ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

A Data Mesh Implementation

Relatore: Prof. Marco Patella

Correlatori: Martino Ongaro, Matteo Marchiori

Anno Accademico: 2021-2022

Candidato: Luca Parigi

23 Marzo, 2023

Contents

Keywords

- **BI**: Business Intelligence
- **ML**: Machine Learning
- **DWH**: Data Warehouse
- **DL**: Data Lake
- **AWS**: Amazon Web Services
- **GCP**: Google Cloud Platform
- **BQ**: BigQuery
- **S3**: Simple Storage System
- **DP**: Data Product

Abstract

I dati sono diventati centrali per l'economia di tutte le aziende, grazie a strumenti BI di sempre più facile utilizzo e ai vantaggi forniti dai modelli di ML. Le organizzazioni che operano in più settori, tuttavia, devono affrontare il problema di gestire ed estrarre valore da una quantità enorme e diversificata di dati. L'architettura Data Mesh, teorizzata da Zhamak Dehghani nel 2018, sta riscuotendo molto interesse perché promette una soluzione a questi problemi favorendo la decentralizzazione dei dati, anziché ostacolarla come è sempre avvenuto nelle strutture tradizionali di gestione dei dati. Il lavoro svolto per questa tesi è suddiviso in due parti: una parte teorica sul Data Mesh, perché è necessario, da cosa ha preso origine e i suoi principi; una pratica, svolta nell'ambito di un progetto interno all'azienda Bip, che ha visto l'implementazione da zero di un'architettura Data Mesh cross-cloud, utilizzando vari strumenti open-source. Il mio contributo si è concentrato sull'area specifica di Data Consumption, implementnado alcuni strumenti – quali GCP Dataplex, Trino e Apache Superset – attraverso i linguaggi Terraform e YAML. La tesi si conclude con un confronto dell'architettura creata in Bip con la teoria e i servizi offerti da altre grandi aziende, come Google e Microsoft.

Data has become central to the economy of all companies, thanks to increasingly easy-to-use BI tools and the benefits of the insights provided by ML models. Organizations with roots in multiple businesses, however, must deal with the problem of organizing and extracting value from an overwhelming and diverse amount of data. The Data Mesh architecture, theorized by Zhamak Dehghani in 2018, is gaining a lot of interest because it promises a solution to these problems by embracing data decentralization, rather than hindering it as has always been done in traditional structures. The work done for this thesis has been divided into two parts: a theoretical part on

Data Mesh, why it is necessary, what it originated from, and its principles; a practical one done inside an internal project at Bip company, involving the implementation of a cross-cloud Data Mesh architecture from scratch, using many open-source tools. My contribution focused on the specific area of Data Consumption, implementing some tools (GCP Dataplex, Trino and Apache Superset) via Terraform and YAML languages. The thesis concludes with a comparison of the architecture created in Bip against the theory and services offered by other large companies, such as Google and Microsoft.

Chapter 1

Introduction

The amount of data is growing every day more, even overnight: in the*Data Never Sleeps 10.0*, a study conducted by Domo in 2022, we know the total amount of data predicted to be created, captured, copied and consumed globally in 2022 is 97 zettabytes. To be clearer, this is one zettabyte: 1,000,000,000,000,000,000,000 bytes. If every gigabyte in a zettabyte were a meter, it could span the distance of the Amazon River (the world's longest river at 6,992 kilometers) more than 150,000 times [1]. Over the past ten years, digital activities have grown exponentially, and the Covid-19 pandemic has also played a significant role in this: we have seen an increase in the use of digital tools to support our personal and work needs, from connecting and communicating to conducting transactions and business [2].

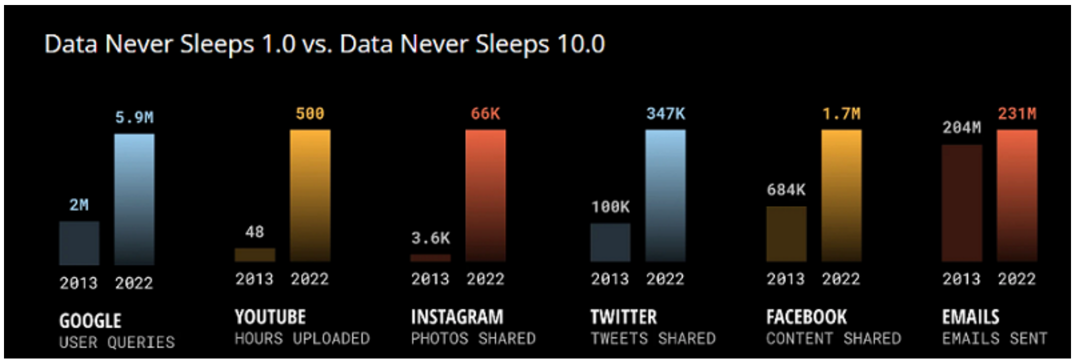


Figure 1-1: How much data is generated every minute? 2013 vs. 2022 [2].

Data is ubiquitous. Data is the by-product of every digital action we take. Everything, every system, every process, every sensor generates data [3]. Today's processes are not efficient enough to keep up with all the data we generate. It's a flood of data, and the challenge is to turn it into a meaningful and consumable stream of information in real time. Since this thesis has been written on the work done during an internship, the context of data is approached from the perspective of a business and an interesting question to ask today is: what is the importance of data in a company today? Is data the new gold? If we continue to view data as the byproduct of our actions, then no: data is only a tool - a shovel - to dig up gold. That was the scope of Business Intelligence: companies wanted to generate reports and dashboards to manage operational risks, respond to compliance, and ultimately make fact-based business decisions. But today, companies' data aspirations have evolved and they want to become data-driven, using Machine Learning in product design, such as automated assistants, service design, and customer experience, such as personalized health care. Large companies need to process huge amounts of data intelligently and efficiently to provide machine learning models with the near real-time data they need. They cannot be satisfied with storing all the data generated in loosely organized data lakes (swamps). Instead, if we begin to think of data as a product, as something that has value on its own, independent from the source, reusable and central to a company's economy, then we could say that data is the new gold. To do that we have to change the way we understand, structure and manage data; this was Zhamak Dehghani's idea in 2018, when studying how organizations were trying to manage the complexity of mass digitization she conceived the idea of Data Mesh.

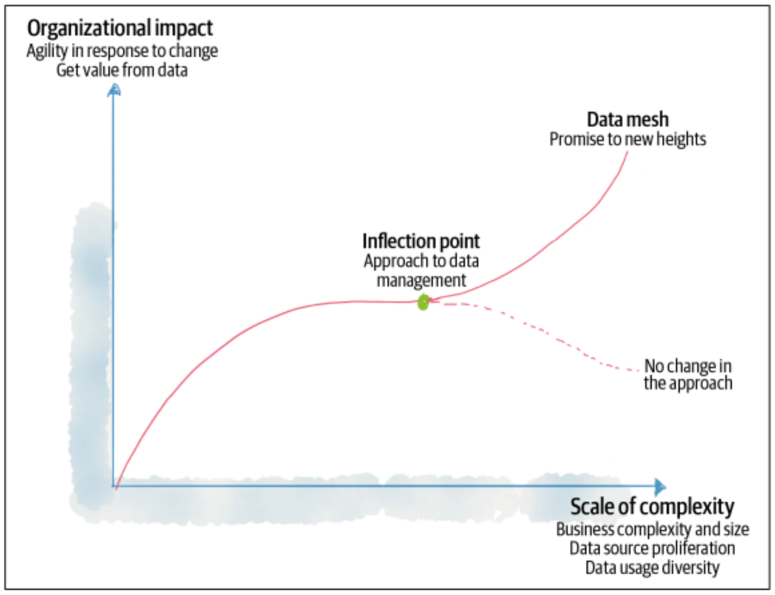


Figure 1-2: The inflection point in the approach to analytical data management [4].

Data Mesh is a decentralized sociotechnical approach to share, access, and manage analytical data in complex and large-scale environments - within or across organization [4]. It is a different way of understanding and composing the technologies we already know and use.

1.1 Bip Consulting

Bip - Business integration partners S.p.A. is an Italian multinational consulting company working for enterprises and public administration. Founded in 2003, it operates in the field of business strategy, information technology, and human resources consulting. The economic sectors in which it operates are: Industry, Finance, Automotive, Energy and Utilities, Telecommunications, Public Administration, E-commerce, Sales and Consumer Goods, Healthcare, and Pharmaceuticals [5]. My internship at Bip took place over three months in the xTech center of excellence, featuring multidisciplinary teams in the areas: Data, Cloud, Platforms, Solutions [6]. I was included in a Cloud Team and gave my contribution within an already started research that involved the development of a Data Mesh infrastructure for testing purposes, from scratch. In particular, my contribution was focused in the consumption area of the data product. This project was initiated as a result of a growing interest by Bip in this new data architecture, which is well suited for use in managing the data of large companies such as banks. The goal was to build a demo - and reusable artifacts - to show to potential customers: in recent month companies have shown particular interest in this architecture, but the negotiation phase and, if successful the subsequent implementation and transformation, will require quite a long time (years). Below I provide an overview of the structure of this thesis. The second chapter discusses the nature of data and the history of the technological evolution to manage it. The third chapter is about the theory of Data Mesh, what inspired this architecture and its principles. The fourth chapter gives an overview of the Data Mesh implementation done as Bip internal project. The fifth chapter illustrates my contribution to the project. The sixth chapter discusses the results of the project. The last chapter contains some reflections on what the Data Mesh project has been and future developments.

Chapter 2

Data Flow

First, what is data? In the pursuit of knowledge, data is a collection of discrete values that convey information, describing quantity, quality, fact, statistics, other basic units of meaning, or simply sequences of symbols that may be further interpreted [7]. Data can be used if they are first contextualized and interpreted, and in the context of companies using the data produced by their business activities, it is essential to know that not all data are the same and there are different ways to store and classify them.

2.1 Drinking Data

Humankind has always used data and information for taking advantages. In the history of companies, data has been essential for taking decisions and keeping track of transactions: real-world interactions in which something was exchanged - money, product, information, request for services, and so on. But the definition of transaction in this context has expanded over the years, especially since the advent of the Internet, to encompass any kind of digital interaction or engagement with a business that can be triggered from anywhere in the world and via any web-connected sensor [8]. At the beginning of the Internet, for example, the focus was on transaction response time. As operations grew on e-commerce sites, we also wanted to know who was using the system, from where, what they were doing, for how long and most importantly how one could offer more value to existing customers and bring in new ones [9]. Precisely because of the need to use this kind of data, the first Business Intelligence software solutions were created: tools to help businesses to make more informed decisions by providing them with data-driven insights.

2.1.1 Business Intelligence

The term Business Intelligence has been around since before the Internet and refers to the set of business processes and technologies related to collecting, organizing data, and extracting information from it. BI is used to understand the capabilities available in the firm; the state of the art, trends and future directions in the markets, the technologies, and the regulatory environment in which the firm competes; and the action of competitors and the implications of these actions [10]. Even today, BI tools are still common and are used to:

- Compactly visualize data from multiple sources through dashboards, charts, graph, maps;
- Explore and analyze data to find patterns and trends, generating reports to share with other team members and stakeholders (data mining and reporting tools).

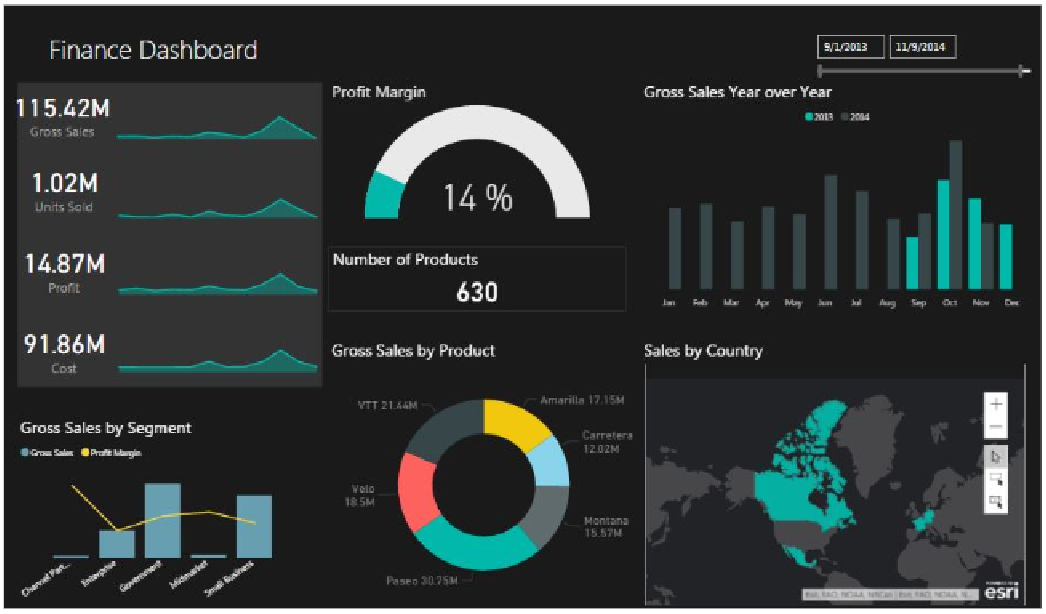


Figure 2-1: PowerBI dashboard example

But where do these tools get their data from?

2.1.2 Operational Data

Operational data supports running the business and keeps the current state of the business with transactional integrity [4], they are the data generated and used in the day-to-day operations of an organization. This type of data is typically transactional in nature, meaning it records the details of specific business transactions or events. Examples of operational data include customer orders, invoices, inventory records, and employee time sheets. Operational data is typically stored in operational databases, designed to support the rapid insertion, updating, and retrieval of individual rows as needed for the organization's daily operations. These databases are usually optimized for transactional processing, meaning they are designed to support high volumes of read and write operations with low latency. In the 90s' companies started to manage multiple sources of data - such as transactional databases, spreadsheets, and flat files - pulling and integrating data from them was time-consuming and error-prone. There was the need for a central architectural model to organize the flow of data from the operational systems to decision support environments (BI tools).

2.1.3 Data Warehouse

Data Warehouse is a central repository of integrated data from one or more heterogeneous sources [11]. The goal is to consolidate data, starting from the data produced from day-to-day operations (operational data) to aggregated data ready to be queried and analyzed by Business Intelligence [12]. Data from many different sources (like transactional databases, log files) are extracted and transformed into a format that is suitable for analysis, then loaded into the Data Warehouse (ETL process): a physical Data Storage can be implemented with a Relational DB or NoSQL DB. The DWH has a Schema: a logical structure which defines how the data is organized (tables, columns, data entities, relationships between data entities) and how it can be queried. Then the data stored can be analyzed through BI tools. A Security Layer is also present, with governance controls to ensure that the data is protected and used appropriately.

