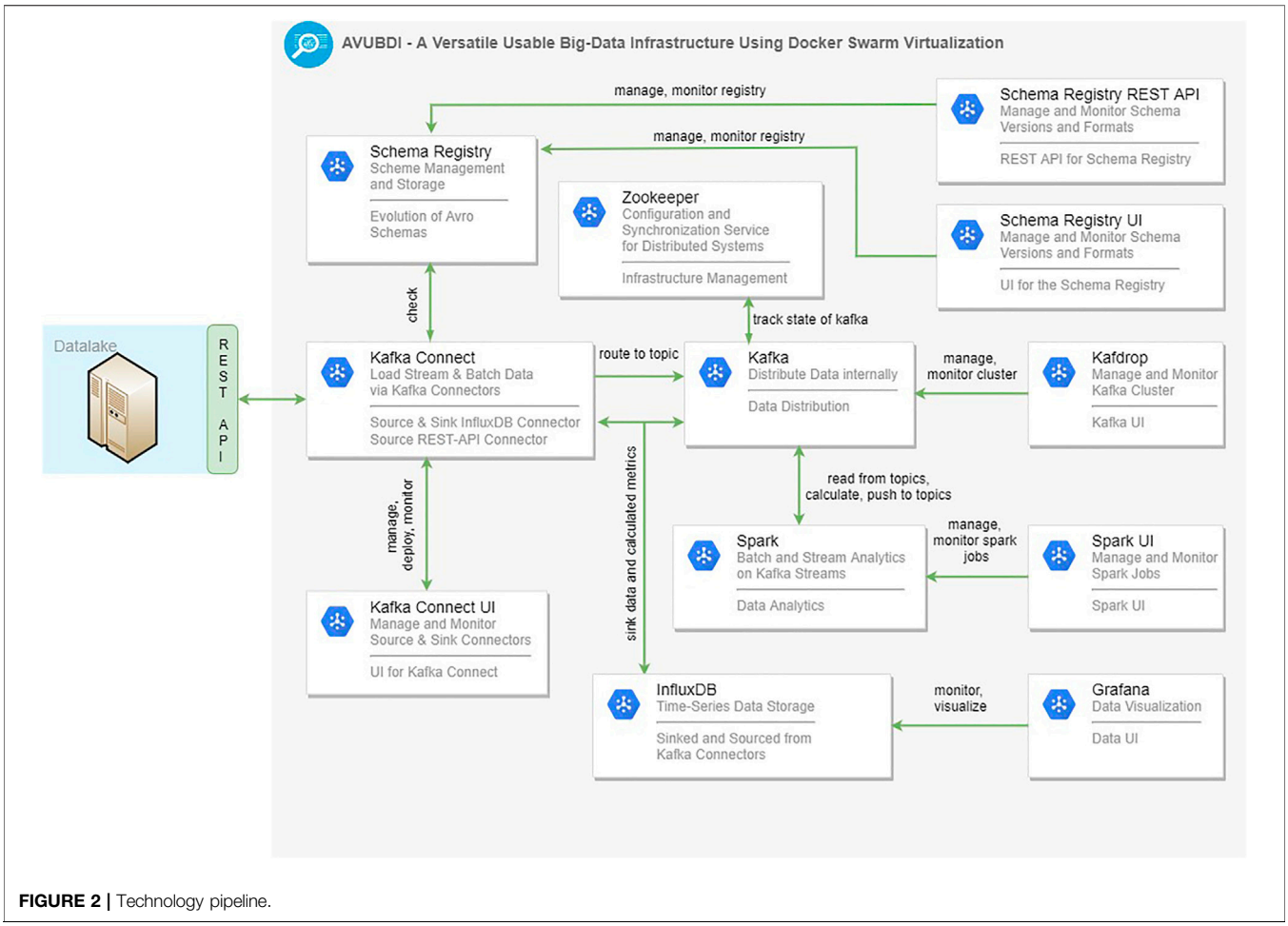

FIGURE 2 | Technology pipeline.

the overall technology-stack with the necessary deployment steps. We should note that, as also described in **Table 3**, the configuration of the technology-stack is governed by the docker-compose. yml file which is going to be described in the forthcoming *Container Configuration*.

