Data Pipelines Assessment: The Role of Data Engine Deployment Models

Claudio A. Ardagna[1,*,†], Valerio Bellandi[1,†], Marco Luzzara[1,†] and Antongiacomo Polimeno[1,†]

[1]*Università Degli Studi di Milano, Department of Computer Science, Via Celoria 18, Milano, Italy*

Abstract

In this paper, we explore different deployment models for data engines and elucidate their implications on data pipeline behavior. Specifically, we examine the impact on data sharing, data protection, pipeline uptime and latency, and the feasibility of moving segments of typical data engines to the edge. Our work demonstrates the consequences of various deployment strategies on non-functional properties of data pipelines, focusing on availability, performance, and privacy. By considering the interplay between data engine deployment and data pipeline requirements, stakeholders can make informed decisions to optimize the efficiency and effectiveness of data-driven systems.

Keywords

Data Engine, Deployment Models, Non-Functional Assessment, Privacy

1. Introduction

The last decades have been characterized by multiple ICT revolutions, from service to cloud-edge computing, from mobile systems to 5G and Internet of Things (IoT), and from big data to machine learning (ML). These technological enhancements brought to a scenario where data production, collection, and analysis are carried out at an unprecedented rate [1, 2], while distributed systems are increasingly non-deterministic and built on miniaturized services composed and executed across the cloud-edge continuum. Data are today the cornerstone of innovation, driving advancements in a variety of sectors from healthcare to finance and beyond. However, as the volume and variety of data continue to expand, so do concerns surrounding privacy and security. In addition, data stands as the lifeblood of ICT infrastructures, driving innovation, decision-making, and efficiency across various domains. The significance of data within ICT infrastructures cannot be overstated, as it serves as the foundation upon which modern systems are built and optimized.

Data engines (aka data platforms) are yet another type of modern system and consist of many components for data management often implemented as micro-services. Different architectural solutions for data engine deployment address the peculiarities of complex data-driven environ-

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy

[*] Corresponding author.

[†] These authors contributed equally.

claudio.ardagna@unimi.it (C. A. Ardagna); valerio.bellandi@unimi.it (V. Bellandi); marco.luzzara@unimi.it (M. Luzzara); antongiacomo.polimeno@unimi.it (A. Polimeno)

0000-0001-7426-4795 (C. A. Ardagna); 0000-0003-4473-6258 (V. Bellandi); 0009-0003-1197-568X (A. Polimeno)

ments. In the past, the choice of the specific deployment model typically hinged on technical feasibility, while not considering the impact of a specific deployment on the non-functional posture (e.g., performance, privacy) of the data-driven systems.

The research community has recently started considering data engine deployment as yet another dimension of data pipeline validation and verification [3], where any changes to the data engine must be assessed on the basis of data (pipeline) peculiarities, including data source, sensitivity, type, and volume, to name but a few. In this work, we claim that a specific data engine deployment model has a direct impact on the behavior of the data engine itself and its ability to satisfy specific non-functional requirements requested by the target data pipeline. We discuss three possible deployments (Sections 3, 4, and 5) and demonstrate their impact on the final data pipeline behavior, particularly concerning the sharing and protection of data, on one side, and data engine availability and performance, on the other side (Section 6). We finally evaluate our deployment models using three data pipelines in the domains of e-commerce, finance, and healthcare (Section 7).

2. Reference Architecture

Our reference architecture incorporates a common data engine [4, 5] that orchestrates essential building blocks for effective data management and analysis: *i)* data storage for extensive data volumes accessed for analytical and operational purposes; *ii)* resource manager for allocating computational resources, ensuring timely data processing; *iii)* data analytics for querying and analyzing large datasets; *iv)* data processing for manipulating data in batch and real-time workflows; *v)* data visualization for engaging and informative data representations.

Initially, the architecture collects data from various sources, directing it to a central data storage repository accessible for processing and analytics. The resource manager coordinates these elements, ensuring structured and efficient resource allocation. Data processing and analytics components can write back to the central storage, which is then accessed for data visualization to present and interpret the processed data.

