Terramycelium: A Reference Architecture for Adaptive Big Data Systems

Pouya Ataei

[a]*School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand*

Abstract

Big Data is a nascent term that emerged to describe a large amount of data that comes in different forms from various channels. In the modern world, users are the ceaseless generators of structured, semi-structured, and unstructured data that, if gleaned and crunched precisely, will reveal game-changing patterns. While opportunities exist with big data, the unprecedented amount of data has brought traditional approaches to a bottleneck, and the growth of data is outpacing technological and scientific advances in data analytics. It is estimated that approximately 75% of the big data projects have failed within the last decade, according to multiple sources. Among the challenges, system development and data architecture are prominent. This paper aims to address these challenges by introducing a novel big data reference architecture. This reference architecture is developed by absorbing the principles of complex adaptive systems, domain-driven design, distributed systems, and event-driven systems. The reference architecture is developed following the guidelines for creating empirically grounded reference architectures and is evaluated using two distinct methods: a case-mechanism experiment and expert opinion. The results of the case-mechanism experiment demonstrate the artefact's capability to meet the requirements of big data systems while being highly maintainable and scalable. Moreover, the feedback collected from the experts suggests that Terramycelium offers advantages for organisations committed to big data, yet its alignment with organisational cultures or structures varies.

Keywords: `Reference architecture, Architecture, Big data reference architecture, Big data architecture, Big data systems, Big data software engineering,`

1. Introduction

With the proliferation of digital devices and the advent of the internet, a significant transformation has occurred in terms of data creation and connectivity, ushering in an epoch marked by the exponential growth of information. This period is characterised by the extensive expansion of data, which presents difficulties for traditional data processing systems and necessitates inventive methods in data architecture [12, 10]. The volume, variety, and velocity of data in the current digital environment necessitate innovative solutions, particularly in the field of Big Data (BD).

Data needs have dramatically evolved, transitioning from basic business intelligence (BI) functions, like generating reports for risk management and compliance, to incorporating machine learning across various organisational facets [11]. These range from product design with automated assistants to personalised customer service and optimised operations. Also, as machine learning becomes more popular, application development needs to change from rule-based, deterministic models to more flexible, probabilistic models that can handle a wider range of outcomes and need to be improved all the time with access to the newest data. This evolution underscores the need to reevaluate and simplify our data management strategies to address the growing and diverse expectations placed on data.

Currently, the success rate of BD projects is low. Recent surveys have identified the fact that current approaches to BD do not seem to be effectively addressing these expectations. According to a survey conducted by Databricks [89], only 13% of organisations are highly successful in their data strategy. Additionally, a report by NewVantage Partners reveals that only 24% of organisations have successfully converted to being data-driven, and only 30% have a well-established big data strategy [67]. These observations, additionally corroborated by research conducted by McKinsey & Company [38] and Gartner [63], emphasise the difficulties of successfully using big data in the industry. These difficulties include the lack of a clear understanding of how to extract value from data, the challenge of integrating data from multiple sources, data architecture, and the need for skilled data analysts and scientists.

Without a well-established big data strategy, companies may struggle to navigate these challenges and fully leverage the potential of their data. One effective artefact to overcome some of these challenges is Reference Architectures (RAs) [23]. RAs extract the essence of the practice as a series of

2

patterns and architectural constructs and manifest it through high-level semantics. This allows stakeholders to refrain from reinventing the wheel and instead focus on utilising existing knowledge and best practices to harness the full potential of their data. While there are various BD RAs available to help practitioners design their BD systems, these RAs are overly concentrated, lack attention to cross-cutting concerns such as privacy, security, and metadata, and may not effectively handle the proliferation of data sources and consumers [10, 12].

To this end, this study presents TerraMycelium, a RA designed specifically for BD systems based on the principles of complex adaptive systems, Domain-Driven Design (DDD), distributed systems, and event-driven systems. TerraMycelium seeks to surpass the constraints of current RAs by utilising domain-driven and distributed approaches derived from contemporary software engineering [13]. This method aims to improve the ability of BD systems to scale, be maintained, and evolve, surpassing the constraints of traditional monolithic data architectures.

2. Background

This section offers the fundamental definitions necessary for understanding the intricacies of the research. This section seeks to establish the conceptual framework required for comprehending the terminology utilised in the study.

2.1. Definition of Big Data

Various academic definitions have been reviewed to define BD in this research. According to Kaisler et al. [43], BD refers to data that exceeds the capacity of current technologies to store, manage, and process effectively. The author Srivastava et al. [84] define big data as the use of extensive data sets for managing the gathering or reporting of data that aids multiple recipients in decision-making. The phrase BD is defined by Sagiroglu et al. [79] as referring to extensive data sets characterised by their huge size, diverse and intricate structure, and the challenges associated with storing, analysing, and visualising them for subsequent processes or outcomes. In this research, BD is defined as "*the process of identifying patterns from large datasets to achieve progress, management, and predictive analysis in specialised fields*".

3

2.2. Reference Architectures

RAs are crucial components in modern system development, providing guidance for building, maintaining, and evolving complex systems [23].

They provide a precise representation of the fundamental elements of a system and the interactions required to achieve broad goals. This clarity encourages the development of digestible modules, each focusing on certain parts of complicated issues, and offers a sophisticated platform for stakeholders to participate, contribute, and work together. The importance of RAs in IT is highlighted by the success of widely used technologies such as OAuth [65] and ANSI-SPARC architecture [9], which have their roots in well-organized RAs.

RAs define the characteristics of a system and influence its development. RAs stand apart by emphasising abstract features and higher levels of abstraction, which are present in any system's architecture. They strive to encapsulate the core of practice and incorporate proven patterns into unified frameworks, covering elements, attributes, and interconnections. RAs play a crucial role in BD by managing communication, controlling complexity, handling knowledge, reducing risks, promoting architectural visions, establishing common ground, improving understanding of BD systems, and enabling additional analysis.

2.3. Microservices and Decentralised, Distributed Architectures

Microservices architecture is a modern approach in software engineering that organises applications as a set of independent services that are loosely connected [19]. This method, derived from the wider idea of Service Oriented Architectures (SOA), concentrates on creating small, self-sufficient modules that work together to create a complete application.

According to Newman [64], microservices improve scalability, enable continuous deployment, and promote a more agile development environment. They allow teams to design, deploy, and scale components of a system separately, enhancing system resilience and enabling quick adaptability to changing requirements. Decentralised and distributed architectures are essential in today's computer environment, featuring systems that are spread out among several nodes, typically in various geographic regions [13].

The architectural style, as emphasised by Richards [75], addresses the drawbacks of conventional monolithic architectures by providing improved scalability, fault tolerance, and flexibility. Distributed systems involve the distribution of data and processing among several nodes that collaborate to

4

execute tasks, as explained by [25]. Decentralisation in this context is the absence of a central controlling node, choosing a more democratic and robust network structure instead. The integration of microservices in these systems signifies an advancement in software engineering.

3. Related Work

This section reviews significant works in BD RAs, detailing their emphasis, research methodologies, and inherent limitations, to endorse the distinctive approach of this study's domain-driven distributed RA, *Terramycelium*. The Lambda architecture design by Kiran [50] and the Kappa architecture by Kreps [53] are important industrial advancements in real-time analytics for BD, setting foundational principles for data processing. Nevertheless, these architectures have faced criticism for their insufficient data management techniques, particularly concerning data quality, security, and metadata [12].

Along the same lines, there have been numerous efforts on RAs in special domains, like the work of Klein on a security driven BD RA [51] and the work of Quintero et al. on BD RAs in the healthcare domain [69]. These RAs, while concentrating on certain domain requirements, sometimes overlook broader concerns such as privacy and interoperability. Academic research, like Viana [90] and Paakkonen [66], has concentrated on enhancing the conceptual comprehension of BD systems through proposed RAs to provide broader views on data analytics ecosystems.

Yet, these recommendations frequently overlook the dynamic and pervasive features of modern data environments, particularly concerning scalability and adaptability. Ataei et al. [10] highlighted the limitations of current BD RAs, emphasising a common trend: a substantial reliance on monolithic data pipeline architectures.

Existing RAs' reliance on centralization is apparent due to their inefficiency in managing data quality, security, privacy, and metadata. Additionally, the inflexible framework of these systems often results in difficulties with scalability and adaptation, making it challenging for them to accommodate changes in data and technology settings. To this end, Terramycelium is presented as a novel artefact to address some of these challenges. Terramycelium addresses the major limitations of current RAs by adhering to tackle complexity and breaking down the bigger problem into smaller, well-managed ones. Moreover, Terramycelium advocates for decentralised data ownership and a modular architecture, as opposed to traditional systems that usually

5

have data management in rigid, monolithic structures. This artefact promotes an event-driven, asynchronous style of communication while empowering domains with simple local rules to embody the idea of complex adaptive systems.

Moreover, Terramycelium enhances scalability and adaptability by addressing cross-cutting concerns such as security, privacy, and data quality. Terramycelium is a departure from traditional BD RAs by focusing on domain-driven distributed processing. This artefact offers a scalable and adaptable framework that aligns well with current data management needs and contemporary software architecture concepts. Terramycelium not only addresses the limitations of present RAs but also paves the way for future research and development in this important field.

4. Research Methodology

Various methods exist for the structured creation of RAs. Cloutier et al. [23] present a methodology for creating RAs by gathering current architectural trends and innovations. Bayer et al. [16] present a method named PuLSE DSSA for generating RAs in product line development. The concept of a pattern-based runtime adaptation for service-based systems is introduced by Stricker et al. [86], who emphasise the usage of patterns as primary entities. The authors Nakagawa et al. [59] present a four-step method for creating and advancing RAs.

Guided by ISO/IEC 26550 [1], Derras et al. [28] provide a four-phase method for creating RAs in the software product line environment. Moreover, Galster et al. [33] suggest a 6-step process based on two main concepts: empirical foundation and empirical validity. Considering all these methodologies, the 6-step methodology of empirically grounded RAs presented by Galster et al. [33] is the most suitable approach for this study. This methodology is preferred over others because, firstly, it's got a good pedigree, and secondly, it aligns with the objectives of this study.

Nevertheless, the methodology has limitations. Additional approaches must be incorporated into the methodology to achieve the appropriate level of rigour and relevance. Specific instructions for gathering empirical data in step three of the methodology are absent. Data collection, synthesis, and modelling is not clearly discussed. For this purpose, Nakagawa et al. [60] presented research guidelines and introduced the RAModel concept. We absorbed the RAModel concept to increase the systematicity and rigour of

6

this study. In addition, a more systematic and robust evaluation technique was needed, as Galster et al.'s methodology lacks details on how to assess the RA. To solve this issue, a case-mechanism experiment is used to assess the artefact as presented in the works of Wirienga [92].

Together, this study adopts the methodology of empirically grounded RAs presesnted by Galster et al. [33], with the addition of the RAModel concept from Nakagawa et al. [61] and the evaluation method of case-mechanism experiment as delineated by Wirienga [92].

4.1. Step 1: Determination of RA Type

The first step in creating the RA was choosing its kind using Angelov et al.'s classification framework [7], which divides RAs into standardisation RAs and facilitation RAs. This decision is fundamental as it directs the following stages of information gathering and risk assessment development:.

The classification framework, which considers context, aims, and design dimensions, was crucial in determining the most suitable RA type for the study's objectives. The method takes a systematic approach by employing key interrogatives such as 'When', 'Where', 'Who' for context, 'Why' for goals, and 'How' and 'What' for design to efficiently categorise RAs.

The study merged Angelov's classification with insights from a recent Systematic Literature Review (SLR) on BD RAs presented by Ataei et al. [10]. The goal of Terramycelium is to facilitate BD system development and enhance an efficient, adaptable data architecture. Therefore, this artefact is classified as a *standardisation RA intended to be adaptable in many organisational settings.*

4.2. Step 2: Design Strategy Selection

The design approach for the RA was influenced by the frameworks proposed by Angelov et al. [8] and Galster et al. [33], which describe two main methodologies: practice-driven (creating RAs from the beginning) and research-driven (building RAs based on pre-existing ones). Practice-driven RAs are uncommon and usually found in emerging areas, while research-driven RAs, which combine current designs, models, and best practices, are more common in established sectors.

This study chooses a research-driven strategy. The RA was created by utilising existing RAs, the contemporary body of academic knowledge, and documented best practices. This method allows for the development of a detailed design theories that combines and expands on existing knowledge.

7

4.3. Step 3: Empirical Data Collection

In the development of the artefact, a foundational aspect of the methodology involved harnessing the insights from two comprehensive SLRs specifically focused on BD RAs. These SLRs ([10] ,[12]) were pivotal not only for their breadth and depth in covering the current state of BD RAs but also for their inclusivity of a wide range of sources, including industrial white papers and grey literature. The deliberate inclusion of such documents in these SLRs provided us with a rich, multifaceted understanding of BD RAs, incorporating both academic research and practical, real-world applications from the industry.

Given the extensive coverage of existing standards, frameworks, and best practices within these SLRs, we deemed it unnecessary to independently revisit or replicate the analysis of such documents. The SLRs had already undertaken a rigorous examination of these areas, ensuring that our RA was built upon a foundation that was both current and aligned with industry standards.

In addition to the insights gained from the SLRs, the methodology was enriched by the inclusion of a study on the application of microservices patterns to BD systems [13]. This study, notable for its use of expert opinion, provided a nuanced perspective on how microservices architectures could be leveraged within the context of BD, offering valuable design patterns and architectural insights. The integration of findings from this study with those of the SLRs allowed us to construct an RA that not only reflects the current landscape of BD architectures but also incorporates forward-thinking design principles enabled by microservices.

By drawing on these recent, comprehensive studies, we were able to forego the repetition of gathering and analysing similar sets of data. Instead, our focus was directed towards synthesising these findings to form the basis of our artefact. This approach ensured that our RA was developed with an appreciation for the current state of knowledge and practice within the BD domain, supported by high-quality, authoritative sources.

4.4. Step4: Construction of the RA

The construction phase of the RA was guided by the findings and elements identified in the previous steps of the research. Based on the ISO/IEC/IEEE 42010 standard [39], the construction of the RA involves selectively choosing and integrating architectural constructs.

8

The artefact is created using the Architectural Definition Language (ADL) called Archimate, which is part of the ISO/IEC/IEEE 42010 standard. Archimate's service-oriented strategy efficiently connected the application, business, and infrastructure layers of the RA. The incorporation of themes, theories, and patterns from the empirical data gathered from the previous phase guaranteed that the artefact design addressed the objective of the study.

Using Archimate's various viewpoints, a comprehensive perspective on the RA is delineated, encompassing technical, business, and consumer context views. This method, in accordance with the ideas presented by Cloutier et al. [23] and Stricker et al. [86], facilitated a thorough comprehension of the RA, guaranteeing its congruence with the study's goals and context.

4.5. Step 5: Activating RA with variability

Integrating variability into RAs is crucial to ensuring its application within organisational-specific legislation and regional policies. Variability management is essential in Business Process Management (BPM) and Software Product Line Engineering (SPLE) to customise business processes and software artefacts according to specific contextual requirements.

Accurate identification and clear communication of variability are essential to encourage stakeholder discussion, maintain decision traceability, and aid the decision-making process [26]. Galster et al. [33] outline three approaches to including variability in RAs: 1) annotating the RA, 2) constructing variability views, and 3) forming variability models.

Current literature lacks in-depth information on the criteria for choosing a method to facilitate variability. This study utilises Archimate annotations to incorporate variability into the RA, following the method proposed by Rurua [78]. The technique involves two stages: first, developing a customised layer to encompass fundamental variability principles, and second, annotating the RA. The aim is to highlight the main architectural variabilities associated with the system for architects to consider in order to improve the design and make it easier to implement the RA.

4.6. Step 6: Evaluation of the RA

Evaluating the RA is essential to ensuring it achieves its developmental goals, especially in terms of efficacy and usefulness [34]. Assessing a RA presents distinct difficulties because of its elevated level of abstraction, lack of clear stakeholder groups, and emphasis on high-level architectural features rather than concrete context-driven implementations.

Common evaluation techniques for concrete architecture, like SAAM [45], ALMA [17], PASA [93], and ATAM [46], are not suitable for RAs because they heavily depend on stakeholder participation and scenario-based assessment. This requires a customisation of these methods for evaluating RA.

To this end, the evaluation approach of this study includes developing a prototype of the RA for a case-mechanism experiment as presented by Wieringa [92].

5. Why Reference Architectures

Viewing the system as a RA aids in comprehending its main elements, behaviour, structure, and development, which subsequently impact quality characteristics like maintainability, scalability, and performance [23]. RAs can serve as a valuable standardisation tool and a means of communication that leads to specific architectures for BD systems.

They also offer stakeholders a common set of elements and symbols to facilitate discussions and advance BD projects. Utilising RAs for system conceptualization and as a standardisation object is a common technique among those dealing with complex systems. Software Product Line (SPL) development uses RAs as generic artefacts that are customised and configured for a specific system domain. IBM and other top IT companies have continually supported the use of RAs as exemplary techniques for solving complex system design difficulties in software engineering [10].

RAs often act as instruments for establishing uniformity in new areas within international standards. The BS ISO/IEC 18384-1 RA [40] for service-oriented architectures showcases the effectiveness of RAs in establishing standardised frameworks in particular domains. In summary, the strategic application of RAs in the development of Big Data systems provides a structured methodology that enhances system design, fosters effective communication among stakeholders, and supports the standardisation of complex architectures.

6. Software and System Requirements

According to Wieringa [92], the requirement specification phase is an essential step in developing a new Information System's (IS) artefact. This phase involves identifying the requirements that the artefact must satisfy to meet the needs of stakeholders and achieve the desired outcomes.

Wieringa's methodology distinguishes between functional requirements, which describe what the artefact should do, and non-functional requirements, which describe how the artefact should do it. Functional requirements are typically expressed as use cases, which describe the specific interactions between users and the system. Non-functional requirements may include performance requirements, security requirements, and usability requirements.

The requirement specification designed for this study is made up of the following phases:

1. Determining the type of requirements

2. Determining the relevant requirements and

3. Identifying the right approach for categorising the requirements

4. Identifying the right approach for the presentation of the requirements

6.1. Determining the type of the requirements:

Defining and classifying software and system requirements is a common subject of debate. Sommervile [83] classifies requirements as three levels of abstraction; user requirements, system requirements, and design specifications. These abstractions are then mapped against user acceptance testing, integration testing, and unit testing. Nevertheless, in this study, a more general framework provided by Laplante [55] is adopted. The adopted approach provides three types of requirements: functional, non-functional, and domain requirements. The objective of this step is to define the high-level requirements of BD systems; therefore, the main focus is on functional and non-functional requirements.

6.2. Determining the relevant requirements:

In an extensive effort, the NIST BD Public Working Group embarked on a large-scale study to extract requirements from a variety of application domains such as Healthcare, Life Sciences, Commercial, Energy, Government, and Defense [21]. The result of this study is the formation of general requirements into seven categories. In addition, Volk et al. [91] categorise nine use cases of BD projects sourced from published literature using a hierarchical clustering algorithm.

Rad et al. [15] focus on security and privacy requirements for BD systems, Yu et al. [96] present modern components of BD systems, using goal-oriented

11

approaches, Eridaputra et al. [30] created a generic model for BD requirements, and Al-Jaroodi et al. [6] investigate general requirements to support BD software development.

By analysing the results of the first SLRs, the studies discussed above, and by evaluating the design and requirement engineering required for BD RAs, the relevant requirements are deemed to be the high-level requirements based on the characteristics of BD presented in the works of Rad et al. [72]. That is, the types of requirements determined for this study are based on the characteristics of BD. This is explored further in subsequent sections.