2.4 Deployment and Configuration of Data Pipeline

In this subsection, we provide additional necessary information for configuring certain parts of the *data pipeline*. In order to better understand the functionality of this infrastructure in terms of the induced data flow, we identified four main components for configuration, namely 1) *Data Ingestion*, 2) *Data Conformance*, 3) *Data Processing* and 4) *Data Visualization*.

2.4.1 Data Ingestion

The first point of configuration is the data ingestion of the *data pipeline*. The AVUBDI uses a customized Kafka Source Connector to first access and gather data from a data-lake *via* a RESTful service. The connector is self-implemented and is able to pull mini-batches from the source in JSON format. To avoid complex processing, the nested JSON data is flattened and mapped on a POJO object for easier further processing. The data is then routed through the infrastructure using Kafka and the publish-subscriber approach for messages.

For the persistence of, e.g., results or raw data, Kafka sink connectors are used. **Table 4** contains the configuration of such a Kafka sink connector, which stores the received messages in a local InfluxDB. The configuration enables the definition of the Kafka topics listened to, the database and measurement table where data and further settings (e.g., value conerters) should be stored. Using the connector defined in **Table 4**, received messages have to include a schema definition and a payload using JSON schemas. Another possibility would be to use Avro schemas in combination with the Schema Registry. Sink connectors for other database types, e.g., Cassandra or Postgres, are also available as well as various source connectors.

2.4.2 Data Conformance

Schemas, which are used to check data conformance and evolution in routed messages, are defined in the Schema Registry. **Table 5** contains a configuration example *Schema_1*, where a message consists of two string and

TABLE 3 | Step by step instructions for installing and deploying the *AVUBDI* stack on *CentOS* systems.

1. Prerequisites
   A. CentOS 8
   B. Internet connection (for pulling docker images and git project repository, but later should be done internally)
   C. SFTP connection
2. Installation
   A. Install the yum-utils package (which provides the yum-config-manager utility) and set up the stable repository
   I. sudo yum install -y yum-utils
   II. sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
   B. Install latest version of docker engine and containerd
   I. sudo yum install docker-ce docker-ce-cli containerd.io
   C. Start docker
   I. sudo systemctl start docker
   D. Install docker compose
   I. sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o/usr/local/bin/docker-compose
   E. Make docker compose binary executable
   I. sudo chmod + x/usr/local/bin/docker-compose
   F. Pull git project repository
   I. sudo git clone https://github.com/software-competence-center-hagenberg/AVUBDI
   G. Switch to project directory
   I. cd/AVUBDI
3. Deploy
   A. Deploy infrastructure standalone on one hosting VM (standalone)
   I. Deploy infrastructure services docker compose up -d --build
   B. Deploy infrastructure cluster on multiple hosting VMs (swarm cluster)
   I. Open ports (2376,7946 TCP and 7946,4789 UDP) in firewall to allow docker swarm communication across different VM nodes
   II. Setup swarm with hosting VM as swarm manager (returns *swarm-token*): docker swarm init
   III. Scale swarm by adding additional VM nodes: docker swarm join < *swarm-token*> <*manager-ip*>
   IV. Set roles of swarm nodes: docker node update --label-add role = <master, stream_processing, batch_processing or analytics>
   V. Deploy infrastructure services to swarm (run on every swarm node): docker stack deploy --compose-file docker-compose.yml cogniplant

TABLE 4 | Configuration of a Kafka Sink Connector for a local InfluxDB.

```
{
"Name": "influxDBSinkConnector",
"config": {
"connector.class": "io.confluent.influxdb.InfluxDBSinkConnector",
"measurement.name.format": "Results",
"influxdb.url": "http://127.0.0.1:8086",
"topics": "storeInInfluxDB",
"tasks.max": "1"
"value.converter.schemas.enable": "true",
"value.converter": "org.Apache.kafka.connect.json.JsonConverter",
"influxdb.db": "database_1"
}
}
```

one double component. Depending on the Kafka settings, each message has to fit to one schema in the Schema Registry to be passed on to subscribers or a schema is adapted on the fly, enabling an insight into its evolution. As already mentioned above, it is possible to include Schema Registry schemas into the Kafka connector configuration to enable conformance checking of incoming messages for storage or further routing, so only conforming messages are further processed.

