future internet

MDPI

Article

Interoperable Data Analytics Reference Architectures Empowering Digital-Twin-Aided Manufacturing

Attila Csaba Marosi [1,*], Márk Emődi [1], Ákos Hajnal [1], Róbert Lovas [1], Tamás Kiss [2], Valerie Poser [3], Jibinraj Antony [3], Simon Bergweiler [3], Hamed Hamzeh [2], James Deslauriers [2] and József Kovács [1,2]

[1] Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH), Kende u. 13-17, 1111 Budapest, Hungary; mark.emodi@sztaki.hu (M.E.); akos.hajnal@sztaki.hu (Á.H.); robert.lovas@sztaki.hu (R.L.); jozsef.kovacs@sztaki.hu (J.K.)

[2] Centre for Parallel Computing, University of Westminster, 115 New Cavendish Street, London W1W 6UW, UK; t.kiss@westminster.ac.uk (T.K.); h.hamzeh@westminster.ac.uk (H.H.); j.deslauriers@westminster.ac.uk (J.D.)

[3] German Research Center for Artificial Intelligence (DFKI), Trippstadter Str. 122, 67663 Kaiserslautern, Germany; valerie.poser@dfki.de (V.P.); jibinraj.antony@dfki.de (J.A.); simon.bergweiler@dfki.de (S.B.)

* Correspondence: attila.marosi@sztaki.hu; Tel.: +36-1-279-6073

Abstract: The use of mature, reliable, and validated solutions can save significant time and cost when introducing new technologies to companies. Reference Architectures represent such best-practice techniques and have the potential to increase the speed and reliability of the development process in many application domains. One area where Reference Architectures are increasingly utilized is cloud-based systems. Exploiting the high-performance computing capability offered by clouds, while keeping sovereignty and governance of proprietary information assets can be challenging. This paper explores how Reference Architectures can be applied to overcome this challenge when developing cloud-based applications. The presented approach was developed within the DIGITbrain European project, which aims at supporting small and medium-sized enterprises (SMEs) and mid-caps in realizing smart business models called Manufacturing as a Service, via the efficient utilization of Digital Twins. In this paper, an overview of Reference Architecture concepts, as well as their classification, specialization, and particular application possibilities are presented. Various data management and potentially spatially detached data processing configurations are discussed, with special attention to machine learning techniques, which are of high interest within various sectors, including manufacturing. A framework that enables the deployment and orchestration of such overall data analytics Reference Architectures in clouds resources is also presented, followed by a demonstrative application example where the applicability of the introduced techniques and solutions are showcased in practice.

Keywords: IoT; digital twin; reference architecture; microservice; algorithm; analytics

Citation: Marosi, A.C.; Emődi, M.; Hajnal, Á.; Lovas R.; Kiss, T, Poser, V.; Antony, J.; Bergweiler S.; Hamzeh, H.; Deslauriers, J.; et al. Interoperable Data Analytics Reference Architectures Empowering Digital-Twin-Aided Manufacturing. *Future Internet* **2022**, *14*, 114. https://doi.org/10.3390/fi14040114

Academic Editors: Vijayakumar Varadarajan, Rajanikanth Aluvalu and Ketan Kotecha

Received: 28 February 2022
Accepted: 30 March 2022
Published: 6 April 2022

1. Introduction

Manufacturing is typically the backbone of every economy and is one of the oldest and most prominent economic sectors worldwide. In Europe, for example, manufacturing accounts for about 15% of the GDP and provides approximately 33 million jobs [1]. Additionally, it is widely estimated that every job in manufacturing creates up to two and a half other job opportunities across the value chain, further increasing the importance of the sector. Although manufacturing is one of the more traditional industries, it has undergone a significant transformation in the past decade. We are currently witnessing the so-called Industry 4.0 (I4.0) [2] revolution, where automated machines have been extended with cutting edge technologies such as smart sensors, intelligent robots, and innovative IT solutions to further improve productivity, increase automation, and improve the overall efficiency of processes.

A central aspect of I4.0 is the concept of Digital Twins (DTs). A DT is typically a real-time digital model of a production line or industrial product that enables self-diagnosis, self-optimization, and self-configuration of the line without the need for human intervention. While there is no universally accepted definition of DTs, they typically represent bidirectional communication with a physical system, where information is received from the physical system, and new information is fed back to influence the operation of the production line.

A Digital Twin is typically a complex application composed of a large number of components or services that are executed over a dispersed and heterogeneous set of resources, including both central cloud computing nodes and resources closer to the edge of the network and to IoT devices and sensors.

The DIGITbrain project [3], funded by the European Commission's H2020 Programme, aims to make DTs more accessible to small and medium enterprises (SMEs)—who account for a large proportion of the Manufacturing industry [4] and have struggled to adopt DTs due to their inherent complexity [5]. To achieve this aim, DIGITbrain develops novel technologies that make the development and deployment of DTs easier, and therefore more affordable for smaller companies.

Reference Architectures (RAs) are another concept that emerged recently in various domains, including both cloud computing environments and I4.0. An RA represents frequently emerging patterns and best practices that can be reused in various settings with relatively small customization effort.

The main contribution of this paper is to explore and demonstrate how the concept of Reference Architectures can be effectively applied for the development and deployment of Digital Twins. According to our best knowledge, the combination of the two concepts has not yet been suggested and formalized. Utilizing the concept of RAs in the development and deployment of DTs has the potential of making the process faster and easier, without the need for specialized expertise in cloud or IoT technologies. Reference Architectures can reduce time and effort in designing and deploying complex applications such as DTs by encouraging re-usability and reducing the likelihood of errors. It will also be seen that Reference Architectures support and encourage the use of cloud orchestration tools to further empower application deployment and execution.

To achieve this objective, we first overview related work and analyze the various forms of Reference Architectures to assess their relevance and applicability to Digital Twins and data analytics (in Section 2). After this, we define the generic core building blocks of Reference Architectures for DTs (with specific focus on data analytics applications), such as Software Architecture Components, Software Architecture Fragments, and Data Analytics Reference Architectures (in Section 3). Typical and emergent categories of such data analytics applications are data provisioning and Machine Learning applications, which are then analyzed (in Sections 4 and 5, respectively), with typical examples of RAs implementing such functionalities. The automated deployment and run-time management of Reference Architectures is covered in Section 6. Section 7 brings together the previously described concepts and solutions to form a representative example for the implementation of a Reference Architecture. This single-file Reference Architecture deploys the Machine Learning stack of one of the Digital Twins of the DIGITbrain Project, and in doing so provides an end-to-end solution for end users looking to deploy a complete Digital Twin in the future. Finally, we conclude our paper and outline future work in Section 8.

2. Related Work

Reference Architectures are defined and interpreted in various ways and can represent various levels of abstractions. For example, "Reference Architectural Model Industry 4.0" [6] or "Industrial Internet Reference Architecture" [7] are high-level concepts representing the joint effort of several organizations and companies with the aim of establishing conceptual foundations, providing norms, and standardizing, implementing, and applying

them. Other RA concepts, including the ones described in this paper, are more specific to "software reference architectures" and provide a lower level of abstraction.

There are several definitions available for Reference Architectures such as in [8–12]. RAs typically provide template solutions derived from a set of existing solutions for a given use case, i.e., they give an answer for the most common questions in a given relationship. The use case—in the area of enterprise architectures or software architectures—may be an end-to-end solution, e.g., data analytics; or a partial one for a well-defined subset of functionalities, e.g., time-series data ingestion. Reference Architectures incorporate industry best practices for serving such specific use cases. These architectures should be reusable, provider-agnostic, and be the amalgamation of reusable, smaller-scale building blocks. RAs may be designed at different abstraction levels—some contain specific technologies in each building block, some may only describe the required functionality; however, they are much more than simple architecture diagrams. For example, they encapsulate non-functional requirements, best practices, and use case-specific further requirements.

In this related work section, we specifically focus on Reference Architectures related to data analytics. We present Reference Architectures using two sub-categories. These are grouped based on their abstraction level, either using (i) low-level or (ii) high-level abstraction.

At first look, the difference between concrete implementations and low-level Reference Architectures might seem minimal; however, concrete implementations are designed for a particular single use case or designed by a group for their particular use cases. On the contrary, even a low-level RA should be designed for re-usability.