3. Centralized Deployment

The centralized deployment deploys the entire stack in a single location (e.g., in the cloud). It is well-suited for scenarios where data proximity is not a critical factor, and the primary concern is the efficient management of resources and data processing.

3.1. Description

Figure 1 presents the architecture of a centralized deployment built on two main blocks: *i)* data sources, *ii)* data engine. Data sources, positioned at various layers of a distributed infrastructure, gather data from sensors, devices, and network (edge and cloud) nodes. These data are transferred to the data engine via communication queues or APIs, and stored in the cloud-based data engine. The data then undergoes processing steps like preparation, analysis, and processing. Centralized deployment integrates key components in a single stack, typically located in a data

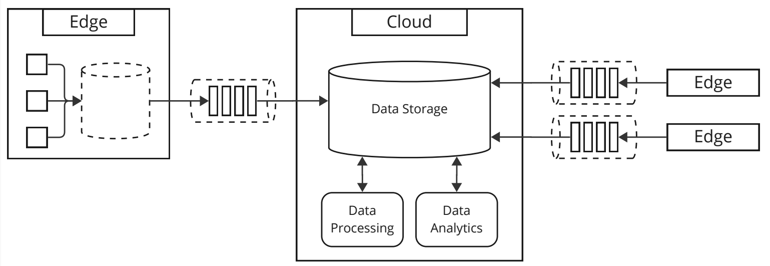

Figure 1: High-level architecture for a centralized deployment.

center or cloud environment, to simplify management and enhance function accessibility for efficient data handling and analysis.

3.2. Pro and Cons

The centralized deployment holds several advantages that contribute to its appeal in various contexts, where simple management and resource availability stand out: i) *single point of management,* the centralized deployment streamlines administrative tasks, reducing complexity and facilitating more efficient oversight of the entire centralized system; ii) *resource availability,* built on cloud functionalities with particular reference to scalability and elasticity. The final user has the impression of having infinite resources at its disposal, further empowering its ability to efficiently execute resource-intensive data processing; iii) *end-to-end data pipeline control,* giving the final user the possibility of managing the entire data pipeline in a single point, leading to a cohesive and orchestrated approach. iv) *centralized authentication and authorization,* where access to data is centrally regulated reducing the complexities associated with user syncing and access management across multiple locations.

The centralized deployment, however, introduces important challenges that call for careful consideration: i) *single point of failure,* posing a significant risk, as any malfunctions or outages in the centralized infrastructure can disrupt the entire system. It can be relieved by adopting disaster recovery and high availability protocols; ii) *transfer costs and scalability limitations,* continuous streams of data towards a centralized cluster for processing can result in significant data transfers, leading to high costs in terms of time and resources. Additionally, scalability limitations may impede the adaptability of the architecture to growing data volumes and increasing demands; iii) *increased latency,* impacting on the ability of the user to carry out real-time processing when data need to be moved from data sources to the cloud; iv) *increased risk of data breaches and data leaks,* when data traveling from distributed edge locations to the centralized cluster, traverse various untrusted network points, where they can be intercepted or accessed by unauthorized parties.

4. Decentralized Deployment

The decentralized deployment deploys the entire stack closer to the data at the edge. It is well-suited for scenarios where data proximity is a critical factor, and the main focus is on

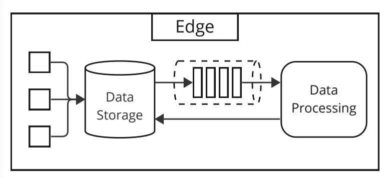

Figure 2: High-level architecture for a decentralized deployment.

efficient resource management and data processing with privacy in mind. It can be complex and costly to implement in certain contexts.

4.1. Description

Figure 2 presents the architecture of a decentralized deployment. The decentralized deployment includes all building blocks defined in Section 2. The entire stack is deployed at the edge, closer to the data sources. From a technical standpoint, it is necessary to limit the complexity of the deployed stack to reduce costs and system complexity. For example, data storage is simpler and deployed on fewer machines, while data resource management only manages the resources available at the edge.