2.4.3 Data Processing

Data processing is conducted within Spark Jobs, which are deployed on the Spark master. The Spark Jobs are separated according to their usage to offline or online Jobs. Offline Jobs cover the generation and training of new models and require higher computational resources. Models created in this environment are using historical data stored, e.g., in InfluxDB and use cross-validation with a training/testing partition of 80/20. Online Jobs are used to process a continuous flow of data for, e.g., predictions.

Spark Jobs are defined using scripts, e.g., written in Scala or Python. The source of information is either a Kafka source connector (offline) or the subscription to Kafka topics for the continuous gathering of data (online). The received data is converted into a Spark Dataset object and reduced to its main features, e.g., removing duplicated or non-informational columns. The filtered data is separated into feature and target columns using a Spark Vector-Assembler and then

TABLE 5 | Configuration example of a schema in the Schema Registry.

```
{
"type": "record",
"name": "Schema_1"
"namespace": "at.cogniplant.schemas",
"fields": [
{
"name": "Value_1"
"type": "string"
}, {
"name": "Value_2"
"type": "string"
}, {
"name": "Value_3"
"type": "double"
}
]
}
```

further processed according to its usage. Trained and adapted models are stored together with their preprocessing pipeline as well as experimental results in MLFlow so processing steps are reproducible. Results of online Jobs are stored *via* Kafka sink connectors.

For the deployment of Spark Jobs within a Docker container, the Job has to be saved as fat jar (Scala, Java) or script (Python) inside the data volume. Further operations, e.g., submitting the job, have to be conducted inside the container for successful deployment. In this case, the creation of a script for automatized submissions is useful to avoid overhead in the development.

2.4.4 Data Visualization

The configuration of data visualizations is dependent on the tool and use-case. Grafana and Chronograf are both able to provide dashboards for various use-cases, each of which is configured independently. The configuration is done by adding panels inside of dashboards, in which users are able to define queries for data retrieval out of the database and visualization types. Queries are directly sent to pre-defined database connections, without the need for direct integration into the infrastructure. It is possible to define queries by hand or concatenated *via* building-blocks, enabling simple usage for non-experienced SQL users. The dashboards can be structured as preferred, allowing various plot types and alignment designs for the overall page. Dashboards or parts of them can be adapted continuously, having no influence on other components. Furthermore, visualized periods of time can be adapted continuously for historical analysis of data or can be viewed live as they are inserted into the database.

3 RESULTS

This section covers various use-cases utilizing our infrastructure in different settings with different targets. The goal was on the demonstration of the functionality of the tool. The specifics of the use-cases considered are not relevant since they are used merely for demonstration purposes of 1) the monitoring of process parameters and 2) predicting and modeling performance surveillance. Those three use-cases are just a sample of usage possibilities for such an infrastructure in the process industry. Further possibilities include the integration of alarm systems, simulation of new scenarios or analysis of machine behavior using process mining. For the experiments, we used a virtual machine with the operating system CentOS Linux, an Intel(R) Xeon(R) Gold 6136 CPU @ 3.00 GHz (4 cores of 12 were used for the VM), 16 GB RAM and 128 GB disk space, 50% of which is used for the services in docker.

Even though the presented experiments have been conducted using the CentOS Linux operating system, it should be noted that Docker runs natively on both Linux and Windows. However, native Windows- and Linux-based containers have to be individually configured. For simplicity and consistency, we focused solely on the deployment of Linux containers. It is important to note that for seamless Linux container stack development on Windows platforms, e.g., developer systems, the Windows Subsystem for Linux (WSL2).[24] is used. *WSL2* builds upon Microsoft's Virtual Machine Platform (Hyper-V), meaning that a Windows version with the virtualization extension is required. In a nutshell, *WSL2* is a minimal and tightly integrated VM with a Linux Kernel that runs on Hyper-V and is supported by x64 systems since Build 18,917.[25,26]. This feature enables generic development and deployment of Linux containers in Windows host systems without worrying about platform-specific container configurations.