Low-level RAs that provide directly deployable components are very popular with public cloud providers. These include Amazon WebServices (AWS), Google Cloud platform,, and Microsoft Azure. They all provide low-level RAs for different use cases to commercialize their services. Azure currently offers 394 architecture blueprints [11], while AWS offers 38 Reference Architectures for Big Data/Data Analytics [13].

The low-level Reference Architecture provided by Microsoft Azure for IoT Data Analytics [14], as an example, uses the concept of Things-Insights-Actions: Things are the devices providing data; we gather Insights by processing the data; we can perform Actions based on the Insights gathered. There are three paths defined for data after ingestion. The *hot path* represents near real-time insights using stream analytics, the *warm path* uses more detailed processing but incurs larger delay, and the *cold path* represents batch processing at longer intervals. Microsoft also provides more generic data processing architectures, for example in [15], a streaming source (Apache Kafka [16]) is used to ingest data. As a best practice, when ingesting IoT data, data are fed into a streaming service, for example, in Azure's IoT, the ingested data are sent to Azure EventHubs (the Azure platform equivalent of Kafka).

Technology providers also create—sometimes partial—low-level Reference Architectures using their core technologies. For example [17] focuses on Apache Kafka for streaming ingestion and Apache Spark for streaming data processing. It describes how these can be combined to create a component for a larger-scale Reference Architecture; however, options for storing the results are only mentioned but not selected.

It can be observed that the architecture described in [17] has much in common with the streaming or IoT parts of the Azure-based architectures [14,15]. This is not a coincidence, since most of the low-level data processing architectures are based on two high-level archetypes, which we are discussing next. High-level RAs represent a higher level of abstraction and may not be directly deployable, similar to low-level RAs. Such high-level RAs represent more generic architecture patterns that can be mapped to low-level RAs and specific technologies on demand.

The first well-known example of a high-level Reference Architecture for data analytics is the Lambda Architecture [18]. It introduces two parallel pipelines for data processing. One pipeline is used for stream computation ("speed layer") and one for batch processing ("batch layer"). An extra layer ("serving layer") stores the output and responds to ad hoc queries. The batch layer utilizes a distributed processing system to perform processing on all available data. As processing takes time, the batch layer can push fewer updates to

the serving layer. To fix computational or algorithmic errors, the system needs to recompute the complete dataset. The speed layer uses a time window and processes just recent data but provides a (near) real-time view. This layer was not intended to do a full computation on the whole dataset.

The Kappa architecture [19] was introduced to overcome the shortcomings of the Lambda architecture, for example the need to duplicate data between the speed and batch layers and to reflect on the evolution of stream processing since the inception of the Lambda architecture. The main idea behind the Kappa architecture is to have a single streaming pipeline, which can process the entire dataset, e.g., perform a re-computation if an algorithm changes.

Another well-known vendor-agnostic RA for data analytics is the FIWARE [20] middleware. It contains both high-level and low-level components as it provides higher-level functionality and multiple low-level alternatives. FIWARE was originally driven by the European Union to develop a platform for the Future Internet. FIWARE is a framework of open-source components for data management and for developing interoperable smart solutions. As part of the framework, FIWARE provides different components such as the "Orion context broker" for data management, IoT agents for connecting smart devices, an identity management component called "Keyrock", components for storing historical data such as "Comet" as well as components for data analytics among many others. FIWARE provides not just a standards-based platform, but an ecosystem as well. Edge devices can be considered as constrained environments in terms of data processing capabilities. Symeonides et al. [21] introduce a high-level architecture that provides a declarative query model. This model allows accessing data independently of the capabilities of the available processing engines available on the edge devices.

Finally, The Open Group Architecture Framework (TOGAF) [12] is a standardized framework for creating and managing software Reference Architectures. It is a widely used framework for enterprise architecture that includes an approach for designing, planning, delivering, and governing an enterprise information technology architecture. TOGAF incorporates both lower-level and higher-level components. It is not restricted to data analytics or any other domain. The TOGAF standard defines a building block as a potentially reusable component that encompasses an architectural or business functionality and can be used with other building blocks to provide architectures or complete solutions. It distinguishes between two building block types within architecture artifacts: Architecture Building Blocks (ABBs) and Solution Building Blocks (SBBs). An architectural artifact is a work product that describes a facet of the architecture. A building block within this is defined as a package of functionality for a given business need; however, how the entities that are packed into a building block can vary and are not defined. The specification of building blocks is an iterative and evolutionary process within the architecture design. ABBs are higher level architecture models. They encapsulate a set of technology, application, business and data requirements. Additionally, they incorporate the relationship and interoperability with other building blocks, including dependencies, interfaces, and even business and organizational aspects. They steer the development of SBBs and should be reusable. SBBs are lower-level components and are directly guided by ABBs. They implement a defined architecture component. A SBB specification must define its relation with ABB(s), its design considerations, including for example its physical architecture and its specific provided functionalities and attributes. SBBs are implementation-aware, and thus utilize a given product or component.

Based on the representative examples provided above, Reference Architectures in general can be defined as architectures, recommendations, or blueprints distilled from previous experiences, representing best-practice solutions in a particular domain and for a particular task. They define a full-stack, end-to-end data analytics configuration (e.g., data storage $\rightarrow$ data acquisition $\rightarrow$ pre-processing $\rightarrow$ processing $\rightarrow$ analytics $\rightarrow$ visualization). In this work, we only consider the data-management-related parts of them, i.e., we look at how individual "components" and "fragments" of Reference Architectures can be exploited in this

context. Since we address a very concrete level of abstraction of Reference Architectures, we call them "Software Reference Architectures" for distinction.

3. Reference Architectures in DIGITbrain

Constructing and implementing complex data analytics systems and pipelines are not straightforward tasks, especially not in heterogeneous environments, working on sensor/factory data requiring (near) real-time processing. DIGITbrain defines the term "data resource" as a primary source or sink of data. This work focuses on the data processing and data analytics parts, as well as the development activities related to data exchange, and interoperability with data sources. In this context, we restrict our scope to the manufacturing and manufacturing-related data processing domains, which include monitoring, visualization tasks, Industrial IoT (IIoT), Digital-Twin-based (DT-based) modeling, and streaming data processing.

The primary target audience is independent software vendors (ISVs); in other words, algorithm and model developers/providers. They can benefit largely from using ready-made Reference Architectures, which can either be used directly, or as an initial template to be customized and tailored according to specific requirements and goals; however, data providers can also benefit from using various components and fragments that deal with data provisioning to apply proven technologies to store and serve data in an efficient, robust, and safe way.

Our aim with providing Reference Architectures is threefold:

1. Aid users, such as ISVs and Software Developers, in constructing complex data services;
2. Ensuring that these complex data services can automatically be deployed and orchestrated in diverse computing infrastructures (e.g., clouds, HPC, edge);
3. Provide a toolset to connect data resources with calculations with least possible effort.

These goals are addressed by offering several "blueprints" for various data analytics tasks for diverse purposes and across different domains. These constructs are tested, ready-to-use, and deployable on various infrastructures. At the same time, they can serve as customizable prototypes for more complex data processing structures.

As was explained earlier, Reference Architectures can generally be defined at different levels of abstraction, which may or may not be technology independent. In DIGITbrain, we address Software Reference Architectures only, having functional boxes filled in with particular implementation solutions—that is, at a concrete level—even if these technologies (software implementing a functional block) are replaceable occasionally.

Our work is specifically aimed at providing Reference Architectures in the form of deployment descriptors (e.g., Dockerfile, Docker-Compose file, Kubernetes manifest, Helm chart, and MiCADO ADT), which include architecture definitions as well as connections and concrete technologies and software solutions for the individual components. This goal is aligned with the orchestration requirements indicated among the following goals:

- Help ISVs accelerate the development of new (SW) products in the form of algorithms and models.
- Provide robust, consistent, highly-interoperable and properly-configured ready-to-use solutions for a large class of common processing tasks for data.
- Provide scalable, highly available and manageable architectures that can be run on clouds, HPC and edge-side infrastructures.
- Provide tested and secure prototypes.

Based on the discussion above, in DIGITbrain we use the following definition for Reference Architecture: *Reference Architecture is an end-to-end architecture consisting of distributed components that incorporates existing knowledge, utilizes best-practice solutions for a particular domain and is reusable. It (a) is designed at an agreed abstraction level; (b) is subject to orchestration; (c) satisfies specific quality criteria; and (d) is use-case driven.*

3.1. Components of Reference Architectures

Figure 1 shows the different components used and their relationship when building Reference Architectures. The components should be thought of as nodes of a graph and are connected by directed edges denoting the flow of data. All components (SACs, SAFs, and DARAs) consist of:

- One or more artifacts (e.g., container images);
- A descriptor for instantiating the component;
- One or more other descriptors for building the artifacts.