6.3. Identifying the right approach for categorising the requirements:

After defining the types of requirements and identifying relevant requirements, an assessment of current BD RAs and their associated requirements was conducted to enhance our understanding of the existing methods for categorizing BD requirements. This assessment revealed a unifying theme across these studies, suggesting a consensus on a methodological approach to requirement classification. Consequently, this theme informed our categorization strategy, leading us to organise the requirements based on intrinsic BD characteristics. Specifically, the requirements were classified according to critical dimensions of BD, namely velocity, veracity, volume, variety, and value, as well as considerations of security and privacy. This classification approach aligns with the findings from key studies in the field, including works by Ataei et al. [10], Bahrami et al. [14], Rad et al. [70], and Chen et al. [22], which collectively underscore the significance of these characteristics in defining and addressing the requirements of BD systems.

6.4. Determining the appropriate strategy for presenting the requirements:

A systematic strategy for presenting software and system requirements that includes informal model verification techniques is recognised due to its established use in both business and academia [44]. The method for illustrating functional requirements adheres to the principles stated in the ISO/IEC/IEEE standard 29148 [41]. The requirements representation is structured based on system modes, detailing the primary components of the system, followed by the requirements. This technique is based on the requirement specification published for the NASA Wide-field InfraRed Explorer (WIRE) system [62] and the Software Engineering Body of Knowledge (SEBoK) [5]. The requirements are outlined in Table 1.

Category	Code	Requirement
Volume	Vol-1	System needs to support asynchronous, streaming, and batch processing to collect data from centralized, distributed, and other sources.
Volume	Vol-2	System needs to provide scalable storage for massive data sets.
Velocity	Vel-1	System needs to support slow, bursty, and high throughput data transmission between data sources.
Velocity	Vel-2	System needs to stream data to data consumers in a timely manner.
Velocity	Vel-3	System needs to be able to ingest multiple, continuous, time-varying data streams.
Velocity	Vel-4	System shall support fast search from streaming and processed data with high accuracy and relevancy.
Velocity	Vel-5	System should be able to process data in a real-time or near real-time manner.
Variety	Var-1	System needs to support data in various formats ranging from structured to semi-structured and unstructured data.
Variety	Var-2	System needs to support aggregation, standardization, and normalization of data from disparate sources.
Variety	Var-3	System shall support adaptation mechanisms for schema evolution.
Variety	Var-4	System can provide mechanisms to automatically include new data sources.

Category	Code	Requirement
Value	Val-1	System needs to be able to handle compute-intensive analytical processing and machine learning techniques.
Value	Val-2	System needs to support two types of analytical processing: batch and streaming.
Value	Val-3	System needs to support different output file formats for different purposes.
Value	Val-4	System needs to support streaming results to the consumers.
Security & Privacy	SaP-1	System needs to protect and retain the privacy and security of sensitive data.
Security & Privacy	SaP-2	System needs to have access control, and multi-level, policy-driven authentication on protected data and processing nodes.
Veracity	Ver-1	System needs to support data quality curation including classification, pre-processing, format reduction, and transformation.
Veracity	Ver-2	System needs to support data provenance including data life cycle management and long-term preservation.

Table 1: Terramycelium Software and System Requirements

7. Theory

An inflection point in mathematics is a critical point where the curvature of a curve changes direction, indicating a shift from one behaviour to another [94]. This significant moment is characterised by the breakdown of the previous structure and the emergence of a new one. Currently, there are concrete indicators and factors that suggest the change is imminent. The 2023 New Vantage Partners research indicates that investments in data are increas-

ing and staying robust, despite possible economic challenges [67]. However, the survey indicates that only 23.9% of organisations describe themselves as data-driven.

This section is aimed at delineating theories that underpin the artefact. These theories are referred to as *design theories* in design science research [77]. These theories form the basis for understanding the underlying principles that govern the design of Terramycelium. This study utilises the architecture definition defined in ISO/IEC 42010 [39], which defines architecture as the underlying concepts or properties of a system within its context, manifested in its elements, relationships, and design and evolution principles.

7.1. The monolith

The techniques and methods of data engineering have experienced rapid growth, similar to the Cambrian explosion, yet the fundamental assumptions guiding data designs have not been significantly questioned [27]. As per Richards [74], architectures can be classified into two main categories: monolithic, deployed as a single entity, and distributed, consisting of individually deployed sub-components. Currently, the majority of data architectures fall under the first type [10].

Beginning with a monolith may be a straightforward and effective method for constructing a data-intensive system, but it becomes inadequate as the system grows. Although this notion is being questioned in the software engineering field with the emergence of microservices patterns [76], data engineering continues to be influenced by monolithic designs. Enabler technologies like data warehouses and data lakes support these concepts.

In contrast, modern data architecture paradigms like Data Mesh [27] and Data Fabric [35] and the works of Ataei et al. [11] have gained traction as alternatives to monolithic designs. Data Mesh advocates for a decentralised approach to data management, where data ownership and governance are distributed across different domains or teams within an organisation. This shift towards federated governance addresses some of the challenges associated with monolithic architectures, such as scalability, agility, and data silos [18].

Data Fabric, on the other hand, emphasises the seamless integration of data from various sources and formats, providing a unified view of data across an organisation. By leveraging technologies like distributed computing, data virtualization, and metadata management, Data Fabric enables organisations

15

to access and analyse data in real-time, regardless of its location or format. This approach enhances data agility, accessibility, and reliability [18].

As organisations navigate the complexities of modern data ecosystems, the transition from monolithic designs to more agile and scalable architectures becomes imperative. These approaches offer a more flexible and decentralised model for managing data, aligning with the evolving needs of data engineering in today's dynamic and data-intensive environments.

Reflecting on this transition, Terramycelium has learned a lesson from the shift towards more agile and scalable data architectures. Embracing a domain-driven distributed architecture, Terramycelium incorporates the concepts of complex adaptive systems and event-driven systems, signifying a move away from traditional, rigid structures towards a more adaptable and responsive data architecture model. This evolution marks a step forward in how data ecosystems are designed and managed.

7.2. The data chasm

Analytical data and operational data are two separate categories of data used in corporate operations. Operational data is utilised for daily corporate operations, whereas analytical data is employed for strategic decision-making by recognising patterns and trends in previous data [49].

The problems of current BD systems stem from the basic assumption of segregating operational and analytical data [10]. Operational and analytical data have distinct characteristics and processing methods. Moving operational data away from its source can harm its integrity, lead to organisational silos, and result in data quality problems.

These two distinct data planes are often managed within separate organisational hierarchies. Data scientists, business intelligence analysts, machine learning engineers, data stewards, and data engineers typically work under the guidance of the Chief Data and Analytics Officer (CDAO) to generate business value from data. Software developers, product owners, and quality assurance engineers typically collaborate with the Chief Technology Officer (CTO) [27].

As a consequence, there are now two separate technology stacks in place, requiring significant resources to connect them (Figure 1). Data extraction from operational databases is often accomplished through a batch ETL process. These ETL processes typically lack a clearly defined agreement with the operational database and solely consume its data. This emphasises the vulnerability of this system, as modifications made to operational databases

16

can impact analytical applications downstream. As time passes, the complexity of ETL jobs grows, making them harder to maintain and leading to a decline in data quality.
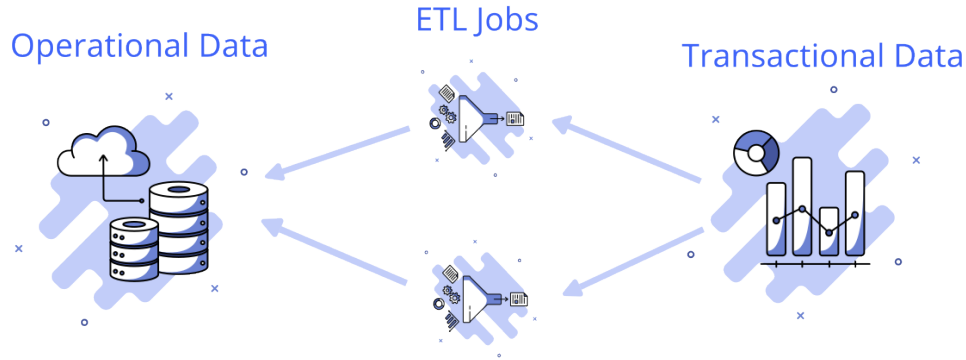

Figure 1: The great divide of data

Many technologies developed over the years are based on this architecture. Although these technologies excel in managing the amount, speed, and diversity of data, current data difficulties revolve around the increase in data sources, data quality, and the complexities of data architecture.

TerraMycelium utilises earlier solutions to improve and overcome their limitations. This artefact tries to move away from highly centralised and rigid data architectures that serve as a bottleneck for coordination. This artefact aims to connect the point of data generation with its utilisation, streamlining the process. This artefact is designed to enhance agility in response to growth and successfully adapt to organisational changes. To achieve that, the first step is the introduction of data into domains.

7.3. Localized autonomy through domain-driven design

Modern enterprises are facing significant complexity. A typical business consists of diverse domains with distinct structures. These domains exhibit varying rates of change and are typically segregated from one another. The business's overall synergy is determined by the relationships and evolution of various domains.

The core of these synergies is characterised by volatility, frequent market changes, and a growing number of rules [68]. How do modern organisations

17

handle the effects of these developments on their data? Should they consistently update ETL tasks, develop new ETL pipelines, and merge data into operational repositories? How can firms generate reliable and high-quality data efficiently? It comes down to accepting the change in the current data environment.

To address this complexity, one approach is to synchronise technology with business objectives. Businesses divide their challenge into smaller components within each domain, integrating technology and data into these areas. This method is well recognised in microservices architectures (Ataei, 2023).

Central to Terramycelium is the dispersal and decentralisation of services into distinct areas with well-defined boundaries. One of the most difficult aspects of designing a distributed system is determining the architectural quanta on which the system should be divided. This matter has been frequently deliberated among proponents of microservices architecture [57].Terramycelium, influenced by the works of Eric Evans [31] and Dehghani [27] places data in close proximity to the product domain it pertains to. This suggests that data is within the product domain and is a component of it.

Most businesses nowadays are structured around their products, which is the fundamental driving force behind this approach. These products represent the business's capacity and are divided into different domains. These domains often establish their defined scope, progress at varying speeds, and are managed by interdisciplinary teams [82]. Integrating data into these specific domains can provide a synergistic effect that enhances the handling of ongoing changes. That is, if data engineering becomes a facet of these products, then there is no need for the sycnrhonization tax that is introduced between operational and analytical systems.

Communication can occur on a micro level between application developers and data engineers over a type definition change in a GraphQL node or on a macro level when application developers consider rebuilding their Protobuff schema in a way that impacts downstream analytical services. Thus, this study integrates the concept of DDD to enhance communication and improve the acceptance, precision, and significance of Terramycelium.

DDD is a software development methodology that emphasises comprehending and representing the problem domain of a software application. The objective of DDD is to develop software that mirrors the language, concepts, and behaviours of the problem domain, prioritising these aspects over technical factors.

DDD can assist Terramycelium by offering a structured method for modelling and handling data that is well-matched with the application's issue domain. DDD can assist data architects in obtaining a more profound comprehension of the necessary data and its structure by concentrating on the language, concepts, and behaviours of the problem domain [27]. Effective communication is crucial in software development projects, as it facilitates the sharing of vital knowledge [87].

Data engineers and business stakeholders frequently do not connect directly. Domain knowledge is conveyed through intermediaries like business analysts or project managers into a set of activities to be completed [48]. This indicates the need for at least two translations from two distinct ontologies.

Each translation results in the loss of crucial subject expertise, posing a danger to the overall data quality. During a data engineering process, the requirements can become twisted, and the data engineer may lack awareness of the specific business domain or problem being addressed [47].

Data engineering challenges are typically complex and wide-ranging, rather than simple mathematics problems or puzzles. An organisation may choose to enhance workflows and processes by consistently making decisions based on data. However, a rigid and inflexible centralised data architecture can increase the likelihood of project failure. After having the problem broken down to domains, we have to apply some system thinking, that is, should the data centralization break into different data domains, how do we ensure that these domains are healthy and evolving? This is where the concepts of complex adaptive systems are applied.

7.4. Complex adaptive systems

Terramycelium architectures have characteristics similar to those of complex adaptive systems (Holland, 1992). The artefact created for this study is inspired by the concept that powerful groups might form based on basic rules that control individual agents. For instance, Reynolds [73] examined a coordinated group of starling birds during the autumn season. The study revealed that each starling bird adheres to three basic rules: alignment, separation, and cohesion. The rules can be mathematically articulated as follows:

Alignment:

$$v_i(t+1) = v_i(t) + \frac{1}{k}\sum_{j=1}^{N}(v_j(t) - v_i(t))$$

19

where $v_i(t)$ is the velocity vector of bird i at time t, k is a normalization factor, and N is the number of neighbouring birds.

Cohesion:

$$v_i(t+1) = v_i(t) + \frac{1}{k}(c_i(t) - p_i(t))$$

where $c_i(t)$ is the center of mass of the neighbouring birds, $p_i(t)$ is the position of bird i at time t, and k is a normalization factor.

Separation:

$$v_i(t+1) = v_i(t) + \sum_{j=1}^{N} \frac{(p_i(t) - p_j(t))}{d_{ij}^2}$$

where $p_i(t)$ is the position of bird i at time t, $p_j(t)$ is the position of bird j at time t, and d_{ij} is the distance between birds i and j.

Starling birds do not require a centralised orchestrator to form this intricate adaptive system. Terramycelium aims to encourage a domain-driven allocation of data ownership. This architecture is designed to offer not just operational data but also analytical data through a standard interface. For instance, the practice management software for veterinarians includes operational APIs for updating animal attributes and analytical interfaces for retrieving animal data within a specific time frame. Each domain owns its own data.

The domain can retrieve data from other domains using a discovery mechanism, process the data, and augment its own data. Aggregate domains are created to collect data from different domains and make it available for a certain purpose.

This aims to eliminate vertical dependency and enable teams to have local autonomy while being equipped with the appropriate amount of discovery and APIs. This architecture emphasises the concept of coequal nodes working together to accomplish the system's overarching objective, rather than relying on a centralised database managed by individuals lacking domain expertise. This idea is influenced by DDD as described by Evans [31], Data Mesh by Dehghani [27], microservices architecture by Newman [64] and Complext Adaptive systems by Lansing [54].

7.5. Event driven services

Terramycelium's decentralised and distributed structure poses difficulties in service communication as the network expands. At first, basic point-

to-point communication via REST API calls is adequate, but it becomes inefficient as the system grows. Synchronous interactions can result in a 'distribution tax', where one service's blocked state, generally caused by costly procedures, leads to delays in dependent services [58].

Distributed systems' extensive network requirements can lead to complications such as tail latency, context switching, and gridlocks. This strong interconnection contradicts the distributed system's objectives of independence and robustness.

Terramycelium uses asynchronous, event-driven communication to address these concerns. This approach allows services to publish and respond to events, separating their interactions. Within the 'publish and forget' paradigm, services broadcast events to designated topics and proceed without waiting for immediate reactions.

Event-driven architectures usually provide eventual consistency, which may not be ideal for some real-time stream processing situations. However, it is generally safe to assume that most data engineering tasks can effectively function within an event-driven framework.

As per Ford et al. [32], event-driven architectures consist of two primary topologies: 1) broker topology and 2) mediator topology. The concept of streaming platforms is further explained in the writings of Stopford [85]. Terramycelium, as a BD architecture designed for processing analytical data, eliminates several issues associated with accomplishing ACID transactions. Terramycelium utilises the architectural principles of distributed, asynchronous, event-driven systems with a mixed topology. Terramycelium is incorporating aspects of broker topology, mediator topology, and several notions from streaming platforms. Terramycelium's CAP theorem [81] prioritises partition tolerance and availability as key assurances.

Terramycelium's hybrid topology consists of five main architectural components: the event, the event consumer, the event producer, the event backbone, and the eventing interface. Events are triggered by the event producer and sent to the relevant topic through the eventing interface. The event is processed by the event broker and stored in a queue-like indexed data structure for later retrieval inside the event backbone. Interested event consumers will listen to the topic of interest using the eventing interface. The event backbone is an internally distributed system composed of numerous event brokers.

Event brokers are services created and provided to enable event communication via Terramycelium. The brokers are synchronised with a decentralised

service coordinator. Brokers also enable the replication of topics. Additionally, the event backbone includes a specialised event archive to ensure fault tolerance and recoverability. The purpose of this archive is to retain all events
635 processed by the brokers for potential restoration in the event of a failure.

These components collaborate to establish a distributed, fault-tolerant, and scalable data system capable of managing both batch and stream processing, as seen in Figure 2.
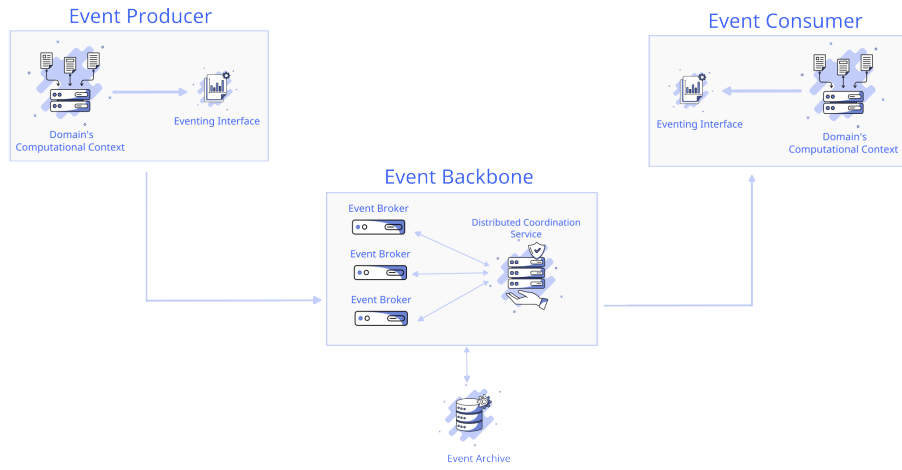

Figure 2: Terramycelium Event-driven Architecture

7.6. Architectural characteristics

640 After discussing the main design theories that underpin Terramycelium, in this section, the overall architectural characteristics of this artefact are discussed. The Terramycelium architecture is distinguished by its emphasis on maintainability, scalability, fault tolerance, elasticity, and deployability. It is in line with contemporary engineering methodologies like automated deploy-
645 ment and continuous integration [32]. The architecture focuses on utilising microservices, which are autonomous components that can be deployed and maintained separately.

This architecture emphasises improved maintainability in comparison to current data architectures. Event-driven microservices architecture enables
650 modular development and independent scaling, facilitating the maintenance

and updating of individual components without impacting the entire system. The design also enables automated deployment processes, which help streamline upgrades and minimise manual interaction.

Scalability and elasticity are significant aspects as well. The architecture allows for horizontal scalability, enabling the addition or removal of services according to demand. This adaptability guarantees that the system can efficiently manage different workloads.

The architecture also demonstrates strong fault tolerance. Interservice communication can affect fault tolerance; however, a redundant and scalable design of Terramycelium, combined with service discovery techniques, helps alleviate this problem. Microservices' standalone and single-purpose design typically results in a high level of fault tolerance [29].

On the other hand, the architecture might be rated lower in terms of cost and simplicity. The decentralised structure of Terramycelium and the possibility of higher communication overhead can create challenges in cost management and optimisation. Implementing strategies like intelligent data caching and replication can help overcome performance issues related to network calls, but managing costs is still a continuous concern.