3.1 Monitoring of Process Parameters

3.1.1 Setting

The first scenario covers the monitoring and visualization of process parameters during production. The data is recorded by various sensors during an industrial process, gathered by Kafka Connectors as mini or micro-batches in near real-time and stored in its raw format next to further preprocessing. The storage of raw data is essential for fault tolerance, replication possibilities and future analysis tasks. Preprocessing steps of raw data cover the filtering of interesting data, flattening of nested structures, aggregations and the transformation into usable data formats, e.g., Spark Dataset. The preprocessed data is stored in InfluxDB to respect its time-series characteristics and simpler handling of it regarding further analysis. Grafana dashboards are used for the visualization of process parameters, see **Figure 4**. They cover the representation of numerical data as a stacked bar plot and in the form of a table, which also contains further non-numerical data, e.g., dates. The dashboards enable insights into current or live process developments as well as developments during specific time ranges.

[24]https://github.com/microsoft/WSL2-Linux-Kernel

[25]https://docs.microsoft.com/en-us/windows/wsl/release-notes\#build-18917

[26]https://docs.microsoft.com/en-us/windows/wsl/install-win10

```
kafka: # data distribution kafka container (standalone)
    image: confluentinc/cp-kafka:latest # docker container specification
    ports:
        - "19092:19092" # host to container port mappings
        - "9092:9092"
    depends_on:
        - zookeeper # depends on zookeeper container
    env_file:
        - ./kafka/config/init.env # environmental variables for the container
    volumes:
        - ./kafka/data:/var/lib/kafka/data # host to container volume mapping
    deploy:
        placement:
            constraints:
                - "node.labels.role==master" # container host placement mapping
        resources: # container computing resource restrictions
            limits:
                cpus: 0.30
                memory: 4096M
        restart_policy: # container restart policies
            condition: on-failure
            delay: 10s
            max_attempts: 10
    networks: # container network
        - cogniplant
```

FIGURE 3 | Example docker-compose service definition for standalone Kafka service.

3.1.2 Results

Figure 4 contains the dashboard and results of a demonstration regarding the monitoring of process parameters using the AVUBDI infrastructure. The visualized data covers a month of recorded process parameters, which represent the temperatures in various temperature zones of a machine hall. The stacked bars give a good insight into the relations between the variables, support the identification of the temperature peaks as well as the lowest values. Furthermore, periods with no recordings of sensor measurements, e.g., during weekends, can be detected.

3.2 Monitoring of Predictions

3.2.1 Setting

The second use-case scenario of AVUBDI covers the monitoring of predictions using a machine learning model for a process criterion, e.g., quality of a product or temperature. Again, the feature data is recorded by industrial sensors, gathered, stored raw and further preprocessed. A machine learning pipeline is used for the prediction or decision part of the infrastructure. The pipeline receives the data *via* Kafka and covers the following points:

- Data cleaning, e.g., handling of null values,
- Feature filtering/selection based on training data,
- Formulating predictions/decisions using trained models,
- Storing the results in InfluxDB.

The machine learning models have been previously trained on historical data and stored as a pipeline for simple online prediction/decision-making. The processing of streaming data takes place in the online part of the infrastructure to provide insights into the target's current development.

3.2.2 Results

Figure 5 contains the visual representation of predictions produced by a regression model within a mock-up scenario regarding the monitoring of predictions using the AVUBDI infrastructure. The visualizations cover a line plot, a table and a gauge with thresholds. The visualized data represents temperature predictions using a regression model, which was developed using various settings of a machine and other sensor measurements. The line plot visualizes the predictions promptly, either as live predictions or historical predictions restricted by time ranges. The table covers the textual representation of the predictions. The gauge calculates the mean over a given time range, e.g., 5 min, and uses thresholds to indicate abnormal behavior, e.g., "too hot or too cold." Using such visualizations, workers can quickly respond to suddenly occurring problems and, therefore, reduce scrap material or low-quality products.

3.3 Monitoring of Model Performance

3.3.1 Setting

The third use-case scenario covers the monitoring of the model regarding its performance over time. The visualized data consists of (preprocessed) data gathered in industrial environments of a process and prediction values as well as measured, real values of the predictions for comparisons. In industrial environments, it is not always possible to measure