The various components are defined and described below.

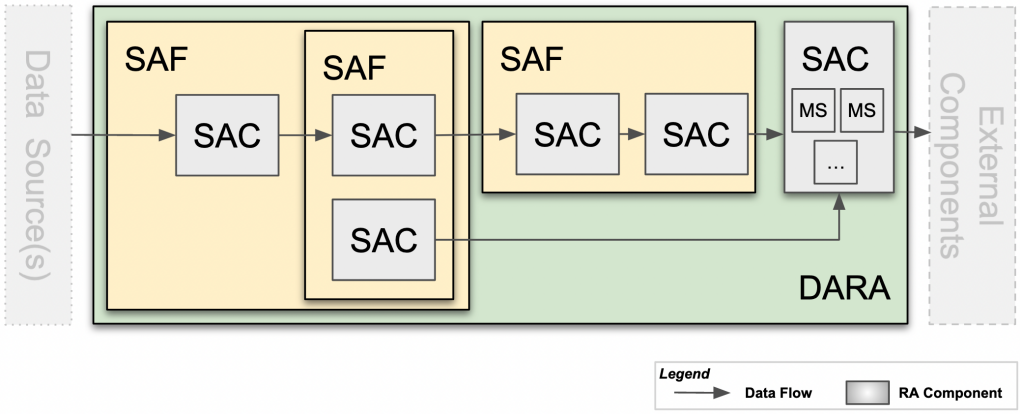

Figure 1. Relationship between Micro-Services (MS'), Software Architecture Components (SACs), Software Architecture Fragments (SAFs), and a Data Analytics Reference Architecture (DARA).

3.1.1. Software Architecture Component (SAC)

A SAC is a single service, a single component, or a small set of strongly connected components that provide a single functionality (for a distributed data analytics system). For example, a single MySQL instance running in a Docker container or Kubernetes Pod can be a SAC; however, a Kafka service consisting of multiple brokers and Zookeeper instances (meaning multiple containers or pods) is also considered a SAC. Internally, SACs contain one or more micro-services, where a micro-service is equivalent to a software container.

3.1.2. Software Architecture Fragment (SAF)

A SAF provides a best-practice solution for one or more interrelated functionalities assembling different low-level distributed components (SACs) in an integrated way. A SAF is a partial architecture tackling a specific task by using SACs, but not providing an end-to-end solution for a use case. A SAF may be a set of services used together for exploratory data analysis as depicted in Figure 2. In this example, a SAF using JupyterLab/Jupyter Notebook is shown that combines an interactive development environment and enables the use of text editors, terminals, data file viewers, and other custom components side by side. A SAF may contain SACs and other SAFs as shown in Figure 1.

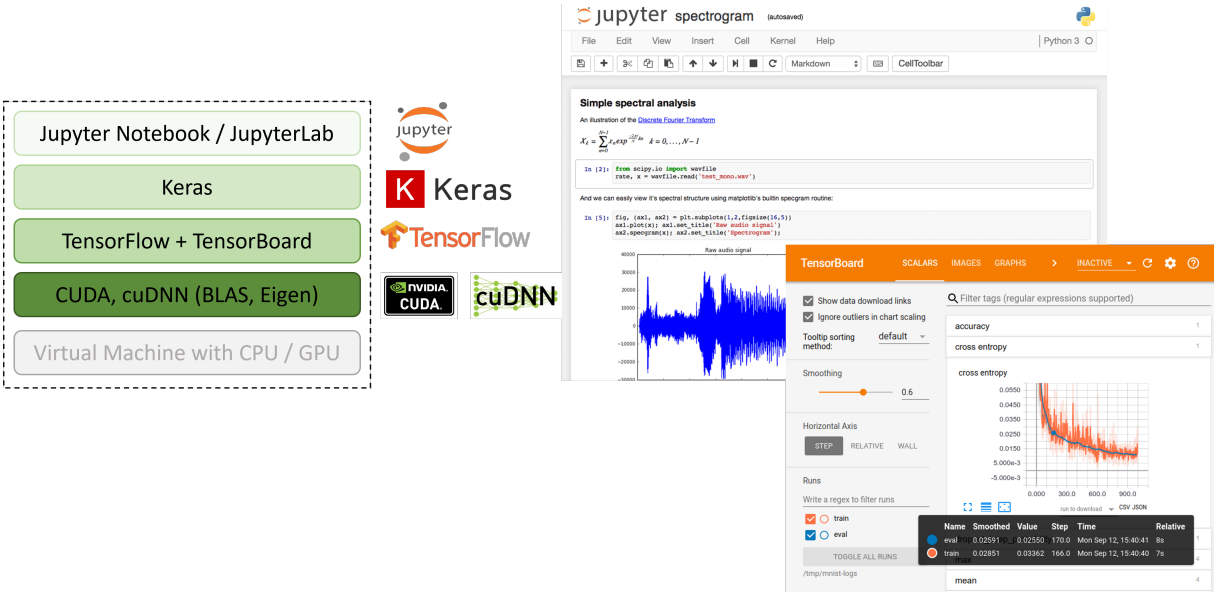

Figure 2. Example of a Software Architecture Fragment (SAF): Data exploration using JupyterLab/Jupyter Notebook.

3.1.3. Data Analytics Reference Architecture (DARA)

A DARA is composed of one or more SACs and SAFs, consisting of distributed components subject to orchestration, forming a concrete, end-to-end data processing architecture, and utilizing best-practice solutions for a particular use case in a particular domain. We define *end-to-end* as from acquiring the data (see Data Source(s) in Figure 1) to visualizing the data for the end user (using data exploratory, business intelligence, or other solutions); however, there may be external entities accessing the data within the DARA (see External Components in Figure 1).

3.1.4. Developer (Authoring) Reference Architecture (DRA)

A DRA is a special kind of Reference Architecture and similar to DARA in its structure (composed of SACs and SAFs); however, in DRAs the primary goal is not to realize data analytics in an end-to-end manner, but to provide a developer environment to support algorithm and model development (i.e., supporting the development of data analytics solution building blocks) by providing an interactive Graphical User Interface (GUI).

4. Data Provisioning Reference Architectures

Data are valuable assets. Cloud providers offer numerous data storage solutions for companies for various kinds of data (BLOBs, databases, IoT), and aim at solving challenges of volume, velocity, variety, veracity, and value of data. There are recent projects addressing data publishing and sharing that incorporate FAIR principles (findable, accessible, interoperable, and reusable) in both scientific and commercial areas. EOSC [22], IDS [23], and Gaia-X [24] are notable initiatives among others.

Although these services are of high quality and are reliable solutions, SMEs are often still reluctant to rely on third-party providers for various reasons. Data sovereignty, governance, fear of vendor lock-in, trust, security/privacy concerns, local regulations, lack of customization, lack of expertise in cloud technologies, and financial considerations are among those reasons. As such, the design and realization of the data management concepts in DIGITbrain were driven by these particular requirements as described below.

4.1. Classification of Data Sources

Three major types of data have been identified in DIGITbrain:

- Structured data;

- Unstructured data;
- Streaming data.

In the first case, data or pieces of data are organized according to a specific schema, potentially, with type and relation information between them (e.g., in the form of tables, rows, and columns, using primary key–foreign key relationships). Such data can be fetched or updated through some query language, such as SQL. Storage of these kinds are called databases, which can be SQL and NoSQL, relational, document, key-value-based, or graph databases. Widely used implementations include MySQL Server [25], MongoDB [26], PostgreSQL [27], Redis [28], InfluxDB [29], TimescaleDB [30], and Apache Cassandra [31].

In case of unstructured data, the information is encapsulated in units called binary large objects (BLOBs) or simply files, which are organized in a flat or some hierarchical structure. Although data can potentially be read in chunks or parts (or from a specific position, respectively), the internal logical structure is typically not explicit. Examples of such storage resources are: SFTP servers [32], Amazon Simple Storage Service (S3), OpenStack Swift, and Apache HDFS [33].

The third type of data addresses high-frequency data provisioning and (near) real-time processing. The primary goal is to forward a series of data packets (messages, events, and logs) of moderate size as fast as possible for a stream processing pipeline. Permanent storage for the streamed data is typically either not required or even infeasible because of the significant volume. A few examples of implementations are: Mosquitto MQTT [34], Apache Kafka [16], and RabbitMQ [35].