The Terramycelium design emphasises the advantages of microservices, focusing on maintainability, scalability, fault tolerance, elasticity, and deployability. The text recognises the difficulties that come with distributed architectures and provides methods to reduce their impact. Architects need to comprehend the principles of architecture in order to skillfully traverse and capitalise on its advantages. An overview of architectural characteristics is portrayed in Table 2

8. Artifact

After exploring the underpinning design theories, a solid theoretical foundation is established for creating and developing the artefact. Terramycelium is generated using Archimate and mostly showcases the RA within the technological layer. The architect has the responsibility to determine the flow and application that should be present in each node when showcasing these services in the technology layer. To ensure thoroughness, a basic business process is assumed, as every piece of software is created to fulfil a business requirement. Terramycelium should possess the flexibility needed to accommodate alternative business models, notwithstanding potential variations in the business layer across different settings.

23

Table 2: Terramycelium Architecture Characteristics

Characteristic	Score
Maintainability	★★★
Scalability	★★★★
Fault Tolerance	★★★
Elasticity	★★★★
Deployability	★★★★
Cost	★★
Simplicity	★
Performance	★★★
Support for Modern Engineering Practices	★★★★

This BD RA does not depict the architecture of any particular BD system. It functions as a versatile tool for describing, discussing, and constructing system-specific designs based on standardised principles. Terramycelium enables in-depth and insightful talks about the needs, frameworks, and functions of BD systems. The system stays vendor-neutral, providing flexibility in choosing products or services, and avoids imposing strict solutions that restrict innovation.

To ensure thoroughness, we have assumed a basic BD business process, as all software is created to fulfil a specific business requirement. Terramycelium consists of 15 primary components and 5 variable components, as seen in Figure 3. The lowercase "a" in the top left corner of the diagram represents the auxiliary view, while the letter "m" represents the master view. When the same entity is utilised in various models, the auxiliary view is employed. This indicates that the entity already exists and is being reused.

While this business layer could vary in different contexts, Terramycelium should be able to have the elasticity required to account for various business models. To ease understanding of the RA, we sub-diagrammed the product domain in Figure 4.
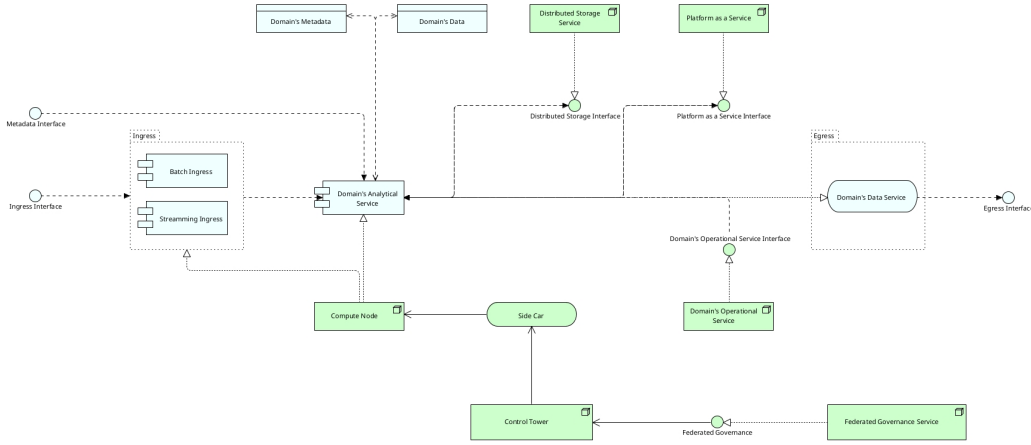
These components are;

Figure 3: Terramycelium

Figure 4: Service Mesh

1. **Ingress Gateway:** The ingress gateway is pivotal in Terramycelium, acting as the main entry point for external requests. It exposes necessary ports and endpoints, intelligently load-balancing traffic to the appropriate processing controllers, thereby enhancing the system's scalability and performance.

    Security is significantly bolstered by the ingress gateway, which prevents direct service access by centralising entry and enabling the enforcement of security policies such as SSL termination and authentication. This approach secures communication with external entities and streamlines SSL processing, offloading this task from downstream services.

    The gateway also allows for incoming data modification or enrichment, offering customisation and system integration capabilities. This architectural component simplifies monitoring and observability, concentrating metrics, logging, and tracing at a singular point for efficient analysis and compliance oversight. Additionally, it supports network segmentation, establishing a clear demarcation between public interfaces and private network resources, thus reinforcing security and control over data flow.

    Addresses requirements: Vol-1, Vol-2, Var-1, Var-3, Var-4, Val-1, Val-3, Val-4, SaP-1, and SaP-2.

2. **Batch Processing Controller:** The primary function of the batch

26

processing controller is to manage batch events by sending them to the event backbone. The dedicated service, which may be a tiny service or a Lambda function, processes batch-processing requests and converts them into events for the event broker. Its unique characteristics enable it to handle batch requests in large quantities and asynchronously, distinguishing it from stream processing controllers.

The batch-processing controller offers various architectural advantages. Firstly, it greatly improves monitoring capabilities, offering clearer insight into the status and performance of batch activities. This efficient monitoring simplifies problem-solving and performance evaluation, guaranteeing that batch processing adheres to defined criteria and performance standards.

The batch processing controller allows for customisation and flexibility in generating batch events. It can carry out supplementary functions apart from computationally demanding activities, including data cleansing or incorporating personalised headers. Businesses can customise batch event production to meet their individual needs and expectations.

A specialised controller for batch processing recognises the distinct needs and attributes of batch events, offering tailored features and enhancements. This component fulfils the criteria.

Addresses requirements: Vel-1, Val-1, and Val-2.

3. **Stream Processing Controller:** The stream processing controller is a crucial component in architecture, handling streaming events and dispatching them to the event backbone. It is distinct from batch processing services as it focuses on non-heavy computations optimized for stream processing requirements. It can enable stream provenance, providing insights for data governance and traceability. It can also leverage one-pass algorithms for stream analytics tasks.

A dedicated stream processing controller allows for the association of custom attributes with stream events, enabling contextual enrichment of streaming data and differentiated treatment based on the system's nature. This enables real-time decision-making or targeted actions based on event content.

27

Simplifying monitoring and discovery within the architecture, the controller simplifies tracking and analysing the performance, latency, and throughput of streaming events. It also enables focused monitoring of stream-specific metrics, providing valuable insights into the behaviour and efficiency of the streaming data pipeline.

In summary, the stream processing controller is an essential component in architectures involving streaming data due to its ability to handle high throughput, apply custom attributes, optimise computations, and simplify monitoring and discovery.

Addresses requirements: Vol-1, Vel-1, Vel-2, Vel-4, Vel-5, and Val-2.

4. **Event Processing Interface:** Event brokers are designed to achieve *inversion of control.* As the company evolves and requirements emerge, the number of nodes or services increases, new regions of operations may be added, and new events might need to be dispatched. As each service has to communicate with the rest through the event backbone, each service will be required to implement its event-handling module.

This can easily turn into a spaghetti of incompatible implementations by various teams, and can even cause bugs and unexpected behaviours. To overcome this challenge, an event broker is introduced to each service of the architecture. Each service connects to its local event broker and publishes and subscribes to events through that broker. One of the key success criteria of the event broker is a unified interface that sits at the right level of abstraction to account for all services in the architecture.

Event brokers, being environmentally agnostic, can be deployed on any on-premise, private, or public infrastructure. This frees up engineers from having to think about the event interface they have to implement and how it should behave. Event brokers can also account for more dynamism by learning which events should be routed to which consumer applications. Moreover, event brokers do also implement circuit breaking, which means if the service they have to break to is not available and does not respond for a certain amount of time, the broker establishes the unavailability of the service to the rest of the services, so no further requests come through. This is essential to preventing a ripple effect over the whole system if one system fails.

Addresses requirements: Val-1, Ver-1.

5. **Event Backbone:** The event backbone is the central component of the Terramycelium, enabling communication between nodes. Distribution and clustering should be implemented to handle the system's scalability. Communication is orchestrated by coordinated activities, like a dance ensemble. Every service monitors and responds to the event backbone, performing the required actions.

   This guarantees an uninterrupted exchange of data between services, ensuring that all systems are in the appropriate condition. The event backbone can combine many event streams, store events in a cache, archive events, and perform other operations, as long as it does not become overly intelligent or function as an Enterprise Service Bus (ESB) in Service-Oriented Architecture (SOA) systems. An architect should see the event backbone as a collection of interconnected nodes that address different subjects of interest. The event backbone can be monitored over time to analyse access patterns and adjusted to optimise communication efficiency.

   Addresses requirements: Vel-1, Vel-2, Vel-3, Vel-4, Vel-5, Val-1, Val-2, Ver-1, Ver-2, and Ver-3.

6. **Egress Gateway:** The inclusion of an egress gateway in Terramycelium offers numerous advantages, especially for external data consumers. In this architecture, data consumers first interact with the discovery component, known as Data Lichen, which serves as a central hub for accessing available data domains. Once connected to Data Lichen, data consumers can navigate to the relevant data domain to retrieve the desired data.

   Furthermore, all external data consumers in the system go through a centralised secret management and centralised authentication and authorization component. This centralised approach brings several benefits to the architecture. Firstly, it ensures a consistent and secure management of secrets, such as API keys, access tokens, or credentials, which are essential for data access and security. This centralised secret management enhances the overall system's security posture by reducing the chances of vulnerabilities or misconfigurations in secret handling.

   Secondly, the centralised authentication and authorization component streamlines the authentication and access control processes for external data consumers. By enforcing a unified authentication mechanism, it

29

ensures that all users are properly authenticated and authorised before accessing the system's data resources. This centralised approach simplifies the management of user access rights, improves security, and provides granular control over data access permissions.

Thirdly, the centralised components simplify the maintenance and scalability of the system. With a single point for managing secrets, authentication, and authorization, it becomes easier to update, monitor, and audit these components. Additionally, this architectural pattern allows for easier scaling and expansion as new data domains or data sources can be seamlessly integrated into the system with consistent authentication and authorization mechanisms.

Overall, the inclusion of an egress gateway in Terramycelium offers a robust and efficient approach for external data consumers. It ensures standardised data access, enhances security, simplifies maintenance, and enables scalability, making it a highly favourable and beneficial architectural design.

Addresses requirements: Vel-2, Vel-4, Val-3, Val-4, SaP-1, and SaP-2.

7. **Product Domain Service Mesh:** Integrating a service mesh as a core element of each product's domain is an effective strategy in the architectural framework. The service mesh consists of essential components that work together to facilitate effective data processing, storage, governance, and communication within the domain. The components consist of batch and streaming data input, analytical and operational services specific to the domain, API access to distributed storage services, an infrastructure API for platform-as-a-service modules, containers hosting the analytical service, a control tower, and integration with a federated governance service's API for policy enforcement using sidecars.

The service mesh's effectiveness is derived from its architectural design and its capability to meet essential needs. By integrating the domain's functionalities into a service mesh, the interdependence across teams is removed, enabling greater team independence. The components of the service mesh collaborate to provide benefits like data ingestion into the analytical service domain, API connectivity to distributed storage services, container hosting for the analytical service domain, a control

tower, and integration with a federated governance service's API for policy enforcement using sidecars.

The service mesh architecture provides the analytical service with direct access to operational data, connecting data analytics and operational systems. The analytical service can obtain real-time insights and a comprehensive overview of the system's activities, which minimises the requirement for manual data extraction and transformation procedures. Accessing operational data directly improves the efficiency and precision of analytical services, leading to more precise analysis and well-informed decision-making.

The service mesh architecture enables a cohesive perspective of the system, promoting smooth cooperation between analytics and operations teams. The system supports scalability, adapts to changes in business environments, validates data, maintains quality and integrity, and meets security and privacy standards by enforcing policies, ensuring secure communication, and implementing data governance processes.

The service mesh's effectiveness lies in its ability to address key architectural concerns. It promotes scalability, allowing the domain to handle large volumes of data and increasing computational resources as needed (Vol-1). It facilitates rapid development and deployment of analytical capabilities (Vel-3, Vel-4, Vel-5). The service mesh architecture accommodates variability in business contexts, supporting the diverse needs and requirements of different product domains (Var-1, Var-2, Var-3). It ensures data validation, quality, and integrity by leveraging advanced analytics and processing techniques (Val-1, Val-2, Val-3, Val-4). Security and privacy requirements are fulfilled through policy enforcement, secure communication, and data governance mechanisms (Sap-1, SaP-2).

Finally, the service mesh architecture allows for the verification of system behaviour, enabling efficient testing, monitoring, and verification of the domain's analytical outputs (Ver-1, Ver-2, Ver-3).

Addresses requirements: Vol-1, Vel-3, Vel-4, Vel-5, Var-1, Var-2, Var-3, Val-1, Val-2, Val-3, Val-4, Sap-1, SaP-2

8. **Federated Governance Service:** Terramycelium's distributed architecture comprises multiple independent services with distinct lifecycles,

developed and deployed by autonomous teams. To ensure coherence and prevent interface conflicts without centralising control, global federated governance is essential. This governance model standardises services, enhances their interoperability, and facilitates team collaboration. It also mitigates risks, such as non-compliance with GDPR, by establishing a unified framework of global policies, metadata standards, API standards, and security protocols. This framework not only supports organisational practices but also aligns with external regulations, safeguarding against potential fines and security threats. The federated governance service, comprising components like global policies, metadata standards, and security regulations, plays a pivotal role in maintaining the architecture's integrity and adaptability, indirectly impacting all requirements. This component can indirectly affect all requirements.

Addresses requirements: All.

9. **Data Lichen:** With an increasing number of products, more data is accessible for users, leading to enhanced interoperability but also posing greater challenges for maintenance. Without an automated method for different teams to obtain their desired data, a tightly connected and sluggish data culture would develop. To overcome these obstacles and enhance the ability to find, work together, and navigate with guidance, the Data Lichen should be put into practice. Gartner has identified data discovery mechanisms such as Data Lichen as essential tools. These mechanisms facilitate improved communication dynamics, quicker data access through services, and intelligent collaboration among services.

Addresses requirements: Vel-4, Var-1, Var-3, and Var-4.

10. **Telemetry Processor:** The decentralised architecture of Terramycelium poses challenges for debugging, fault identification, and maintenance as it requires tracking transactions across several services. Consolidating the handling of telemetry data in Terramycelium provides architectural advantages by streamlining data processing, offering a cohesive perspective of logs, metrics, and traces, facilitating thorough analysis and correlation, and guaranteeing optimal resource distribution. Centralising the service guarantees uniform monitoring and governance processes throughout the architecture, encouraging compliance with best

practices.

The centralised service can act as a data source for different users, like the Data Lichen, which uses processed telemetry data to create important insights. This service may process data that can be used by customised systems and dashboards, offering adaptability and scalability to fulfil particular business needs. This promotes a data-driven environment, enabling individuals to get valuable information and insights from the analysed telemetry data. The flexibility and ease of access to the telemetry data increase the benefits of the centralised service in the larger BD framework.

Addresses requirements: Vol-1, Vel-1, Val-1, and Ver-1.

11. **Event Archive:** As the quantity of services expands, the subjects in the event backbone rise, leading to a surge in the number of events. If such circumstances occur, there is a possibility of failure leading to a timeout and subsequent loss of a sequence of events. This can lead to the system being in an incorrect state, causing a harmful chain reaction that affects all services. Terramycelium often addresses these issues by utilising an event archive. The event archive is in charge of recording occurrences for future retrieval in case of failure.

During a blackout in a specific area, if the event backbone is down, it can restore itself and return the system to its correct state by retrieving events from the event archive. The event interface manages circuit breaking to prevent services from sending additional requests to the backbone while it is not functioning. The determination of the duration till expiration and the selection of events to be stored are made according to the specific circumstances in which Terramycelium is utilised.

Addresses requirements: Vol-1, Vel-1, Val-1, and Ver-1.

12. **Distributed Storage Service:** Terramycelium supports decentralised and distributed systems but does not require each product domain to have its own unique data store. To avoid duplication, conflicting data storage methods, reduced interoperability across services, and absence of consistent data storage techniques, The distributed storage service is intended to store vast amounts of data in its original format until it is ready to be retrieved for analysis and other uses.

Data can be stored in the distributed storage service with the appropriate domain ownership before being accessed and used by different services. Data of various structures, such as structured, semi-structured, unstructured, and pseudo-structured, can be stored in a distributed storage service before being processed in batches or streams. However, not all data should be sent straight to this service; the data flow depends on the specific context in which the system operates.

Addresses requirements: Vol-2, Vel-1, Var-1, Var-3, Var-4, Val-3.

13. **Platform as a Service:** The Platform as a Service (Paas) component acts as a central hub, providing an API to all other system components. This PaaS component is essential for facilitating the independence and expandability of the entire infrastructure.

    The primary design value of this PaaS component is its capacity to simplify the underlying infrastructure difficulties. By offering a standardised API, each component can autonomously handle and allocate necessary resources like computing power, storage, and networking without having to deal with the complex specifics of the underlying infrastructure. This abstraction layer encourages a minimal connection between components, making it simpler to create, deploy, and maintain the entire system.

    The PaaS component places importance on scalability and elasticity as key design values. It allows for the flexible allocation and deallocation of resources according to the changing requirements of diverse data domains. The PaaS component facilitates effective infrastructure utilisation by providing an API that allows components to request resources as required. The system may adjust resources as needed to maintain peak performance and cost efficiency throughout the whole architecture.

    Data domains can utilise the PaaS API to request, set up, and oversee resources, allowing them to function autonomously and effectively. Decentralising infrastructure management improves agility and flexibility in the system.

    Addresses requirements: SaP-1, SaP-2, Var-1, Var-3, Var-4, Vel-1, and Vol-2.

14. **Identity and Access Management:** The role of the Identity and

Access Management (IAM) component is to guarantee secure and regulated access to the system's resources and data. The concept includes a range of architectural principles necessary for upholding data integrity, privacy, and regulatory compliance.

The IAM component places significant emphasis on authentication and authorization as core design concepts. The system offers strong methods to verify the identities of users, components, and services in the structure, guaranteeing that only approved entities can get the resources and carry out particular tasks. These measures aid in preventing unauthorised access, reducing security risks, and protecting sensitive data.

Another architectural value of the IAM component is its emphasis on centralised user and access management. It acts as a centralised authority for overseeing user identities, responsibilities, and permissions throughout Terramycelium. Centralization optimises access control management, speeds user onboarding and offboarding, and maintains uniform security policy enforcement system-wide.

The IAM component guarantees meticulous access control and privilege management, enabling the creation of precise access policies. The system enables strong authentication using standard protocols such as OAuth and SAML, simplifying user access through Single Sign-On (SSO). It supports audits by recording access occurrences, strengthening compliance, security oversight, and incident management. The Figure 5 provides a detailed representation of how the component functions and how it interacts with other aspects of the system.

Addresses requirements: SaP-1, and SaP-2.

15. **Secret Management System:** The central secret management system is a crucial component for securely storing and maintaining sensitive information, including passwords, API keys, cryptographic keys, and other secrets.