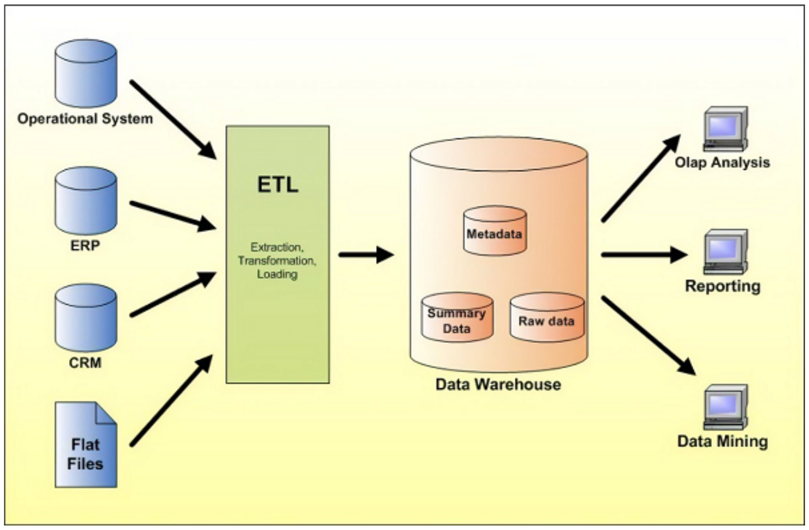


Figure 2-2: - Flow of data, from sources to BI tools, through Data Warehouse.

So, a flow of data began from the Operational Systems to the DWHs, where they were stored after a complex process of processing and transformation to fit predefined data models designed to serve departmental BI and reporting use cases. In the '90s this was experienced as a revolution, enabling IT systems to support large-scale business decisions. Zhamak Dehghani, in her book on Data Mesh [4], reflects on how the nomenclature regarding the data world takes its cue from water: data lakes, data flows, data pipelines; it is particularly the latter that convey data from the sources to the DWHs and only here do they become meaningful and useful. The data flowing from the springs were cleaned and bottled, ready for drinking and gaining insights on the companies' business.

Figure 2-3: Drinking Data, AI Art Generator.

2.2 Back to the source

Over the next decade, it was realized that operational data were not sufficient to enable companies to make truly data-driven decisions. With the advent of technologies such as Machine Learning there was a need for much more raw data and in an unprecedented amount.

2.2.1 Machine Learning

ML is a method of teaching computers to learn and make decisions on their own, without explicit programming. It is a subfield of AI that focuses on the development of algorithms and models that can analyze and interpret data and make predictions: this means it can be used in a variety of business applications to improve decision-making, increase efficiency, and enhance the customer experience. Today, for example, ML models are used to analyze clients' behaviors and trends, for personalizing the customer experience, but also to identify suspicious activities and prevent frauds. With the spread of this technology, however, the need for data changed: these models require a huge amount of raw data to train and test them. As a result, DWHs, which collect a relatively small amount of operational data, were not suitable for this purpose. There was a need to go back to the data sources.

2.2.2 Analytical Data

Analytical data is the temporal, historic, and often aggregated view of the facts of the business over time: it is the byproduct of running the business. It is modeled to provide retrospective or future-perspective insight. Analytical data is optimized for analytical logic: training machine learning models and creating report and visualizations [4]. For example, analytical data inside a commercial store can refer to the number of customers who visited the store each day, the average amount they spent per purchase, and the types of products they bought. With a certain amount of these information, we can understand the behavior of the customers and make better decisions for the business.

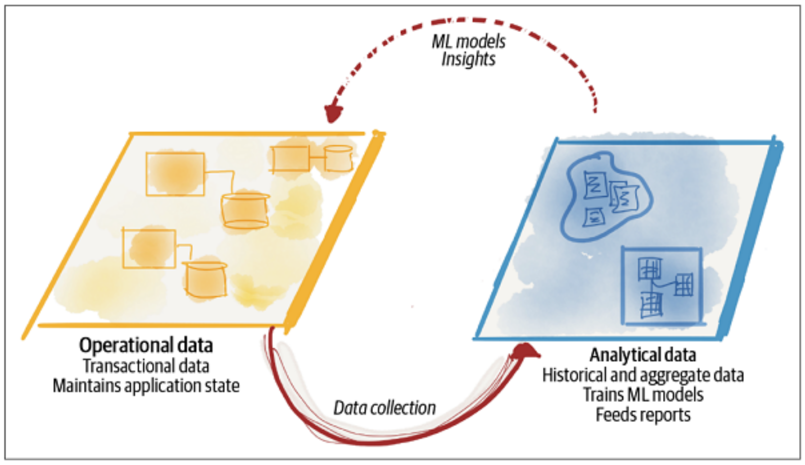


Figure 2-4: The two planes of data [4].

Therefore, by changing the type and amount of data, there was a need for a different architecture from DWHs. Siloed paradigm brought a huge cost to manage increased master data complexity and to implement cross domain analytical use cases. Organization started collecting data of interest in centralized file systems regardless of their raw format to develop and deliver data-driven applications.

2.2.3 Data Lake

Data Lakes can host heterogeneous and unstructured data: by reducing the number of data operations and taking some of the quality out of the structure, it was possible to find a place for the huge amount of data generated every day by companies. The Data Lake is an amalgamation of all the various kinds of data found in the corporation. It has become the place where enterprises offload all their data, given its low-cost storage systems with a file API that holds data in generic and open file formats, such as Apache Parquet and ORC. The use of open formats also made data directly accessible to a wide range of other analytics engines, such as machine learning systems [13]. The Data Lake pattern starts with the Data Ingestion from a variety of sources into a Data Storage, only after that data is eventually transformed (cleaning and filtering) for analysis purposes. Data Lakes also presents features like:

- **Data Catalog** that holds metadata: information about the data sources, the data schema and the transformations that have been applied;
- **Data Access**: making the data accessible to users and systems, usually through APIs;
- **Data Governance**: involves the management and control of data, usually including data quality, security and compliance.

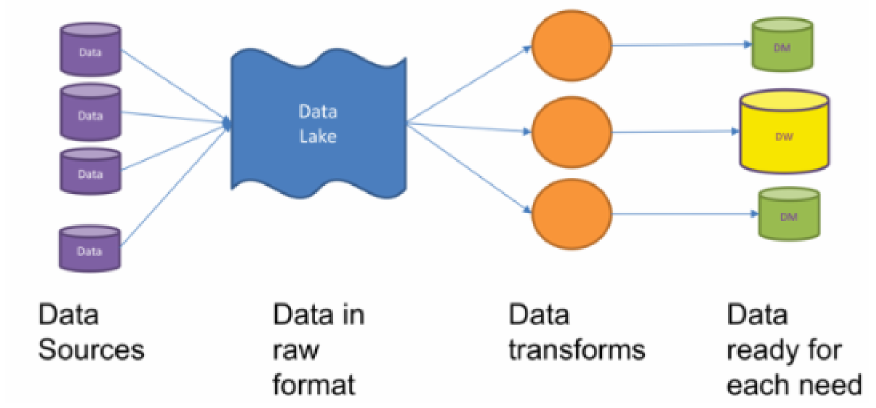


Figure 2-5: Data Lake Pattern.

Thus, there was a shift from using bottled data to channeling all source data into artificial lakes to collect all the necessary data for the company.

2.3 Data Flood

The companies soon realized that managing data storage in a single centralized architecture was expensive due to the increasing volume and number of data sources. Data Lakes also lacked support for transactions, no enforcement of data quality and poor performance optimizations. As a result, most of the data lakes in the enterprise had become data swamps. But not only that, they were not efficient enough to use AI and BI directly on the data stored (also not many tools supported them): since all the data is stored and managed as files, they do not provide fine-grained access control on the contents of files, but only coarse-grained access governing who can access what files or directories [14]. For these reasons, many companies started to move subsets of data from Lakes into Data Warehouses, still with problems in managing semi-structured data, and with limited support for ML.

2.3.1 Next Gen Data Platforms

In the next years new architectures started to be implemented, trying to solve the problem of managing a big amount of data. Data Fabric is a distributed data management architecture that enables data to be accessed and shared across a variety of systems, devices and locations. It allows the creation of a single, integrated view of data from multiple sources taking advantage of virtualization techniques and frameworks. At the same time, cloud service providers, such as AWS, GCP and Microsoft Azure, became increasingly popular, supplying storage resources and computing power to corporate clients to create their own "houses" of data, which can also be processed through ETL and explored through ML and BI tools. As a result, we had Data Lakehouses solutions: a hybrid data management platform that combines the scalability of a Data Lake with the structured querying and data governance capabilities of a Data Warehouse. It allows organizations to store and manage both structured and unstructured data in a single, centralized repository, and provides a consistent and unified view of data across the organization.

2.3.2 The bottleneck

These new data management solutions are perfect for companies that run a business with a certain size and a specific type of data. Instead, big companies must manage a wide range of data: different in type (structured or not) and domain (customer, financial, sales, operational, ...). The central team responsible for managing data across the organization, typically a data engineering one, has nothing to do with either the data producers or the consumers. Because they do not have the same business skills to understand the meaning of the data, they are struggling to create a structure to house it and to ease the analysts' work.

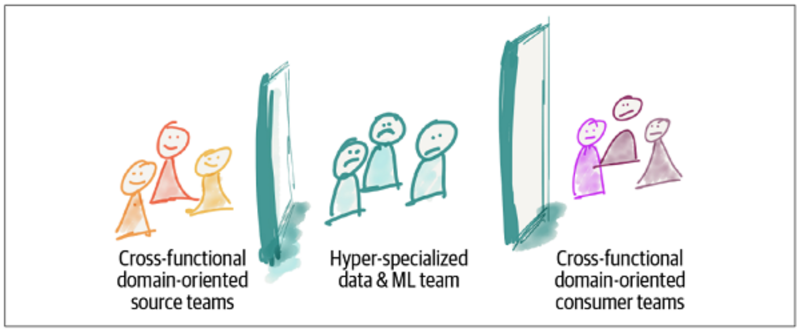

Cross-functional domain-oriented source teams
Hyper-specialized data & ML team
Cross-functional domain-oriented consumer teams

Figure 2-6: Teams with different skills working on data in company.

Data Fabrics and Lakehouses continue to have the same limitations: the centralized data governance model they propose is a bottleneck. For these systems that are complex and continuously changing, this centralized strategy used since the ‘90s is not working and it won’t work. The levees that we have been trying to build no longer hold, data is overflowing: a new architecture is needed.

Chapter 3

Mesh the Flow

We live in an infodemic era: in our daily lives we struggle to manage this continuous flow of information derived mainly from social networks. Continually distracted by few-second videos, superficial news, and clickbait posts, we cannot manage this exaggerated amount of information, failing to translate it into knowledge. The same is happening within companies: digitization has led to an excessive number of data sources producing a quantity of information that, without proper management, generates confusion. It is necessary to select the data to be used. This is one of the foundations of the Data Mesh thesis formulated by Zhamak Dehghani in 2018, after observing common failure patterns in getting value from data in large and technologically forward companies that had made substantial investments in their data technologies. As we have seen in the previous chapter, the ambitious goal of organizations is becoming data-driven for:

- Providing the best customer experience based on data and hyper-personalization.
- Reducing operational costs and time through optimizations.
- Giving employees "superpowers" with trend analysis and business intelligence [15].

The main problem is the centralized monolithically structure that organizations still use. The central data team and the monolithic architecture had become a bottleneck in response to the proliferation of data sources - inside and outside of the company - and the diversity of their use cases. We need a new architecture that embraces the reality of ever present, ubiquitous and distributed nature of data [15].

3.1 Inspirations

Zhamak Dehghani defines Data Mesh as a sociotechnological approach, meaning that it does not imply a technological innovation because it does not invent anything, but it is just a mix of already existing technologies, designs, and architectures, also from neighbor tech fields such as Agile for IT. So, we are still talking about Data Warehouses and Data Lakes, what changes is the way of using them together with other tools. Here follows the description of the two models that inspired this approach to data management.

3.1.1 Domain-Driven Design

Domain-Driven Design (DDD) is a major software design approach, focusing on modeling software to match a domain according to input from that domain's experts [16]. This design was described by Eric Evans in 2004, and the purpose was to crunch knowledge into models. DDD has fundamentally changed how technology teams form and has led to the alignment of business and technology. It has greatly influenced how organizations scale, in a way that a team can independently and autonomously own a domain capability and digital services. DDD is based on multiple models each contextualized to a particular domain, called a bounded context, meaning the delimited applicability of a particular model gives members a clear and shared understanding of what has to be consistent and what can develop independently [4]. A domain, in the context of software engineering, is the business on which the application is built. Application logic, therefore, must be built around the knowledge area of that business and models should describe aspects of a domain reducing the distance between the reality of the business and the code. This is translated into Data Mesh by dividing data into domains; there will be, for example, a domain for user data, a domain for supply data, and a domain for orders. The goal is to subdivide a complex system to achieve the desired architectures (lighter, understandable, modular). The disadvantage of this type of design is that requires professionals who have strong expertise in the identified domains; in Data Mesh this results in the data being processed by the data specialist and no longer by the data engineer, who instead is more specialized in dealing with the structure that makes the data products work transparently [17]. Once the various contexts are identified, it is possible, and recommended, to create a map of these domains (context mapping) to investigate how they are connected, the boundaries of the system and to make decisions about the design deployment and the choice of technology to adopt. Domains, moreover, can be decomposed into subdomains (representing a specific business problem) to manage their complexity. The concept of Ubiquitous Language is also present, in order to encapsulate typical business terminology within the model (and code) and to make the system easier for those not used to working with code.

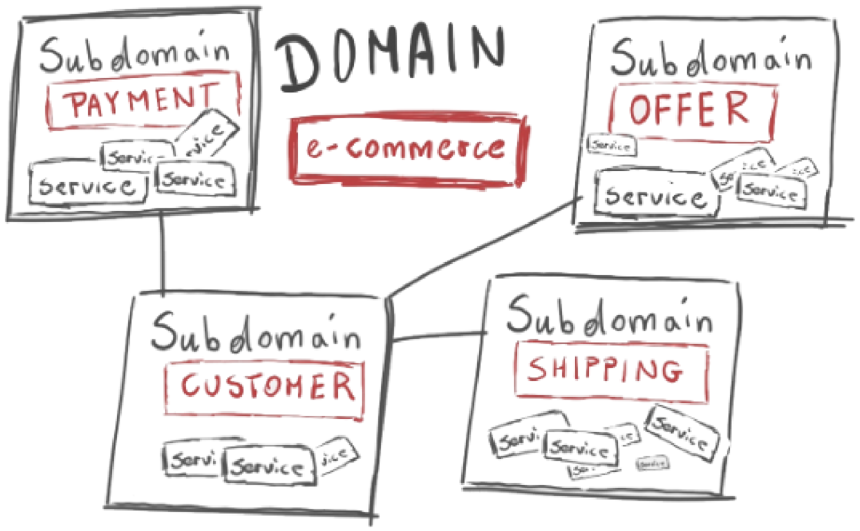


Figure 3-1: Dividing data into domains.

3.1.2 The mesh architecture

In a mesh architecture, every node is connected to every other node in a mesh-like lattice work of connections. An increase in the number of nodes increases the quality of the mesh [17].