We note that the categorization above is not rigid, for instance, object stores do not prohibit the storing of files that are structured internally (e.g., XML and JSON); databases can contain unstructured contents inside (e.g., NoSQL); time-series databases can be a good choice for handling streamed data that can also be structured or unstructured, respectively. Our primary motivation was merely to ease the identification and selection of the appropriate solutions for building blocks in more complex Reference Architectures and assist in answering how to realize the speed layer, the batch layer, or stream processing in Lambda and Kappa architectures presented in Section 2.

4.2. Data Provisioning and Processing Scenarios

As mentioned earlier, for data governance and sovereignty reasons, the potential administrative and spatial separation of the data source and the processing logic is an important aspect. This decision leads to various configurations options, discussed in the following paragraph.

In the most typical setup in our context, **(a)** the data source resides at the factory, whereas the computation, i.e., the actual consumer runs in a remote cloud or HPC. Data stemming from the data source can be delivered to the site of the computation by proactively feeding or pushing data to the remote algorithm from the shop-floor (e.g., by intelligent devices or PLCs), or the remote algorithm can fetch and pull data from the data source. This setup includes a special case, when the processing algorithm is fed by a human user interactively (e.g., uploading data files in a web GUI or adjusting parameters of a simulation in a web form) and the user is considered to be a "data source".

In other cases, **(b)** data provisioning and data processing can be co-located. This configuration is justified when the owner of the data has no proper infrastructure to host and maintain a data provisioning service (e.g., because of the lack of public IP address, storage capacity, or IT expertise), and therefore the data storage is hosted in the cloud too. The storage can be run in the same cloud where the computation is carried out, and often operated for the time of the analysis only. Initialization and uploading of data to the cloud storage (prior to or during the computations), as well as saving results before the cloud storage is discarded are the tasks of the data source operator. This setup is also advantageous when the computation is to be performed multiple times on the same data (e.g., in machine learning), and so data transfers can be optimized.

Another case of co-location **(c)** is when the infrastructure containing the data source is capable of performing computation as well, which is called edge-side processing. In this case, the algorithm is transferred to this location and the data are thus processed close to the data source. Use cases for such configurations are when the volume or frequency of data are prohibitively large with regard to required latency, control loop, or feedback (e.g., processing high-resolution camera images in near real-time).

The DIGITbrain Platform supports all the aforementioned data provisioning and processing scenarios and enables various configurations and combinations, most efficient for the specific goal of the customer.

It is important to note that the platform itself does not store any raw data coming from any devices, it merely stores *metadata* about them. Similarly to data processing applications, data sources have to be registered first by providing some necessary information that will be required to connect data sources with the data processing algorithms. These metadata include administrative data (creator, data asset owner, contact details), access details (host/IP, port, communication protocol, database, path, topic names, authentication modes), and descriptive data (kind, type, version, or additional metadata such as schema, encoding, or semantic information). The latter group of metadata helps users to browse among data sources and assist in finding, matching "compatible" data sources, and processing algorithms.

4.3. Data Provisioning and Processing Reference Architectures

As described above, some data providers already have their own data provisioning service, which only needs to be connected to the processing algorithms; however, this may not always be the case. Although customers on the DIGITbrain Platform (manufacturing companies) may have their data sources (sensors, PLCs, etc.) set up on-premise and may have some local data collectors, these data are not accessible for remote processing over the Internet. They can neither run data analytics on their own data, nor they can offer their data for other parties.

The DIGITbrain Platform offers support for solving this problem for all three scenarios described in Section 4.2 and at different levels of abstraction, by providing them with a set of data provisioning building blocks (SACs, SAFs, and DARAs) to choose from. These building blocks are Reference Architecture components representing best-practice techniques and blueprints that are pre-configured, tested, and ready to be integrated into larger data analytics architectures. The method of support varies depending on whether:

1. The customer has the infrastructure to host a data provisioning service (hardware, Internet connection, public IP address) and wants to keep data on-premises;
2. The customer has no proper infrastructure but wishes to use DIGITbrain services to deploy the data storage to be operated in the cloud.

To support case 1, the platform provides guidelines, which are essentially textual descriptions of necessary installation and configuration steps with supplementary figures and videos. These installation instructions are composed in way that they require moderate-level IT expertise and are mostly based on Docker and Docker-Compose technologies for quick and portable deployments. For databases, DIGITbrain offers deployment of: MySQL Server, MongoDB, PostgreSQL, Redis, InfluxDB, TimescaleDB, and Apache Cassandra. For object stores, it offers MinIO [36], SFTP Server, Apache HDFS, and NFS File Server [37]. For streaming data, the users can currently choose among: Mosquitto MQTT, Apache Kafka, and RabbitMQ.

In case 2, the deployment is even more straightforward for customers. The same set of data provisioning software is available as a set of applications (APPs, implementing Reference Architectures or its building blocks) that can be started with one click from the DIGITbrain Platform. The users only have to provide some basic configuration information, such as what databases and topic names are to be created at startup, what password to use for access, and what certificate files to use for encryption. The storage will be deployed in the cloud and will run until the user explicitly stops the application.

While designing such data provisioning Reference Architecture components, security considerations were of high importance. All the setups include mandatory authentication and traffic encryption, and can easily be further extended or customized on demand with replication, load balancing, automated backups, and other useful features.

5. Machine Learning Reference Architectures

Machine Learning (ML) is a field of Artificial Intelligence (AI) that focuses on independently inferring relationships based on real-world example data. It originated from the ability of computers to recognize patterns in data, and the industry has extended this to automatizing specific industrial tasks without needing them to be explicitly programmed. This field of AI brings more adaptability for industrial applications through iterative exposure to data, and thereby enables the ML algorithms to fit and learn from the data patterns, generating ML models to produce reliable and repeatable decisions and answers. Recent advancements in the ability of ML in handling complex calculations in Big data further increases the scope of ML applications in industrial use cases.

Industries operating on enormous amounts of data have recognized the value of ML technologies in terms of analytic capability and optimization potential, and the manufacturing industry is a significant one which is being accelerated in this Industry 4.0 digital transformation. The manufacturing industry utilizes various I4.0 technologies in many applications such as smart manufacturing, fault diagnosis and prediction, robotic assembly, quality monitoring, job shop scheduling, and more [38]. Deep transfer learning approaches help to create tailored Deep Learning (DL) models suitable for specific use cases, by updating the current data from a physical entity and by modifying the existing knowledge that the model learned from a similar use case or system. This transfer learning approach is well suited to domains where the lack of large-scale, well-annotated datasets limit the development of a DL approach [39]. Similar technological advancements in the field of AI support the *Big data and analytics* domain, which is recognized as one of the nine pillars of I4.0 transforming industrial production, as proposed by Boston Consulting Group [40]. Big data, together with IoT technologies, give more opportunities for enterprises to seek new collaborations and improve industrial performance.

The DIGITbrain Platform enables the execution of ML (and other) algorithms within a Reference Architecture on the one hand, and serves to optimize algorithms on the other. The required sensor data from manufacturing is provided via the platform. Furthermore, predefined algorithms (ML/DL) can be used to train models on HPC. After training, the generated ML/DL models can be stored in ML framework-specific formats and executed on less powerful hardware. To support model portability, the open-source community has proposed the Open Neural Network Exchange (ONNX) standard [41]. With this approach, we propose a standard for exchanging and transferring trained ML models to different devices for applications. As another option, we rely on the *SavedModel* format [42] favored by Google.

The DIGITbrain Platform provides multiple ML frameworks to assist the end user and help to deliver a stable solution through different environments. Currently, the following machine learning software stacks are supported as Reference Architecture fragments (SACs and SAFs): TensorFlow [43], Keras [44], PyTorch [45], ScikitLearn [46], and RStudio [47]. These frameworks were prepared with the commonly utilized ML library stacks and tested to achieve a stable solution in various environments. Depending on needs, the stacks can be used in a single desktop environment via Docker, and they are also deployable with MiCADO [48], an application-level cloud orchestrator tool described in Section 6.

All these stacks contain JupyterLab, pandas, NumPy, and matplotlib. These packages can help to provide a development environment with a graphical web interface. In addition, the authoring interface can interact with the different Python packages and be integrated with varying plotting tools in the UI. The software stacks can be used without any infrastructure knowledge, and the software versions are tested to help avoid incompatible software. Furthermore, each software stack contains specific ML examples related to

the specific ML framework used. The aim of the example is to try and experiment with the underlying libraries and fundamental usage of these frameworks.