The central secret management system places a strong emphasis on securely storing and encrypting secrets as a fundamental design principle. The system uses strong encryption algorithms and processes to safeguard sensitive data while it is kept, guaranteeing that confidential information is securely protected and cannot be accessed by unauthorised
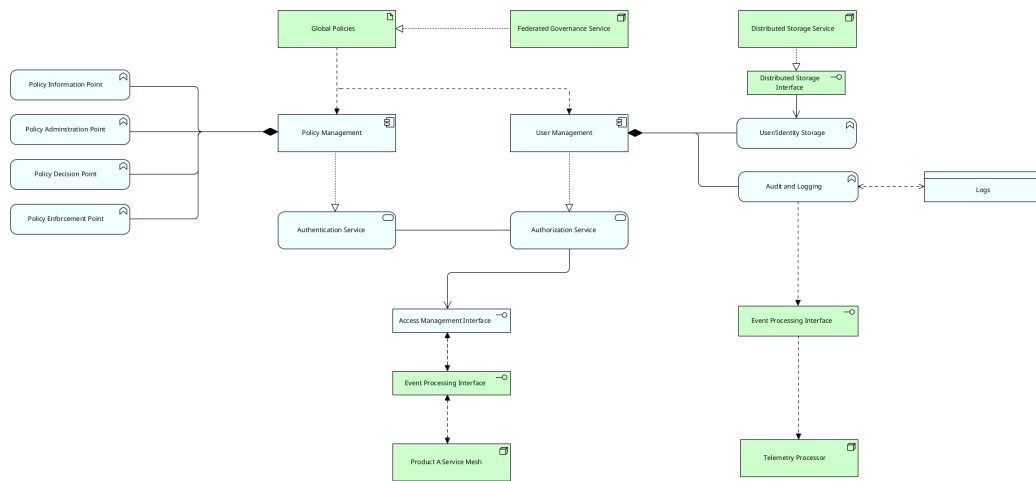
35

Figure 5: Identity and Access Control Management System

parties. This value aids in preventing unauthorised access to secrets, reducing the risk of data breaches, and guaranteeing confidentiality.

The secret management system also facilitates the secure distribution and retrieval of secrets to authorised components or services. It offers safe APIs or client libraries that allow secrets to be retrieved securely during runtime. This value ensures that confidential information is only available to authorised entities who need it, preventing the unauthorised disclosure of secrets.

The secret management system also encourages integration with other components and services in Terramycelium. The system offers APIs and integration points that enable smooth integration of secrets into different data domains. This integration value enables authorised components to securely and conveniently access secrets, ensuring operational efficiency and minimising obstacles in development and deployment procedures.

Addresses requirements: SaP-1, and SaP-2.

The variable components in Terramycelium can be altered, adapted, or excluded according to the architect's discretion and the specific characteristics of the situation. This RA aims to enhance data architects' decision-making process by introducing established patterns and best practices from several perspectives, rather than restricting their innovation. Alternative

36

choices for each variable module are not detailed due to the dynamic nature of the industry, and architects strive to create systems that tackle new issue areas.

9. Evaluation 1: Case Mechanism Experiment

The validation of treatment is crucial in its research, development, and implementation phases [92]. Validating a treatment involves determining its possible impact on stakeholder objectives when applied to the specific problem at hand.

Clearly outlining and justifying the treatment needs establishes a straightforward avenue for validation. This implies that the treatment may be verified by showing that it fulfils certain predetermined criteria. A key challenge in treatment validation is the absence of real-world implementation to assess the treatment's impact on stakeholder objectives. This challenge emphasises the need for thorough, diverse, and well-thought-out assessment methods, as discussed more in the next sections.

The initial phase of evaluating Terramycelium is a single-case mechanism experiment based on the parameters outlined by Wirienga [92]. Although Wirienga's works reference a single experiment, this study considers a few experiments. A multi-case experiment is required because the artefact is complex, which improves the thoroughness and significance of the evaluation process, leading to a more detailed and nuanced comprehension of the artefact's capabilities and possibilities.

9.1. Prototyping an Architecture

Prototyping is the creation of a working model that physically represents the structure, demonstrating the design principles and ideas. This is frequently referred to as the *concrete architecture*. This technique enables researchers to validate and develop the design to assess its practicality, performance, and alignment with research objectives.

Implementing the essential components and functionalities stated in the RA, Terramycelium, is part of prototyping the architecture. The selected technologies, supported by academic justification and in line with the research goals, serve as the basis for developing the prototype. The architecture's ability to manage large-scale data processing and meet various needs can be evaluated by systematically implementing its architectural components. Figure 6 shows a concrete structure created with Terramycelium.
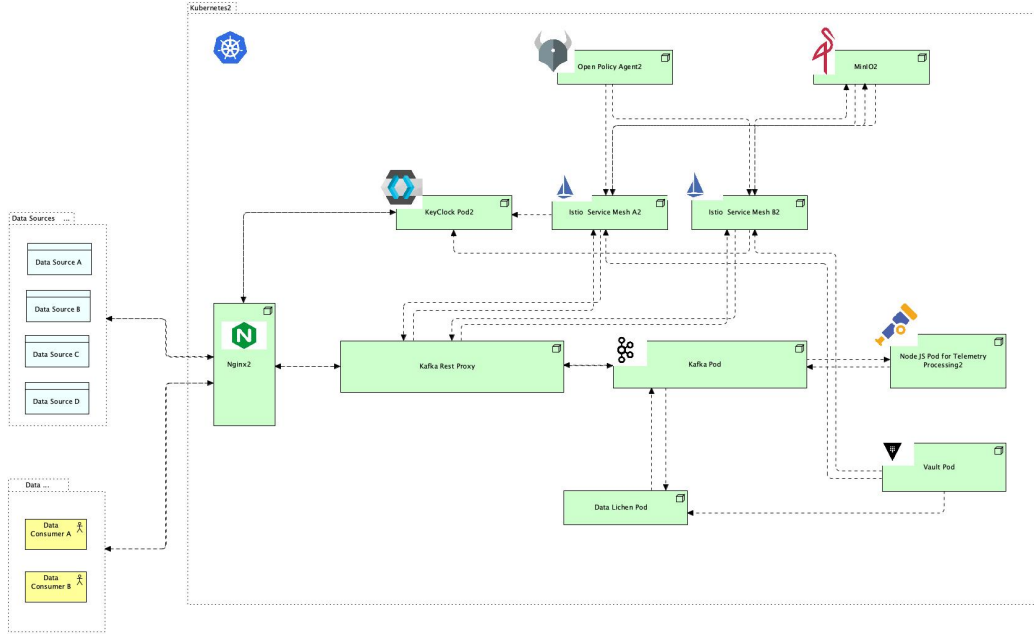
37

Figure 6: Terramycelium Prototype

9.2. Stage 1: Formation

All components of this prototype are integrated within a single Kubernetes cluster. Therefore, KIND [80] has been selected as the local Kubernetes cluster for this prototype to function.

The reproducibility of this technique is crucial. The scripts required to execute, download, and set up artefacts are located in the scripts folder of the Github repository, cited as [3]. These scripts are designed to exclusively operate on Unix-like operating systems.

After establishing the fundamental scripts for initialising the cluster, the procedure commenced with the creation of various services within the Kubernetes cluster. Helm Charts from Artefact Hub [88] are utilised for enhanced maintainability and scalability of Nginx ingress and Kafka. Terraform was subsequently used to implement the charts on the Kubernetes cluster. It is located in the IaaC folder of the repository.

We conducted an extensive search for helm charts for all artefact components and selected a mature and well respected chart from Bitnami for Nginx. Using this method is favoured over developing unique directives for Nginx ingress in Kubernetes and linking services to each other through Kubernetes

architectural elements such as Services or StatefulSets.

After this, a search has been done for other charts for other components of the concrete architecture on Artefact Hub. After selecting an appropriate chart for Nginx ingress, Keycloak, and Vault, the telemetry processor service is created.

Initially, a FastAPI app is set up. The programme is containerised using Docker, hosted on Github sites, and integrated into Terraform Helm release resources for deployment on the local KIND cluster. The Kubernetes ingress controller is configured as Nginx and deployed to the local cluster using Terraform.

Once the ingress controller is established, the ingress resources are configured to direct traffic to Kafka-rest-proxy. The Kafka-rest proxy is included in the artefact to simplify data access from the cluster. Kafka uses the binary protocol instead of HTTP, whereas most services depend on HTTP for network connections.

Upon successful configuration of the Nginx ingress and Kafka REST proxy, Keycloak is installed and seamlessly integrated into various components of the architecture. The Vault is incorporated into the cluster using HashiCorp's official helm chart [37].

Next, the major system to be implemented is Data Lichen. Data Lichen required a new frontend to be developed from scratch. A template engine for Node.js called EJS was chosen for this purpose. This template engine was selected for its ability to simplify development and speed up the operations of this trial.

The goal is to create a simple user interface for Data Lichen that successfully showcases the framework's potential design and capability, rather than building a comprehensive frontend. Various teams may opt to adopt Data Lichen in diverse ways. Data Lichen must be developed from scratch because there is no existing open-source technology that offers the necessary features.

Once Data Lichen has been organised and implemented in the cluster, an Istio service mesh is established. To achieve this goal, various Kubernetes namespaces are established, and various services are categorised within the same namespace for the service mesh to manage and function. Istio's Kiali dashboard is implemented to enhance the development experience and observability.

Minio's chart is installed, and the dashboard is active. The last component that needed to be deployed was Open Policy Agent (OPA). There

are multiple methods to incorporate OPA into an architecture. This can be accomplished via a solo Envoy proxy, a Kubernetes operator, a load balancer with Gloo Edge, or through Istio.

Istio was chosen because it has already been selected and deployed on the cluster. Nevertheless, automatically injecting policies into services inside a particular service mesh is a complex operation. This was done by extending the Envoy sidecar with an external authorizer API to implement inbound and outbound policies.

OPA is incorporated into the service mesh as an independent service known as a sidecar proxy. The OPA functions as an admission controller, evaluating requests sent to the Kubernetes API server.

OPA assesses these requests based on established policy guidelines. Approved requests are executed by the API server, whereas refused requests result in an error message sent to the user. OPA's policies manage all interactions among services and pods within a namespace. OPA assists in ensuring adherence to predetermined security and operational regulations within a Kubernetes setting. This is depicted in Figure 7.
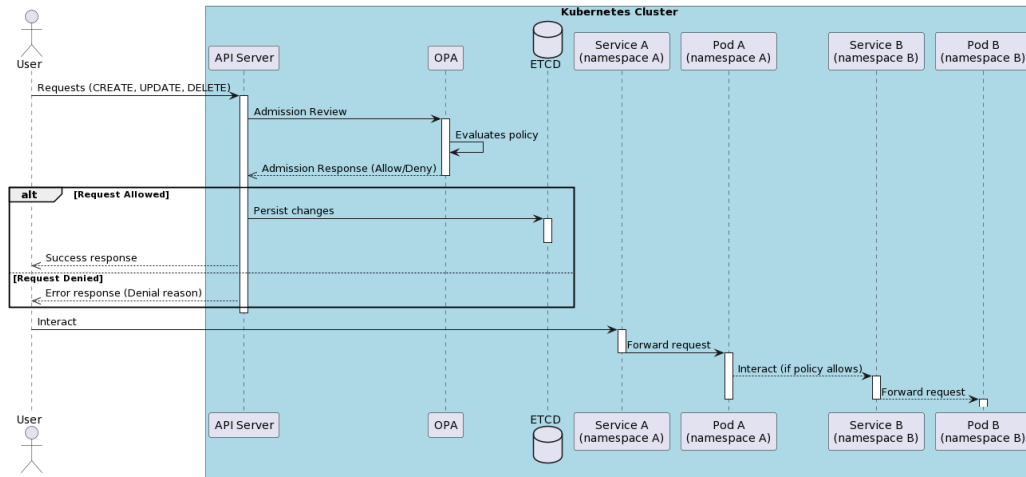

Figure 7: Open Policy Agent Communication Flow with Other Services

Once OPA was implemented, the analytics service for the domain was created using the Python framework FastAPI. To meet the requirements of Terramycelium, a domain incorporating operational and analytical systems must be established. Since this experiment is not a genuine business, it is logical to establish a simulated operational service that mimics real-world

40

activity, such as generating large amounts of data at a high speed.

₁₁₈₀ Two important data sources studied and selected for creating these domains are the Yelp Academic Dataset [95] and the data from the U.S. Climate Resilience Toolkit's Climate Explorer [24]. This decision is based on the extensive and detailed data provided by these sources, their accessibility, and their ability to generate valuable analytical insights when used together.

₁₁₈₅ The Yelp Academic Dataset is selected for its comprehensive compilation of restaurant reviews, containing structured and semi-structured data that encompasses customer feedback, business information, and temporal patterns. Another option, the MovieLens dataset [36], has been considered as well, but was found less suitable because it focuses only on movie ratings, ₁₁₉₀ limiting the scope of potential analysis.

The Climate Explorer dataset, which provides a detailed record of past and predicted climatic factors in the United States, is selected as an appropriate option for studying possible connections between weather trends and restaurant evaluations. Although options like Visual Crossing Weather ₁₁₉₅ Data Services [4] have been examined, Climate Explorer's extensive global coverage and historical data are preferred.

The datasets qualify as BD because of their amount, variety, and velocity. The Yelp Academic Dataset comprises millions of reviews for over a hundred thousand businesses, making it a data source with a large volume and variety. ₁₂₀₀ The Climate Explorer dataset, updated frequently, adds a speed component to the data structure. Collectively, they offer a chance for sophisticated analytical investigations in a data-rich setting.

9.3. Stage 2: Amalgamation

The initial phase of developing the prototype involves establishing the ₁₂₀₅ foundational services within the cluster. This step focuses on establishing connections and ensuring that the architecture can effectively react to external stimuli. Selected situations are applied to the system to achieve this.

Scenarios are commonly used to evaluate new architectural designs because they offer a robust way to simulate real-world situations, allowing for ₁₂₁₀ the assessment of the architecture's performance in different scenarios.

This experiment is crucial as it simulates various behaviours, system loads, data quantities, and processing kinds to analyse the architecture's resilience, adaptability, scalability, and interoperability.

Based on the templates in Carroll's work [20] and the examples in Kaz- ₁₂₁₅ man's ATAM [46], a set of scenarios is established to evaluate the prototype's

41

Table 3: Scenario S1: High Volume Data Ingestion Scenario

Scenario Description:	The system is subjected to a high volume of data, simulating peak data ingestion rates. The scenario is implemented when Domain A retrieves high volumes of data from Domain B, and processes it.
Relevant Quality Attributes:	Performance, Scalability.
Expected Outcome:	The system can ingest, store, and process large quantities of data without significant performance degradation.

Table 4: Scenario S2: High Velocity Data Ingestion Scenario

Scenario Description:	The system is subjected to a high velocity of data, simulating peak data ingestion rates. The scenario is implemented when Domain A streams data into domain B and domain B stream processes the data.
Relevant Quality Attributes:	Performance, Scalability.
Expected Outcome:	The system can ingest, store, and process a continuous stream of data without significant performance degradation.

abilities. The scenarios, which include various aspects of system performance and security, are outlined in Tables 3 through 8.

9.4. Stage 3: Scenario Testing

During this step, after defining scenarios, the system is initialised, and each scenario is executed. Each specific situation is evaluated to determine how the system handles different conditions. This method entails initialising the system and methodically testing each scenario against it.

By doing this, important measurements are recorded, leading to a more thorough comprehension of how the system reacts to different stimuli. This method assesses the system's ability to withstand challenges and its efficiency,

Table 5: Scenario S3: Data Variety Scenario

Scenario Description:	The system is exposed to a diverse range of data types and formats. This scenario is implemented when domain B retrieves files in a different format from domain A and processes it.
Relevant Quality Attributes:	Variety, Interoperability.
Expected Outcome:	The system can handle and process different data types and formats efficiently.

Table 6: Scenario S4: Complex Query Scenario

Scenario Description:	The system processes complex queries that involve multiple large datasets. This scenario happens when an external data scientist queries both domains and tries to understand a relationship between the datasets.
Relevant Quality Attributes:	Computational Efficiency, Performance.
Expected Outcome:	The system can efficiently handle and process complex queries.

Table 7: Scenario S5: Secret Management Scenario

Scenario Description:	This scenario underscores the system's ability to manage secrets securely and efficiently, focusing on storage, retrieval, and rotation using Hashicorp Vault in conjunction with OpenID Connect's standard flow with bearer tokens.
Relevant Quality Attributes:	Confidentiality, Integrity, Availability.
Expected Outcome:	The system securely manages secrets, ensuring timely storage, retrieval, and rotation while maintaining confidentiality and integrity.

Table 8: Scenario S6: Data Security Scenario

Scenario Description:	The system's capability is evaluated regarding ensuring data security throughout access, processing, and transmission. OpenID Connect with bearer tokens is leveraged for authentication.
Relevant Quality Attributes:	Confidentiality, Data Integrity, Authentication.
Expected Outcome:	The system guarantees data security, with secure data access, processing, and transmission. Unauthorised access attempts are effectively detected and mitigated.

providing valuable information about its functioning and opportunities for improvement.

9.4.1. Scenario S1: High Volume Data Ingestion Scenario

Before establishing the inter-domain link, it is important to review the intra-domain flow. Each operational service in the domain handles data by initially receiving an event from the analytical service and then requesting the data. The data is kept in Minio after being processed. A message is sent to a particular topic to indicate that the data storage process has been completed.

The message is transmitted to a Kafka-rest proxy and then to the Kafka cluster. This event contains the object's address in MinIO and other pertinent metadata. The analytics service establishes a consumer group and subscribes to that particular topic. The analytical service retrieves any new event transmitted to that topic on a background thread.

Upon receiving the event, the analytical service uses the metadata within the event's body to extract the object's address. Once the address is known, the analytical service directly accesses MinIO to retrieve the data. The service processes the data and saves it in an embedded SQlite database.

At this stage, the required data quality metadata and bitemporality metadata are generated. The metadata, service name, service address, and unique identifier are sent as an event to Kafka on a specified topic. Data Lichen establishes a consumer group, subscribes to that particular topic, retrieves the data, and subsequently presents the information for each domain. Data Lichen presents this information in a specially designed table, as shown in Figure 8. Utilising a distinct identification for each service guarantees that the listing and presentation of services are uniform and dependable, yielding consistent outcomes with every action. The flow is depicted in Figure 9.

Metadata Information

Unique Identifier	Service Name	Service Address	Completeness	Validity	Accuracy	Processing Time	Actual Time	Processing Duration
6c055c9a-7cdf-4c47-bc6c-aefb135ec9d6	Weather domain data	http://localhost:8500	87.9049676025918	87.9049676025918	87.9049676025918	2023-08-10 20:42:47	2023-08-10 20:42:47	0.08 seconds
d02aa17d-7cac-4695-bdab-3dc7f3194b8a	Customer domain data	http://localhost:8000	100	100	100	2023-08-13 15:32:03	2023-08-13 15:32:02	0.72 seconds

Figure 8: Data Lichen Dashboard in Local Development Environment
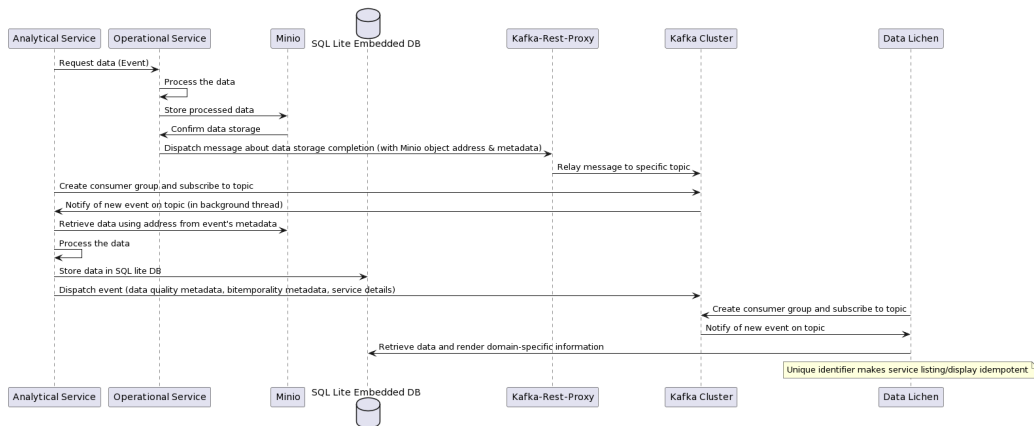
45

Figure 9: Intra-domain communication in Terramycelium's prototype

Domain A requests a 5GB JSON file from domain B in this instance. Domain A possesses the meteorological data, but Domain B possesses the customer ratings and business data. An event is sent from the domain A to Data Lichen to obtain a list of all datasets along with metadata such as actual time, processing time, completeness, and correctness.