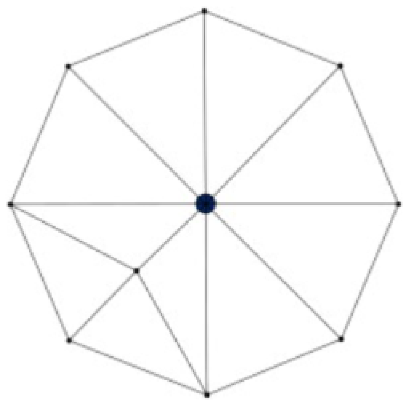


Figure 3-2: Mesh architecture.

In a centralized structure the most obvious problem is the waste of resources generated by the flow of data between various teams: all data must converge in a "single" point (like a Star Schema). The idea behind adopting a Mesh Architecture is to keep the data as close as possible to their sources, because the teams working on it may understand better the meaning of that data. Data then will be reachable though APIs and not moved. It is therefore convenient to structure the data within different domains, as explained in the previous section. The single node of the mesh is a Data Product, an entity that is autonomous and self-contained (we will delve into this in the next section). Data Products are then connected according to the domain they belong to, in this way it is possible to establish the federation policies that allow them to be accessed. Data Warehouses, and later Data Lakes, have imposed patterns on business domains, the latter chosen and tailored to the technology adopted. Data Mesh reverses this trend: the idea is first to identify business domains and then create an architecture consistent with the structure of the company's business. This concept was already expressed decades ago by Conway's Law: "any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure". Data Mesh, then, takes the concepts expressed above to delineate a division, classification, and exchange of data based on (sub)domains, which may cover a larger or smaller area of business. This decentralized approach brings data ownership back to those who have the expertise over them (Data Analysts), whereas previously it had been moved away with the flow to the Lakes and centralized structures (whose expertise lies with the Data Engineer).

3.2 The four principles

Up to this point the history of the Cloud and Data world has been covered: what architectures have been used, what has worked and what has not, and consequently what has inspired the principles that Zhamak Dehghani defines in her book in order to provide guidelines of a Data Mesh architecture.

3.2.1 Domain Ownership

The first principle is inspired from DDD, and it is based on decentralizing the data ownership: giving the responsibility to create, model, maintain, transform, and share the analytical data to the business (sub)domain - can be the source (who produce the data) or the consumers. This principle is intended to categorize data and delegate responsibility to teams: a first attempt to bring order within large companies. In contrast to traditional architectures, where there is a flow of data from operational systems to central collectors of analytical data (whether lakes or warehouses), the data here remains at the source, it is shared with "neighbors" (aggregates) and transformations are limited to fit multiple use cases. This way the teams are free to model their data according to the context, embracing the complexity and continuously changing reality of analytical data, refusing to search a single model to describe all data domains (as it was traditionally). Inside a domain it is possible to identify three big class of (analytical) data:

1. **Source-Aligned Domain Data**, created and shared by the teams who own the source operational systems. They represent the facts and reality of the business, are expected to be permanently captured and made available, closely representing the raw data at the point of creation and not modeled for a particular consumer.

2. **Aggregate Domain Data**, a composition of multiple data products, since many systems are transversal to more domains, some source-aligned data can make more sense if aggregated and used together.

3. **Consumer-Aligned Domain Data**, design to satisfy one or a small group of closely related use cases, of course they are different in nature with respect to the Source data, since they must go through

changes and transformations (pipelines) to fit a particular use case (fit-for-purpose) [4].

The result of applying the DDD to a Data Organization is to have multiple models of shared entities (data products) that are linked to each other, while removing the pipelines from the foundation of the architecture. Since it is no longer necessary to move data, pipelines are not a central concern but are hidden as an internal implementation of the data domain.

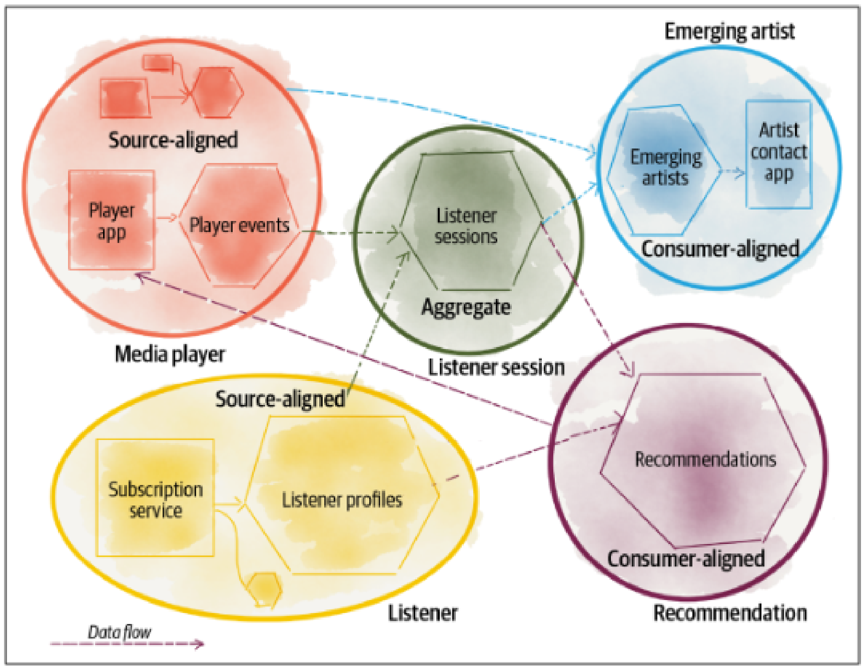


Figure 3-3: Example of domain data archetypes.

3.2.2 Data as a Product

The second principle of Data Mesh is born as a response to the data siloing challenge that may arise from the distribution of data ownership to domains, that can create frictions in using data, such as incompatibility between data sets. This is the cardinal principle of the Data Mesh, as it expresses a shift in the way to conceive data inside a company: it is no longer a byproduct of business processes, but a product essential to its business. Applying the Product Thinking to data means creating something that should be feasible, usable and valuable on its own: a Data Product to deliver to a user - ideally data scientists and analysists inside the company. In order to guarantee the best data user experience, a Data Product should meet some characteristics:

- **Discoverable**: the DP itself should provide information to consumers and catalog, about its source of origin, owner, runtime and contributions made by the consumers, such use cases and applications.
- **Addressable**: the DP offers a permanent and unique address to the user to access it.
- **Understandable**: the DP should provide documentation of the business concept, the data, and the code to use it, so the user can understand the entities encapsulated in it, their relationships, and the adjacent DPs.
- **Trustworthy and Truthful**: the DP should communicate and guarantee an acceptable level of quality and trustworthiness, through SLOs (Service-Level Objectives) - objective measures that remove uncertainty surrounding the data – plus cleansing and running automated data integrity tests. Also, providing provenance and data lineage helps consumers gain further confidence in the DP.
- **Interoperable**: the DP should follow some standards to link data, so the user could easily correlate data across domains and stitch them together in insightful ways – through join, filter, aggregate operations.
- **Secure**: the DP contains the policies that directly validate access control in the data flow, encryption of data, confidentiality levels, data retention, regulations and agreements.
- **Natively Accessible**: the DP should be accessible by the authorized data users with their native tools, since inside a big company there is a large profile of users: analysts explore data in spreadsheets, analysts visualize data through query languages, scientists structure data to train their ML models, developers expect to pull data with APIs. This can be implemented by building multiple read adapters on the same data.

- **Valuable (on its own)**: the DP should be valuable and meaningful on its own. For example, machine optimizations such as indices or fact tables must be created by the platform and do not appear as DPs.

[4]

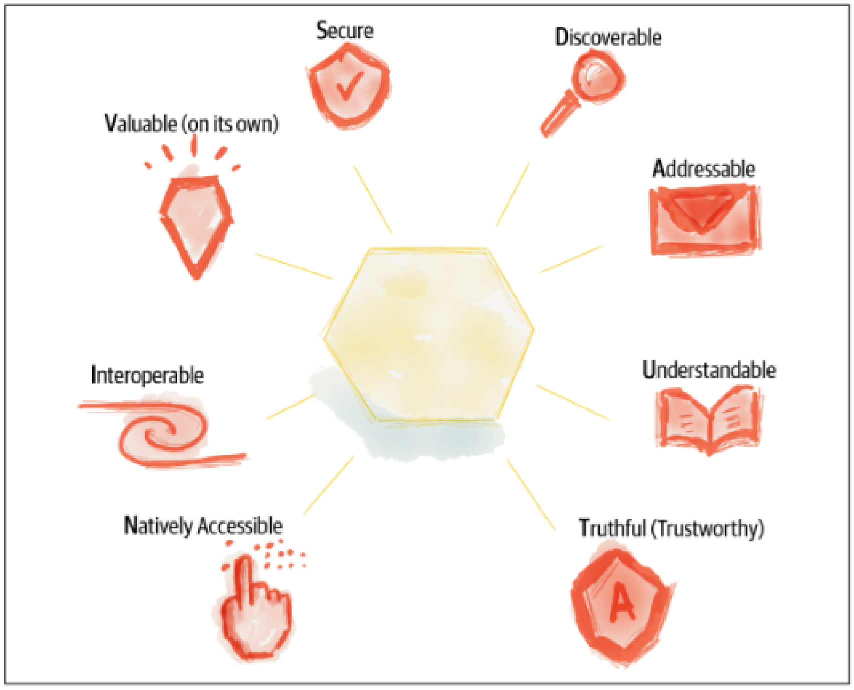


Figure 3-4: Attributes of a Data Product.

These characteristics make the Data Product directly usable, independent, self-sufficient and a federable element of the mesh. It is the architectural quantum of the Data Mesh, a single deployable unit made of code, data and policies, that is structurally complete to do its job: providing the high-quality data of a particular domain. Deploying data and code together is not something new, since it is already present in Microservices, but here the code serves the data: transforming it, maintaining its integrity, governing its policies, and serving it. This principle accentuates the responsibility given to the Data Domain Teams; for this reason, there is the need for two new roles:

1. **Data Product Developer**, who is responsible for developing, serving and maintaining the domains' Data Products for their entire lifecycles.
2. **Data Product Owner**, someone with an intimate understanding of the domain's data and its consumers, who is accountable for the success of a domain's data products in delivering value, satisfying and growing the data users.

3.2.3 Self-Serve Data Platform

As explained in the previous chapter, existing platforms focus on centralization: centralized pipelines orchestration tools, centralized catalogs, centralized warehouse schema, centralized allocation of compute/storage resources, and so on. These solutions give too many responsibilities to a few teams, and they become a bottleneck that slows down the speed of change. From here the need for a decentralized view, that is introduced with the first two principles. However such principles create a risk of duplication of efforts in each domain, increasing costs and inconsistencies. For this reason, the third principle focuses on the development and provisioning of a self-serve data platform, where to store, compute, and exchange data: teams can share their refined and structured data in the form of products with the rest of the organization and the users can easily access and analyze them. Since the Data Product is a self-contained entity - encapsulating data (with metadata), code (with pipelines), policy, and infrastructure dependencies - it can be easily shared with the rest of the company and gives sense of the effort produced by the data domain team which built it. This platform is designed to be user-friendly, so that the consumer can easily extract insight from data through data visualization tools and machine learning algorithms. To understand how to structure this system, it is important to understand what are the focuses of the two main involved roles.

1. **Data Product Developers.** Their focus is the creation of the data product, so the platform must implement all the necessary capabilities to allow him to build, test, deploy, secure, and maintain a data product through continuous delivery, without worrying about the underlying infrastructure provisioning (storage, compute, accounts, . . .). In addition, the platform should abstract complexity: developers should be able to express their domain-agnostic wants without specifying the

low level implementation details; for example, the platform will take care to create the data structures, provision the storage, perform the encryption. A nice way to abstract complexity is through declarative modeling of the target state, such as:

- Container orchestrators, like Kubernetes;
- Infrastructure provisioning tools, like Terraform;
- Query language, like SQL.

Other processes, like data verification, can be automated through Machine Learning techniques.

2. **Data Product Users.** Their focus is to discover, access, and explore data products, so the platform should work as an organizational data marketplace: exchanging value between data products (on the mesh) but also end products (at the edge of the mesh), such as ML models, reports, dashboards and DPs [4]. The platform should ease the user work: access to data, for example, should be automatically evaluated by the platform, without asking permission to the governance team; continuously processing new data to keep the DPs up to date, useful for users working on ML models. Also, the platform should provide a standardized way of identifying, addressing, and connecting data products: creating a mesh of heterogeneous domains with homogeneous interfaces. In this way it's also possible to achieve interoperability with external platforms and "scale out data sharing". The implementation of APIs inside the platform is important, since both the Developers and the Users can deal with the Data Product as an object.

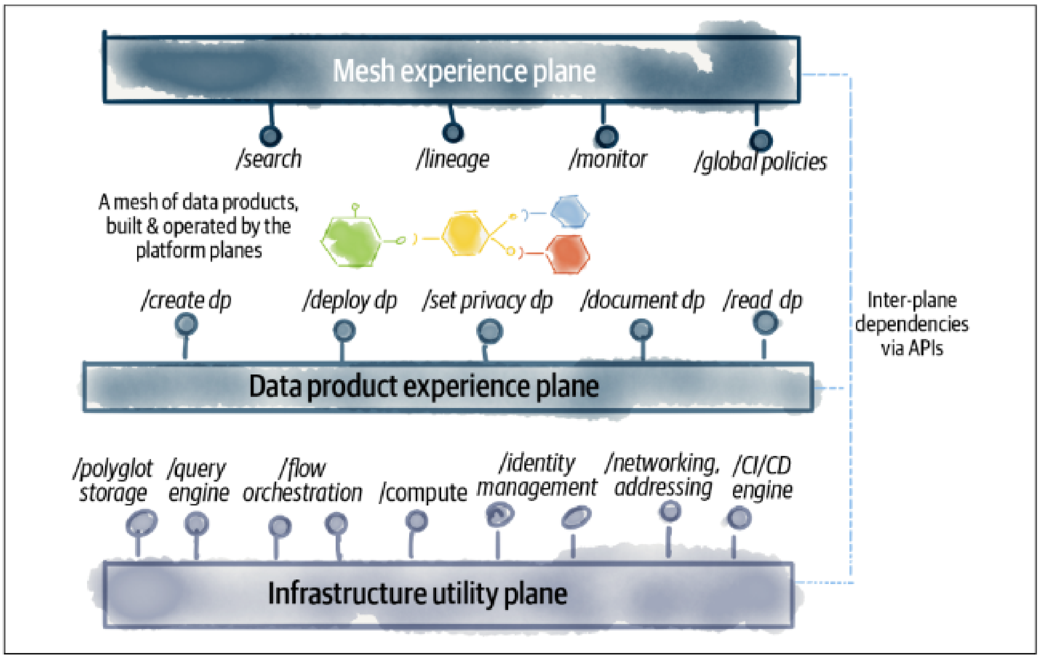


Figure 3-5: Multiple planes of the self-serve data platform.