All these stacks are extended with an additional software component named Rclone [49], to aid data exchange (e.g., fetch training and test data, persist-trained model) from/to tens of various types of remote data storage (FTP, S3, Google Drive, etc.). Rclone provides a web interface where users can specify the remote storage configuration, start data transfer, and obtain details about the ongoing active transfer.

6. Deployment and Run-Time Management of Reference Architectures

Reference Architectures, either for ML or for data provisioning, need to be described with suitable deployment descriptors and then deployed and managed automatically by an appropriate orchestrator component. There are various alternatives for such orchestrators available, including vendor and technology-specific solutions, such as AWS Cloud-Formation [50], OpenStack HEAT [51], MicrosoftAzure's Resource Manager (ARM) templates [52], or Google's Deployment Manager [53]. There are also cloud-agnostic solutions, including Cloudify [54], Cloudiator [55], Alien4Cloud [56], MODAClouds [57], and MiCADO [58]. In DIGITbrain, the MiCADO application-level multi-cloud orchestrator has been selected as a component of the architecture and used for the deployment and run-time management of various applications, including Reference Architectures. While this technology choice makes the DIGITbrain architecture specific to this particular solution, it needs to be emphasized that Reference Architectures, as described earlier in this paper, can be deployed with any suitable orchestrator solution and are not specific to a particular technology; therefore, while we apply MiCADO in this paper to demonstrate how Reference Architectures can be deployed and utilized, other technology solutions can equally be used by modifying the deployment descriptor and feeding it into a different orchestration framework. In the remaining part of this section, a short overview of the MiCADO orchestrator is given, followed by the presentation and analysis of a combined data provisioning and Machine Learning case study in the next section.

The MiCADO Orchestration Framework

MiCADO is an application-level cloud-agnostic orchestration and auto-scaling framework that was developed in the European COLA (Cloud Orchestration at the Level of Application) Project [59]. A number of cloud service providers and middlewares are supported by MiCADO, including both commercial clouds such as Amazon AWS, Microsoft Azure, Google Cloud Platform, or Oracle Cloud Services, as well as private cloud systems based on OpenStack or OpenNebula. MiCADO is fully open-source and implements a microservices architecture in a modular way. The modular design [60] supports varied implementations where any of the components can easily be replaced with a different realization of the same functionality. The concept of MiCADO is described in detail in [48]. In this section, only a high-level overview of the framework is provided to explain its architecture, building blocks, and modular implementation.

The high-level architecture of MiCADO is presented in Figure 3. The input to MiCADO is a TOSCA-based (Topology and Orchestration Specification for Cloud Applications [61]) Application Description Template (ADT) [62], defining the application topology (containers, virtual machines, and their interconnection) and the various policies [63] (e.g., scaling and security policies) that govern the full life-cycle of the application. MiCADO consists of two main logical components: Master Node and Worker Node(s). The Submitter component on the MiCADO Master receives and interprets the ADT as input. Based on this input, the Cloud Orchestrator creates the necessary virtual machines in the cloud as MiCADO Worker Nodes and the Container Orchestrator deploys the application's microservices in Docker containers on these nodes. After deployment, the MiCADO Monitoring System monitors the execution of the application and the Policy Keeper performs scaling decisions based on the monitoring data and the user-defined scaling policies. Optimiser is

a background microservice performing long-running calculations on-demand for finding the optimal setup of both cloud resources and container infrastructures.

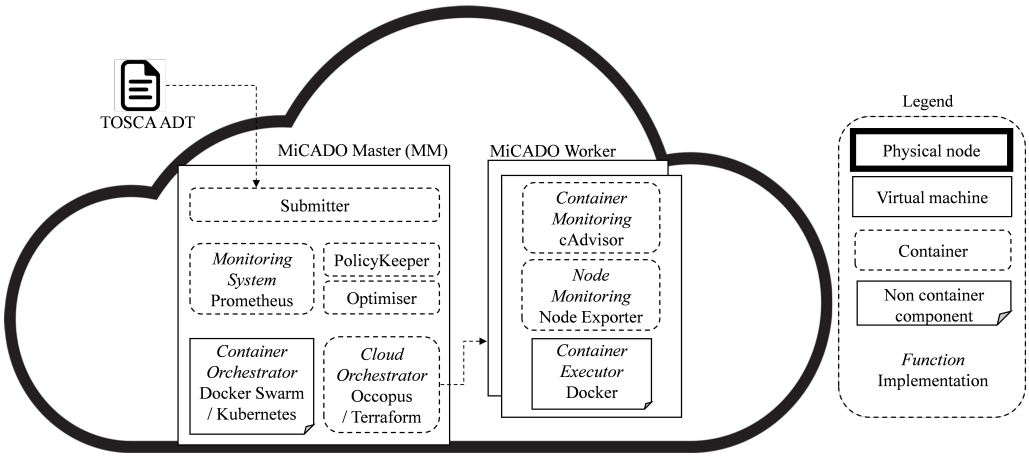

Figure 3. High-level architecture of the MiCADO orchestration framework.

Currently, there are various implementations of MiCADO based on its modular architecture, which enables changing and replacing its components with different tools and services. As a Cloud Orchestrator, the latest implementation of MiCADO can utilize either Occopus [64] or Terraform [65]. These are both capable of launching virtual machines on various private or public cloud infrastructures; however, as the clouds supported by these two orchestrators differ, MiCADO can support a wider variety of targeted resources. For Container Orchestration, MiCADO uses Kubernetes. The monitoring component is based on Prometheus [66], a lightweight, low resource consuming, but powerful monitoring tool. The MiCADO Submitter, Policy Keeper [67] and Optimiser components were custom implemented.

The latest development of MiCADO, specifically based on the requirements of the DIGITbrain project, is to extend its deployment and run-time orchestration capabilities from cloud resources towards edge and fog nodes [68]. Such an extension is especially advantageous and attractive in the manufacturing domain, where local sensors and other IoT (Internet of Things) devices collect a large amount of data, and where the processing of these data needs to be intelligently distributed between local (edge and fog) processing nodes and central cloud resources. The extension, implemented by utilizing KubeEdge [69]—an open-source Kubernetes extension—enables application owners to map the execution of certain microservices to local non-cloud processing nodes and also to specify scaling policies that enable the migration of these computations between the various layers to achieve more efficient application execution. In the context of this paper, MiCADO ADTs were applied to describe the various Reference Architectures and were then fed into the orchestrator to deploy and manage these RAs at run-time, as illustrated in the next section.

7. Experimental Results

To demonstrate the applicability of the previously described concepts, a Machine Learning Reference Architecture, including both data provisioning and data analytics steps was implemented. The various building blocks (SAC and SAF) of the RA were created independently from each other and can be reused outside the presented example.

When creating the building blocks, the logic of the DIGITbrain Platform was followed. This means that the application developer (the person who creates the SACs and the SAFs) only needs to provide a rich set of metadata (as described in Section 4.2) describing the Reference Architecture and its building blocks at a higher level of abstraction (e.g., describing the algorithm, its composing microservices, the data resources, and the AI models applied, for each individual SAC and SAF).

This metadata description was then translated into an executable deployment descriptor in MiCADO ADT format, individually for each SAC and SAF, and then combined into a final DARA description by concatenating the individual deployment descriptors. The translation and concatenation will be fully automated in the final version of the DIGITbrain Platform, which is currently under development; therefore, for this particular example, we translated the metadata into a MiCADO ADT manually, and also manually concatenated the various segments into a final RA description; however, the result, an executable/deployable ADT, is independent from the actual translation process and therefore suitable to demonstrate the feasibility of our concepts. In this section, we first introduce the ML application example and then describe how it was implemented, deployed, and tested using the Reference Architecture concept.

7.1. ML Image Classification Use Case

This section details a simple use case of an image classification to validate the concepts explained in the paper. The use case described in this paper is the part of a production line consisting of a conveyor system and several stations that assemble a product in a series of steps. At the end of the production line, there is a quality assurance module, which ensures the quality of the manufactured product with the aid of a camera sensor mounted directly above the conveyor system. The Digital Twin of the production line describes the overall function of the system in detail as well as the features of the quality assurance setup. A simple matching procedure is used to match the compatibility of the quality assurance module and the installed camera system with the requirements of the parameterized algorithm. The input data to the quality assurance module are provided by a data stream, which is being delivered through a Kafka server. The definition and the necessary adaptation of the end points of this communication is also carried out via the Digital Twin.