4.2. Pro and Cons

Some of the advantages of a centralized deployment (i.e., *single point of management*, *end-to-end pipeline control*, and *centralized authentication and authorization*) are also valid in a decentralized deployment. The decentralized deployment provides two additional benefits: i) *increased data protection,* since sensitive information does not have to travel to external servers, minimizing the risk of data breaches; ii) *reduced data transfer costs* by deploying processing components in proximity of data sources.

The decentralized deployment, however, introduces important challenges that call for careful consideration: i) *increased complexity,* due to the shift of the entire stack to the edge; ii) *resource limitations,* where the system performance is constrained by the computational resources at the edge, which are typically less powerful than those available in a data center; iii) *decreased security,* because data at the edge lacks the protection provided by a typical data center with no resource limitations, making it more vulnerable to physical attacks; iv) *environmental factors* (e.g., temperature and humidity) that can impact the performance of the system.

5. Hybrid Deployment

The hybrid deployment combines centralized and decentralized deployments to fully unleashes the potential of microservices technologies. It deploys the building blocks of the data engine where convenient.

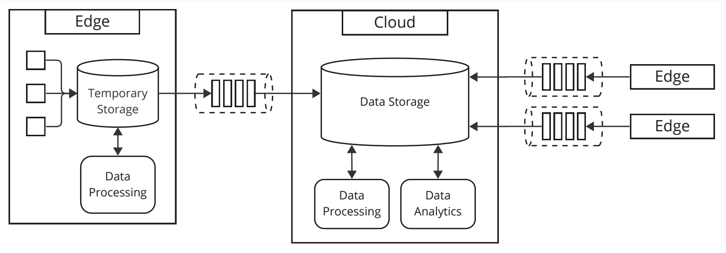

Figure 3: High-level architecture for a hybrid deployment.

5.1. Description

Figure 3 presents the architecture of a hybrid deployment. It is built on two main parts, one deployed in the cloud and the other deployed at the edge. The same stack discussed for the centralized deployment (Section 3) is used in the cloud. We note that the resource manager might be compatible and support the communication with its counterpart at the edge. A minimal version of the stack with *i)* a temporary storage to manage ingestion and *ii)* a resource manager that receives instructions from the one located in the cloud is used at the edge. A distributed resource manager permits some autonomy at the edge, enabling it to independently manage its own resources and tasks, such as job scheduling or data transformation operations.

5.1.1. Pros and Cons

The hybrid deployment provides a balanced solution that merges the advantages of both centralized and decentralized deployment models: i) *optimized resource allocation and monitoring,* where resources are dynamically allocated based on workload demands, maximizing utilization and performance across the distributed environment. By partially processing data at the edge, computational tasks can be offloaded from the centralized cloud infrastructure, reducing latency and bandwidth usage; ii) *increased fault tolerance,* because the system remains operational, albeit potentially with limited functionality, in the event of central server failure; confining data within explicitly defined processing boundaries to reduce the likelihood of accidental data exfiltration.

The decentralized deployment, however, introduces an additional challenge: i) *increased complexity,* due to the need to support data partitioning, fault tolerance, coordination, resource management, and troubleshooting.

6. Mapping Non-Functional Properties on Architectural Deployments

We discuss how architectural deployments can impact the non-functional properties availability, performance, and privacy of the specific data engine and the data pipelines executed on it. Each property is assigned a level of strength in {*low*, *medium*, *high*}, as presented in Table 6.1.

High	Multiple replicas, multiple zones
Medium	Multiple replicas, single zones
Low	Single replica, single zone

a) Availability

High	Resources can scale without any limits
Medium	Resources have a limited ability to scale
Low	Resources do not scale

b) Performance

High	No data are transferred
Medium	Only secondary data are transferred
Low	Both primary and secondary data are transferred

c) Privacy

Table 1
Property strength in {*low*, *medium*, *high*}

6.1. Non-Functional Properties

6.1.1. Property Availability

It is the availability that a specific deployment model can guarantee to a data engine and, in turn, to the data pipelines executed on it. We define property availability as follows.