3.2.4 Federated Computational Governance

The fourth and last principle tries to solve the problem of making sure that any data product complies with a set of common policies, without centralizing all responsibilities on a single team - which has been seen to be a bottleneck in all architectures. Traditionally, governance relied heavily on manual interventions, complex central processes of data validation and certification, and established global canonical modeling of data with minimal support for change, often engaged too late after the fact [4]. Data Mesh, in contrast, embraces change over constancy: change from the continuous arrival of fresh data, change of data models, rapid change in use cases and users of the data, new data products being created, and old data products being retired [4]. Still, there is the need for global standards to grant a certain level of security, legal conformance, interoperability standards and other policies applied to all data products, but without sacrificing local optimization. For achieving this it is necessary to organize the Mesh as a Federation: every domain is a division owning and controlling a certain number of data products. To also fulfill local optimization, the domain needs to have autonomy over modeling and serving Data Products (as a consequence of the first principle) and over the application of most of the policies like self-registration, observability, and discoverability capabilities. However, there are a set of standards and global policies – defined by the core team of the Federation – that all domains must adhere to as a prerequisite to be a member of the Mesh. In this way, it is possible to grant interoperability, security and a coherent experience of the Data Products. These are a small set of cross-functional policies, such as interconnectivity and data ownership, that must be defined globally, since they have an impact over the whole Mesh. This core is a Federated Team that takes the responsibility of deciding:

- what policies must be implemented by all data products;
- how the platform must support these policies computationally;
- how data products adopt the policies.

Given the different know-how these tasks require, the team must be cross-functional, composed by figures like:

- Domain Representatives, contributing to the definition of policies that govern the data products;
- Data Platform Representatives, designing the experience and implementation of computational policy configuration and execution;
- Subject Matter Experts, influencing prioritization and design of platform features, having the knowledge about security, compliance and legal concerns;
- Facilitators and Managers, supporting the process of governance under the federated and computational model.

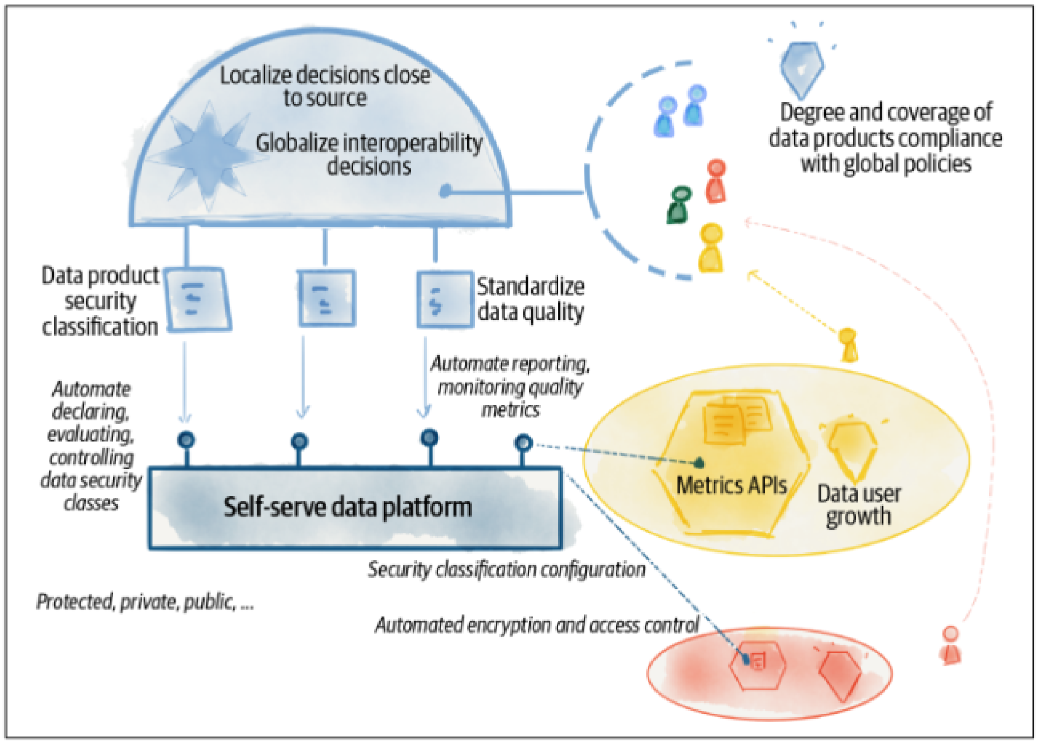

Localize decisions close to source
Globalize interoperability decisions
Degree and coverage of data products compliance with global policies
Data product security classification
Standardize data quality
Automate reporting, monitoring quality metrics
Automate declaring, evaluating, controlling data security classes
Self-serve data platform
Metrics APIs
Data user growth
Security classification configuration
Protected, private, public, ...
Automated encryption and access control

Figure 3-6: Example of data mesh governance operating model.

The last, but difficult, goal is to make the governance function as invisible as possible, automating it through the platform, that can manage the life cycle of the Data Products and embed the execution of the policies. There are different ways to implement governance policies computationally:

- Standard as Code, for standards categories that are expected to be implemented in a consistent way across all data products, for example: APIs to expose and discover data; modeling semantics and syntax of the data and the query language; modeling lineage (traces of data flow and operations across connected DPs).
- Policies as Code, for embedding policies like compliance, access control, access audit, and privacy in all the Data Products.
- Automated Tests and Monitoring, to make sure that the Data Product

complies with its guarantees in terms of data quality, integrity, global policies, and rapidly detected errors (could be set in CI/CD pipelines).

As we have seen, the principles are closely interrelated: the Data as a Product stems from the need to avoid data siloing caused by the decentralization introduced in the Domain Ownership; the Self-Serve Data Platform gives meaning to the efforts of domain teams who see themselves with increased responsibilities because of the first two principles; the Federated Computational Governance guarantees a central core inside the architecture, which must define global policies concerning quality and security that all Data Products should adhere to be part of the Mesh.

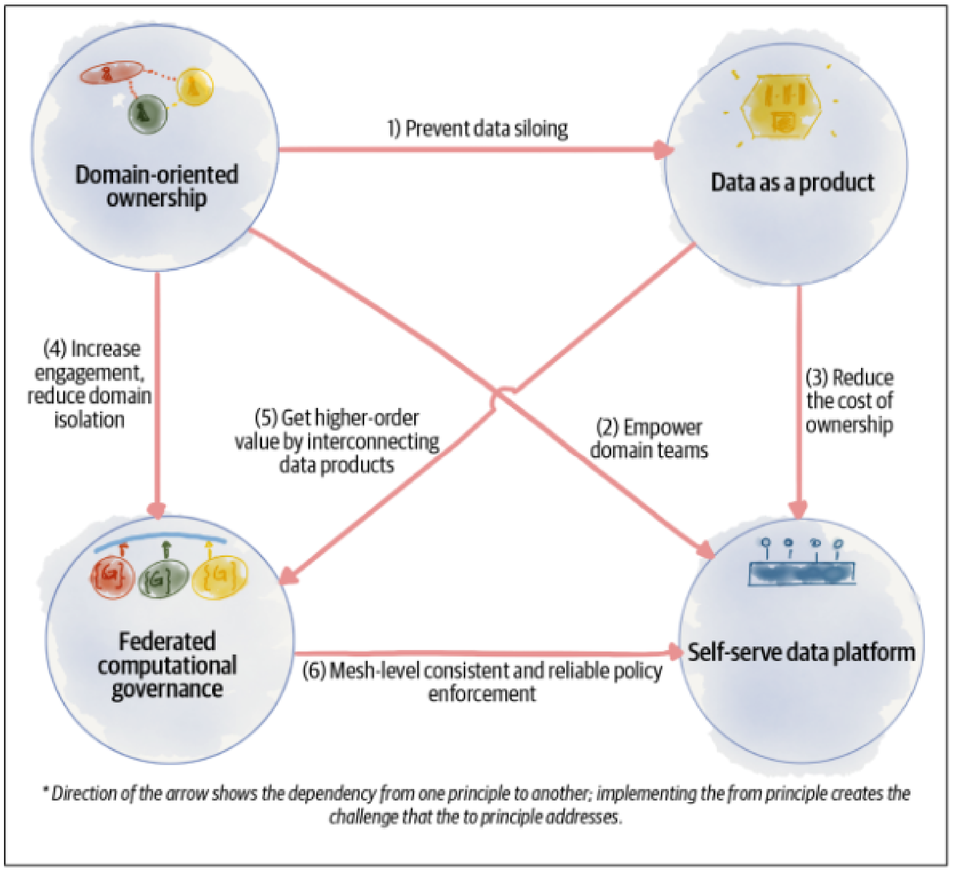

Domain-oriented ownership
1) Prevent data siloing
Data as a product
(4) Increase engagement, reduce domain isolation
(5) Get higher-order value by interconnecting data products
(2) Empower domain teams
(3) Reduce the cost of ownership
Federated computational governance
(6) Mesh-level consistent and reliable policy enforcement
Self-serve data platform
* Direction of the arrow shows the dependency from one principle to another; implementing the from principle creates the challenge that the to principle addresses.

Figure 3-7: Four principles of Data Mesh and their interplay.

3.3 When to Mesh?

Does it always make sense moving to a Data Mesh architecture? As described in this chapter, the Data Mesh appears to be a rather complex system and the transformation from current systems to this type of architecture can require more than two years, since it involves:

- a good study of the company's businesses and their division into domains and subdomains;
- creation of new teams and roles, assigning responsibilities;
- studying the implementation of data products, an efficient exchange platform, and an infrastructure that allows for the smoothest experience as well as including all the appropriate standards to make it interoperable and secure (policies management).

For this reason, the Data Mesh architecture is not a good solution for all companies, rather only for large ones that must deal with a wide variety of data, produced by different types of businesses, or for companies involved in a digital native medium. How to figure out whether it makes sense to implement this new architecture? There are several factors to consider:

- Number of data sources;
- Size of the team(s) working on the data;
- Number of data domains: how many functional teams (marketing, sales, operations) rely on different data sources to make decisions? Do teams struggle to collaborate effectively?
- How often does the data engineering team turn out to be a bottleneck?
- How much priority is given to Data Governance? [18]

These questions give an indication of how complex and challenging the requirements of the desired data infrastructure are: if the structure is very complex, then it means that many teams work with a large amount of data sources and need to experiment with the data (i.e., transform it at very high speeds), which is one reason why the Data Mesh application might be a good evolution of the system.

Chapter 4

A Data Mesh Implementation

This chapter is about the Data Mesh project in Bip, an overview of the implementation and the tools used. The goal of the project is to design and develop a self-service data platform for the development, cataloging, and consumption of Data Products in a federated fashion, while exploiting automation to govern its use. Such infrastructure will enable non-technical end-users to develop new, trustworthy, and high-quality data products that are ergonomic, business relevant and improve time and quality of business decisions. Learnings from this project can be applied to current architectures, to apply best practices as:

- Democratization of the data product development and deployment.
- Automated generation and ingestion of Data Products' metadata into data catalogs.
- Improved discoverability and observability of Data Products.
- Transparency and monitoring in usage and adoption.

Such best practices are best tailored to complex, multi-domain environments in which many and different data professionals work on polyhedric data, for example, the production of global reporting for the top management. The implementation of this architecture has been designed on tools – mostly open-sources – in order to grant the four principles outlined in the previous chapter and to ensure the following six characteristics that have been identified as essential to make a Data Product as a viable element of the Mesh:

1. **Self-contained**: it should be able to run without requiring external tooling or infrastructure provisioning.

2. **Stateless**: compute and storage should be separated from the Data Product runtime, to allow for versioning and resilience.

3. **Self-descriptive**: any metadata about the product should be served by the Data Product itself through API.

4. **Federable**: the Data Product should execute in concert with other DPs, in autonomy and without any centrally defined DAG.

5. **Monitorable**: execution, access and adoption to federated products is to be monitored under standard governance.

6. **Addressable**: the Data Product should serve its produced data through protocols, with an option for smart defaults.

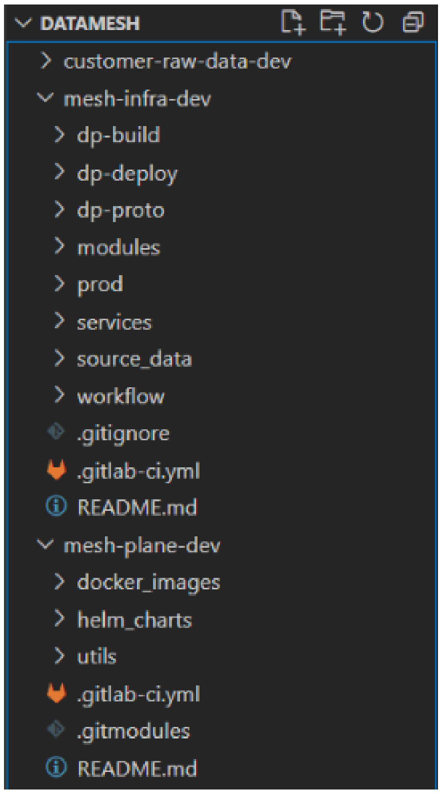
4.1 Architecture development

4.1.1 An overview of the project

Two are the main repositories with the necessary configuration for the architecture:

1. **mesh-infra**: containing all the code necessary for building the ground-level infrastructure, GitLab pipelines for deploying modules and services, containers' specifications, project's variables and the workflows to build a Data Product.

2. **mesh-plane**: containing the "blueprints" for cross Data Products – Docker images and Helm charts – and an implementation of Fast API.



```
DATAMESH
  customer-raw-data-dev
  mesh-infra-dev
    dp-build
    dp-deploy
    dp-proto
    modules
    prod
    services
    source_data
    workflow
    .gitignore
    .gitlab-ci.yml
    README.md
  mesh-plane-dev
    docker_images
    helm_charts
    utils
    .gitlab-ci.yml
    .gitmodules
    README.md
```

Figure 4-1: Structure of the project.

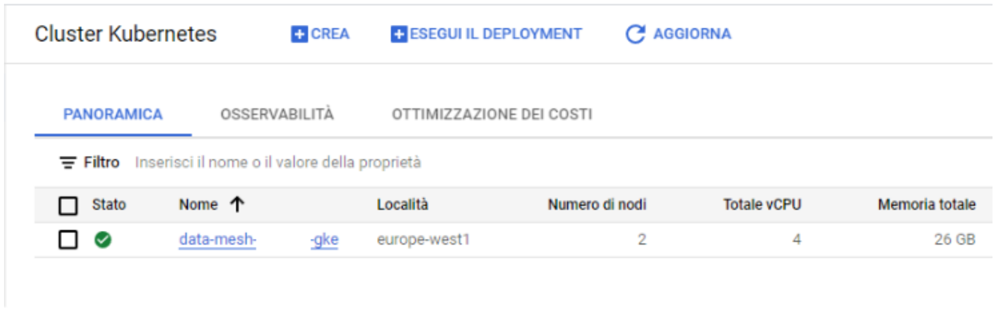
The other repositories are Data Products in the form of dbt projects.

4.1.2 Kubernetes

The infrastructure, with all its services, is deployed on Google Kubernetes Engine, which enables the creation of clusters where Kubernetes applications can run. A cluster is made up of:

- machines, called nodes, which run the services supporting the containers that make up the workloads;
- a control plan, which decides what runs on those nodes, including scheduling and scaling [19].