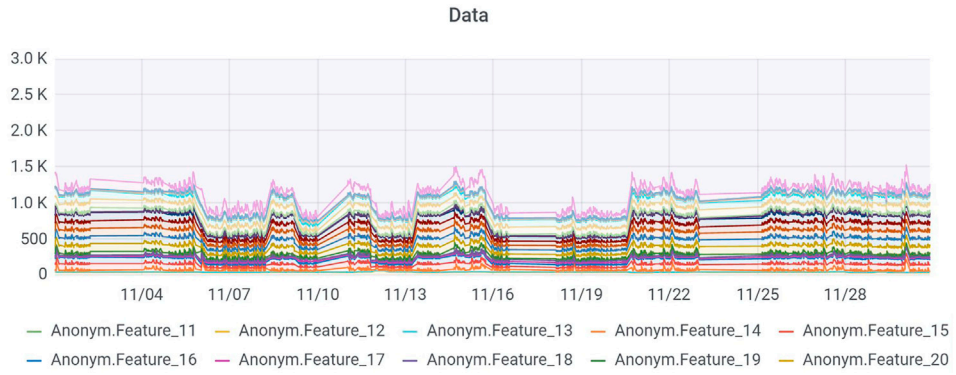

FIGURE 4 | Visualization of recorded process parameters in Grafana using a stacked bar plot. The features of the mock-up scenario resemble temperature measurements in different temperature zones recorded over a month. Users are able to identify measurement stops as well as correlated sensor data.

product or process criteria, which makes predictions necessary to get an overview on it. It is very likely though, that values are measured in predefined time intervals to check if the prediction models are (still) fitting. This use-case scenario is a mixture of the previous ones, whereat for each predicted value, a real value is available.

In this scenario, we want to check if the model performance is still suitable or if a model has to be adapted/a new model has to be trained. The dashboard created in Grafana, see **Figure 6**, compares the predictions of a regression model with their real-world counterpart measurements using different visualization types.

3.3.2 Results

Figure 6 contains the monitoring of model performances within a mock-up scenario using the AVUBDI infrastructure. The dashboard covers four different visualization types and illustrates the prediction of temperatures as well as the real values for the visualizations. The predicted values follow the same process as the predictions in the previous section and are compared to the target values gathered in an unscheduled manner. There are visual and textual comparisons using line plots and a table, as well as the difference between the values as a line plot and a gauge with predefined thresholds, thus allowing us to detect poor fitting. The use-case is rather relevant for Industry 4.0 as environments, processes and sensors may continuously change. It is possible to see the precision over a variable time range, enabling insights into developments under various circumstances and therefore decide whether a model was not fitting for a longer period or just recently.

4 DISCUSSION

In this article, we have presented a versatile usable big data infrastructure (AVUBDI), which is able to handle the full-stack of historic and real-time data processing, i.e., gathering, transforming, analyzing and visualizing in a user-friendly manner. Our main goal is the utilization of AVUBDI to improve the supervision of the production, through which performance indicators can be improved. In the first part of this paper, various open-source state-of-the-art technologies for data management, routing, storage, processing and visualizations were presented and compared. The most promising of those technologies were used to describe the development of AVUBDI in more detail, where we focused on both the technology stack and the data pipeline. Topics ranging from data gathering *via* Apache Kafka to processing with Apache Spark and visualizations with Grafana were covered. Finally, we have demonstrated its utility through two industrial use-cases, namely 1) monitoring of predictions and 2) monitoring of model performance.

The presented big data infrastructure can easily be deployed and adapted to various use-cases in industrial environments due to its versatile and solid structure, focusing on a user-friendly environment for non-experts as well as experts. In addition, the usage of containerization enables a straightforward scaling and management of the infrastructure services. On the other hand, containerization slightly reduces the overall achievable performance due to additional software and networking layers on top of the used services, see Truyen et al. (2019). We have chosen to provide the infrastructure in Docker as it can be configured, adapted and deployed in a straightforward manner, contrary to a manual development/deployment or Kubernetes. Furthermore, the usage of Docker in combination with WSL2 enables seamless Linux container development and deployment on Windows and Linux host systems without extensive adaptations of the used services. However, in order to address challenges related to fault tolerance and extensive scalability in a production environment, in the future we aim to also investigate Kubernetes (instead of Docker Swarm) as a container orchestrator. Furthermore, it would be part of our future work to evaluate the computational performance of the proposed Big Data schema.