The proposed quality assurance module features an image classification model, which has been fine-tuned on a set of images, similar to the expected user inputs. A generalized image classification model [70] based on TensorFlow, has been used as a basis for this use case. In order to make this classification model executable through MiCADO, it has been prepared as a Docker container (Image Classification Container) running a simple Python script using TensorFlow 2.0, which provides interfaces to accept images and publish the results of the model. The results of the model have been communicated to the Digital Twin of the process in order to make decisions on the acceptance criteria of the manufactured component.

To make use of the published ML algorithm (shown as Image Classification Container in Figure 4), the end user provides the input images (in jpg format) from the camera sensor mounted on the conveyor system through an application interface, and publishes them through an Apache Kafka stream (topic) to the cloud. Once the Image Classification Container retrieves the image data from the stream, it analyzes the image and publishes the result (including the identified class and the confidence score) to another Apache Kafka stream, where the end user can check the result. On the end-users' side, a simple shell script is used to pass images to an Apache Kafka stream, using Apache Kafka's producer interface. The Image Classification Container, listens on this particular Kafka Stream and is informed whenever a new image arrives. It then downloads the image and passes it to the fine-tuned ML model, which in turn performs the actual classification. The generated results are extracted and forwarded to the second Apache Kafka stream, where the end-user can inspect it. Figure 4 shows an abstract representation of this use case. The AKHQ GUI and the Zookeeper Cluster assist the operations and the monitoring of the processes executed in the cloud.

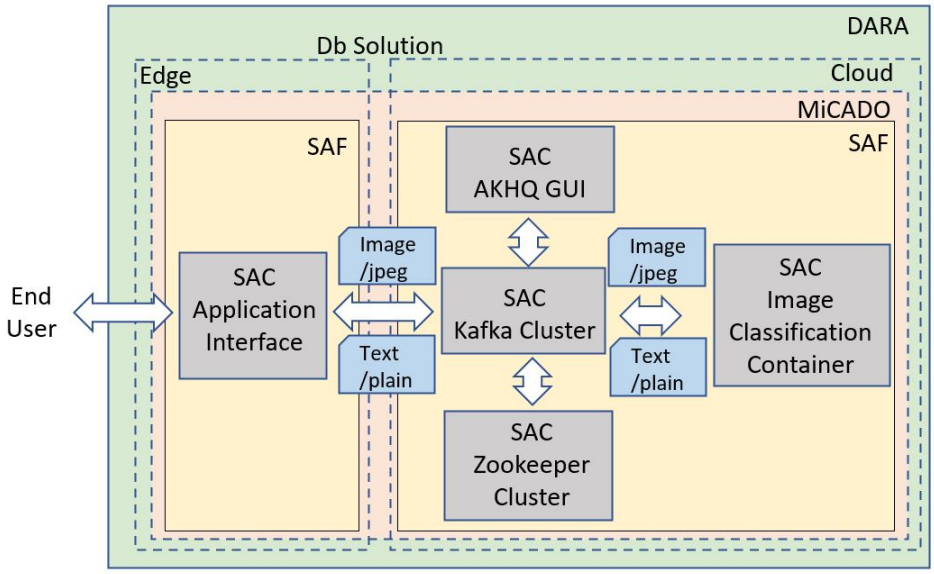

Figure 4. The graphical representation of the ML Image Classification use case indicating the SAC (marked in grey) and SAF (marked in orange) components, the DARA (marked in green), responsibilities of MiCADO (marked in red), and the stream payloads with MIME types (marked in blue).

In the above detailed use case, the individual microservices (Kafka cluster, Zookeper cluster, AKHQ Kafka GUI, and the Image Classification Container) in the application stack can be represented as SACs and thus create a SAF for the implementation (right-hand side of Figure 4). Additionally, the Application Interface, shown on the left-hand side of the figure, is implemented as a separate SAF, consisting of only one SAC. To demonstrate a realistic user scenario, the application interface is deployed at the Edge (close to the user and the data sources in production line), while the application containers are deployed in the cloud. To ensure the security of the data, the connections to the Apache Kafka server are secured by applying mutual authentication using TLS. The entire architecture, including both SAFs, represents a DARA, an end-to-end solution that is accessible for end users. As demonstrated in the following section, the entire DARA, spanning both edge and cloud resources, can be deployed automatically by the MiCADO cloud orchestrator.

7.2. Deployment

The deployment of the above Machine Learning Reference Architecture is fully managed by MiCADO and targets both cloud and edge resources from a single deployment descriptor (ADT). On submission of the ADT, MiCADO provisions and connects to infrastructure and deploys micro-services. The cloud resources that comprise the hosting infrastructure for this RA are described in the ADT and are automatically provisioned by Terraform at the time of deployment. The edge resource required for this RA is configured with a public floating IP that MiCADO can connect with via SSH. Both the cloud and edge resources are then configured on-the-fly—by cloud-init scripts and an Ansible Playbook, respectively—and are connected to a Kubernetes cluster prepared by MiCADO.

With the nodes of the Kubernetes cluster ready, MiCADO then orchestrates the individual containers (Pods) that make up each SAC, ensuring that they support the necessary interconnections with other containers and are deployed to the correct cloud or edge nodes.

In this deployment, MiCADO orchestrates four services in the cloud side and one container in the edge-side. On the edge side, MiCADO deploys a container with Kafka's consumer interface that represents the *Application Interface* SAC. Typically, the query layer (charts, visualization) is deployed in the cloud, now this is in the GUI—Application Interface. On the cloud side, MICADO deploys containers (Pods) for four SACs:

- *Image Classification* (Image Classification Container);
- *Kafka Cluster* (Kafka Broker container);
- *Zookeeper Cluster* (Zookeeper server container);
- *AKHQ GUI* (AKHQ container).

The experiment was conducted using a medium-sized cloud-based virtual machine running in AWS EC2 with 2 VCPUs and 4 GB RAM in the default VPC. The attached volume is a 20 GB General purpose SDD (gp2). Various ports have been opened in a security groups as per micro-service and MiCADO component requirements. The single virtual machine was sufficient for hosting the four containers for the duration of the experiment. To mimic an edge node, a small Ubuntu 20.04 virtual machine with 1 CPU core and 2 GB of RAM was provisioned on a private cloud (EGI Cloud). A single container was deployed to this node.

The input for the experiment was a MiCADO ADT written in YAML. As mentioned, the ADT was manually created using provided metadata, which included a Docker-Compose file. The ADT describes each micro-service, specifying all necessary parameters such as image, environment variables, and exposed ports. The Zookeeper, inference, consumer, and AKHQ microservices are served by the Kafka cluster microservice by defining appropriate port numbers.

MiCADO itself was installed on a small-sized virtual machine in AWS using the official installation documentation. The installation also handles connecting the edge node to the MiCADO cluster once ready. The ADT was then submitted to the REST API of MiCADO for deployment and execution by its underlying tools. The output and status of the deployment are accessible via the stdout logs in the MiCADO Kubernetes dashboard. For interacting with the deployment, the AKHQ microservice is exposed via a Kubernetes NodePort service, which permits access to a graphical interface/UI via a web browser.

With all of the SACs deployed and running, an end-user can use the Application Interface on the edge side to push images to the Kafka stream, where they can be consumed by the Image Classification Container. The end user can then see the results of the classification using the same Application Interface. Administrators can also access the AKHQ GUI to debug issues or view more detailed information about the Kafka streams.

MiCADO will continue to manage the RA at runtime, re-deploying containers in case of application errors, re-allocating cloud resources in the case of node failure and re-establishing the connection to the edge node if connectivity is temporarily lost.

7.3. Discussion

The deployment of the Machine Learning Reference Architecture described above demonstrates the many benefits of the proposed solution as well as highlighting certain limitations. This use case exemplifies the three broad aims of Reference Architectures introduced in Section 3:

1. The ISVs and Software Developers are supported in that they need only provide a rich set of metadata describing their services, without needing to invest effort in learning the underlying tools used for deployment and execution.
2. The generated deployment descriptor (MiCADO ADT) is automatically deployed and the described containers are orchestrated by MiCADO across cloud and edge.
3. The SACs for Kafka, Zookeeper, and AKHQ can be imagined as generic tools that support connecting data with computation. Reused, these SACs can reduce time and effort involved in future developments.