The cluster will contain all the services needed for building, discovering, accessing, and exploring the Data Products.



Stato	Nome	Località	Numero di nodi	Totale vCPU	Memoria totale
✓	data-mesh- -gke	europe-west1	2	4	26 GB

Figure 4-2: Data Mesh cluster on GCP.

Kubernetes (K8s) is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates both declarative configuration and automation [20]. It enables the deployment of Pods: an environment for running a group of one or more containers, with shared storage and network resources, and a shared context – a specification for how to run these containers. The configuration is often written in YAML.

4.1.3 YAML

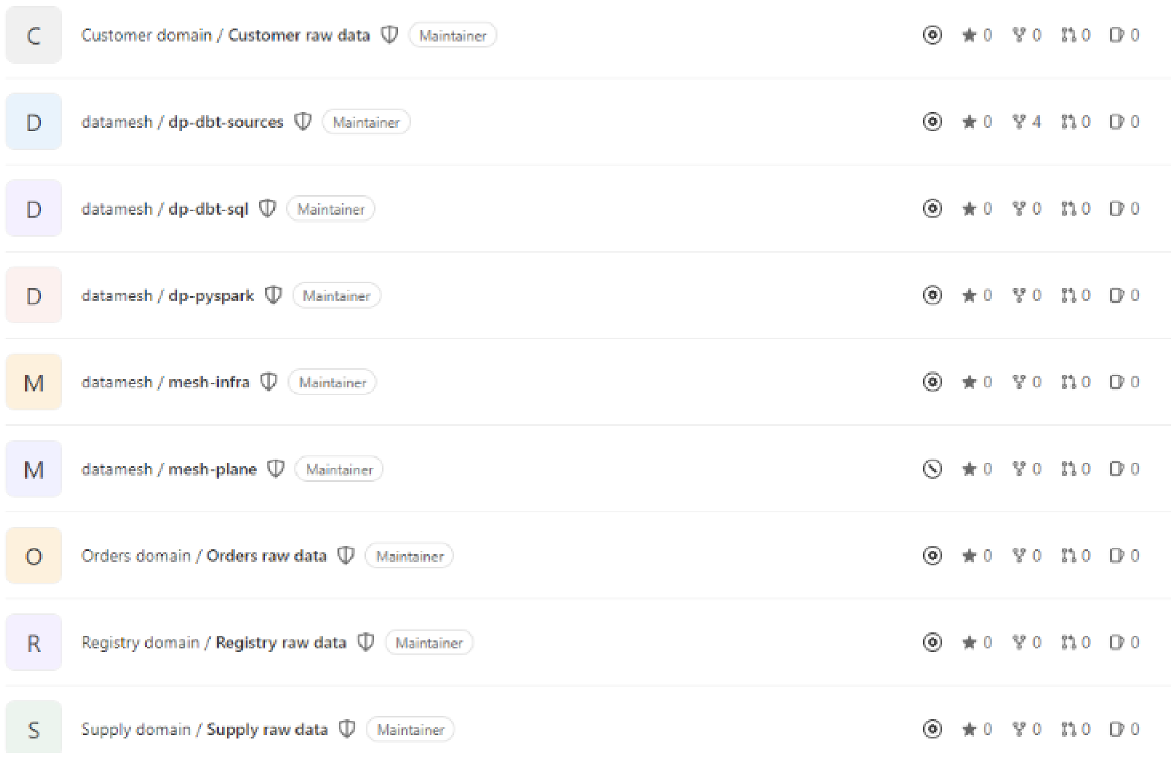
YAML - Yet Another Markup Language - is a human-readable data-serialization language, commonly used for configuration files in applications where data is being stored or transmitted [21]. Right now, it is very popular in the DevOps domain, and it is used for some tools present in this architecture, such as Kubernetes and Terraform.

4.1.4 Terraform

Terraform is an open-source infrastructure-as-code software tool created by HashiCorp. Users define and provide data center infrastructures using a declarative configuration and human-readable language known as HashiCorp Configuration Language (HCL), or optionally JSON [22]. In this project, it has been used to manage Kubernetes resources. As seen in the third chapter, Terraform and Kubernetes are useful tools for abstracting complexity while building a platform and orchestrating containers, since they use a declarative modeling of the target state. The Infrastructure as Code can be written as modules, promoting reusability and maintainability, important concepts for the implementation of Data as a Product and Self-Serve Data Platform principles and through these we can achieve Self-contained data products.

4.1.5 GitLab

GitLab is an open-source platform published in 2011, and it has been used in this project because it allows management of Git repositories, and so is very useful for working in teams on different features, but also has Continuous Integration/Continuous Delivery (CI/CD) and DevOps workflows built in.



```
C  Customer domain / Customer raw data   Maintainer   ★ 0  ⑂ 0  0  0
D  datamesh / dp-dbt-sources             Maintainer   ★ 0  ⑂ 4  0  0
D  datamesh / dp-dbt-sql                 Maintainer   ★ 0  ⑂ 0  0  0
D  datamesh / dp-pyspark                 Maintainer   ★ 0  ⑂ 0  0  0
M  datamesh / mesh-infra                 Maintainer   ★ 0  ⑂ 0  0  0
M  datamesh / mesh-plane                 Maintainer   ★ 0  ⑂ 0  0  0
O  Orders domain / Orders raw data       Maintainer   ★ 0  ⑂ 0  0  0
R  Registry domain / Registry raw data   Maintainer   ★ 0  ⑂ 0  0  0
S  Supply domain / Supply raw data       Maintainer   ★ 0  ⑂ 0  0  0
```

Figure 4-3: The repositories at the end of the internship.

Implementing CI/CD in a project means continuously building, testing and deploying iterative code changes. This iterative process helps reduce the chance of developing new code based on buggy or failed previous versions.

This method strives to have less human intervention or even no intervention at all, from the development of new code until its deployment [23]. Pipelines have been implemented in order to

1. **Build the platform**:
    a. A pipeline to create the cluster through Google Kubernetes Engine and loading the workflows for the Data Products in Argo Workflows.
    b. Pipelines for deploying the other services, such as DataHub for discoverability of the DPs; Dagster and Grafana for monitoring and Data Governance; Trino and Superset for virtualization and consumption.
    c. A pipeline for destroying all the deployments and the cluster.
2. **Build the images**:
    a. Docker Images, a read-only template with instructions for creating a Docker container.
    b. Helm Charts, a collection of files that describe a related set of Kubernetes resources.
    c. Argo Workflows, configuration files for CI/CD pipeline management.

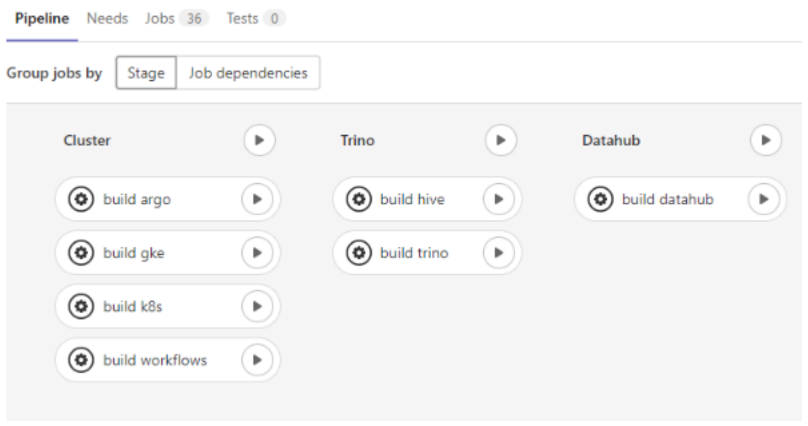


Figure 4-4: Pipelines in GitLab.

The pipelines are defined inside the repository in a file called 'gitlab-ci.yml'; this file contains the default values for job keywords, the configuration from other YAML files (include), the names and order of the pipeline stages, the variables and workflows (what type of pipeline run). GitLab allows to give a company a strong control over its infrastructure, building a CI/CD foundation in which to script lifecycle events (build, deployment, test, stop) and automatable tasks such as configuration and documentation, enforcing standards, and best practices. Since the Data Product should be stateless and balancing simplicity for user and full autonomy required by the platform, Bip experimentation suggests introducing another layer, decoupled from the CI/CD one, that grants a certain degree of freedom to the end user through a Data Product Control API, created at the beginning of its lifecycle. This represents a single point of contact with the Data Product, one that accepts lifecycle requests and can address them with some degree of self-awareness. As recommended in the principle of Self-Serve Data Platform, it automates most data engineering tasks, freeing the user from learning the job. This is achieved through Argo Workflows and Fast API.

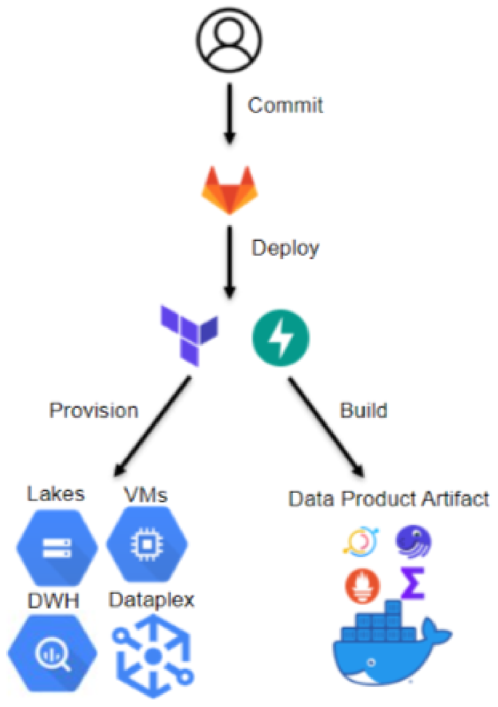


Figure 4-5: Two decoupling layers.

4.1.6 Argo Workflows

Argo Workflows is an open-source container-native workflow engine for orchestrating parallel jobs on Kubernetes. It is implemented as a Kubernetes CRD (Custom Resource Definition), allowing to:

- Define workflows (with YAML files) where each step is a container.
- Model workflows as a sequence of tasks or capture the dependencies between tasks using a directed acyclic graph (DAG).
- Run CI/CD pipelines natively on Kubernetes [24].

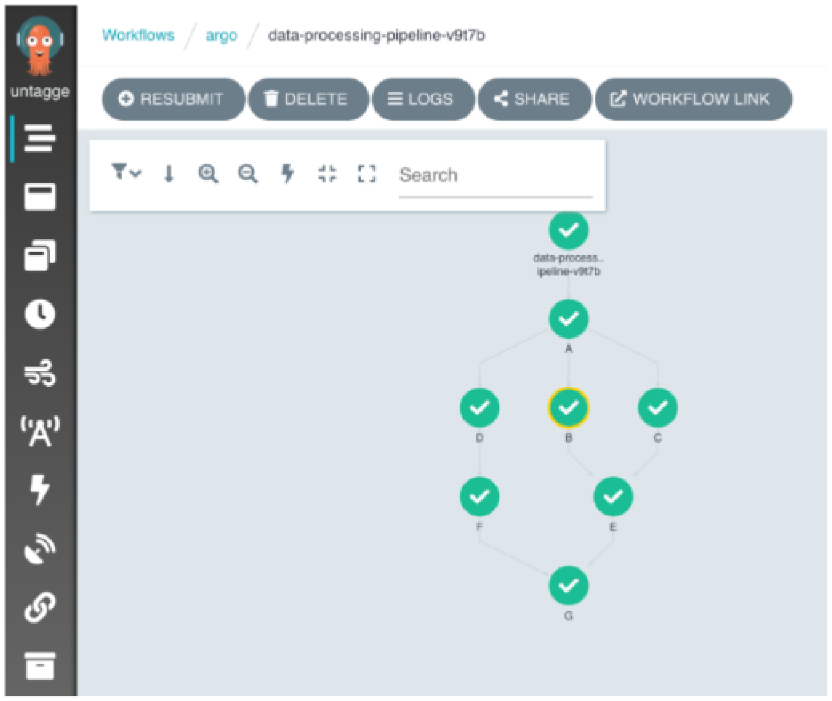


Figure 4-6: Example of an Argo workflow interface.

It was decided to use Argo to manage the lifecycle of the Data Product.

4.1.7 Fast API

Fast API is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints [25]. The use of APIs is strongly recommended for building an effective Self-Serve Data Platform, as their common scope is to simplify programming by abstracting the underlying implementation and only exposing objects or actions the developer needs [26], and it is a good way to simplify the life of the data user. In this implementation, it facilitates interaction with Argo, and can be extended to other tools.

4.1.8 Compute and Storage

Compute can be any technology required by the domain (DWH, analytics engines, streaming, ...), both ephemeral or always-on. For storage, it was decided to follow a Data Lake or Data Lakehouse pattern, because raw and unstructured data are both present. Since this first implementation is designed on GCP services, the computing is based on Big Query, while the storage is based on Google Cloud Services.

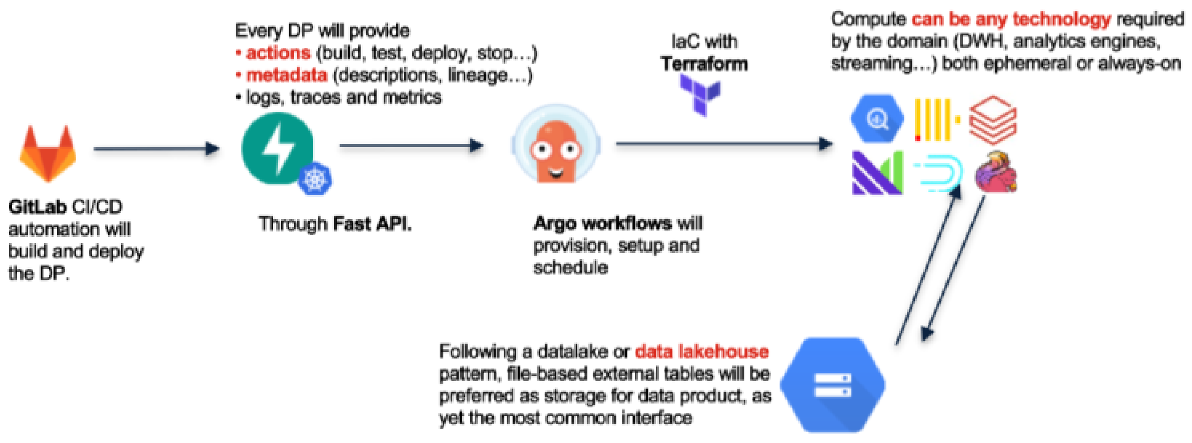

Every DP will provide
• actions (build, test, deploy, stop…)
• metadata (descriptions, lineage…)
• logs, traces and metrics
IaC with Terraform
Compute can be any technology required by the domain (DWH, analytics engines, streaming…) both ephemeral or always-on
GitLab CI/CD automation will build and deploy the DP.
Through Fast API.
Argo workflows will provision, setup and schedule
Following a datalake or data lakehouse pattern, file-based external tables will be preferred as storage for data product, as yet the most common interface

Figure 4-7: How the tools described work together.