DATA AVAILABILITY STATEMENT

The proposed Big Data infrastructure is publicly available at the following link: https://github.com/software-competence-center-hagenberg/AVUBDI. The paper has used a synthetic data source

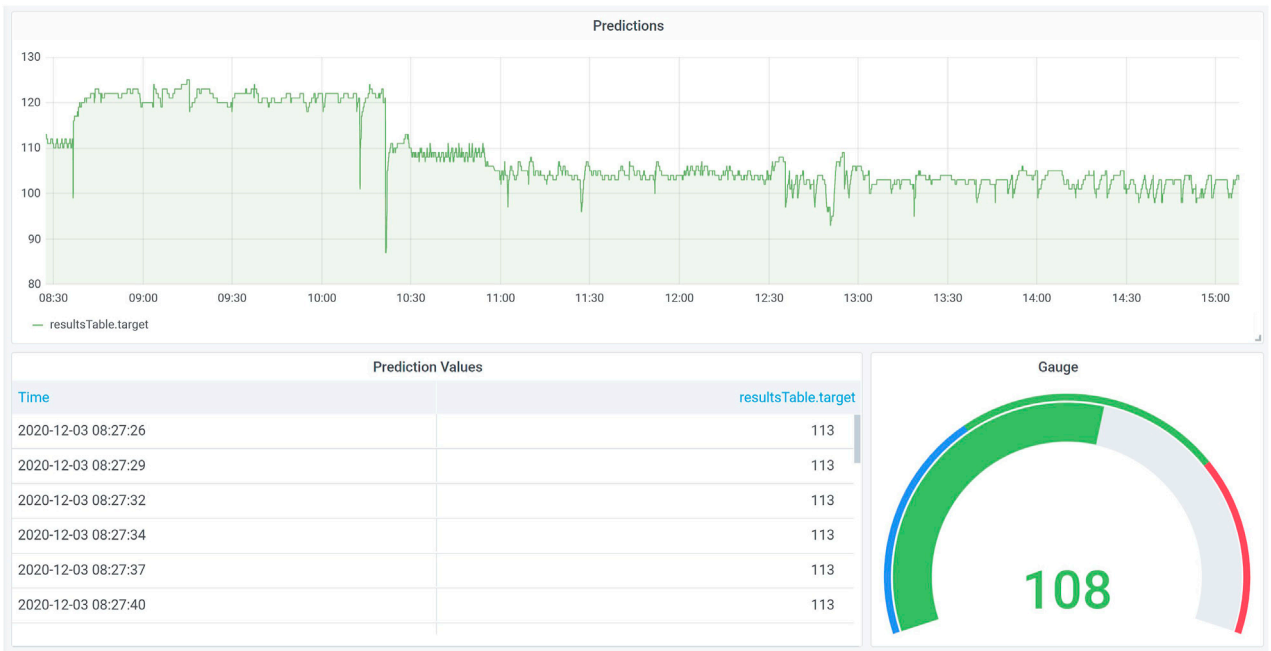

FIGURE 5 | Visualization of model predictions in Grafana including line plot, textual representation in table and gauge with thresholds. The predicted value of the mock-up scenario represents the temperature prediction within a machine hall. The gauge indicates whether the temperature is within a suitable range using the mean predicted value of the past 10 min.

FIGURE 6 | Model performance visualization in Grafana. This mock-up scenario covers the visualization of predicted and real values when estimating the performance of a prediction model. The performance surveillance is necessary to initiate changes prior to arising problems and indicates whether amodel still fits or it has to be adapted.

for demonstration purposes, however the functionality of the infrastructure is independent of the specifics of this data source.

AUTHOR CONTRIBUTIONS

LS and MM have contributed equally to the design and implementation of the AVUBDI infrastructure as well as the scientific research and the documentation process. GC has contributed partially to the design of the AVUBDI infrastructure, the scientific research and the documentation process. MP has contributed partially to the scientific research and the documentation process.

FUNDING

This paper is supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 869931, project COGNIPLANT (COGNITIVE PLATFORM TO ENHANCE 360° PERFORMANCE AND SUSTAINABILITY OF THE EUROPEAN PROCESS INDUSTRY). It has also been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

REFERENCES

Ardagna, C. A., Ceravolo, P., and Damiani, E. (2016). "Big Data Analytics As-A-Service: Issues and Challenges," in 2016 IEEE International Conference on Big Data (Big Data), December, 5-8 2016, Washington, DC. Piscataway Township, United States: IEEE, 3638.–3644. doi:10.1109/BigData.2016.7841029

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. (2015). Apache Flink: Stream and Batch Processing in a Single Engine. *Bull. IEEE Computer Soc. Tech. Committee Data Eng.* 36.