The focus of the proposed solution on concrete, software architectures reduces the likelihood of errors in the deployment expects since the Docker container prepared by the ISV or Developer beforehand is deployed as-is by MiCADO. This also raises a potential limitation—higher-level RAs may be able to offer more opportunity for optimization of certain software if a particular orchestrator were able to optimize for different container run-times or image formats. Another limitation exists in that, while there is no hard restriction on the cloud provider imposed by MiCADO, support for a given cloud provider must be developed

in MiCADO in the form of plugins. While many clouds are supported already, MiCADO does not support every available cloud.

Applying a Reference Architecture to facilitate the deployment and execution of the Machine Learning component of the DIGITbrain Digital Twin empowers the use case owners with a flat, cohesive interface for describing a complex application. Once deployed, their application is afforded all the benefits provided by a cloud orchestrator such as MiCADO—smart scheduling and self-healing of containers, distributed computations across cloud and edge, enhanced security, and easy migrations—none of which would be available to a more traditional local or otherwise non-orchestrated deployment.

One of the greatest benefits of Reference Architectures is their potential for reuse. Because of this, communities should naturally gravitate towards RAs. As more ISVs and Software Developers in a given community embrace a Reference-Architecture-oriented approach to designing their applications, more and more examples will become available to guide the rest of the community. At the same token, this can limit initial growth, since an initial effort is required by a community to describe the first SACs, SAFs, and RAs that will appear again and again as examples.

8. Conclusions and Future Work

Applying Digital Twins is one of the crucial concepts of Industry 4.0. DTs can significantly improve the manufacturing process by providing near real-time monitoring and feedback to manufacturing lines; however, developing DTs is a complex process requiring significant time and expertise. In this paper, we analyzed how Reference Architectures, especially data analytics RAs combined with Machine Learning applications, can speed up the development process of RAs. Typical structures and building blocks of Reference Architectures were introduced, and their applicability in data analytics and Machine Learning scenarios were detailed. Finally, an application example from the DIGITbrain project was provided, where a full end-to-end data analytics and Machine Learning Reference Architecture was constructed from high-level metadata and deployed and managed from a single deployment descriptor (a MiCADO ADT).

Reference Architectures and Digital Twins are emerging concepts in various domains. To our best knowledge the combination of both concepts along with efficient support for development and deployment has not yet been suggested and formalized. The paper pays special attention to the potential of applying Machine Learning techniques in various data analytics tasks which is also of high interest.

The implementation of the full concept with automated deployment descriptor generation is currently ongoing in the H2020 DIGITbrain project. The DIGITbrain Solution will provide a high-level graphical user interface to publish the application (data, model, algorithm) specific metadata, and then a specific component (ADT Generator) will automatically create the executable ADT from this.

Finally, the platform will deploy dynamically an instance of the MiCADO orchestrator for each application, which will then deploy and manage the RA at run-time. Although MiCADO does not support every available cloud and container image format, it already supports several and supports extensibility with the development of new plugins. The solution is currently being prototyped in 21 application experiments within the project.

Author Contributions: Conceptualization, Á.H., A.C.M. and T.K.; methodology, Á.H., A.C.M. and T.K.; Software, H.Á., H.H., J.D. and J.K.; writing—original draft, Á.H., A.C.M., T.K., J.D., H.H. and R.L.; writing—reviewing and editing, A.C.M. and T.K.; supervision, A.C.M.; related works RA: A.C.M.; RA components: A.C.M.; related works ML: V.P., J.A. and S.B.; ML RA and use case: V.P., J.A., S.B., M.E., T.K., H.H., J.D. and J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the European Union's Horizon 2020 project titled "Digital twins bringing agility and innovation to manufacturing SMEs, by empowering a network of DIHs with an integrated digital platform that enables Manufacturing as a Service (MaaS)" (DIGITbrain) under grant agreement No. 952071.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Manufacturing Sector in Europe. Labor Market Briefing Series. Available online: https://cdn4.euraxess.org/sites/default/files/labor_market_information-_manufacturing_sector.pdf (accessed on 16 January 2022).
2. Adolph, L.; Anlahr, T.; Bedenbender, H. *German Standardization Roadmap: Industry 4.0.*, Version 2; DIN eV: Berlin, Germany, 2016.
3. Digitbrain h2020 Project. 2020. Available online: https://digitbrain.eu/ (accessed on 16 January 2022).
4. Eurostat: Statistics on Small and Medium-Sized Enterprises. Available online: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Statistics_on_small_and_medium-sized_enterprises (accessed on 25 January 2022).
5. European Commission. *Digital Transformation Scoreboard 2018. EU Businesses Go Digital: Opportunities, Outcomes and Uptake*; Publications Office of the European Union: Luxembourg, 2018; Volume 10, pp. 10–2826.
6. Schweichhart, K. Reference Architectural Model Industrie 4.0 (Rami 4.0). An Introduction. 2016; Volume 40. Available online: https://www.plattform-i40 (accessed on 16 January 2022).
7. Lin, S.W.; Murphy, B.; Clauer, E.; Loewen, U.; Neubert, R.; Bachmann, G.; Pai, M.; Hankel, M. Architecture Alignment and Interoperability: An Industrial Internet Consortium and Platform Industrie 4.0 Joint Whitepaper. 2017. Available online: https://www.iiconsortium.org/pdf/JTG2_Whitepaper_final_20171205.pdf (accessed on 30 October 2021).
8. What Is a Reference Architecture?—Enterprise IT Definitions. Available online: https://www.hpe.com/us/en/what-is/reference-architecture.html (accessed on 30 October 2021).
9. Pääkkönen, P.; Pakkala, D. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Res.* **2015**, *2*, 166–186. [CrossRef]
10. Wikipedia: Reference Architecture. Available online: https://en.wikipedia.org/wiki/Reference_architecture (accessed on 30 October 2021).
11. Microsoft Azure Documentation—Reference Architectures. Available online: https://docs.microsoft.com/en-us/azure/architecture/browse/ (accessed on 30 October 2021).
12. The TOGAF Standard, Version 9.2 Overview. Available online: https://www.opengroup.org/togaf (accessed on 18 November 2021).
13. AWS Architecture Center—Architecture Best Practices for Analytics & Big Data. Available online: https://aws.amazon.com/architecture/analytics-big-data/?cards-all.sort-by=item.additionalFields.sortDate&cards-all.sort-order=desc&awsf.content-type=content-type%23reference-arch-diagram&awsf.methodology=*all (accessed on 30 October 2021).
14. Microsoft Azure IoT. Available online: https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot (accessed on 14 November 2021).
15. Microsoft Azure Real Time Analytics. Available online: https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/real-time-analytics (accessed on 14 November 2021).
16. Kreps, J.; Narkhede, N.; Rao, J. Kafka: A distributed messaging system for log processing. In Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece, 12–16 June 2011; Volume 11, pp. 1–7.
17. Real-Time End-to-End Integration with Apache Kafka in Apache Spark's Structured Streaming. Available online: https://databricks.com/blog/2017/04/04/real-time-end-to-end-integration-with-apache-kafka-in-apache-sparks-structured-streaming.html (accessed on 14 November 2021).
18. Marz, N. How to Beat the CAP Theorem. 2011. Available online: http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html (accessed on 30 October 2021).
19. Kreps, J. Questioning the Lambda Architecture. 2014. Available online: https://www.oreilly.com/radar/questioning-the-lambda-architecture/ (accessed on 30 October 2021).
20. Cirillo, F.; Solmaz, G.; Berz, E.L.; Bauer, M.; Cheng, B.; Kovacs, E. A Standard-Based Open Source IoT Platform: FIWARE. *IEEE Internet Things Mag.* **2019**, *2*, 12–18. [CrossRef]
21. Symeonides, M.; Trihinas, D.; Georgiou, Z.; Pallis, G.; Dikaiakos, M. Query-Driven Descriptive Analytics for IoT and Edge Computing. In Proceedings of the 2019 IEEE International Conference on Cloud Engineering (IC2E), Prague, Czech Republic, 24–27 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–11. [CrossRef]
22. Appleton, O.; Asmi, A.; Bird, I.; Dekker, R.; Blomberg, N.; Dimper, R.; Ferrari, T.; Grant, A.; Jones, S.; Manola, N.; et al. EOSC—A tool for enabling Open Science in Europe. 2020. Available online: https://www.egi.eu/wp-content/uploads/2020/09/2020-09-17-EOSC-SRIA-Cluster-and-e-infra-statement-1.pdf (accessed on 30 October 2021).
23. Otto, B.; ten Hompel, M.; Wrobel, S. International data spaces. In *Digital Transformation*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 109–128.
24. Biegel, F.; Bongers, A.; Chidambaram, R.; Feld, T.; Garloff, K.; Ingenrieth, F. *GAIA-X: Driver of Digital Innovation in Europe*; Germany's Federal Ministry for Economic Affairs and Energy (BMWi): Berlin, Germany, 2020.
25. Oracle. MySQL Database Server. 2021. Available online: https://www.mysql.com/ (accessed on 26 November 2021).
26. Kyle, B. *MongoDB in Action*; Manning Publications Co.: Shelter Island, NY, USA, 2011.
27. Momjian, B. *PostgreSQL: Introduction and Concepts*; Addison-Wesley: New York, NY, USA, 2001; Volume 192.