4.2 User Experience of the Data Product

4.2.1 dbt - Building the Data Product

Data Build Tool is an open-source command line tool that helps analysts and engineers transform data in their warehouses more effectively, by writing select statements into tables and views [27]. It enables anyone who knows SQL to build production-grade data pipelines, clean and prepare data for analytics purposes.

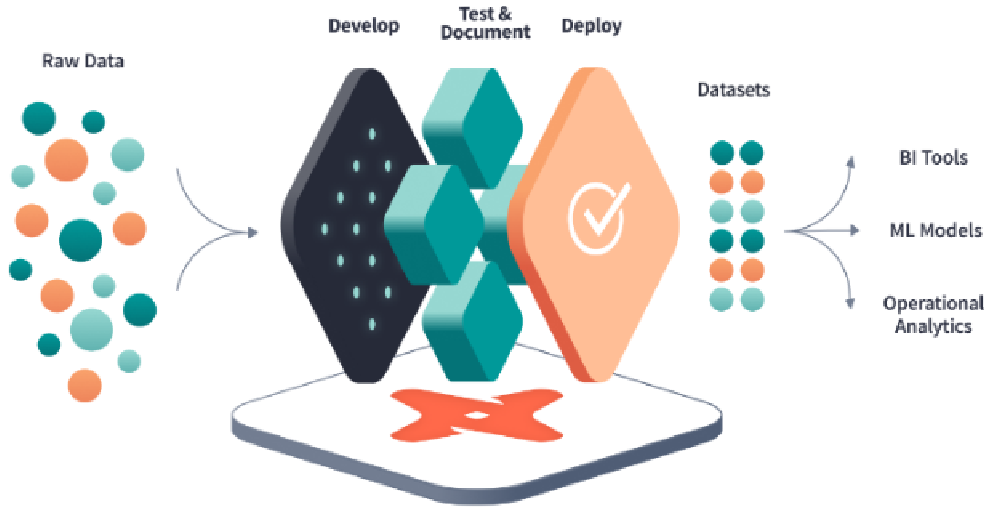
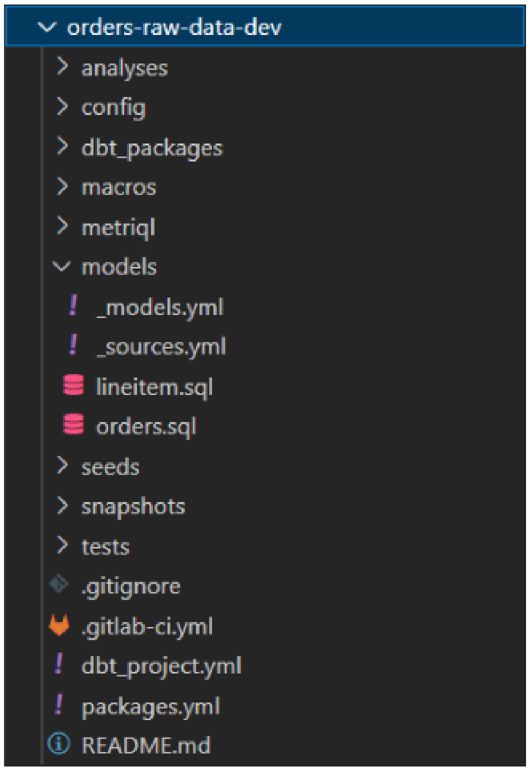


Figure 4-8: dbt capabilities.

The choice to implement dbt in this Data Mesh experiment is because users can define a Data Product by requiring as a prerequisite only knowledge of SQL, which is now recognized in the data world as a "lingua franca". Here is the example of a Data Product built as a dbt project: a collection of SQL and YAML files, that informs dbt about the context and how to transform the data and build the data sets. By design, dbt applies the top-level structure of a dbt project, such as the 'dbt_project.yml' file, the 'models' directory, the 'snapshots' directory, and so on [28].

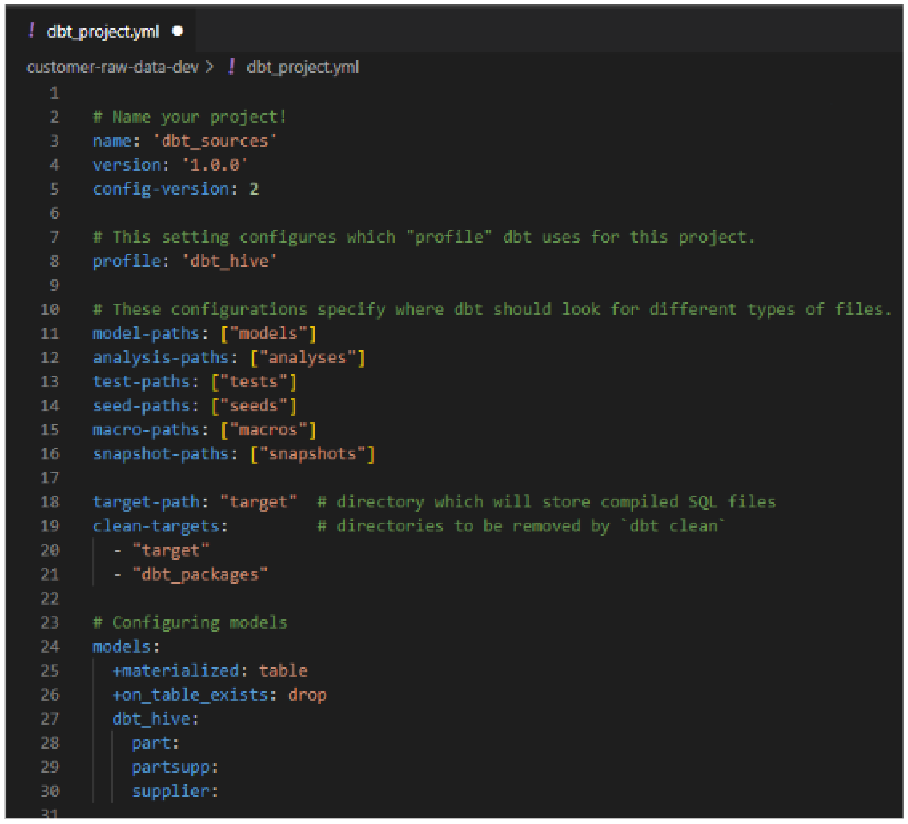


```
orders-raw-data-dev
  analyses
  config
  dbt_packages
  macros
  metriql
  models
    _models.yml
    _sources.yml
    lineitem.sql
    orders.sql
  seeds
  snapshots
  tests
  .gitignore
  .gitlab-ci.yml
  dbt_project.yml
  packages.yml
  README.md
```

Figure 4-9: Repository of a Data Product as a dbt project.

The core is 'dbt_project.yml', a configuration file containing:

- project name;
- profiles that dbt uses to connect to the data platform;
- destination directory for the compiled SQL files;
- project variables to be used for data compilation.



```yaml

# Name your project!
name: 'dbt_sources'
version: '1.0.0'
config-version: 2

# This setting configures which "profile" dbt uses for this project.
profile: 'dbt_hive'

# These configurations specify where dbt should look for different types of files.
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

target-path: "target"  # directory which will store compiled SQL files
clean-targets:         # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"

# Configuring models
models:
  +materialized: table
  +on_table_exists: drop
  dbt_hive:
    part:
    partsupp:
    supplier:
```

Figure 4-10: *dbt project configuration file.*

A dbt project can contain different types of files containing blocks of reusable code (macros) or ways to define metrics, but, among others, the more important are models. Each model lives in a single file and contains logic that either transforms raw data into a dataset that is ready for analytics or, more often, is an intermediate step in such a transformation. More in detail, it contains:

- A models.yml file containing the name of the table and its columns;
- A sources.yml file containing the source of data - with name of the table, location and format;
- One or more .sql files containing the logic to transform data.

Summarizing, a user can define the Data Product via modular SQL code, then the infrastructure will deploy computing and storage needed to build and publish it on the Mesh.

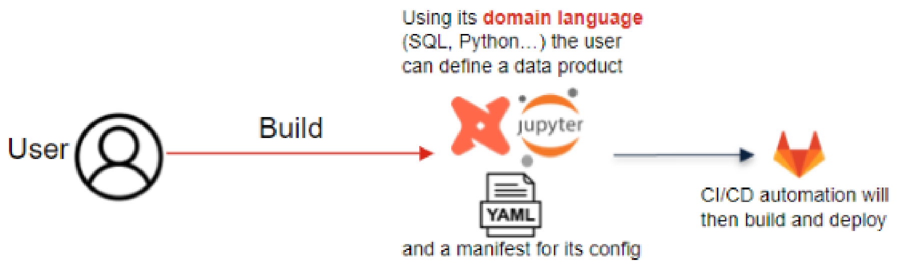

Using its domain language (SQL, Python...) the user can define a data product
User
Build
jupyter
YAML
and a manifest for its config
CI/CD automation will then build and deploy

Figure 4-11: define DP with dbt and YAML.

dbt is not the only way to build a Data Product, this architecture can be extended to more complex modeling use-cases with Python or PySpark.

4.2.2 DataHub - discovering the Data Product

DataHub is an open-source metadata platform for the modern data stack [29]. It is a modern data catalog built to enable end-to-end data discovery, data observability, and data governance. This extensible metadata platform is built for developers to tame the complexity of their rapidly evolving data ecosystems and for data practitioners to leverage the total value of data within their organization [30].

DataHub grants an optimal data user's experience through search and discovery features, like:

- Unified search of results across databases, Data Lakes, BI platforms, orchestration tools.
- End-to-end lineage across platforms, datasets, charts, . . .
- Dependencies make easier to identify which entities may be impacted by a breaking change.

DataHub has been chosen because it supports both push and pull based integrations. In this implementation DataHub extract metadatafrom dbt and expose them, allowing to build Self-descriptive Data Products.

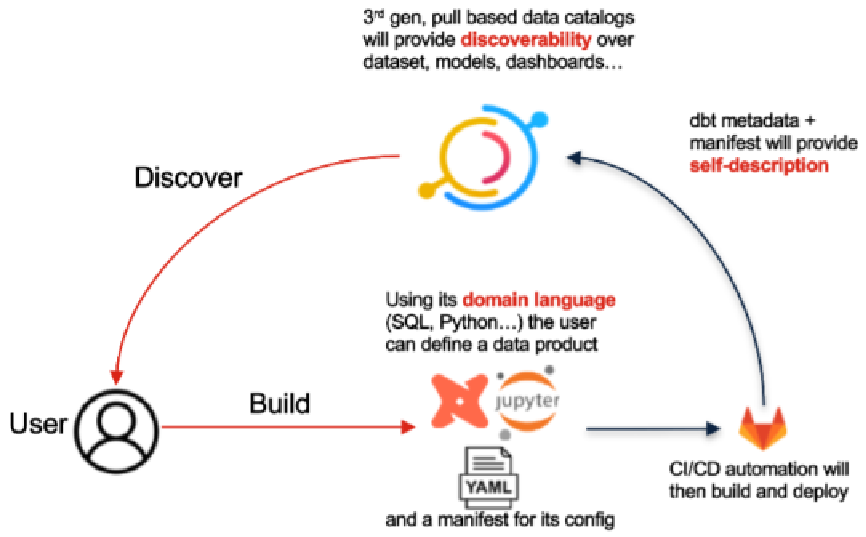

3rd gen, pull based data catalogs will provide discoverability over dataset, models, dashboards...
dbt metadata + manifest will provide self-description
Discover
Using its domain language (SQL, Python...) the user can define a data product
User
Build
jupyter
YAML
and a manifest for its config
CI/CD automation will then build and deploy

Figure 4-12: dbt + Datahub.

4.2.3 Dagster - orchestrating the Data Product

Dagster is a next-generation open-source orchestration platform for the development, production, and observation of data assets [31]. Each Data Product, aiming at full autonomy, will contain a local orchestration component (a client, with methods defined inside a Python script) to manage the execution of any internal process bound to an output port. The same component will then communicate its metadata (i.e., pipeline and trigger definitions, as well as past jobs results) to a central instance of the orchestrator. The central instance will provide a complete overview on the execution of the entire mesh, enabling monitoring and tracing over the entire data product lineage. In this way, each Data Product can be a federable element of the Mesh, as intended in the Data as a Product principle.

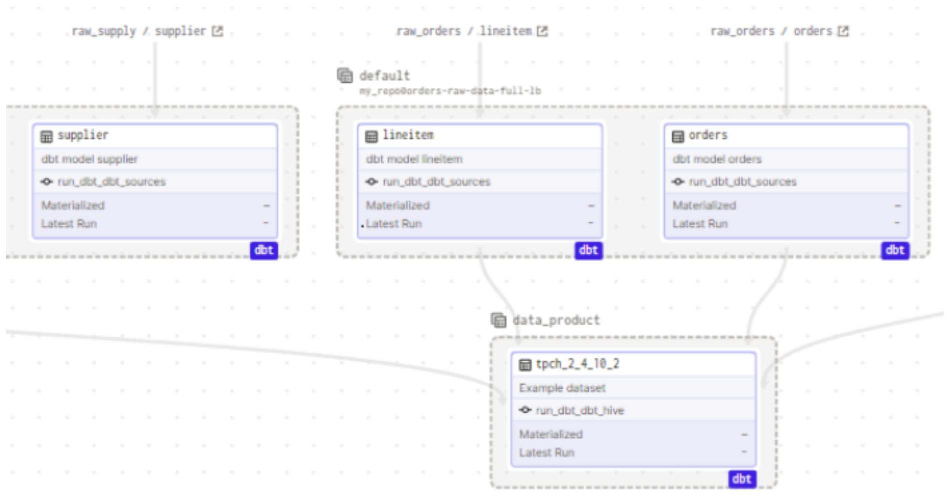


Figure 4-13: Dagster orchestration interface.

4.2.4 Grafana - monitoring the Data Product

Grafana is a multi-platform open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources [32]. It is currently a service yet to be implemented within the infrastructure. Once a Data Product has been deployed, it should make itself available to Grafana, which will monitor it and visualize the results through Prometheus, another open-source technology. This will make the Data Product monitorable.