Definition 1 (Property Availability). *Property Availability p_a models the system uptime as a function of the number of system replicas (i.e., single, multiple) and their deployment across different zones (i.e., single zone, multiple geographically-distributed zones).*

The levels of strength associated with property availability are as follows. *Low availability* refers to a single replica with data stored on a single zone. If a node fails, the data may become inaccessible until the node is repaired or replaced, resulting in significant downtime. Low availability is generally not suitable for critical systems, but may be acceptable for non-critical data or systems, where cost savings are a priority. *Medium availability* refers to multiple replicas with data distributed across a single zone. While there is some level of fault tolerance, the system may experience downtime or slower response time if a node fails, as the remaining nodes may become overloaded with requests, or the entire zone may experience downtime. Medium availability may be acceptable for systems where occasional downtime or slower response times can be tolerated. *High availability* refers to multiple replicas with data distributed across different geographically-distributed zones. Even if one or more nodes or zones fail, the system can still serve the data from the remaining nodes or zones. This ensures that the data are always accessible, providing a high level of fault tolerance. High availability is often associated with systems that cannot afford downtime and where data loss is unacceptable.

6.1.2. Property Performance

It is the performance that a specific deployment model can guarantee to a data engine and, in turn, to the data pipelines executed on it, as follows. We define property performance as follows.

Definition 2 (Property Performance). *Property Performance p_p models the system performance as a function of the amount of available resources and the time required for moving data from the sources to the data engine. The former aspect considers both static scenarios where resources are assigned a priori and dynamic scenarios where resources can elastically scale. The latter aspect depends on the volume of data to be moved and the corresponding network resources.*

The levels of strength associated with property performance are as follows. *Low performance* refers to resources that do not scale. This means that the amount of resources is fixed and cannot be increased to handle higher demand. If the demand exceeds the capacity of the resources, the system may experience significant performance issues. *Medium performance* refers to resources that have a limited ability to scale. The system can handle moderate increases in demand, but may struggle or experience performance degradation if the demand increases significantly. *High performance* refers to resources that can scale virtually with no limits. The system can add more resources to meet an increasing demand. This is often seen in cloud-based systems where resources can be added or removed as needed.

6.1.3. Property Privacy

It is the level of privacy that a specific deployment model can guarantee to a data engine and, in turn, to the data pipelines executed on it. We define property privacy as follows.

Definition 3 (Property Privacy). *Property Privacy p_{pr} models the level of data protection guaranteed by a specific deployment model as a function of the amount and type (either primary or secondary) of data that are exchanged between different systems.*

The levels of strength associated with privacy are as follows. *Low privacy* refers to a scenario where data transfer is requested for both primary data collected from the source and secondary data that have been previously pre-processed. These data can include sensitive information, so it is important to have robust security measures in place to protect them during transfer. *Medium privacy* refers to a scenario where data transfer is requested for secondary data only. Primary data collected from the source are not transferred, reducing the risk associated with sensitive information leakage. Security measures are still important to protect data during transfer. *High privacy* refers to a scenario where no data transfer is requested. This is the highest level of privacy, as it eliminates the risk of sensitive information being intercepted or misused during data transfer. However, it also means that the benefits of data sharing, such as collaboration and data analysis, are forbidden.

6.2. Mapping

Table 2 describes the correlation between non-functional properties in Section 6.1 and the architectural deployments in Sections 3–5. We use the following symbols to denote the support provided by a deployment model for a specific property level: ✓ to denote full support, ~ to denote that the property level can be supported with certain limitations, ✗ to denote no support. The subsequent analysis delves into the details of this mapping.