28. Gade, A.N.; Larsen, T.S.; Nissen, S.B.; Jensen, R.L. REDIS: A value-based decision support tool for renovation of building portfolios. *Build. Environ.* **2018**, *142*, 107–118. [CrossRef]
29. Ahmad, K.; Ansari, M. Hands-on InfluxDB. In *NoSQL: Database for Storage and Retrieval of Data in Cloud*; Chapman and Hall/CRC: London, UK, 2017; pp. 341–354.
30. Freedman, M. TimescaleDB: Re-Engineering PostgreSQL as a Time-Series Database. 2019; Volume 24. Available online: https://www.percona.com/live/18/sites/default/files/slides/TimescaleDB-Percona-2018-main.pdf (accessed on 26 November 2021).
31. Lakshman, A.; Malik, P. Cassandra: A decentralized structured storage system. *ACM Sigops Oper. Syst. Rev.* **2010**, *44*, 35–40. [CrossRef]
32. Rout, J.K.; Bhoi, S.K.; Panda, S.K. Sftp: A secure and fault-tolerant paradigm against blackhole attack in manet. *arXiv* **2014**, arXiv:1403.0338.
33. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The Hadoop Distributed File System. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 3–7 May 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 1–10.
34. Light, R.A. Mosquitto: Server and client implementation of the MQTT protocol. *J. Open Source Softw.* **2017**, *2*, 265. [CrossRef]
35. Richardson, A. *Introduction to RabbitMQ, An Open Source Message Broker That Just Works*; Google: London, UK, 2008.
36. MinIO. MinIO Object Storage. 2021. Available online: https://min.io (accessed on 26 November 2021).
37. Hitz, D.; Lau, J.; Malcolm, M.A. File System Design for an NFS File Server Appliance. In Proceedings of the USENIX Winter, San Francisco, CA, USA, 17–21 January 1994; Volume 94.
38. Rathore, M.M.; Shah, S.A.; Shukla, D.; Bentafat, E.; Bakiras, S. The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities. *IEEE Access* **2021**, *9*, 32030–32052. [CrossRef]
39. Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; Liu, C. A Survey on Deep Transfer Learning. In *Proceedings of the Artificial Neural Networks and Machine Learning—ICANN 2018*; Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., Maglogiannis, I., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 270–279. Available online: https://link.springer.com/chapter/10.1007/978-3-030-01424-7_27 (accessed on 30 October 2021).
40. Vaidya, S.; Ambad, P.; Bhosle, S. Industry 4.0–a glimpse. *Procedia Manuf.* **2018**, *20*, 233–238. [CrossRef]
41. Bai, J.; Lu, F.; Zhang, K. ONNX: Open Neural Network Exchange. 2019. Available online: https://github.com/onnx/onnx (accessed on 16 January 2022).
42. Using the Savedmodel Format: Tensorflow Core. Available online: https://www.tensorflow.org/guide/saved_model (accessed on 16 January 2022).
43. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
44. Gulli, A.; Pal, S. *Deep Learning with Keras*; Packt Publishing Ltd.: Birmingham, UK, 2017.
45. Ketkar, N. Introduction to pytorch. In *Deep Learning with Python*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 195–208.
46. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
47. Gandrud, C. *Reproducible Research with R and RStudio*; Chapman and Hall/CRC: London, UK, 2018.
48. Kiss, T.; Kacsuk, P.; Kovacs, J.; Rakoczi, B.; Hajnal, A.; Farkas, A.; Gesmier, G.; Terstyanszky, G. MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator. *Future Gener. Comput. Syst.* **2019**, *94*, 937–946. [CrossRef]
49. Rclone. Rclone Syncs Your Files to Cloud Storage. 2021. Available online: https://rclone.org/ (accessed on 28 February 2022).
50. Aws Cloudformation: Speed up Cloud Provisioning with Infrastructure as Code. Available online: https://aws.amazon.com/cloudformation/ (accessed on 18 October 2020).
51. OpenStack: Openstack Orchestration. Available online: https://wiki.openstack.org/wiki/Heat (accessed on 18 October 2020).
52. Azure Resource Manager (ARM) Templates. 2020. Available online: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview (accessed on 19 October 2020).
53. Google Deployment Manager. 2020. Available online: https://cloud.google.com/deployment-manager/docs (accessed on 19 October 2020).
54. Cloudify Orchestration Platform—Multi Cloud, Cloud Native & Edge. 2020. Available online: https://cloudify.co/ (accessed on 10 April 2020).
55. Cloudiator. 2020. Available online: http://cloudiator.org/ (accessed on 10 April 2020).
56. Alien 4 Cloud. 2020. Available online: https://alien4cloud.github.io/ (accessed on 10 April 2020).
57. Modaclouds Multi-Cloud Devops Alliance: Modaclouds Releases Multi-Cloud Devops Toolbox. 2020. Available online: http://multiclouddevops.com/ (accessed on 10 April 2020).
58. MiCADO. Available online: https://micado-scale.eu/ (accessed on 10 April 2020).
59. COLA: Cloud Orchestration at the Level of Application. Available online: https://project-cola.eu/ (accessed on 10 April 2020).
60. DesLauriers, J.; Kiss, T.; Ariyattu, R.C.; Dang, H.V.; Ullah, A.; Bowden, J.; Krefting, D.; Pierantoni, G.; Terstyanszky, G. Cloud apps to-go: Cloud portability with TOSCA and MiCADO. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e6093. [CrossRef]
61. Oasis Topology and Orchestration Specification for Cloud Applications. 2020. Available online: https://www.oasis-open.org/committees/tosca (accessed on 10 April 2020).

62. Pierantoni, G.; Kiss, T.; Terstyanszky, G.; DesLauriers, J.; Gesmier, G.; Dang, H.V. Describing and processing topology and quality of service parameters of applications in the cloud. *J. Grid Comput.* **2020**, *18*, 761–778. [CrossRef]
63. Kiss, T.; DesLauriers, J.; Gesmier, G.; Terstyanszky, G.; Pierantoni, G.; Oun, O.A.; Taylor, S.J.; Anagnostou, A.; Kovacs, J. A cloud-agnostic queuing system to support the implementation of deadline-based application execution policies. *Future Gener. Comput. Syst.* **2019**, *101*, 99–111. [CrossRef]
64. Kovács, J.; Kacsuk, P. Occopus: A Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures. *J. Grid Comput.* **2017**, *16*, 19–37. [CrossRef]
65. Terraform. 2020. Available online: https://www.terraform.io (accessed on 10 April 2020).
66. Prometheus. 2020. Available online: https://prometheus.io/ (accessed on 10 April 2020).
67. Kovács, J. Supporting Programmable Autoscaling Rules for Containers and Virtual Machines on Clouds. *J. Grid Comput.* **2019**, *17*, 813–829. [CrossRef]
68. Ullah, A.; Dagdeviren, H.; Ariyattu, R.; DesLauriers, J.; Kiss, T.; Bowden, J. MiCADO-Edge: Towards an Application-level Orchestrator for the Cloud-to-Edge Computing Continuum. *J. Grid Comput.* **2021**, *19*, 47. [CrossRef]
69. KubeEdge. 2020. Available online: https://github.com/kubeedge/kubeedge (accessed on 19 October 2020).
70. Imagenet (ILSVRC-2012-CLS) Classification with MobileNet V3 Small. Available online: https://tfhub.dev/google/imagenet/mobilenet_v3_small_100_224/classification/5 (accessed on 24 January 2022).