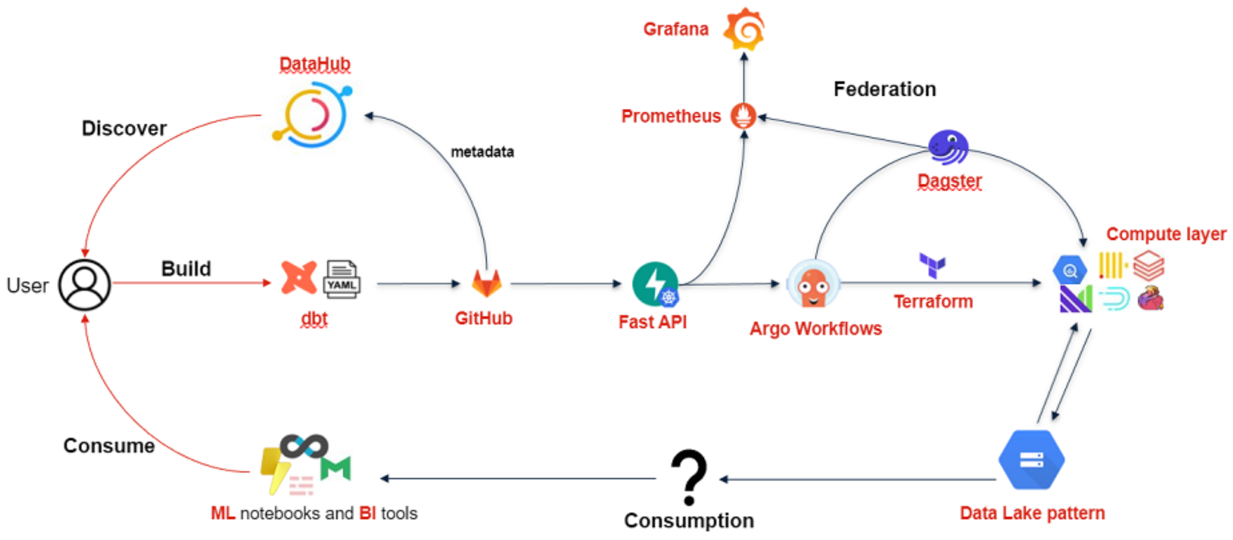


Figure 4-14: Data Mesh implementation, missing the consumption of the Data Product.

At this point of the development of the infrastructure, the only thing missing is the implementation of a tool for the consumption of Data Product, such as Microsoft PowerBI and Superset, and a virtualization tool enabling the Addressability of the Data Product. These topics will be explored in more detail in the next chapter.

Chapter 5

Data Consumption

The implementation activity as the focus of the thesis involved the deployment and testing of the consumption part within the Data Mesh architecture exposed in the previous chapter. We talk about data consumption to highlight the difference with data ingestion: the subtle difference is that the upstream data is already cleansed, processed and served ready for the user to analyze with BI tools or notebooks. The change of language creates a new cognitive framing that is more aligned with the principle of serving Data as a Product. The challenge of the Consumption is to make it federated, while maintaining the autonomy and heterogeneity of Data Products, through a unique user interface. This problem is quite complex and to solve it, these three requirements must be met:

1. Abstract away the specific backend of each Data Product;
2. Allow consumption though any access pattern;
3. Allow access according to the role of the user.

Any candidate tool must therefore provide a virtualization layer over the individual backend, both for warehouses, streaming platforms and specialized databases (i.e. document, graph, and timeseries DBs) and Data Lakes (i.e. parquet, iceberg over buckets). For an appropriate consumption, this virtualization layer must be directly accessible from the users' preferred environment and tools: SQL, Python, BI dashboards or streaming platforms. Lastly, the tool must also support user impersonation using a federated identity provider such as Google and AWS IAM or Azure AD, to avoid replication of rules and synchronization issues. In this way, the Data Product would be Addressable. The experimentation was held with two tools:

- Dataplex, which proved to be not very fit for the purpose, and will be explored further in the next chapter;
- Trino, a SQL federated compute engine also offered as SaaS.

5.1 Trino

Trino is an open-source, distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources, be Distributed Object Storages, RDBMS or NoSQL; basically, Trino is a SQL-on-Anything system. Its original name was Presto, designed and developed at Facebook.

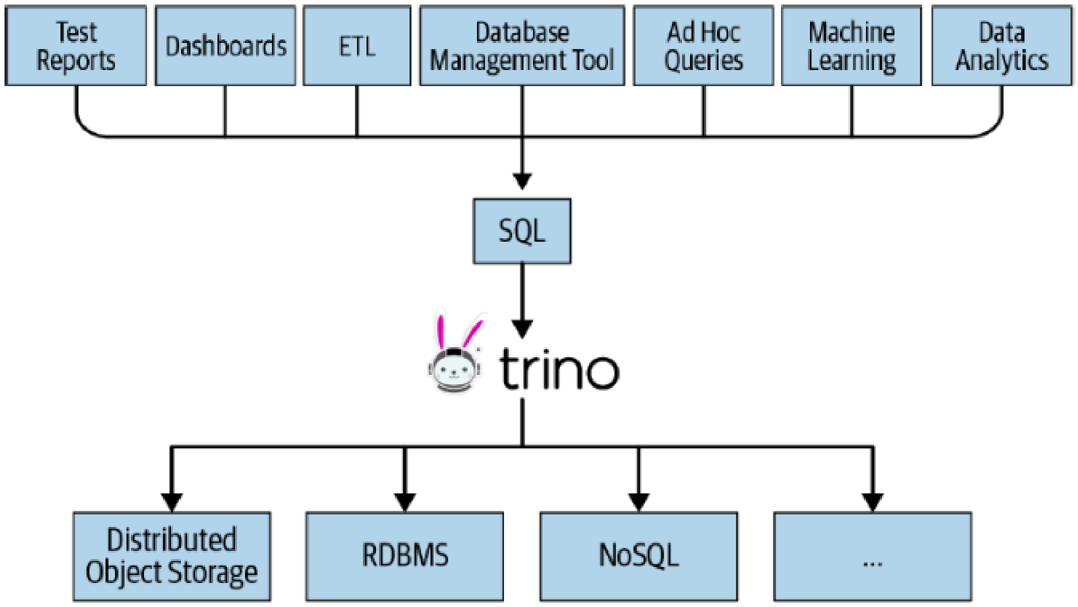


Figure 5-1: Trino is basically a SQL-on-Anything system [33].

Trino was conceived to address the problem of accessing increasingly decentralized data, a process that will continue in the coming years, with data stored all over different systems. Trino is not an operational database, since it is designed to represent the compute layer of a a data management architecture, while the underlying data sources represent the storage layer. This decoupling allows to scale up and down the compute resources for query processing basing it on analytics demand to access data, independently from storage. Trino supports a lot of different use cases: in this project the goal of using it was having one SQL Analytics Access Point, exposing all databases in one location. In this way all the BI and analytics tools can point to Trino to have access to all data in the organization. With Trino it is also possible to do a federated query, i.e. an SQL query that references and uses databases and schemas from entirely different systems in the same statement.

Trino leverages techniques for distributed query processing, like:

- In-memory parallel processing;
- Pipelined execution across nodes in the cluster;
- Multithreaded execution model to keep all the CPU cores busy;
- Efficient flat-memory data structures to minimize Java garbage collection;
- Java bytecode generation [33].

5.1.1 Trino Cluster

Trino is able to distribute all processing across a cluster of servers horizontally. A cluster is made up of multiple nodes running Trino, which are configured to collaborate with each other:

- **Coordinator**, a server that handles incoming queries and manages the workers. It is also responsible for fetching results from workers and returning the output to the client.
- **Worker**, a server responsible for executing tasks assigned by the coordinator, including retrieving data through connectors and processing them.

A discovery service runs on the coordinator: when a worker starts up, it advertises itself to this service, which registers it into the cluster and makes it available for task execution. All communication and data transfer between clients, coordinator and workers uses REST-based interactions over HTTP/HTTPS.

5.1.2 A connector-based architecture

At the heart of the separation of storage and compute in Trino is the connector-based architecture. A connector provides Trino with an interface to access an arbitrary data source. Each connector provides a table-based abstraction over the underlying data source and is used by a catalog configuration to access a specific data source. A catalog is associated with a specific connector, defines the details for accessing a data source and contains schemas – exposed by the data source – that are a way to organize tables. Each schema contains tables that provide the data as set of unordered rows, which are organized into names columns with data types.

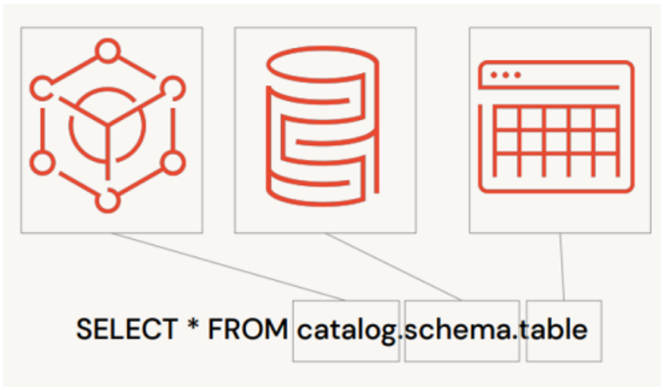


Figure 5-2: A catalog and a schema together define a set of table that can be queried.

5.1.3 Query execution model

The request for a query can be done by a user through CLI (Communication Line Interface) or by a client using the ODBC (Open DataBase Connectivity) or JDBC (Java ...) driver. The coordinator accepts a SQL statement, then parses and analyzes it to create a query plan. Using metadata (about tables, columns and data types), data statistics (row counts, table sizes), and data location, the coordinator can break up the query plan in one or more stages: each stage contains tasks that are planned and scheduled across the workers. Each task works on a unit of data called split, that is fetched by the worker though a connector, then processed and provided to the coordinator.

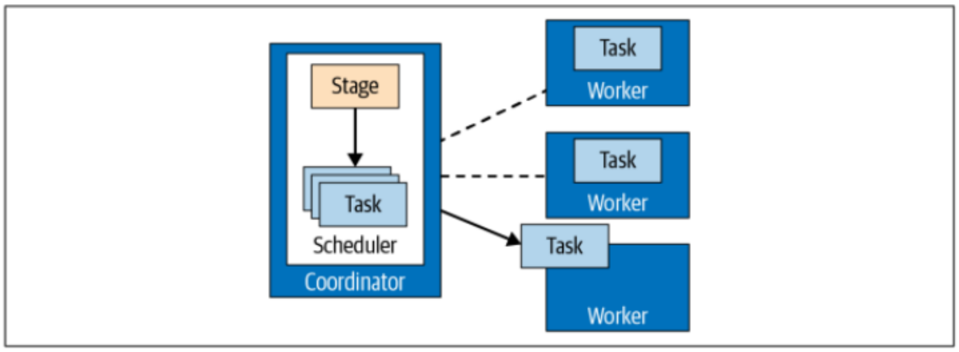


Figure 5-3: Task management performed by the coordinator [33].

5.1.4 Hive connector

Trino uses connectors to reach the data located in different storages and specifically it uses the Hive connector to read data from distributed object storages that are organized according to the rules laid out by Hive – including GCS, HDFS, Amazon S3, Azure Storage – without using the Hive runtime code [34]. To use this connector, therefore, it is not necessary to implement Hive, but the presence of Hive Metastore is required, along with an associated relational DB. Before Trino could be deployed within the infrastructure, it is necessary for these two services to be active.

5.1.5 Hive Metastore

Data is stored in a multitude of different formats, in different locations, under different access restrictions, with different structures. It is necessary to be aware of them all, so it is needed to have one place managing all the information about the data stores. Hive Metastore is this place: a service that stores metadata - like table columns, file locations, file formats - related to Apache Hive and other services, in a backend RDMBS, such as MySQL or PostgreSQL. Trino — and other clients — communicates with the Hive Metastore Thrift server to retry information about the underlying data storages.

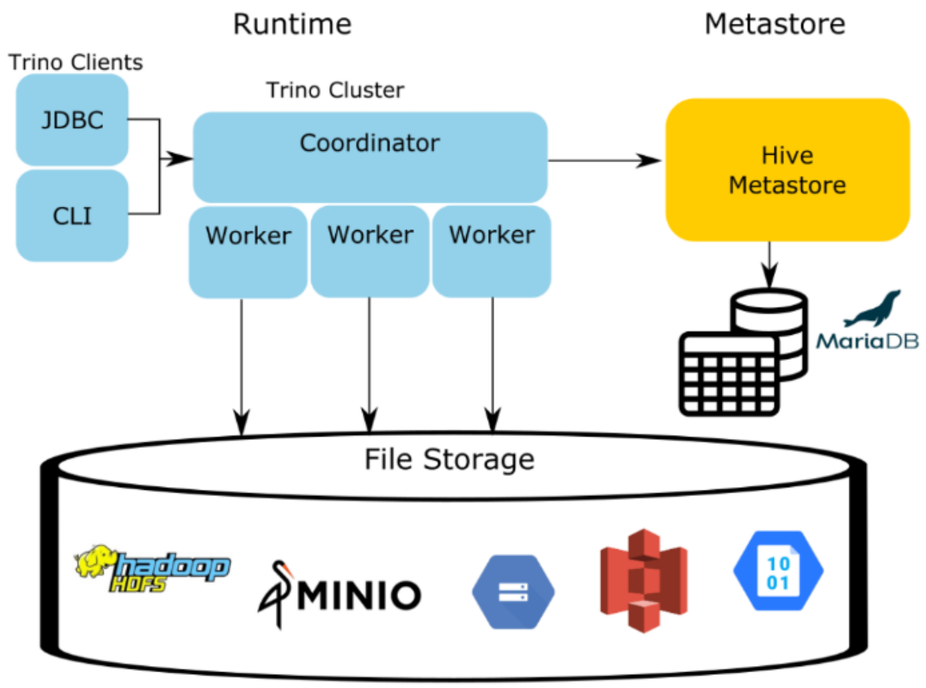
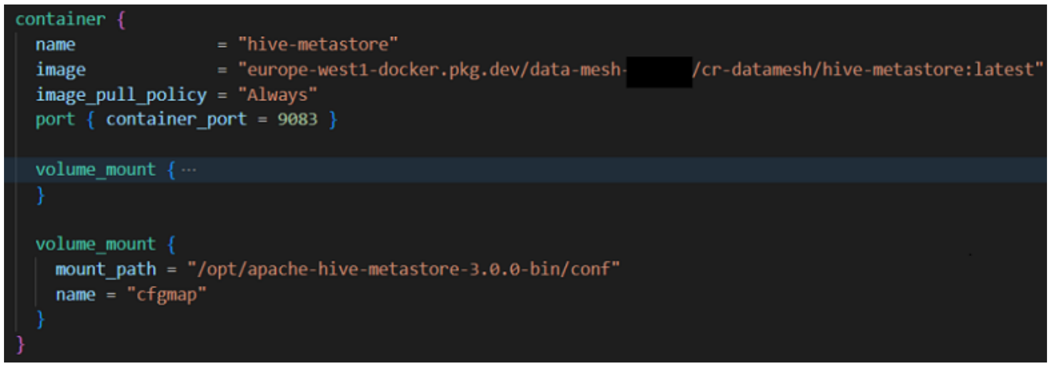


Figure 5-4: Trino + Hive Metastore architecture [34].