Centralized Deployment takes full advantage of cloud capabilities, ensuring optimal availability and solid performance, while introducing several privacy challenges. It supports all availability levels due to the native support for geographically distributed replicas provided by the cloud. We note that the support for high availability (i.e., multiple replicas stored across geographically distributed zones) mitigates the impact of a single point of failure. It streamlines resource allocation and monitoring, facilitating system scalability and elasticity without

Deployment Model	Availability			Performance			Privacy		
	L	M	H	L	M	H	L	M	H
Centralized	✓	✓	✓	✓	✓	~	✓	✗	✗
Decentralized	✓	~	✗	✓	~	✗	✓	✓	✓
Hybrid	✓	✓	✓	✓	✓	~	✓	✓	✓

Table 2
Mapping between non-functional property strengths and architectural deployment models

constraints. However, performance could degrade (high latency) when large volumes of data must be transferred from the sources to the cloud. For this reason, while the *low* and *medium* performance levels are fully supported, the *high* level might not be always feasible. Finally, the frequent transfer of raw and unprocessed data to the central server in the cloud raises concerns about unauthorized access or interception, limiting privacy to the *low* level.

Decentralized Deployment executes on a restricted set of resources at the edge, which negatively affects availability and performance. On the other hand, data locality guarantees high privacy. It lacks support for geographically distributed zones. Nevertheless, it can accommodate multiple replicas, sustaining *low* and *medium* levels of availability. Operating within a decentralized environment entails coping with limited and less powerful resources. However, proximity to the data source can mitigate latency, thereby increasing the efficiency of data processing. Decentralized deployment thus supports performance levels *low* and *medium.* Finally, leveraging decentralized data storage can fortify privacy measures by reducing exposure to a single point of attack and minimizing data transfer. Decentralized deployment ensures privacy across all levels, from *low* to *high.*

Hybrid Deployment offers better properties on average. It enjoys the benefits given by cloud resources availability, as well as the protection/anonymization of sensitive data at the edge. Hybrid deployment breaks the monolithic approach followed by centralized and decentralized deployment models, distributing critical components across the cloud-edge continuum. It supports availability across all levels, from *low* to *high.* Although this approach may ensure performance at all levels, from *low* to *high*, potential bottlenecks in data transfer can still survive. Finally, it enhances privacy by minimizing exposure to a single point of attack and reducing data transfer. Source data can be first pre-processed at the edge and then transferred to the cloud. Hybrid deployment ensures privacy at all levels, from *low* to *high* .

7. Evaluation

We propose three reference scenarios that evaluate the application of our deployment models in varying contexts. Throughout this section, we also present selected code excerpts illustrating the deployment processes using Docker Compose or Kubernetes.

7.1. Centralized Deployment Scenario - E-commerce Platform Analytics

Context: An e-commerce platform that analyzes users' behavior, sales data, and product trends to optimize its offerings and marketing strategies.

Use Case: Using the centralized deployment model, the platform can aggregate data from various sources into a single data center or cloud environment, facilitating complex analytics and machine learning processes to derive actionable insights.

Benefits: The centralized deployment model offers high analytics performance and availability. It handles large volumes of data and supports intensive computational tasks, which are crucial for real-time analytics and decision-making in a dynamic e-commerce environment. Analytics performance comes at the cost of an increasing cost in data transfer.

Technological Architecture: The centralized deployment model for e-commerce analytics employs Hadoop Distributed File System (HDFS) as data storage, Yet Another Resource Negotiator (YARN) as resource manager, Apache Spark for data processing and analytics, Apache Superset for data visualization, and Hive as query engine.