The information contains the address of the data domain to be retrieved. The address is utilised to send an event to domain B for data retrieval. This activates the internal process of domain B, leading to the production and storage of datasets in the distributed storage service. After the procedure is finished, an event is sent to the data processing completion topic, to which domain A is subscribed. Figure 10 provides an overview of the Kafka topics utilised in this investigation.

Figure 10: Overview of kafka topics

The event signalling the conclusion of data processing also includes the

data address. Domain A utilises this URL to retrieve the data. After retrieving the data, it is subsequently processed. The flow is illustrated in Figure 11.
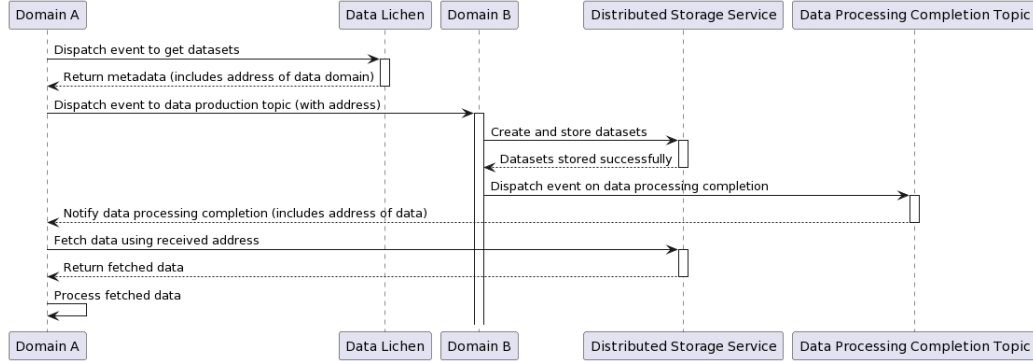

Figure 11: Scenario S-1 Flow Diagram

Understanding system performance and resource utilisation is crucial in this scenario. Data ingestion rate is a direct indicator of the system's throughput, showing how effectively the system can handle large volumes of incoming data. A high rate indicates efficient data processing capability, which is crucial for situations with an anticipated fast data influx. Latency, divided into intake and processing components, provides an understanding of system responsiveness. Minimal latency is essential for ensuring prompt data availability and processing, especially in systems where delays can cause subsequent effects.

CPU usage and memory utilisation are key metrics for assessing the system's resource efficiency. Increased CPU or memory usage could signal bottlenecks, inefficiencies, or places that could be optimised, particularly during times of high demand. The error rate provides information on the system's resilience and dependability. A low error rate, even with large amounts of data, indicates the system's ability to continuously manage and process data without any issues.

These metrics collectively evaluate the system's capacity to sustain performance and stability when faced with high data input requirements. Each measure will be described in connection to the corresponding system mechanism in the following sections.

47

9.4.1.1. Data Ingestion Rate

Each service is instrumented using OpenTelemetry SDKs to capture telemetry data for FastAPI requests and manual spans for specified processes. The collected data is then exported to Kafka using the KafkaRESTProxyExporter class to measure the data intake rate.

Figure 12 illustrates the open telemetry Kafka topic and a large number of events recorded in this topic. The open telemetry processor service consumes all events sent to this topic. The service gathers and maintains metrics, which are then extracted by Prometheus and subsequently sent to Grafana. The flow is depicted in Figure 13. The JSON displayed in Figure 14 depicts an example of a trace span JSON that is communicated over Kafka to the open telemetry service.
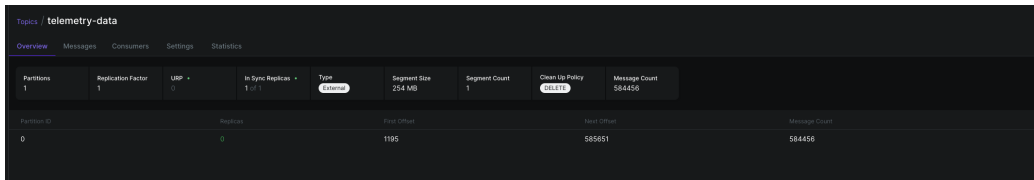

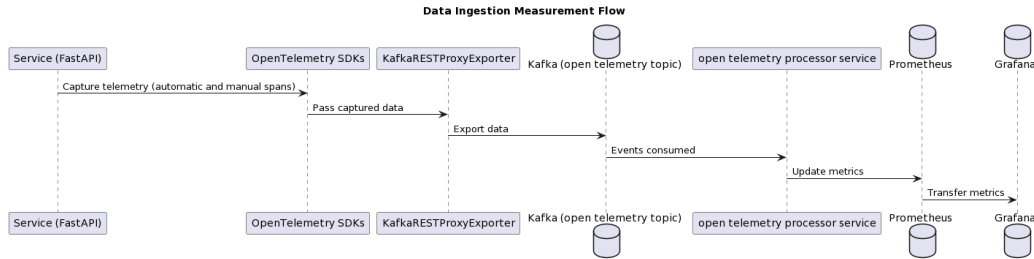Figure 12: Open Telemetry Topic

Figure 13: Open Telemetry Flow

Before presenting the experiment results, it is important to define the specific meaning of heavy load of data' in the context of this study. In BD, there is no commonly accepted criterion for defining high volume', particularly in decentralised, domain-driven architectures such as Terramycelium. There is no single standard benchmark that universally defines what qualifies as a heavy data load in these systems.
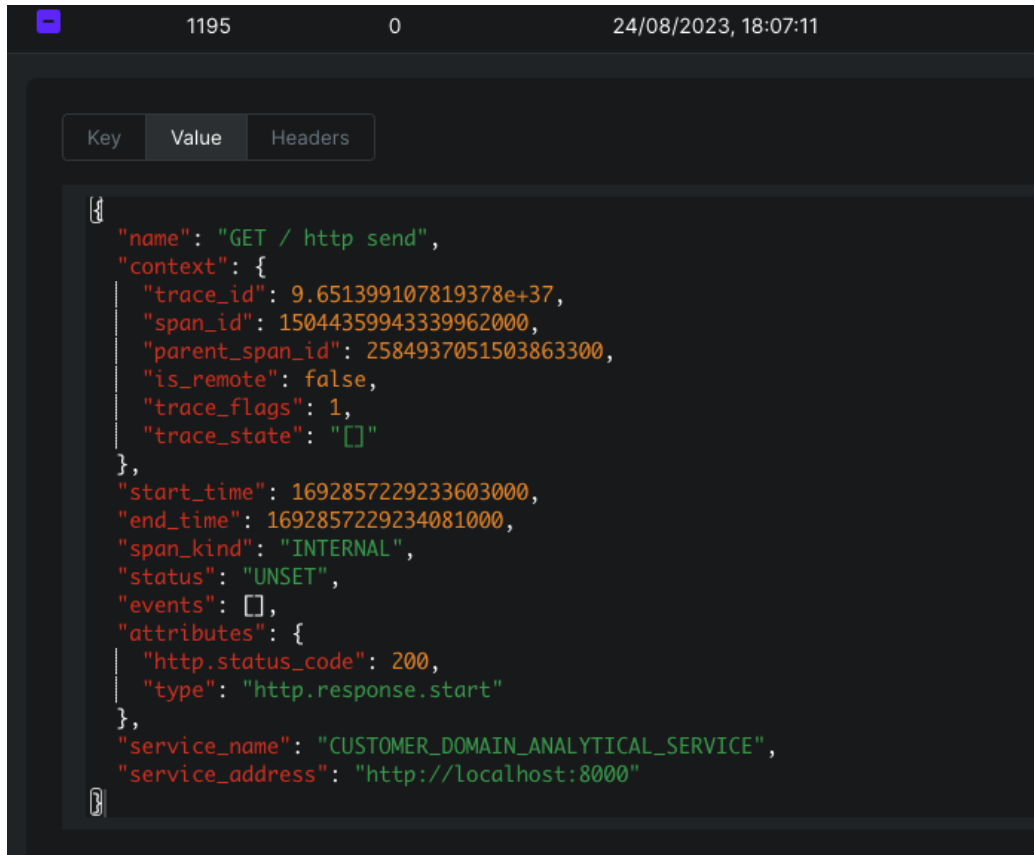
```json
{
  "name": "GET / http send",
  "context": {
    "trace_id": 9.651399107819378e+37,
    "span_id": 15044359943339962000,
    "parent_span_id": 2584937051503863300,
    "is_remote": false,
    "trace_flags": 1,
    "trace_state": "[]"
  },
  "start_time": 1692857229233603000,
  "end_time": 1692857229234081000,
  "span_kind": "INTERNAL",
  "status": "UNSET",
  "events": [],
  "attributes": {
    "http.status_code": 200,
    "type": "http.response.start"
  },
  "service_name": "CUSTOMER_DOMAIN_ANALYTICAL_SERVICE",
  "service_address": "http://localhost:8000"
}
```

Figure 14: Open Telemetry Trace Span

Academic research and business reports suggest that daily data intake can reach terabytes or more, particularly in high-velocity streams such as social media or e-commerce transactions [71].

Transferring many gigabytes of data in JSON format between domains is a substantial stress test due to its verbosity and demands compared to binary formats. The Yelp academic dataset is divided into 400KB chunks and transferred many times to test the prototype's capability and resilience against increased data traffic.

This configuration is suitable for practical scenarios where systems need to consistently handle continuous streams of incoming data from different sources and varying quantities, while maintaining high performance and data integrity. Terramycelium's decentralised, event-driven architecture is

49

designed to allow data to be smoothly ingested, processed, and retrieved,
regardless of the frequency of dataset transfers or accesses. This assesses the
system's capacity to manage large amounts of data and its ability to perform
well in a setting that replicates the rapid data flow typical of modern BD
scenarios.

All data sent during the experiment is stored in MinIO. Upon success-
ful storage, a confirmation is always sent back. The process of streaming,
storing, and validating data was repeated several times to demonstrate its
reliability, especially when dealing with large amounts of data. Reliable and
accurate communication in challenging situations supports the importance
of consistent communication in remote systems, as shown in the works of
Kleppmann [52]. The process can be formally represented using logical im-
plications to determine the system's resilience.

Let's denote the following:

- P as the event where "data is transferred from Domain A to Domain B".

- Q as the event where "data is successfully stored in MinIO".

- R as the event where "confirmation of successful storage is communicated back".

From the empirical observations, it is deduced that:

$$P \quad \text{implies} \quad Q$$

$$Q \quad \text{implies} \quad R$$

Thus, utilising the transitive property of implications [42], it is inferred
that:

$$P \quad \text{implies} \quad R$$

Every data transmission from Domain A to Domain B ensures the certi-
fication of successful storage. The link highlights the system's effectiveness
and dependability in handling extensive amounts of data, aligning with the
conclusions of Lee [56] who stressed the need to maintain data consistency
and integrity in distributed storage systems.

The graph in Figure 15 shows that the Kafka data ingestion reached its highest point at 1,693,816,791 bytes on 2023-09-04 at 20:40:00 in the analytical service of the customer domain, demonstrating the transfer of data from the operational service to the analytical service. The bar chart illustrates the incremental rise in the service's workload. The entire quantity of data ingested during the trial was 1.6 billion bytes, as shown in Figure 16.

Figure 15: Kafka Data Ingestion in Bytes in Customer Domain's Analytical Service

Figure 16: Total Data Ingested in Bytes in Customer Domains' Analytical Service

The data ingestion rate for the weather domain's analytical service peaked at 1,693,893,416 bytes on September 5, 2023, at 18:00:00, as shown in Figure 17. This service has processed a total of 1.6 billion bytes.

The system's performance during large data input is empirically evaluated by visualising metrics in Grafana using the screenshots. The monitoring system offers timestamped data points showing the amount of data consumed

Figure 17: Total Data Ingested in Bytes in Weather Domain's Analytical Service

at different intervals. The Table 9 records the data points collected within a particular time frame in the analytical service of the customer domain.

In addition, the data acquired in the analytical service of the weather domain is presented in the Table 10. After capturing the ingestion rate in bytes, it is essential to further explore other metrics to provide a comprehensive analysis of the system's performance during high data load conditions.

9.4.1.2. Ingestion Latency

Ingestion latency measures the time between when a data record is published to Kafka and when it is processed by the domains. It is calculated for each Kafka record that is ingested using a specific formula.

$$Latency = current\_time - publish\_time$$

Where:

- *current_time* denotes the time at which the record is being processed.

- *publish_time* represents the timestamp when the record was published to Kafka. If the Kafka record lacks a timestamp, the current time is assigned as a default, making the latency calculation a fallback measure.

The latency numbers captured are presented in Figure 18. The figure shows that the latency in both domains' analytical services is negligible, with one spike reaching 400 seconds and the rest maintaining a constant ingestion latency of roughly 50 to 100 seconds. The weather domain experiences lower

52

Table 9: Data Ingested in Customer Domain's Analytical Service

Timestamp	Data Ingested (bytes)
2023-09-04 20:40:00	1693816791
2023-09-04 20:45:00	1693816791
2023-09-04 20:50:00	1693816791
2023-09-04 20:55:00	1693816791
2023-09-04 21:00:00	1693816791
2023-09-04 21:05:00	1693816791
2023-09-04 21:10:00	1693816791
2023-09-04 21:15:00	1693816791
2023-09-04 21:20:00	1693816791
2023-09-04 21:25:00	1693816791
2023-09-04 21:30:00	1693816791
2023-09-04 21:35:00	1693816791

latency than customer latency, averaging 50 seconds. This is because this domain uses a less amount of data.

The findings are significant given the intricacies and asynchronous nature of the domain-driven distributed BD system utilising Kafka. The occasional increase to 400 seconds in ingestion latency is noteworthy but falls within acceptable limits, particularly given Kafka's internal features like indexing methods, offset management, and sticky partition behaviour.

9.4.1.3. CPU Usage

The telemetry processing service is crucial to the infrastructure and offers valuable information when monitoring CPU utilisation. It is crucial to monitor this service since it provides immediate insights into data processing efficiency. Therefore, increased CPU utilisation in this service implies potential processing bottlenecks.

Furthermore, CPU utilisation metrics are calculated at certain intervals.

Table 10: Data Ingested in Weather Domain's Analytical Service

Timestamp	Data Ingested (bytes)
2023-09-05 18:00:00	1693893416
2023-09-05 18:15:00	1693893416
2023-09-05 18:30:00	1693895179
2023-09-07 20:45:00	1694076236
2023-09-07 21:00:00	1694076236
2023-09-07 21:30:00	1694076236
2023-09-08 11:45:00	1694130131
2023-09-08 12:00:00	1694130131
2023-09-08 12:15:00	1694130131
2023-09-08 12:30:00	1694130131
2023-09-08 12:45:00	1694130131
2023-09-08 13:00:00	1694130131

Due to the asynchronous nature of receiving telemetry data from different services, monitoring the CPU consumption of individual services may not provide an accurate representation of the current system load. The telemetry service consistently processes data to ensure that its measurements accurately reflect the system's load.

It is important to consider this finding while assessing scalability and forecasting future infrastructure needs. CPU consumption measurements can be affected by the operating system and environment in which the services are running. The given statistics may not be exact, but they provide a reliable comparison of system resource usage.

Figure 19 illustrates the CPU utilisation trends of the telemetry processing service, emphasising the interplay between data intake rates and resource consumption.

The CPU utilisation in the Open Telemetry service metrics shows a continuous level of around 24 percent, with occasional peaks reaching 25.8 per-
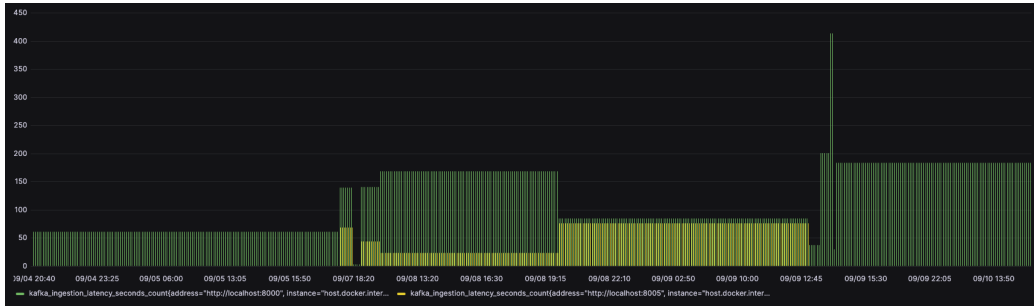
Figure 18: Customer and Weather Domain's Analytical Service Latencies
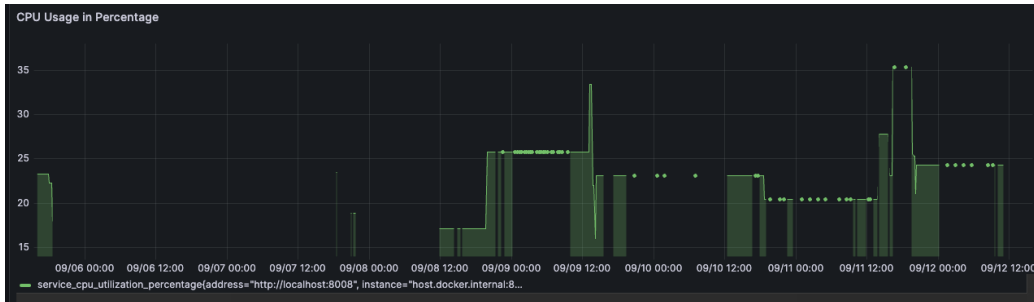

Figure 19: Open Telemetry Service CPU Usage

cent at certain times. The behaviour is a result of the architecture in which events are consistently transmitted to the associated services. A full table showing the CPU utilisation during a certain period can be found in Table 11 to clarify this issue. The tabulation offers insights into the variation in CPU usage at different times and under varied conditions, showing periods of stable CPU usage, periods of inactivity, and occasional spikes in utilisation.

The constant and regular CPU usage indicates a reliable processing environment. Stability can indicate optimised data streaming, where the telemetry service efficiently manages incoming events without excessive resource usage or sudden computational spikes.

The Kubernetes cluster CPU use consistently stays at 43%, with occasional spikes up to 250% as seen in Figure 20. These spikes typically occur when a large amount of data is being sent across the network and into Minio.

Similarly, as shown in Figure 21, an average of 5.37GB of data was read and 12.2GB of data was written at various intervals. Furthermore, the network has received 11.6 gigabytes of data and delivered 13.9 megabytes (Fig-

55

Date and Time	Average CPU Usage (%)
09/06 00:00	23%, 24%
09/06 12:00	0 (no processes)
09/07 00:00	0 (no processes)
09/07 12:00	0 (no processes)
09/08 00:00	24%, 18%
09/08 12:00	17%
09/09 00:00	26%
09/09 12:00	26% (with a spike to 34%)
09/10 00:00	24%
09/10 12:00	24%
09/11 00:00	22%
09/11 12:00	22% (with spikes to 27% and 35.2%)
09/12 00:00	24%
09/12 12:00	24%

Table 11: Telemetry Processing Service CPU Utilisation

ure 22).

After thoroughly analysing the telemetry and infrastructure data, it is
clear that the asynchronous and event-driven architecture of this BD archi-
tecture positively impacts its performance dynamics. In this asynchronous
event-driven paradigm, it is important to acknowledge that the amount of
data handled, measured in gigabytes, is less important than the system's
efficiency in handling incoming data streams.