Christudas, B. (2019). "Activemq," in *Practical Microservices Architectural Patterns* (Springer), 861–867. doi:10.1007/978-1-4842-4501-9_25

[Dataset] Cogniplant (2019). Available at: https://www.cogniplant-h2020.eu/ https://cordis.europa.eu/project/id/869931 (Accessed May 2021).

[Dataset] Confluent Schema Registry (2020). Available at: https://docs.confluent.io/current/schema-registry/index.html (Accessed May 2021).

Dobbelaere, P., and Esmaili, K. S. (2017). "Kafka versus Rabbitmq: A Comparative Study of Two Industry Reference Publish/subscribe Implementations: Industry Paper," in Proceedings of the 11th ACM international conference on distributed and event-based systems, June, 19-23 2017, Barcelona Spain. New York City, United States: ACM, 227–238.

[Dataset] EU Climate and Energy Framework (2020). Available at: https://ec.europa.eu/clima/policies/strategies/2030_en (Accessed May 2021).

[Dataset] European Environment Agency (2017). Final Energy Consumption by Sector and Fuel in Europe. Available at: https://www.eea.europa.eu/data-and-maps/indicators/final-energy-consumption-by-sector-10/assessment (Accessed May 2021).

Garg, N. (2013). *Apache Kafka*. Birmingham, UK: Packt Publishing Ltd. doi:10.5005/jp/books/12257

Ge, Z., Song, Z., Ding, S. X., and Huang, B. (2017). Data Mining and Analytics in the Process Industry: The Role of Machine Learning. *IEEE Access* 5, 20590–20616. doi:10.1109/ACCESS.2017.2756872

Hausenblas, M., and Nadeau, J. (2013). Apache Drill: Interactive Ad-Hoc Analysis at Scale. *Big data* 1, 100–104. doi:10.1089/big.2013.0011

He, Q. P., and Wang, J. (2018). Statistical Process Monitoring as a Big Data Analytics Tool for Smart Manufacturing. *J. Process Control.* 67, 35–43. doi:10.1016/j.jprocont.2017.06.012

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). "Zookeeper: Wait-free Coordination for Internet-Scale Systems," in USENIX annual technical conference. June, 23-25 2010, Boston, USA. Berkeley, United States: USENIX Association, Vol. 8.

Iqbal, M. H., and Soomro, T. R. (2015). Big Data Analysis: Apache Storm Perspective. *Int. J. Comput. Trends Technol.* 19, 9–14.

Khan, A. (2017). Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. *IEEE Cloud Comput.* 4, 42–48. doi:10.1109/MCC.2017.4250933

Lakshman, A., and Malik, P. (2010). Cassandra. *SIGOPS Oper. Syst. Rev.* 44, 35–40. doi:10.1145/1773912.1773922

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 2.

Miloslavskaya, N., and Tolstoy, A. (2016). Big Data, Fast Data and Data lake Concepts. *Proced. Computer Sci.* 88, 63. doi:10.1016/j.procs.2016.07.439

Momjian, B. (2001). *PostgreSQL: Introduction and Concepts*, 192, New York: Addison-Wesley.

Naik, N. (2016). "Building a Virtual System of Systems Using Docker Swarm in Multiple Clouds," in 2016 IEEE International Symposium on Systems Engineering (ISSE), October 4-5, 2016, Edinburgh. Piscataway Township, United States: IEEE 3.