Deployment Configurations: This section presents the configurations for a basic centralized setup. For brevity, we propose a Docker-Compose that only includes HDFS, YARN, and Spark. We use YARN as resource manager, because the availability of (potentially unlimited) resources in the cloud makes it a robust and widely adopted solution. Modern orchestrators like Kubernetes can be used, while introducing additional maintenance overhead and complexity.

```
services:
  namenode:
    image: apache/hadoop
    command: ["hdfs", "namenode"]
    ports:
      - 9870:9870
    env_file:
      - ./config
  datanode:
    image: apache/hadoop
    command: ["hdfs", "datanode"]
    env_file:
      - ./config
  resourcemanager:
    image: apache/hadoop
    command: ["yarn", "resourcemanager"]
    ports:
      - 8088:8088
    env_file:
      - ./config
  nodemanager:
    image: apache/hadoop
    command: ["yarn", "nodemanager"]
    env_file:
      - ./config
  sparkmaster:
    image: spark
    entrypoint: ["bash", "-c",
      "$$SPARK_HOME/sbin/start-master.sh
      --host sparkmaster && sleep inf"]
  sparkworker:
    image: spark
    entrypoint: ["bash", "-c",
      "$$SPARK_HOME/sbin/start-worker.sh
      sparkmaster:7077 && sleep inf"]
```

The Docker-Compose file creates a Namenode, which manages the metadata for the HDFS file system, and a Datanode, which contains the actual user data. Regarding YARN, the Docker-Compose file creates a Resourcemanager and a Nodemanager to efficiently manage the resources across the cluster nodes. Finally, it configures a Spark cluster with two containers: the first one becomes the master upon executing the `start-master.sh` script, while the second one becomes a worker upon executing the `start-worker.sh` script, passing the master endpoint as a parameter.

7.2. Decentralized Deployment Scenario - Patients' Health Monitoring

Context: A healthcare system that monitors patients' health data in real-time across various devices and locations to provide immediate care and intervention.

Use Case: The decentralized deployment model allows patient data to be processed locally at each healthcare facility or via patient monitoring devices, ensuring quick response times and reducing the need to transfer sensitive data over the network.

Benefits: The decentralized deployment model enhances data protection and privacy, both of which are critical in the healthcare sector. It also facilitates real-time monitoring and decision-making by processing data close to their sources.

Technological Architecture: The decentralized deployment model for the patients' monitoring system employs Minio as data storage, Kubernetes as resource manager, Spark for data processing, Apache Superset for data visualization, and Trino as query engine.