The telemetry service demonstrates its resilience in data processing by
maintaining a continuously moderate CPU consumption, even when faced
with fluctuating data input rates. Similarly, the Kubernetes cluster occa-
sionally experiences surges in CPU utilisation during data-intensive tasks,
but overall it maintains a well-distributed workload. This is evidence of the
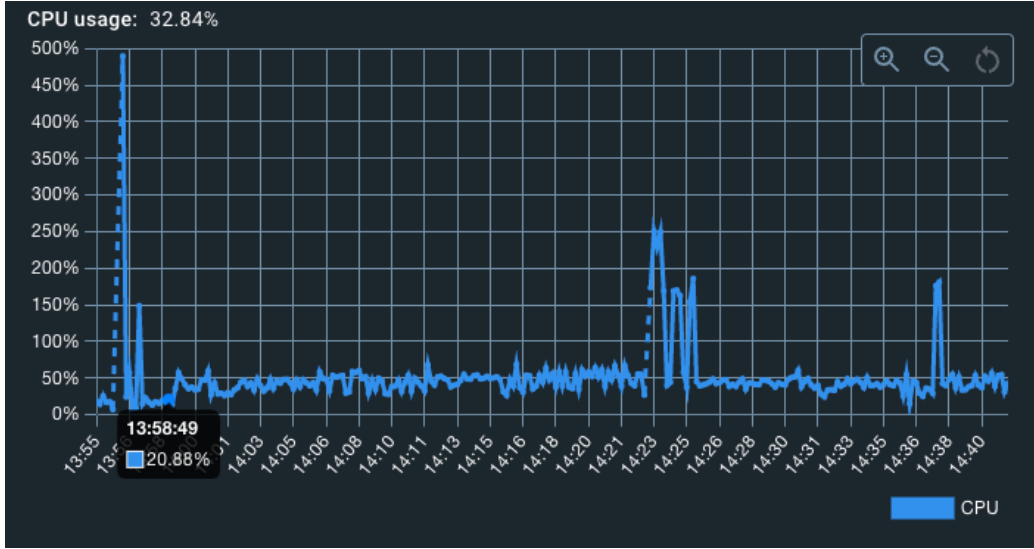
56

Figure 20: Kubernetes Cluster CPU Usage

system's proficiency in handling and allocating work among its nodes.

The data statistics about read, write, and network transfers provide additional clarification on this matter. Although the amount of data being read, written, or transported in gigabytes may appear substantial, what truly matters is not the sheer number but rather the system's efficiency in managing these processes. The average data read is 5.37 GB, and the average data written is 12.2GB. These figures, when compared to the network statistics shown in Figure 22, demonstrate the system's effectiveness in managing data and performing network operations without becoming overloaded.

The metrics not only demonstrate the stability and efficiency of the BD infrastructure but also offer empirical evidence that, in an event-driven asynchronous system like Terramycelium, the focus is not solely on the amount of data processed but rather on the effectiveness and reliability with which it is managed. This observation indicates possible consequences for scalability considerations in future implementations of similar architectures.

9.4.1.4. Memory Utilisation

Memory utilisation is a crucial measure when assessing system efficiency. An efficient memory management system demonstrates a system's capability to process real-time data and adapt to varying workloads. The memory usage
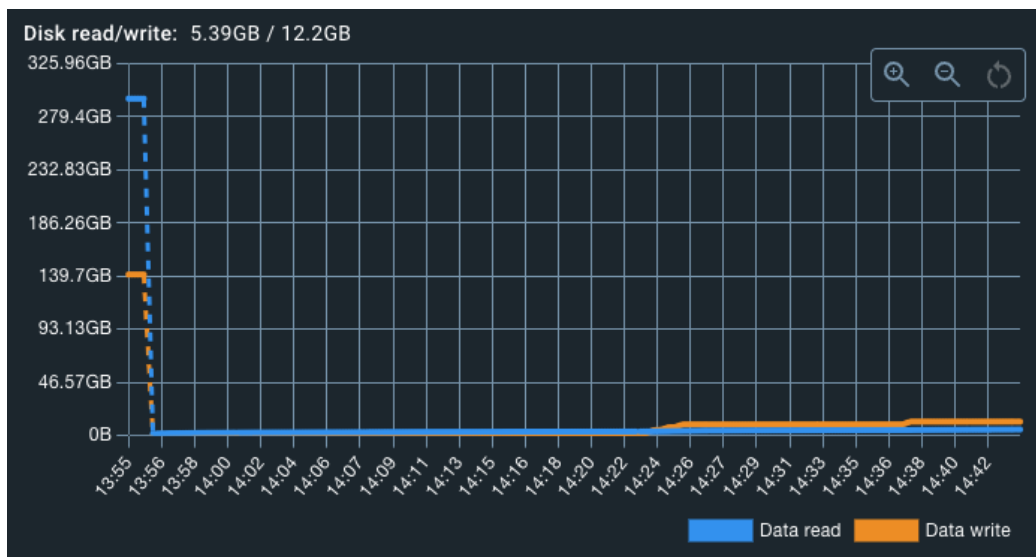
57

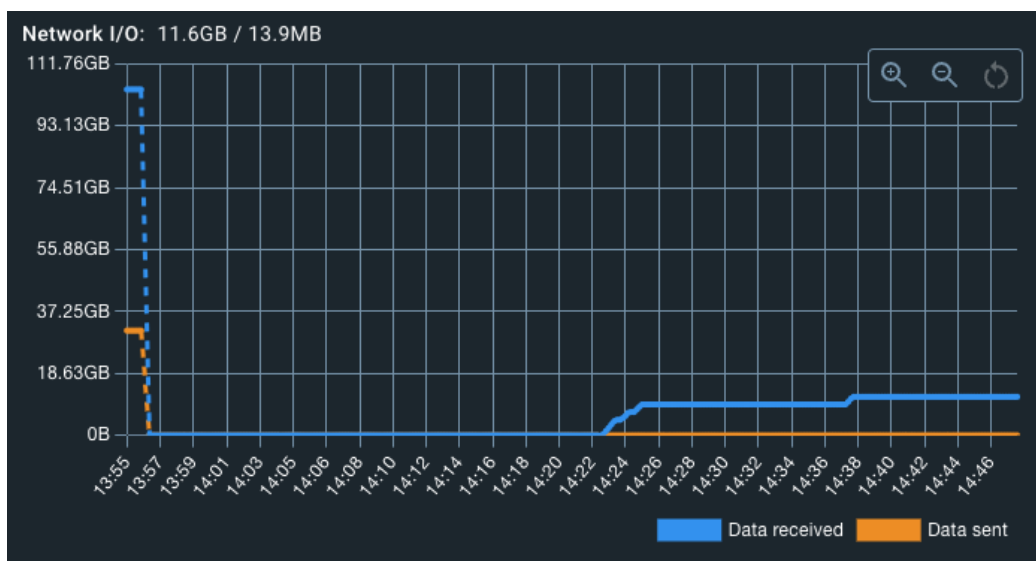Figure 21: Kubernetes Cluster Read and Write Statistics
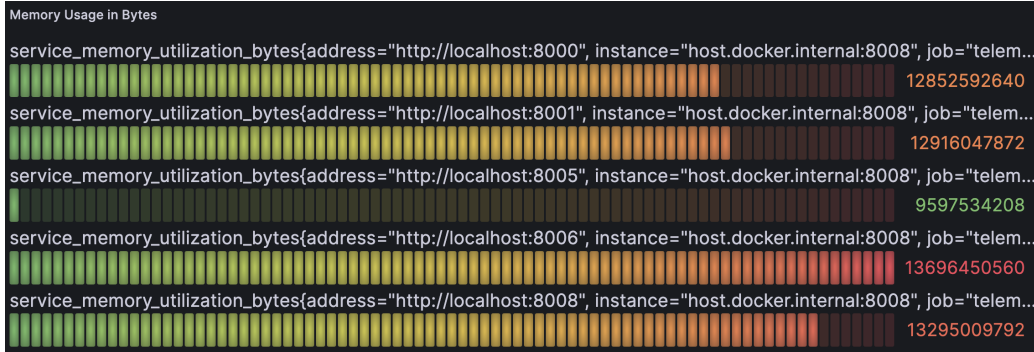

Figure 22: Kubernetes Cluster Network Statistics

Figure 23: Memory Usage for Various Services in the Cluster

seen in the telemetry processing service, a vital element of the infrastructure, offers precise insights into the system's ability to handle data. Examining these memory patterns provides a more distinct comprehension of the service's effectiveness in handling different data rates and identifies places where memory optimisation may be required to improve overall performance.

In the weather domain, memory utilisation can be categorised into two primary types: operational services and analytical services. Operational services that serve static data have low memory usage because they require less processing power. On the other hand, analytical services can exhibit diverse memory patterns based on the intricacy of data analysis.

The customer domain has the same pattern, wherein operational services have consistent memory use as a result of predominantly handling static client data. On the other side, the analytical services, which thoroughly examine client habits and preferences, may experience occasional surges, particularly when dealing with large datasets.

The memory utilisation trends across the various services are visualised in Figure 23. The address for each service is specified in the respective service definitions in the Github repositories [3, 2]. It is important to note that the service running on port 8000 is the operational service in the customer domain, while the service on port 8001 is the analytical service for the customer domain. The service running on port 8005 is the operational service for the weather domain, whereas the service on port 8006 is the analytical service for the same domain. The service running on port 8008 is the application responsible for processing telemetry data.

The association between CPU and memory usage is further explained in

59

Table 12. This table illustrates the allocation of memory across several services at specific instances of CPU use. It is evident that periods of increased CPU activity align with higher memory allocation, especially in the telemetry processing service. This alignment clarifies the system's ability to scale and respond effectively during moments of high computing demand.

Similarly, as the CPU utilisation decreases, there is also a decrease in the amount of memory being used, which highlights the effective management of resources during periods of less demanding processing. The ability to efficiently use memory in a way that can adapt to different needs is crucial for sustaining the performance of a system and ensuring that real-time data is managed smoothly across the entire infrastructure.

Date & Time	Svc A (B)	Svc B (B)	Svc C (B)	Svc D (B)	Svc E (B)
09/06 00:00	H (12,750,000,000)	H (12,800,000,000)	L (4,700,000,000)	H (14,000,000,000)	H (13,500,000,000)
09/06 12:00	L (6,350,000,000)	L (6,400,000,000)	L (4,650,000,000)	M (10,850,000,000)	M (10,650,000,000)
09/07 00:00	L (6,375,000,000)	L (6,425,000,000)	L (4,675,000,000)	M (10,870,000,000)	M (10,670,000,000)
09/07 12:00	L (6,360,000,000)	L (6,410,000,000)	L (4,680,000,000)	M (10,860,000,000)	M (10,660,000,000)
09/08 00:00	M (9,560,000,000)	M (9,610,000,000)	L (4,720,000,000)	H (13,800,000,000)	H (13,300,000,000)
09/08 12:00	M (9,580,000,000)	M (9,630,000,000)	L (4,750,000,000)	H (13,850,000,000)	H (13,350,000,000)
09/09 00:00	H (12,780,000,000)	H (12,830,000,000)	L (4,770,000,000)	H (14,100,000,000)	H (13,600,000,000)
09/09 12:00	H (12,760,000,000)	H (12,810,000,000)	L (4,760,000,000)	H (14,050,000,000)	H (13,550,000,000)
09/10 00:00	M (9,570,000,000)	M (9,620,000,000)	L (4,730,000,000)	M (10,900,000,000)	M (10,700,000,000)
09/10 12:00	M (9,590,000,000)	M (9,640,000,000)	L (4,740,000,000)	M (10,920,000,000)	M (10,680,000,000)
09/11 00:00	M (9,565,000,000)	M (9,615,000,000)	L (4,710,000,000)	M (10,880,000,000)	M (10,670,000,000)
09/11 12:00	H (12,770,000,000)	H (12,820,000,000)	L (4,775,000,000)	H (14,110,000,000)	H (13,610,000,000)
09/12 00:00	M (9,585,000,000)	M (9,635,000,000)	L (4,745,000,000)	M (10,930,000,000)	M (10,690,000,000)
09/12 12:00	M (9,600,000,000)	M (9,650,000,000)	L (4,755,000,000)	M (10,950,000,000)	M (10,700,000,000)

Table 12: Correlated Memory and CPU Usage for Various Services. Legend: B - Bytes, H - High Usage, M - Medium Usage, L - Low Usage. Service A corresponds to localhost:8000, B to localhost:8001, C to localhost:8005, D to localhost:8006, and E to localhost:8008.

9.4.1.5. Error Rate

Error rates within the FAST API services are determined through analysis of logs and monitoring of exception blocks. Every domain is assigned a specific Kafka topic exclusively for the purpose of reporting errors. However, as depicted in Figure 24, only the OpenTelemetry service encountered issues. However, the weather and customer domains did not have any errors.
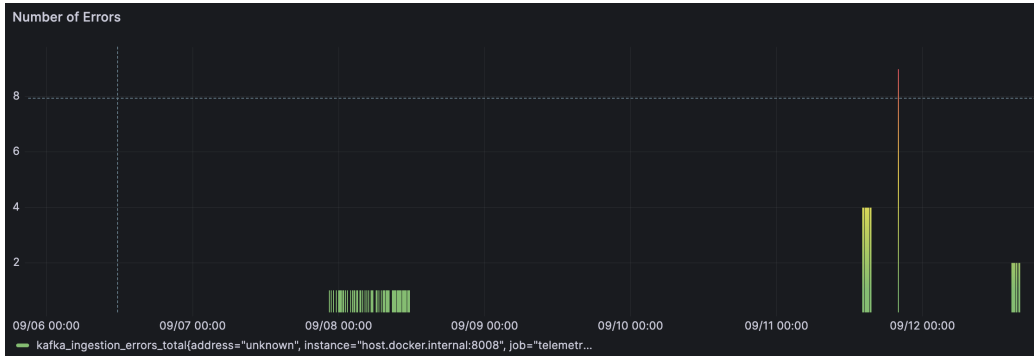
Figure 24: Number of Errors in Telemetry Processing Service

The intrinsic ability of event-driven systems to recover quickly and effectively adds greatly to the low occurrence of errors. The decoupling between components in these systems guarantees that transitory errors in one service are typically contained, preventing them from spreading and causing interruptions throughout the entire system. This tendency is clearly seen in the prototype, highlighting the inherent advantages of event-driven architectures in ensuring the resilience of the system. The term "peak" refers to the highest point or maximum value of something. It represents the culmination, or apex, of a certain phenomenon or trend.

With the above metrics in conjunction with the ingestion rate, a comprehensive understanding of the system's operational efficiency, resilience, and overall performance can be obtained.

9.4.2. Scenario S2: High Velocity Data Ingestion Scenario

Scenario S2 tests the system's capabilities under high data velocity conditions, approximating peak ingestion rates. This experiment involved Domain A (Customer Domain) streaming data into Domain B (Weather Domain), with Domain B subsequently processing the streamed data. To facilitate this, a dedicated Kafka topic, named customer-domain-stream-data, is established, earmarking it exclusively for the streaming operations between the two domains.

Concurrently, within the customer domain, a novel endpoint is created. This endpoint is responsible for streaming out the data that had been previously stored. In synchrony with this, the weather domain analytical service adopts a long-polling mechanism. By doing so, it could efficiently subscribe to and intake the continuous data stream relayed from the customer domain.

61

A methodical analysis is conducted based on the metrics of interest:

9.4.2.1. Volume and Size of Messages

The system processed a total of 771,305 messages. Collectively, these messages have a combined volume of around 1 GB, and they are all sent to the customer-domain-stream-data topic. The statistical data related to the streaming subject is illustrated in Figure 25.

9.4.2.2. Memory and CPU Utilisation

Memory usage is crucial in this high-speed data scenario. Regularly monitoring memory consumption was crucial in assessing the overall health and efficiency of the system. The memory usage in the customer domain reached a significant peak, measuring 12,852,592,640 bytes. The complexity of the chunking logic used in the streaming operation is the reason for this (Figure 26). In contrast, the weather domain, which functions as the recipient of data, indicated a more conservative use of memory. This is believed to be a result of its more straightforward data processing logic, which does not include the additional steps of chunking (Figure 27).

Furthermore, the CPU consumption has only grown by 5% compared to the statistics mentioned in the section titled CPU Usage for both the Kubernetes cluster and its associated services. Therefore, no further data is provided for this metric.

9.4.2.3. Duration of Processing and Latency of Ingestion

The latency of ingestion, defined as the time delay between receiving a data packet and processing it, remained constant during the observed timeframe. The delay continuously registers a value of 0.0000148 seconds at various time intervals throughout the monitoring period. This observation suggests that the system's data ingestion mechanism does not experience substantial delays in processing. However, a more comprehensive evaluation can be obtained by comparing its performance benchmarks with those of other platforms (Figure 28). The system's ability to maintain consistent latency indicates its capacity to sustain performance levels without interruption, especially during periods of heavy data flow.

Furthermore, the processing duration provides us with information on the amount of time it takes for the system to process the data that has been received. The Weather Domain consistently processes data in 1,694,491,558
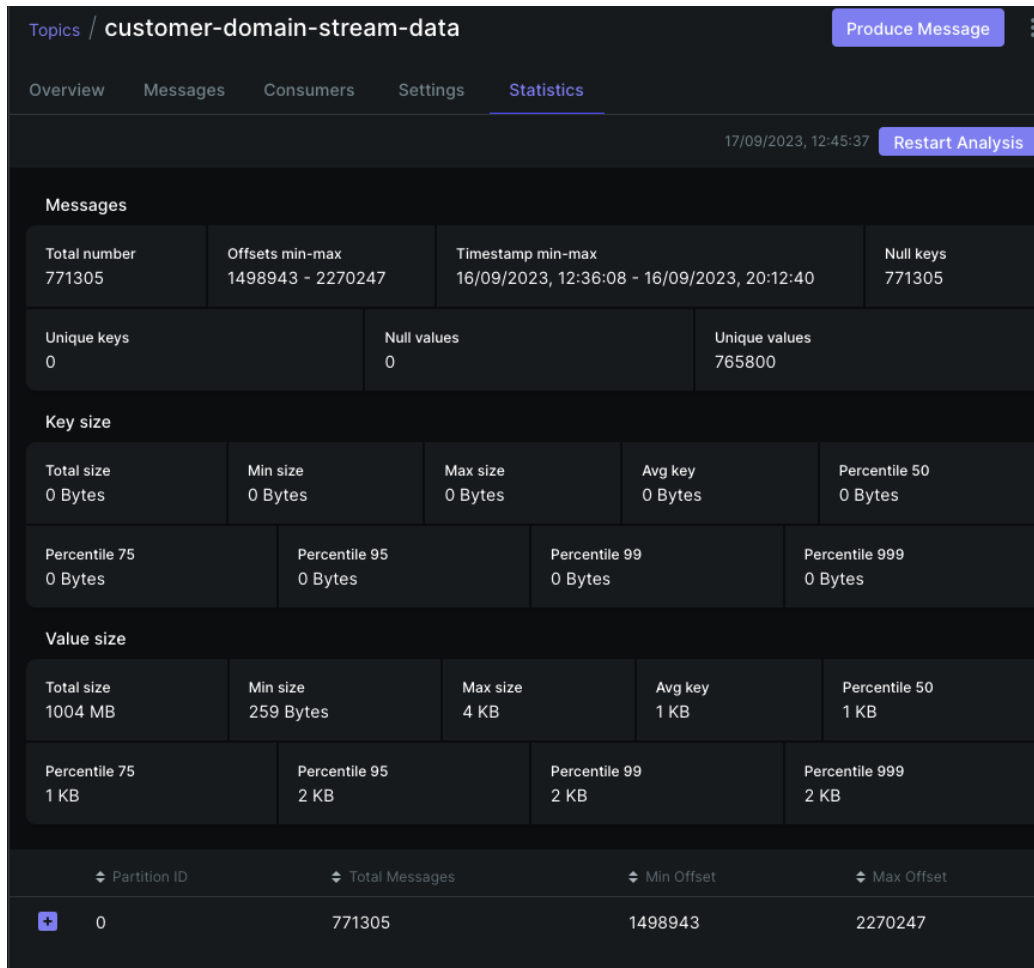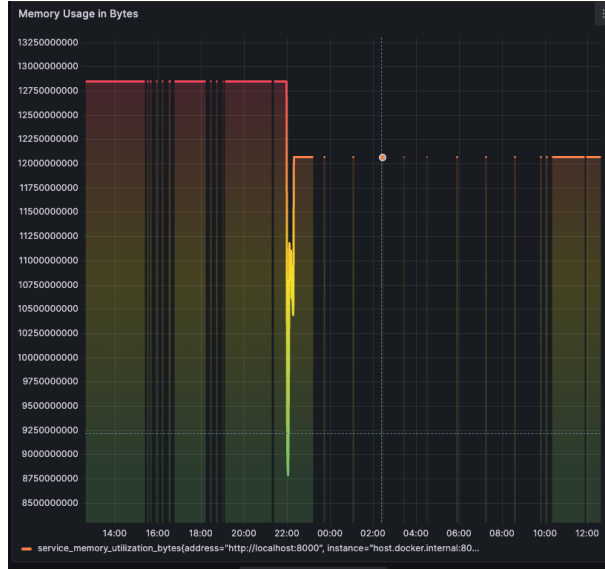
Figure 25: Customer Domain Streaming Topic Statistics

Figure 26: Memory Utilisation in Customer Domain in High Velocity Case

nanoseconds (or approximately 1.6945 seconds) for various timestamps (Figure 29). The consistency seen suggests a reliable data processing method that remains unaffected by any potential variables encountered during the streaming process.