Naqvi, S. N. Z., Yfantidou, S., and Zimányi, E. (2017). *Time Series Databases and Influxdb*. Studienarbeit, Université Libre de Bruxelles. doi:10.17501/wcosm.2017.2106

Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., et al. (2017). Samza: Stateful Scalable Stream Processing at Linkedin. *Proc. VLDB Endowment* 10, 1634–1645. doi:10.14778/3137765.3137770

Pahl, C., Brogi, A., Soldani, J., and Jamshidi, P. (2019). Cloud Container Technologies: A State-Of-The-Art Review. *IEEE Trans. Cloud Comput.* 7, 677–692. doi:10.1109/TCC.2017.2702586

Pan, Y., Chen, I., Brasileiro, F., Jayaputera, G., and Sinnott, R. (2019). "A Performance Comparison of Cloud-Based Container Orchestration Tools," in 2019 IEEE International Conference on Big Knowledge (ICBK), November 10-11, 2019, Beijing, China. Piscataway Township, United States: IEEE 191–198. doi:10.1109/ICBK.2019.00033

Poddubny, B., Kupin, A., Muzika, I., and Savytskyi, O. (2017). Information Technology for Processing of Industrial Bigdata with Distributed Infrastructure on the Basis of Smart Agents and Parallel Algorithms. *Int. J. Eng. Tech. Res.* 7.

Reis, M. S., and Gins, G. (2017). Industrial Process Monitoring in the Big Data/industry 4.0 Era: From Detection, to Diagnosis, to Prognosis. *Processes* 5, 35. doi:10.3390/pr5030035

Rios, L. G., and Diguez, J. A. I. (2014). "Big Data Infrastructure for Analyzing Data Generated by Wireless Sensor Networks," in 2014 IEEE International Congress on Big Data June 27 - July 2, 2014, Anchorage, United States. Piscataway Township, United States: IEEE, 816–823.

Salloum, S., Dautov, R., Chen, X., Peng, P. X., and Huang, J. Z. (2016). Big Data Analytics on Apache Spark. *Int. J. Data Sci. Analytics* 1, 145–164. doi:10.1007/s41060-016-0027-9

Sarnovsky, M., Bednar, P., and Smatana, M. (2018). Big Data Processing and Analytics Platform Architecture for Process Industry Factories. *Big Data Cogn. Comput.* 2, 3. doi:10.3390/bdcc2010003

Shah, J., and Dubaria, D. (2019). "Building Modern Clouds: Using Docker, Kubernetes Google Cloud Platform," in 2019 IEEE 9th Annual Computing and Communication Workshop and Conference January 7-9, 2019, Las Vegas, United States. Piscataway Township, United States: IEEE, 0184–0189. doi:10.1109/CCWC.2019.8666479

Smith, R. (2017). *Docker Orchestration*. Packt Publishing Ltd.

[Dataset] Srinivasan, N., and Rajeev, N. (2012). *Harnessing the Big Data Opportunity*.

Truyen, E., Landuyt, D. V., Lagaisse, B., and Joosen, W. (2019). "Performance Overhead of Container Orchestration Frameworks for Management of Multi-Tenant Database Deployments," in Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. April 8-12, 2019, Limassol, Cyprus. New York City, United States: ACM doi:10.1145/3297280.3297536

ur Rehman, M. H., Yaqoob, I., Salah, K., Imran, M., Jayaraman, P. P., and Perera, C. (2019). The Role of Big Data Analytics in Industrial Internet of Things. *Future Generation Computer Syst.* 99, 247–259. doi:10.1016/j.future.2019.04.020

Venkatram, K., and Geetha, M. A. (2017). Review on Big Data & Analytics – Concepts, Philosophy, Process and Applications. *Cybernetics Inf. Tech.* 17, 3–27. doi:10.1515/cait-2017-0013

Wang, J., Zhang, W., Shi, Y., Duan, S., and Liu, J. (2018). *Industrial Big Data Analytics: Challenges, Methodologies, and Applications*. https://arxiv.org/abs/1807.01016.arXiv

Xu, L. D., and Duan, L. (2019). Big Data for Cyber Physical Systems in Industry 4.0: a Survey. *Enterprise Inf. Syst.* 13, 148–169. doi:10.1080/17517575.2018.1442934

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., et al. (2018). Accelerating the Machine Learning Lifecycle with Mlflow. *IEEE Data Eng. Bull.* 41, 39–45.

Zeng, Y., and Yin, Y. (2017). Virtual and Physical Systems Intra-referenced Modelling for Smart Factory. *Proced. CIRP* 63, 378–383. doi:10.1016/j.procir.2017.03.105

Zhou, K., Fu, C., and Yang, S. (2016). Big Data Driven Smart Energy Management: From Big Data to Big Insights. *Renew. Sustainable Energ. Rev.* 56, 215–225. doi:10.1016/j.rser.2015.11.050

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.