Deployment Configurations: This section outlines the configurations for a decentralized setup utilizing Kubernetes, Minio, and Spark. Initially, Kubernetes establishes a pod for the Minio server, which is then accessible as a service named `minio-service`. Subsequently, Spark is deployed on Kubernetes to leverage the dynamic resource allocation it provides. However, integrating Spark with Kubernetes isn't straightforward and requires the Spark-Operator, an *operator* designed specifically for managing Spark applications within the Kubernetes ecosystem. The most efficient installation method for the Spark-Operator is through Helm.[1] Following this, a Docker image that contains the Spark code to be executed is created. The final step involves defining a Kubernetes resource type `SparkApplication`, introduced by the Spark-Operator, for deploying the applications. Below is an example YAML configuration used in this setup.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
    name: spark-with-minio
spec:
    type: Python
    pythonVersion: "3"
    mode: cluster
    image: "spark-app"
    mainApplicationFile: local:///app/main.py
    sparkVersion: "3.3.1"
driver:
    cores: 1
    memory: "1024m"
    labels:
        version: 3.3.1
executor:
    cores: 1
    instances: 2
    memory: "1024m"
    labels:
        version: 3.3.1
```

The location of the Spark application is specified in the `mainApplicationFile` property, while `driver` and `executor` specify the system requirements for the driver and the executors, respectively.

[1] https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/quick-start-guide.md#installation

7.3. Hybrid Deployment Scenario - Financial Services Risk Analysis

Context: A financial institution that analyzes transaction data for real-time fraud detection, while also conducting deeper, historical risk analysis to refine its fraud detection algorithms.

Use Case: The hybrid deployment model can be used to initially process transaction data at the edge (local bank servers) for immediate fraud detection. Simultaneously, data are sent to a centralized cloud server for more complex, long-term risk analysis and model refinement.

Benefits: The hybrid deployment model leverages low-latency processing at the edge for immediate fraud detection and robust computational resources in the cloud for deep analytics. It offers a solution that ensures real-time responsiveness and advanced analytical capabilities.

Technological Architecture: The hybrid deployment model for the risk analysis system employs Minio as data storage at the edge and HDFS as data storage in the cloud, Kubernetes as resource manager, Spark for data processing, Apache Superset for data visualization, and Hive as query engine. The configurations for a hybrid deployment resemble and extend those presented for centralized and decentralized deployments. The configurations of the decentralized deployment are used as is; the centralized deployment is migrated to Kubernetes with the need to define all Kubernetes configuration files. The hybrid model introduces several complexities in the setup process due to the orchestration layer shared between cloud and edge nodes. A detailed explanation of how to efficiently deploy hybrid architectures using Kubernetes is beyond the scope of this paper.

8. Related Work

Distributed systems have been studied from several angles across the ICT evolution, focusing on their design, development, deployment, and the evaluation of their non-functional behavior. Recently, particular emphasis has been given to big data architectures, focusing on the design and implementation of big data systems, their deployment on the cloud-edge continuum, as well as the evaluation of their performance and scalability (e.g., [6, 7]). In the literature, solutions based on the Apache ecosystem are still widespread. Aissi et al.[8] present an architecture based on HDFS, Spark, and Hive to process and analyze data from a smart farm. However, the necessity for enhanced resource utilization, especially in edge systems, has spurred the design and implementation of complex frameworks centered around orchestrators like Kubernetes. An example of these frameworks is described by Corodescu et al. [9] and outlines the importance of data locality when data must be processed in a distributed environment. Mosa et. al [10] propose MICADO, another platform designed for the deployment of scalable and autonomously managed solutions. The focus of their work has been the containerization of the Hadoop stack, enabling more effective orchestration in a cloud-native environment. Considering the challenges associated with the design and implementation of a Big Data architecture, Iatropoulou et. al [11] introduce the Big Data Apps Composition Environment (BDACE), a set of components, tools, and best practices, that improve the reliability and flexibility of the solution being developed. BDACE is one of the few approaches that address the non-functional property of security, albeit focusing solely on the aspect of authorization. The impact of distributed systems on the safety, security, and privacy of humans has then been considered, with particular reference to system trustworthiness in terms of governance, risk, and compliance. Several assurance techniques [12] have been defined with the aim of proving a specific system behavior in terms of non-functional properties support. Today, certification is considered by policymakers, regulators, and industrial stakeholders as the most suitable assurance technique for the verification of non-functional properties (e.g., availability, confidentiality, privacy) of distributed systems [13]. Certification