The assessment of the High Velocity Data Intake Scenario (S2) demonstrated that the system consistently handled the rapid intake of data within the specified parameters, indicating its effective capability to manage such scenarios. By establishing a specialised Kafka topic and strategically integrating a new endpoint in the customer domain to enable data streaming, we are able to accurately replicate peak data input rates. The system's memory use patterns demonstrate its ability to handle large amounts of data. More precisely, the consumer domain, influenced by its chunking logic, exhibits more memory consumption in comparison to the weather domain. However, both domains efficiently managed the processing and intake of a significant total of 771,305 messages (about 1GB in size).

Moreover, the continuous delay in consuming data and the time taken for processing highlight the system's resilience. Overall, the system effectively showcases its ability to handle a constant flow of fast-moving data without any noticeable decrease in performance, fulfilling the anticipated results of Scenario S2. Furthermore, the system has been continuously checked
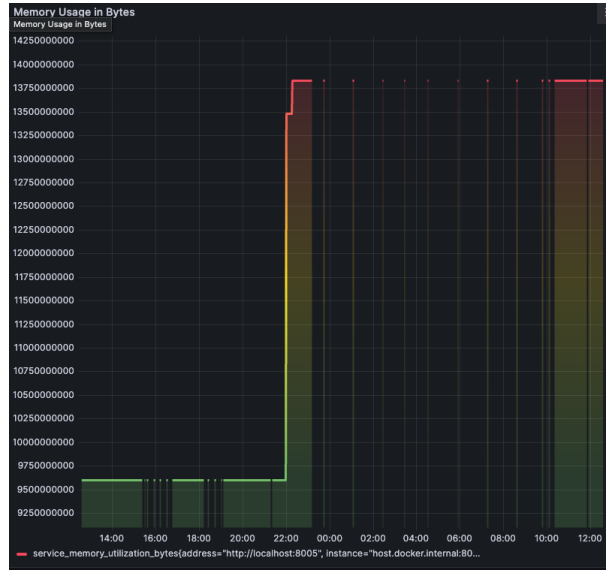
Figure 27: Memory Utilisation in Weather Domain in High Velocity Case

to guarantee that the allocated memory for streaming is released after the transmission is finished. No memory leak was detected during the monitoring process, and all memory was successfully released after the streaming was completed.

Based on the extensive results obtained from Scenarios S1 and S2, which include both high-volume and high-velocity data ingestion processes, it is concluded that Scenario S3's emphasis on data variety is inherently resolved and confirmed, eliminating the need for a separate testing phase for Scenario S3.
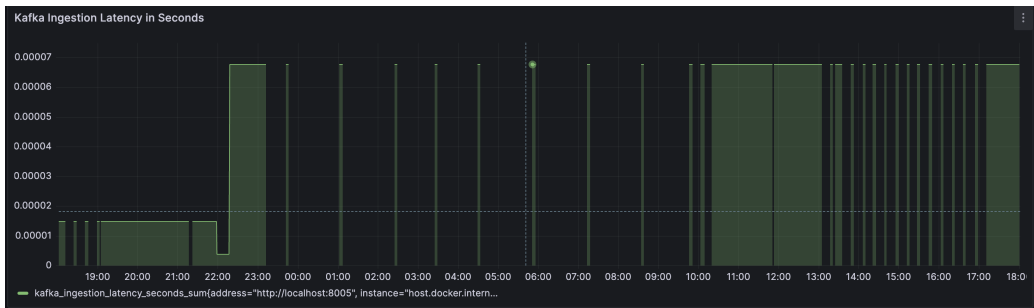

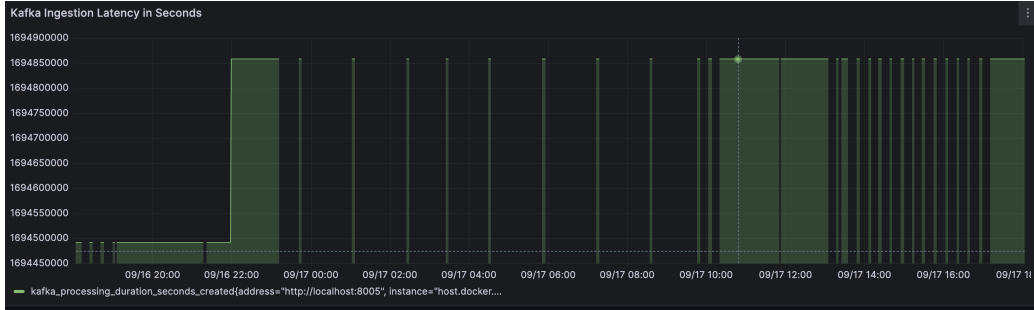Figure 28: Kafka Ingestion Latency in Streaming Case

Figure 29: Processing Duration in Streaming Case in the Weather Domain

9.4.3. Scenario S4, S5, S6

Subsequently, an evaluation is conducted for Scenarios S4, S5, and S6 as
a whole. The decision to test these scenarios concurrently is based on the
fact that they have common system operations and the capacity to provide
a thorough system evaluation while maximising efficiency. Furthermore, the
interconnected relationship between OpenID Connect and bearer token sys-
tems implies that doing a simultaneous test would provide a comprehensive
understanding of the system's performance.

A Python service is initialised to replicate the behaviour and workload of
an external data scientist. While Jupyter Notebook was originally designed
to replicate the tasks performed by data scientists, a separate Python service
has emerged as the best option. Its inherent low-level characteristics allow for
precise management and enhance the efficiency of capturing metrics. Similar
to previous services, this Python service is equipped to provide telemetry
data to a telemetry processing application via Kafka REST proxy.

Keycloak, the authentication and authorization server, plays a crucial role
in this ecosystem. Upon initiation, the Python service acquires authorization
by creating a client in Keycloak. The following confidential information,
crucial for the functioning of the service, is securely stored in and retrieved
from the HashiCorp Vault.

The authentication process, enabled by OpenID Connect standards, al-
lows the simulated data scientist to effortlessly access Data Lichen. After
being registered, the data scientist can access and see datasets from many
domains. They can then send network requests to retrieve the specific data
they need. The flow is illustrated in Figure 30.

It is noteworthy that the Minio secret key and access key could also be
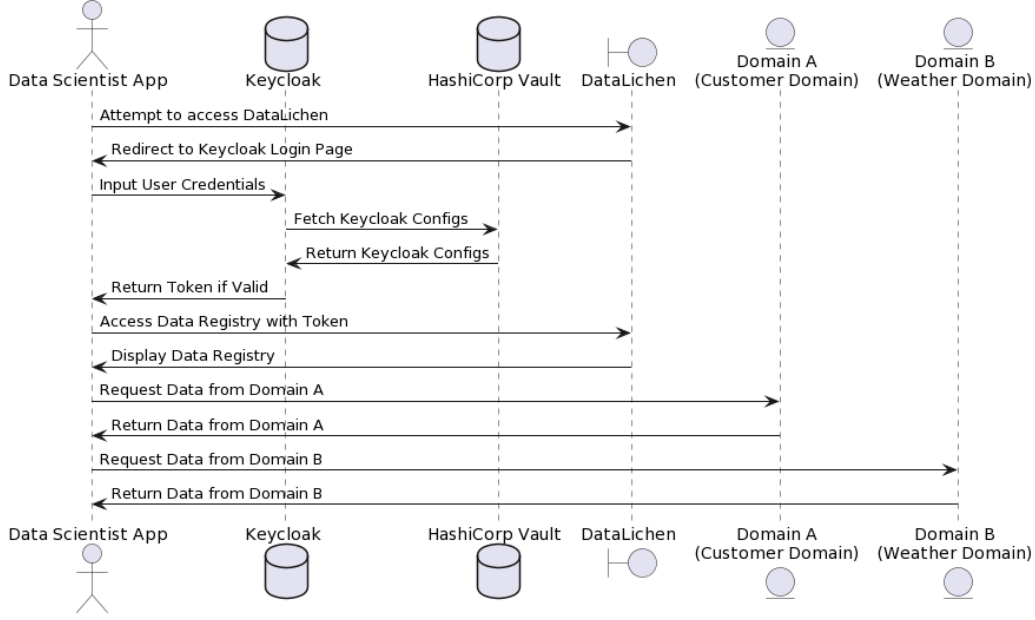
66

Figure 30: Data Scientist Flow with Authentication

stored in the vault and retrieved in each service; nevertheless, this is not done due to time and resource constraints. This does not risk the integrity of this experiment. For scenarios S4, S5, and S6, the metrics below are selected:

1. **Query Processing Time (QPT):** Measures the duration from when a complex query is initiated to when its results are returned. It encapsulates computational efficiency and performance.

$$QPT = t_{querycompleted} - t_{queryinitiated} \tag{1}$$

2. **Secret Retrieval Latency (SRL):** Captures the time taken to retrieve secrets. It summarises the system's efficiency in secret management.

$$SRL = t_{secretretrieved} - t_{retrievalinitiated} \tag{2}$$

3. **Data Integrity Verification (DIV):** Assesses the integrity of data during transmission and processing, ensuring data security.

$$DIV = Hash(DataSent) \stackrel{?}{=} Hash(DataReceived) \qquad (3)$$

After obtaining the domain data using a secure method utilising OpenID Connect and Vault on many occasions and over various days, the following conclusions arise:

9.4.3.1. Duration of Secret Retrieval:

The recorded metrics consistently displayed a length of 3 seconds throughout the timestamps. This demonstrates a dependable and consistent secret management system, as depicted in Figure 31.

Figure 31: Data Scientist Secret Retrieval Delay in Seconds

The CPU utilisation remains constant at 18.6% during the recorded time period. The behaviour can be linked to the special resource allocation of the operating system on the Apple M2 chip, which enables a consistent usage of the central processing unit (CPU), as shown in Figure 32. As a result, CPU consumption may change in various circumstances.

Figure 32: CPU Utilization in Data Scientist Application

68

9.4.3.2. Memory Utilisation in the Data Scientist Application:

The memory utilisation remains consistently at around 12.56 GB during all observations. This suggests that the application possesses efficient memory management and does not encounter memory leaks or substantial variations. This is illustrated in Figure 33.

Figure 33: Memory Utilization in Data Scientist Application

9.4.3.3. Query Processing Duration

The duration of query processing consistently stayed at 10 seconds throughout the measured time period. Nevertheless, there is a decrease to 2 seconds at the conclusion of the observation period (Figure 34). This abrupt decline indicates potential opportunities for optimising or modifying the characteristics of the processed queries.

Figure 34: Query Processing Duration in Data Science Application

An essential element contributing to the system's reliable performance is its domain-driven asynchronous and event-driven architecture, which is in line with the research conducted by Braun (2021) on distributed systems.

69

This architecture demonstrates good proficiency in handling simultaneous data access and ensuring the integrity of domain models in asynchronous scenarios, concepts that are also discussed in Kleppmann's renowned book, "Designing Data-Intensive Applications" (2017). The prototype's capacity to manage simultaneous updates and uphold data integrity demonstrates the effectiveness of this architectural approach in the construction of distributed systems.

9.5. Challenges and Duration of the Experiment

The experimentation phase lasts around four months and involves a methodical process to assess each situation in order to thoroughly validate the Terramycelium system according to the specified specifications. The experiment faced numerous architectural and technical obstacles, which greatly influenced the design decisions and comprehension of the system's behaviour. The issues are classified into four categories: Infrastructure, Integration, Data Management, and Performance.

9.5.1. Challenges in Infrastructure

(a) The Confluent Helm charts were incompatible because they relied on an alpha version of a Kubernetes resource called PodResourceBudget. The resource was in the alpha stage until V1.22, however, this experiment was conducted on version 1.27. The Nginx Ingress attempted to utilise the cloud manager to establish a load balancer, which led to the ingress service's external IP being stuck in a perpetual pending state. This happened because the cluster was initiated in a local development environment, resulting in the unavailability of a cloud service to deploy a load balancer such as EKS or AKS.

(b) A further infrastructure obstacle emerged with the Kafka pod, which had recurring failures. Due to Kafka's extensive use of file storage, such as segment files and log files, the initial allocation of 64GB for the Kafka instance was inadequate. Afterwards, the configuration was modified, increasing the cluster storage capacity to 300GB.

(c) During the development process, Prometheus and Grafana need a host address for the purpose of intercommunication. The same applies to both the telemetry processing service and the Prometheus metrics scraper. Nevertheless, due to the fact that the prototype was created

70

within a solitary computational setting, a significant portion of the communication relied on the main system. This presented challenges, as the loopback address or localhost in each Docker container resolved to the container's loopback interface rather than the host's interface. In order to address this problem, a separate DNS was used that specifically pointed to the host.

9.5.2. Integration Challenges

(a) An important obstacle was encountered when attempting to establish a connection between the Keycloak helm chart and the Postgres DB, despite specifically configuring a login and password. The cluster necessitated repeated restarts. The final answer entailed transitioning from Minikube to KIND.

Once the entrance was established, the objective was to precisely route traffic to different services. The majority of services had their own dedicated servers, which required the proper configuration of the reverse proxy to ensure that upstream servers could reply effectively. For example, the Keycloak server experienced difficulties in identifying some host paths, as it expected a root path for serving files. Both the server and proxy required modifications to ensure proper functioning. Most services encountered similar problems because many open-source software needed to be configured in order to deliver their assets on paths other than the root.

The user's text is a single item. Hosting Vault UI through an ingress encountered difficulties because Vault-UI carries out several redirections on different application paths, making rewriting or pass-through methods ineffective. This problem elicited inquiries on prominent online platforms such as StackOverflow and Vault's GitHub repository. As of February 7th, 2023, no feasible answer had become apparent. The only feasible method entailed allocating a domain to vault-ui and directing traffic through a load balancer. Nevertheless, as Vault-UI primarily served as a graphical interface for interacting with Vault and not a core architectural element, the experiment's integrity remained unaffected.

The user's text is a single item. Despite being deployed as a helm chart, the use of Prometheus for metrics and log retrieval by Kiali made the network connection complex. Difficulties emerged as a result

71

of intricate networking issues and the inherent service accounts of Kubernetes. Understanding the external authorizer API used by Envoy and enabling it in a Kubernetes cluster's service mesh within a specified namespace was not easy. There was widespread confusion over the lifecycle of service mesh objects, the admission reviews, and the admission controller.

9.5.3. Challenges in Managing Data

(a) One of the main difficulties in designing the data quantum was figuring out the precise nature of the data resolution provided by the analytical service. Using FastAPI, which is inherently event-driven, it is not advisable to block the main thread by making continuous requests to an HTTP endpoint for events. A solution utilised a function that was executed conditionally to retrieve all events related to a certain topic, either at the system's initialization or through an endpoint.

Another problem connected to the creation of the local-host-based cluster came in the context of ingress. In order to route incoming traffic to the Kafka REST proxy, nginx was configured to perform target rewriting. While this method is suitable for incoming network requests, outgoing requests present challenges. For instance, when the Kafka REST proxy API provided the consumer URL, it excluded the rewriting rules, which required the use of custom code to alter the URL.

Handling the data flow for domain B, specifically the customer reviews domain, presented difficulties because of the substantial size of the Yelp academic datasets, with some reaching up to 5GB. In order to provide version control and avoid server and stream timeouts, it was necessary to divide these JSON files into smaller chunks. A bespoke Python script was created to manage the division of JSON files into smaller chunks, guaranteeing that no file surpassed 99MB.

(b) SQLite, used in analytical services to store many types of data, presented difficulties with semi-structured data such as JSON. The lack of uniformity in JSON data formats within the customer domain required the development of separate tables to accommodate various semi-structured data sets. In order to address this issue, a bespoke function was created to dynamically generate SQLite tables.

A noticeable inefficiency arose when initially gathering metrics from different services using normal Python and Node APIs. It was quickly

determined that many traces were missing important metadata, such as service name, service version, and environment, which are necessary ₁₇₅₅ for proper visualisation. This mistake required a repetition of the first scenario in order to provide the best possible visualisation and narrative consistency.

An associated matter pertained to the automated removal of outdated Kafka consumers by the Kafka-rest-proxy service. As a consequence ₁₇₆₀ of this activity, there was a failure in the service when making further queries to Kafka. In order to alleviate this issue, a bespoke function was created to modify the consumer and restore a connection to the REST server.

9.5.4. Performance Challenges

₁₇₆₅ (a) While transferring operational data from the operational service to the analytical service, storage information was sent along with the success event. This strategy presented causality concerns. The following queries, which were intended to process this data and record it in Datalichen, relied on the distributed storage data. To tackle this is- ₁₇₇₀ sue, it was necessary to take into account a centralised configuration pattern.

The user's text is a single item. The Apple M2 chip architecture presented an extra performance difficulty by hindering Psutil's ability to accurately represent CPU utilisation.

₁₇₇₅ Notwithstanding these challenges, the system exhibits proficiency in managing intricate settings. Nevertheless, the scarcity of time and infrastructure constrains the extent of testing. The selected situations offer a thorough comprehension of the system's capabilities, but a more exhaustive examination is possible under more advantageous circumstances. Despite its limitations, ₁₇₈₀ the experiment demonstrates the resilience and promise of the Terramycelium prototype.

9.6. Final Notes on Experiment

The comprehensive evaluation of the Terramycelium system across several scenarios has definitively demonstrated its ability to fulfil, and in some cases, ₁₇₈₅ go beyond, the specified criteria. The experimental data lead to the following conclusions:

1. **Volume**: The large volume data import scenario, also known as Scenario S1, showcases the system's ability to handle both asynchronous and batch processing, hence fulfilling the requirements outlined in Vol-1 and Vol-2. The system offers a flexible storage solution for extensive datasets, efficiently managing the influx of data during periods of high demand.

2. **Velocity**: In Scenario S2, which focused on fast data ingestion, the system demonstrated its ability to handle various data transmission speeds, satisfying requirements textbfVel-1 to textbfVel-5. The system's ability to efficiently deliver streaming data to users, along with its capacity for rapid search and real-time data processing, is very commendable.

3. **Diversity**: While Scenario S3 is not explicitly evaluated, insights from Scenarios S1 and S2 strongly indicate the system's capacity to handle a wide range of data forms, ranging from structured to unstructured data. This substantiates the assertions made by Var-1 to Var-4. The system's inherent ability to gather, standardise, and adapt its structure to changing needs is a notable advantage.

4. **Value**: The Complex Query Scenario (Scenario S4) demonstrates the system's computing capabilities, confirming the criteria outlined in **Val-1** to **Val-4**. The system's versatility in facilitating both batch and streaming analytical processing, as well as its adaptability in managing various output formats, distinguishes it from others. This distinction is emphasised throughout the experiment, as each domain contains a unique form of semi-structured data.

5. **Security & Privacy**: Scenarios S5 and S6 thoroughly demonstrate the system's commitment to fulfilling SaP-1 and SaP-2 by focusing on secret management and data security, respectively. By strategically using OpenID Connect with Hashicorp Vault, the highest standard of security, data preservation, and multi-tier authentication systems are ensured.

6. **Veracity**: The domain-specific character of Terramycelium naturally supports the fulfilment of requirements **Ver-1** and **Ver-2**, emphasising the significance of ensuring high-quality data and maintaining its origin and history.

9.7. Threats to Validity

This section discusses potential threats to the validity of our study on the development and evaluation of Terramycelium, taking into account the identified challenges and limitations.

9.8. Infrastructure and Technical Challenges

The study encountered infrastructure challenges, including issues with Kubernetes resources, difficulties with Kafka storage, and problems related to DNS configuration for container communication. These issues highlight the potential impact of technical constraints on the prototype's performance and the generalizability of our findings.

9.9. Integration and Data Management Difficulties

Integration challenges with third-party services like Keycloak and the management of large datasets underscore the complexities of building BD systems. These difficulties reflect on the RA's adaptability and its applicability across different domains and data scales.

9.10. External Validity

The experimental design, focusing on a prototype and selected scenarios for RA evaluation, may not cover the breadth of potential real-world applications. This limitation could affect the generalizability of our findings to other contexts or systems with different requirements.

Addressing these threats to validity involves further research, particularly exploring alternative technologies, expanding the experimental design to cover a wider range of scenarios, and conducting comprehensive performance and scalability testing.

9.11. Construct Validity

Construct validity is a cornerstone of our research methodology, particularly in ensuring that Terramycelium accurately reflects and addresses the multifaceted nature of BD systems. To this end, we have undertaken several critical steps:

1. We meticulously align the operational definitions of key BD characteristics – including velocity, veracity, volume, variety, value, security, and privacy – with the constructs integrated into the Terramycelium RA. This alignment is essential to mitigate ambiguity and uphold the RA's construct validity [70].

75

2. Validation of the RA against established best practices and industry standards is undertaken rigorously. Such an alignment is crucial for ensuring that the RA adheres to widely recognised norms and practices, a key aspect of construct validity.

3. Lastly, we assess the adaptability and scalability of Terramycelium in response to evolving data needs and varying workloads. This evaluation demonstrates the RA's flexibility and robustness, underscoring its long-term relevance and thereby enhancing construct validity in dynamic BD environments.

10. Discussion

This study explored Terramycelium, a novel approach to managing the complexities of BD systems. Utilizing a complex adaptive system model, Terramycelium proposes a distributed, domain-driven architecture aiming to enhance scalability, maintainability, and adaptability beyond what traditional monolithic architectures offer. This initiative is commendable for its effort to reconcile the inherent volatility and diversity of BD with the principles of modern software engineering. However, as the following discussion highlights, the real-world applicability and scalability of such an architecture across varying domains and data scales require rigorous testing.

The initial evaluation, conducted through prototyping and case mechanism experiments, suggested potential benefits in data management strategies. However, the limited scope of these evaluations necessitates further investigation to represent diverse BD scenarios and workloads. The compatibility with microservices and the support for engineering practices also need to be contextualized within the broader spectrum of BD applications, considering varying operational and business requirements.

The domain-driven and distributed nature of Terramycelium elicited diverse viewpoints regarding its compatibility with existing industry practices and anticipated future developments. While experts acknowledged the promising and advanced approaches (i1, i2), they also highlighted potential challenges related to team configurations, organizational scale, and industry readiness (i2). The architecture's complexity might pose adoption hurdles for some organizations, even larger ones (i3).

Experts commended Terramycelium for its unique characteristics, particularly the integration of domain-driven design and data engineering (i1).

76

This holistic approach, along with the architecture's adaptability and comprehensive ecosystem, were highlighted as potential strengths (i2, i3). The use of immutable logs and the ability to self-provision components were also recognized as valuable features (i3). Additionally, the modular architecture ₁₈₉₅ with clear boundaries between components was praised for its clarity and potential benefits (i2).

The broader implications of this study, touching upon organizational and operational aspects of BD systems, highlight a shift towards more agile and responsive data management strategies. This envisioned transition aligns ₁₉₀₀ with the demand for data-driven organizational processes. Nonetheless, the transition to such decentralized architectures raises significant considerations regarding data governance, security, and privacy, areas that demand detailed scrutiny.

Future research directions proposed for integrating Terramycelium with ₁₉₀₅ emerging technologies and validating its effectiveness across different sectors are well-founded. The incorporation of artificial intelligence and machine learning could indeed amplify the architecture's capabilities. However, as suggested earlier, the tangible impact of such integrations on the architecture's performance, usability, and cost-effectiveness needs clear articulation, ₁₉₁₀ supported by empirical evidence.

Terramycelium presents a promising vision for managing complex BD systems. Its focus on domain-driven design, adaptability, and integration with modern engineering practices offers significant advantages. However, real-world applicability, scalability, and the practicalities of implementation ₁₉₁₅ require further investigation. Addressing the potential challenges identified through expert feedback and further research are crucial steps towards realizing the full potential of this innovative architecture.

11. Conclusion

This study introduced Terramycelium, a RA for BD systems, utilizing ₁₉₂₀ a distributed, domain-driven approach. Evaluation through prototyping suggests that Terramycelium may improve scalability, maintainability, and adaptability, offering an alternative to traditional architectures.

Compatibility with microservices and event-driven services highlights Terramycelium's alignment with current software engineering practices, suggest- ₁₉₂₅ ing its potential utility in BD system design. Moreover, the architecture's

capacity for integration with evolving technologies underscores its applicability across various BD scenarios.

The research indicates a shift towards agile data management practices, aligning with the strategic value of BD. Future work includes assessing the ¹⁹³⁰ integration with advanced technologies and broader empirical validation to ascertain Terramycelium's effectiveness.

In conclusion, Terramycelium contributes to BD system architecture discussions, suggesting the viability of distributed architectures. Its practical implementation and scalability in diverse ecosystems warrant further inves-¹⁹³⁵ tigation, emphasizing the need for ongoing research in BD architecture.

References

[1] ISO/IEC 26550: 2015-software and systems engineering–reference model for product line engineering and management, 2015.

[2] Github repository for terramycelium helm charts and applications. https://anonymous.4open.science/r/helmChartsAndApplicationsForMetamycelium-D440/, 2024.

[3] Github repository for terramycelium infrastructure. https://anonymous.4open.science/r/InfrastructureForMetamycelium-8F00/, 2024.

[4] Visual crossing. `https://www.visualcrossing.com/`, 2024.

[5] Alain Abran, James W Moore, Pierre Bourque, Robert Dupuis, and L Tripp. Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*, page 25, 2004.

[6] Jameela Al-Jaroodi and Nader Mohamed. Characteristics and requirements of big data analytics applications. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 426–432, 2016.

[7] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 141–150. IEEE, 2009.

[8] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431, 2012.

[9] ACM ANSI. X3/sparc study group on dbms, interim report. *SIGMOD FDT Bull*, 7(2), 1975.

[10] Pouya Ataei and Alan Litchfield. The state of big data reference architectures: a systematic literature review. *IEEE Access*, 2022.

[11] Pouya Ataei and Alan Litchfield. Towards a domain-driven distributed reference architecture for big data systems. In *AMCIS 2023*, 2023.

[12] Pouya Ataei and Andrew Litchfield. Big data reference architectures: A systematic literature review. In *2020 31st Australasian Conference on Information Systems (ACIS)*, pages 1–11. IEEE, 2020.

[13] Pouya Ataei and Daniel Staegemann. Application of microservices patterns to big data systems. *Journal of Big Data*, 10(1):1–49, 2023.

[14] Mehdi Bahrami and Mukesh Singhal. *The role of cloud computing architecture in big data*, pages 275–295. Springer, 2015.

[15] B Bashari Rad, N Akbarzadeh, P Ataei, and Y Khakbiz. Security and privacy challenges in big data era. *International Journal of Control Theory and Applications*, 9(43):437–448, 2016.

[16] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: A methodology to develop software product lines. In *Proceedings of the 1999 symposium on Software reusability*, pages 122–131, 1999.

[17] PerOlof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. Architecture-level modifiability analysis (alma). *Journal of Systems and Software*, 69(1-2):129–147, 2004.

[18] A. Bode and Others. The evolution of data architectures: From monoliths to meshes and fabrics. *Journal of Data Engineering Innovation*, 1(1):100–115, 2023.

[19] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Patricia Lago, Manuel Mazzara, Victor Rivera, and Andrey Sadovykh. Microservices. *Science and Engineering. Springer*, 2020.

[20] John M. Carroll. *Scenario-Based Design: Envisioning Work and Technology in System Development.* Wiley, 1995.

[21] Wo L Chang and David Boyd. Nist big data interoperability framework: Volume 6, big data reference architecture. Technical report, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2018.

[22] Wo L Chang, Nancy Grady, et al. Nist big data interoperability framework: volume 1, big data definitions. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2015.

[23] Robert Cloutier, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole, and Mary Bone. The concept of reference architectures. *Systems Engineering*, 13(1):14–27, 2010.

[24] NEMAC+FernLeaf Collaborative. U.s. climate resilience toolkit: Climate explorer, 2023. Available from: `https://crt-climate-explorer.nemac.org/`.

[25] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 4 edition, 2005.

[26] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wasowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*, pages 173–182, 2012.

[27] Zhamak Dehghani. *Data Mesh: Delivering Data-Driven Value at Scale.* O'Reilly Media, 2022.

[28] Mustapha Derras, Laurent Deruelle, Jean Michel Douin, Nicole Levy, Francisca Losavio, Yann Pollet, and Valérie Reiner. Reference architecture design: a practical approach. In *13th International Conference on Software Technologies (ICSOFT)*, pages 633–640. SciTePress-Science and Technology Publications, 2018.

[29] N. Dragoni, S. Giallorenzo, A. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. 2016.

[30] Hanif Eridaputra, Bayu Hendradjaya, and Wikan Danar Sunindyo. Modeling the requirements for big data application using goal oriented approach. In *2014 international conference on data and software engineering (ICODSE)*, pages 1–6, 2014.

[31] Eric Evans and Eric J Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[32] Neal Ford, Mark Richards, Pramod Sadalage, and Zhamak Dehghani. *Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures*. O'Reilly Media, 1 edition, 11 2021.

[33] M Galster and P Avgeriou. Empirically-grounded reference architectures: a proposal. *Joint ACM*, 2011.

[34] Matthias Galster and Paris Avgeriou. Empirically-grounded reference architectures: a proposal. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 153–158. ACM, 2011.

[35] Gartner. Essential guide to data fabric. `https://www.gartner.com/en/publications/essential-guide-to-data-fabric`, 2023. Accessed: 2023-03-15.

[36] GroupLens. Movielens 20m dataset. `https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset`, 2024.

[37] HashiCorp. Vault helm chart. https://artifacthub.io/packages/helm/hashicorp/vault, 2023.

[38] Nicolaus Henke, Jacques Bughin, Michael Chui, James Manyika, Tamim Saleh, Bill Wiseman, and Guru Sethupathy. The age of analytics: Competing in a data-driven world. Technical report, McKinsey & Company, 2016. Accessed: [insert date here].

81

[39] IEEE International Organization for Standardization (ISO/IEC). Iso/iec/ieee 42010:2011, 2017.

[40] ISO ISO. Information technology — reference architecture for service oriented architecture (soa ra) — part 1: Terminology and concepts for soa. *International Organization for Standardization*, page 51, 2016.

[41] ISO/IEC. Iso/iec 29148:2018. systems and software engineering — life cycle processes — requirements engineering, 2018.

[42] Mark Johnson. *Principles of Logic in Computer Science*. TechPress, 2019.

[43] Stephen Kaisler, Frank Armour, J Alberto Espinosa, and William Money. Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE, 2013.

[44] Mohamad Kassab, Colin Neill, and Phillip Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014.

[45] Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb. Saam: A method for analyzing the properties of software architectures. In *Proceedings of 16th International Conference on Software Engineering*, pages 81–90. IEEE, 1994.

[46] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 98EX193)*, pages 68–78. IEEE, 1998.

[47] Vlad Khononov. *Learning Domain-Driven Design*. " O'Reilly Media, Inc.", 2021.

[48] Sergey Khrononov. *Learning Domain-Driven Design: Aligning Your Architecture with the Business using Context Maps, Strategic Design, and Agile Techniques*. Packt Publishing, Birmingham, UK, 2021.

[49] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.* Wiley, 2013.

[50] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792. IEEE, 2015.

[51] John Klein, Ross Buglak, David Blockow, Troy Wuttke, and Brenton Cooper. A reference architecture for big data systems in the national security domain. In *2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE)*, pages 51–57. IEEE, 2016.

[52] Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems.* " O'Reilly Media, Inc.", 2017.

[53] Jay Kreps. Questioning the lambda architecture. Blog post, 2014.

[54] J Stephen Lansing. Complex adaptive systems. *Annual review of anthropology*, 32(1):183–204, 2003.

[55] Phillip A Laplante. *Requirements engineering for software and systems.* Auerbach Publications, 2017.

[56] Kevin Lee and Lucy Wang. Ensuring data consistency in big data systems. In *Proceedings of the 2020 International Conference on Big Data*, pages 456–467, 2020.

[57] Paulo Merson and Joseph Yoder. Modeling microservices with ddd. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 7–8. IEEE, 2020.

[58] Fabrizio Montesi and Janine Weber. Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*, 2016.

[59] Elisa Yumi Nakagawa, Rafael Messias Martins, Katia Romero Felizardo, and Jose Carlos Maldonado. *Towards a process to design aspect-oriented reference architectures.* 2009.

[60] Elisa Yumi Nakagawa, Rafael Messias Martins, Katia Romero Felizardo, and Jose Carlos Maldonado. Towards a process to design aspect-oriented reference architectures. In *XXXV Latin American Informatics Conference (CLEI) 2009*, 2009.

[61] Elisa Yumi Nakagawa, Flavio Oquendo, and Martin Becker. Ramodel: A reference model for reference architectures. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 297–301. IEEE, 2012.

[62] NASA. Reference architecture for space data systems, 2008.

[63] Harvey Nash. Cio survey 2015. *Association with KPMG*, 2015.

[64] Sam Newman. *Building microservices.* ” O’Reilly Media, Inc.”, 2021.

[65] OATH. Oath reference architecture, release 2.0 initiative for open authentication. *OATH*, 2007.

[66] Pekka Pääkkönen and Daniel Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big data research*, 2(4):166–186, 2015.

[67] New Vantage Partners. New vantage partners annual report. Technical report, New vantage Partners, 2023.

[68] Aleš Popovič, Ray Hackney, Rana Tassabehji, and Mauro Castelli. The impact of big data analytics on firms’ high value business performance. *Information Systems Frontiers*, 20:209–222, 2018.

[69] Dino Quintero, Frank N Lee, et al. *IBM reference architecture for high performance data and AI in healthcare and life sciences.* IBM Redbooks, 2019.

[70] Babak Bashari Rad and Pouya Ataei. The big data ecosystem and its environs. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):38, 2017.

[71] Babak Bashari Rad and Pouya Ataei. The big data ecosystem and its environs. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):38, 2017.

[72] Babak Bashari Rada, Pouya Ataeib, Yasaman Khakbizc, and Nafisseh Akbarzadehd. The hype of emerging technologies: Big data as a service. *Int. J. Control Theory Appl*, 9(43):1–18, 2017.

[73] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[74] Mark Richards and Neal Ford. *Fundamentals of software architecture: an engineering approach*. O'Reilly Media, 2020.

[75] Chris Richardson. *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.

[76] Chris Richardson. *Microservices Patterns: With examples in Java*. Manning; 1st edition, 2018.

[77] J. Rose, H. Göbel, S. Cronholm, J. Holgersson, E. Söderström, and C. Hallqvist. Theory-based design principles for digital service innovation. *E-Service Journal*, 11:1, 2019.

[78] Nino Rurua, Rik Eshuis, and Maryam Razavian. Representing variability in enterprise architecture. *Business & Information Systems Engineering*, 61(2):215–227, 2019.

[79] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47. IEEE, 2013.

[80] Kubernetes Special Interest Group (SIG). kind - kubernetes in docker. https://kind.sigs.k8s.io/, 2023.

[81] Salomé Simon. Brewer's cap theorem. *CS341 Distributed Information Systems, University of Basel (HS2012)*, 2000.

[82] Matthew Skelton and Manuel Pais. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution, 2019.

[83] Ian Sommerville. *Software Engineering, 9/E*. Pearson Education India, 2011.

[84] Kriti Srivastava and Narendra Shekokar. A polyglot persistence approach for e-commerce business model. In *2016 International Conference on Information Science (ICIS)*, pages 7–11. IEEE, 2016.

[85] Ben Stopford. *Designing Event-Driven Systems.* O'Reilly Media, Inc., 2018.

[86] Vanessa Stricker, Kim Lauenroth, Piero Corte, Frederic Gittler, Stefano De Panfilis, and Klaus Pohl. Creating a reference architecture for service-based systems–a pattern-based approach. In *Towards the Future Internet*, pages 149–160. IOS Press, 2010.

[87] Goparaju Purna Sudhakar. A model of critical success factors for software projects. *Journal of Enterprise Information Management*, 2012.

[88] Artifact Hub Team. Artifact hub. https://artifacthub.io/, 2023.

[89] MIT technology review insights in partnership with Databricks. Building a high-performance data organization, 2021.

[90] Phillip Viana and Liria Sato. A proposal for a reference architecture for long-term archiving, preservation, and retrieval of big data. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 622–629. IEEE, 2014.

[91] Matthias Volk, Daniel Staegemann, Ivayla Trifonova, Sascha Bosse, and Klaus Turowski. Identifying similarities of big data projects–a use case driven approach. *IEEE Access*, 8:186599–186619, 2020.

[92] Roel J Wieringa. *Design science methodology for information systems and software engineering.* Springer, 2014.

[93] Lloyd G Williams and Connie U Smith. Pasasm: a method for the performance assessment of software architectures. In *Proceedings of the 3rd international workshop on Software and performance*, pages 179–189, 2002.

[94] V. Yadav, F. Oury, N. Suda, Z. Liu, X. Gao, C. Confavreux, K. Klemenhagen, K. Tanaka, J. Gingrich, X. Guo, L. Tecott, J. Mann, R. Hen, T. Horváth, and G. Karsenty. A serotonin-dependent mechanism explains the leptin regulation of bone mass, appetite, and energy expenditure. *Cell*, 138:976–989, 2009.

[95] Yelp. Yelp dataset, 2023. Available from Yelp: `https://www.yelp.com/dataset`.

[96] Jing-Huan Yu and Zi-Meng Zhou. Components and development in big data system: A survey. *Journal of Electronic Science and Technology*, 17(1):51–72, 2019.

2200