followed the distributed system evolution. It was initially used to verify traditional software-based systems [14] and later applied to service- and cloud-based system certification [12]. In this context, Anisetti et al. [15] proposed a multi-dimensional certification scheme for distributed systems, which evaluates distributed applications across several dimensions, including the development process, the verification process, and the target distributed application itself. Anisetti et al. [16] also presented a security assurance methodology for big data pipelines grounded on DevSecOps paradigm to support reliable security and privacy by design. To the best of our knowledge, there is a lack of studies like the one in this paper that systematically investigate the relationship between big data architectures and deployment models, and their impact on non-functional properties. A first solution has been discussed in [3], where a novel assurance process for Big Data holistically evaluates the big data pipelines and the ecosystem underneath to provide a comprehensive measure of their trustworthiness. However, the proposed assurance process does not evaluate the impact of deployment models on the overall trustworthiness.

9. Conclusions

We explored different deployment models for data engines in the cloud-edge continuum and shed light on their impact on data analytics pipeline behavior. Critical aspects, such as data sharing and protection, pipeline uptime and latency, have been explored considering non-functional properties availability, performance, privacy. Our results highlighted the significance of data engine deployment as a critical dimension of data pipeline validation and verification.

Acknowledgments

Research supported, in parts, by *i)* project “BA-PHERD - Big Data Analytics Pipeline for the Identification of Heterogeneous Extracellular non-coding RNAs as Disease Biomarkers”, funded by the European Union - NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.1: “Fondo Bando PRIN 2022” (CUP G53D23002910006), *ii)* project MUSA - Multilayered Urban Sustainability Action - project, funded by the European Union - NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5: Strengthening of research structures and creation of R&D “innovation ecosystems”, set up of “territorial leaders in R&D” (CUP G43C22001370007, Code ECS00000037), *iii)* project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU, *iv)* Università degli Studi di Milano under the program “Piano di Sostegno alla Ricerca”. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.

References

[1] European Commission, D2.5 Second Report on Policy Conclusions, Data Market Study D2.5, European Commission, 2023. European Data Market Study 2021–2023.

[2] Domo, Data never sleeps 11.0, https://web.archive.org/web/20230315000000/https://www.domo.com/learn/data-never-sleeps-11, 2022. Accessed: 2024-03-18.

[3] M. Anisetti, C. A. Ardagna, F. Berto, An assurance process for Big Data trustworthiness, Future Generation Computer Systems 146 (2023) 34–46. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23001371. doi:https://doi.org/10.1016/j.future.2023.04.003.

[4] J. Wang, Y. Yang, T. Wang, R. S. Sherratt, J. Zhang, Big data service architecture: a survey, Journal of Internet Technology 21 (2020) 393–405.

[5] T. R. Rao, P. Mitra, R. Bhatt, A. Goswami, The big data system, components, tools, and technologies: a survey, Knowledge and Information Systems 60 (2019) 1165–1245.

[6] J. Dongarra, B. Tourancheau, D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, M. Parashar, Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows, Int. J. High Perform. Comput. Appl. 33 (2019) 1159–1174. URL: https://doi.org/10.1177/1094342019877383. doi:10.1177/1094342019877383.

[7] J. C. S. Dos Anjos, K. J. Matteussi, P. R. R. De Souza, G. J. A. Grabher, G. A. Borges, J. L. V. Barbosa, G. V. González, V. R. Q. Leithardt, C. F. R. Geyer, Data processing model to perform big data analytics in hybrid infrastructures, IEEE Access 8 (2020) 170281–170294. doi:10.1109/ACCESS.2020.3023344.

[8] M. E. M. El Aissi, S. Benjelloun, Y. Lakhrissi, S. E. H. B. Ali, A scalable smart farming big data platform for real-time and batch processing based on lambda architecture", Journal of System and Management Sciences 13 (2023) 17–30.

[9] A.-A. Corodescu, N. Nikolov, A. Q. Khan, A. Soylu, M. Matskin, A. H. Payberah, D. Roman, Big data workflows: Locality-aware orchestration using software containers, Sensors 21 (2021) 8212.

[10] A. Mosa, T. Kiss, G. Pierantoni, J. DesLauriers, D. Kagialis, G. Terstyanszky, Towards a cloud native big data platform using micado, in: 2020 19th International Symposium on Parallel and Distributed Computing (ISPDC), IEEE, 2020, pp. 118–125.

[11] S. Iatropoulou, P. Petrou, S. Karagiorgou, D. Alexandrou, Towards platform-agnostic and autonomous orchestration of big data services, in: 2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService), IEEE, 2021, pp. 1–8.

[12] C. Ardagna, R. Asal, E. Damiani, Q. Vu, From Security to Assurance in the Cloud: A Survey, ACM CSUR 48 (2015).

[13] C. A. Ardagna, N. Bena, Non-functional certification of modern distributed systems: A research manifesto, in: Proc. of IEEE SSE 2023, Chicago, IL, USA, 2023.

[14] D. S. Herrmann, Using the Common Criteria for IT security evaluation, CRC Press, 2002.

[15] M. Anisetti, C. A. Ardagna, N. Bena, Multi-dimensional certification of modern distributed systems, IEEE Transactions on Services Computing 16 (2023).

[16] M. Anisetti, N. Bena, F. Berto, G. Jeon, A devsecops-based assurance process for big data analytics, in: Proc. of IEEE ICWS 2022, Barcelona, Spain, 2022.