5.1.6 Implementation and configuration

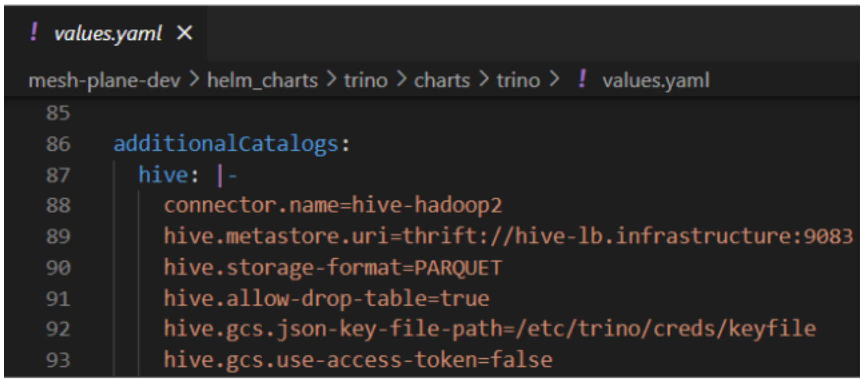
Hive Metastore and Postgres were implemented as containers via declaration in Terraform, and some configurations were defined in the 'metastore-site.xml' file, such as the Thrift URI to access the metastore, the connector to PostgreSQL (JDBC driver), any credentials (username and password), the connector to GCS, the project ID, and the secret access key.



```
container {
  name              = "hive-metastore"
  image             = "europe-west1-docker.pkg.dev/data-mesh-[redacted]/cr-datamesh/hive-metastore:latest"
  image_pull_policy = "Always"
  port { container_port = 9083 }

  volume_mount { ···
  }

  volume_mount {
    mount_path = "/opt/apache-hive-metastore-3.0.0-bin/conf"
    name = "cfgmap"
  }
}
```

Figure 5-5: Hive Metastore terraform declaration.

To implement Trino within the infrastructure, a Helm Chart has been used, that is a collection of files that describes a related set of Kubernetes resources [35]. This Chart uses ConfigMaps, an API object used to store nonconfidential data in key-value pairs, to perform configurations and define certain properties, and then deploy a chosen number of coordinators and workers. These configurations can be defined within the 'values.yaml' file. In this case, it has been necessary to define the catalogs to be used: for example, a Hive catalog, defining the Hive connector, the HMS URI and the key to access GCS.



! values.yaml ×
mesh-plane-dev > helm_charts > trino > charts > trino > ! values.yaml

```

additionalCatalogs:
  hive: |-
    connector.name=hive-hadoop2
    hive.metastore.uri=thrift://hive-lb.infrastructure:9083
    hive.storage-format=PARQUET
    hive.allow-drop-table=true
    hive.gcs.json-key-file-path=/etc/trino/creds/keyfile
    hive.gcs.use-access-token=false
```

Figure 5-6: Configuration of catalogs for Trino.

5.1.7 Compute Layer + Consumption

Trino proved useful not only to serve as a storage access point for various consumer tools, but it has also been used as the execution engine for the SQL statements defined with dbt, making it possible to define, within the Data Product, its own catalog, schemas, and tables, automating work that would have had to be done manually by the user through Trino's CLI and reinforcing the concept of self-description. Also, Trino and dbt are complementary when one needs to access different sources from a single SQL query [36]. Although this functionality is not currently exploited within the implementation (using only GCS as the storage service), it is possible to define a Data Product with dbt with values from different storages, and the query is made possible by the Trino engine. The federated query, a feature of Trino, references and uses databases and schemas from entirely different systems in the same statement.

5.1.8 Apache Superset

After implementing Trino, it was sufficient to implement a service to connect to it and consume the data. It was decided to deploy Superset as the first BI tool because it is open-source, is widely used, and supports Trino. Superset is fast, lightweight, intuitive, and loaded with options that make it easy for users of all skill sets to explore and visualize their data, from simple line charts to highly detailed geospatial charts [37]. In order to implement Superset, it has been used a Helm Chart.

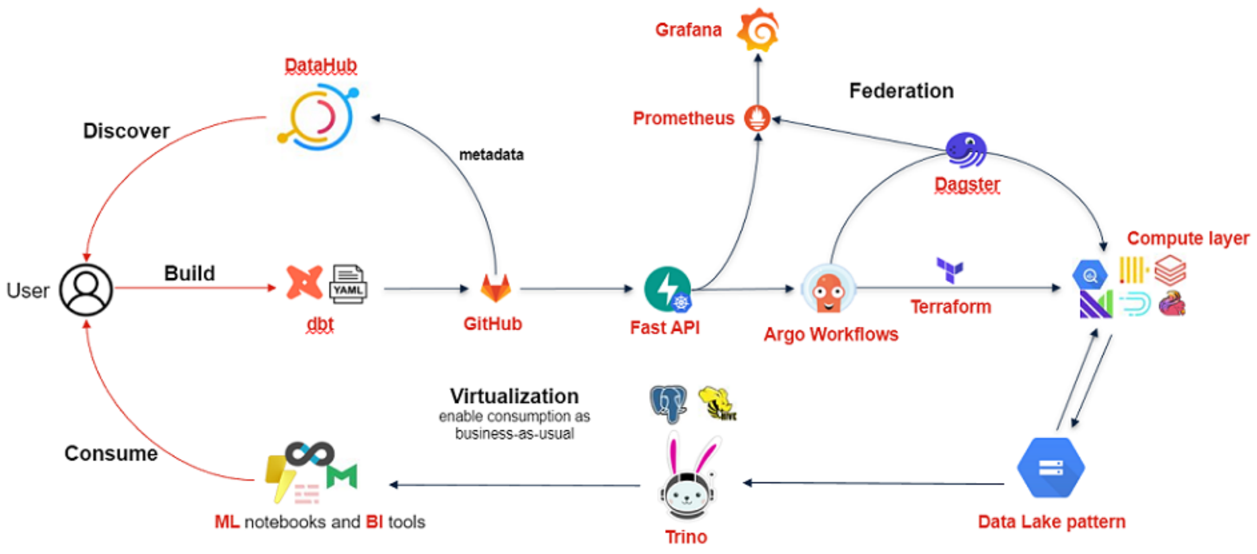
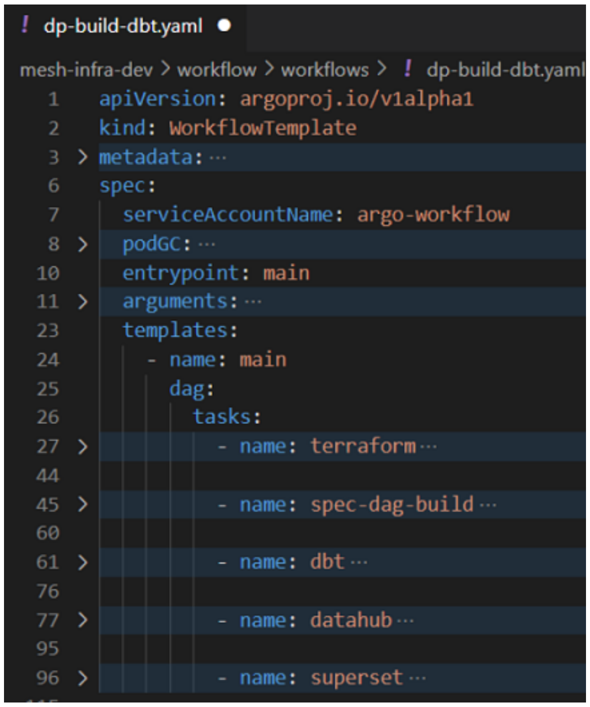


Figure 5-7: Full Data Mesh architecture with Trino.

5.1.9 Build of a Data Product

After the implementation of Trino, Hive Metastore, and Superset, the last part of the work for this project involved the development of an Argo Workflow for the creation and deployment of a Data Product that would be automatically linked to Trino and Superset via the Rest API. The build of a Data Product defined through dbt is assigned to an Argo Workflow, composed of many tasks.



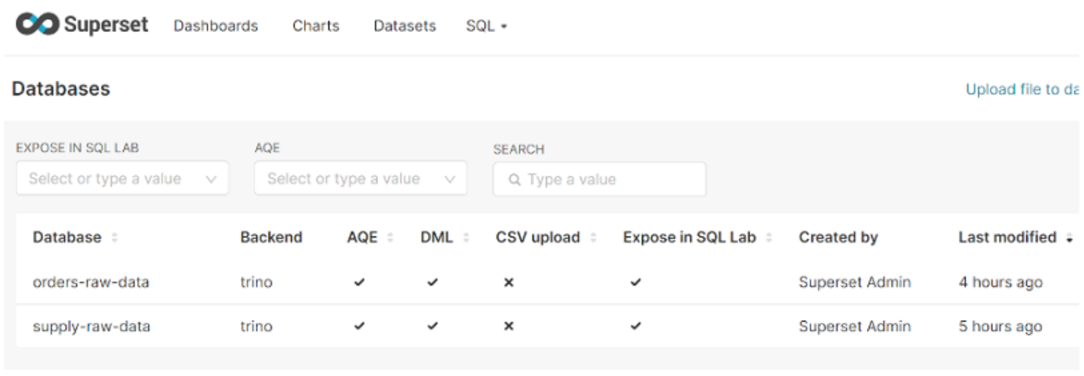
```
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate
metadata: ···
spec:
  serviceAccountName: argo-workflow
  podGC: ···
  entrypoint: main
  arguments: ···
  templates:
    - name: main
      dag:
        tasks:
          - name: terraform ···

          - name: spec-dag-build ···

          - name: dbt ···

          - name: datahub ···

          - name: superset ···
```

Figure 5-8: dp-build-dbt tasks.

The Terraform task is responsible for the plan and apply of a terraformed environment, which builds any needed infrastructure. The Spec-Dag-Build is a docker-in-docker task, allowing the client side of Dagster present in the Data Product to have it orchestrated. It builds a self-contained, fully functional data product image as artifact. The dbt task connects to gcs, executes .sql files, and generates documents and other information needed by Mesh. The Datahub task does ingestion of the metadata generated by the precedent task. The Superset task is itself composed of three tasks:

- **Login**: via API Rest, logs in to Superset and saves the access token, which is needed in subsequent tasks - username and password also distinguish access rules.

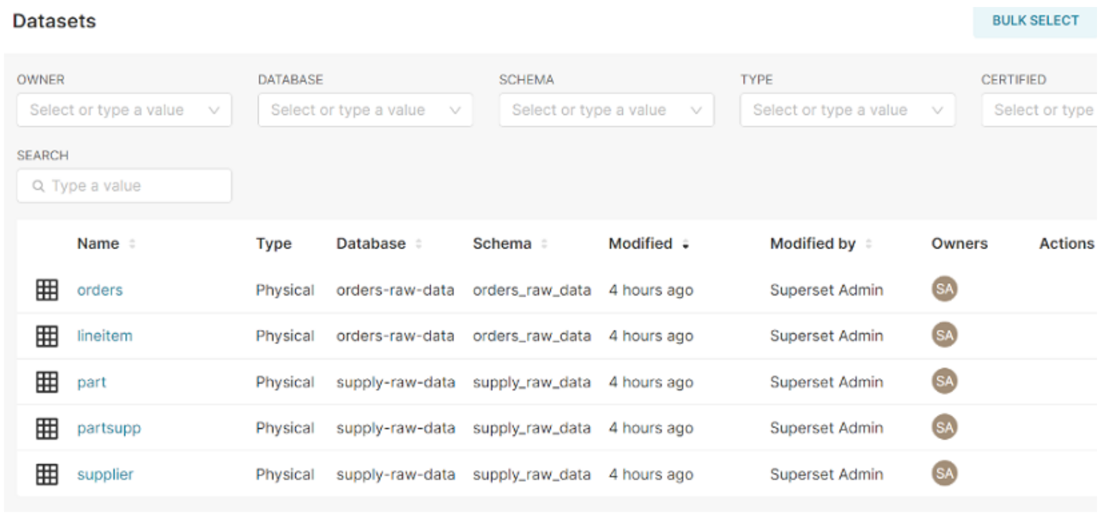
- **Database**: a database with the name of the DP is created via link to Trino, which queries the data in the location indicated by the DP. If such a database already exists, this task retrieves the id from the name and passes it to the next task.



Database	Backend	AQE	DML	CSV upload	Expose in SQL Lab	Created by	Last modified
orders-raw-data	trino	✓	✓	×	✓	Superset Admin	4 hours ago
supply-raw-data	trino	✓	✓	×	✓	Superset Admin	5 hours ago

Figure 5-9: Two databases on superset representing two different data products.

- **Dataset**: a dataset is created for each table in the Data Product. Now it is possible to execute SQL queries on data and create visuals.



Name	Type	Database	Schema	Modified	Modified by	Owners	Actions
orders	Physical	orders-raw-data	orders_raw_data	4 hours ago	Superset Admin	SA	
lineitem	Physical	orders-raw-data	orders_raw_data	4 hours ago	Superset Admin	SA	
part	Physical	supply-raw-data	supply_raw_data	4 hours ago	Superset Admin	SA	
partsupp	Physical	supply-raw-data	supply_raw_data	4 hours ago	Superset Admin	SA	
supplier	Physical	supply-raw-data	supply_raw_data	4 hours ago	Superset Admin	SA	

Figure 5-10: Datasets on Superset, representing tables from data products.

In this first and limited implementation, since BigQuery and GCS are the backends, the catalog identifies the Hive connector, while the schema identifies the individual Data Product, as ‘orders-raw-data’ and ‘supply-raw-data’. Another Argo workflow is then executed to deploy the Data Product, where it is connected to Dagster and orchestrated together with the other ones.

Chapter 6

Results

In the previous chapters, the implementation of the Data Mesh as a Bip project was exposed in some detail. This chapter sets out the results of this experimentation and a direct comparison with theory to verify if the four principles described by Zhamak Dehghani have been met. In a second phase other Data Mesh implementations will be presented, made by big companies in the Cloud industry.

6.1 Theoretical Comparison

In general, the implementation achieves many of the principles and goals set out by Zhamak Dehghani in her book about Data Mesh [4].

6.1.1 Domain Ownerhsip

Data remain "at the source": via the virtualization offered by Trino, they are not moved from the storage systems where they are saved, while only the expertise teams can build them together into Data Products, whether they are Source-Aligned, Aggregated or Consumer-Aligned. dbt allows Data Products to be modeled at will by defining schemas, while preserving interoperability, and to perform transformations or aggregations (via federated queries): pipelines are then removed from the center of the architecture and the task is left to the teams who can easily define them via SQL.

6.1.2 Data as a Product

The tools presented in the previous chapters serve to facilitate users in developing Data Products conforming to the characteristics that make them of central value in a company's economy. DataHub, using the metadata and the YAML documentation provided by dbt awhen a DP is built, serves as a catalog and first point of scouting of the data in an organization's structure, making the DP discoverable and understandable. The virtualization layer introduced by Trino allows the data to be accessed without moving it from the various source backends, the DP is therefore addressable. Together with the configurations made possible by dbt, the Data Product is moldable and yet interoperable (being a SQL-on-Anything system) and Native Accessible, as it can potentially be explored by any BI or ML tool. It also currently covers a small part of security, in that roles can be defined to access and modify data, but that is not enough. Dagster, Grafana, and the metrics on DataHub will make the DP trustworthy and truthful, while it will be a responsibility of the team working on it to make the DP valuable on its own.

6.1.3 Self-Serve Data Platform

The platform is deployed on Kubernetes clusters via CI/CD tools such as GitLab, and IaC such as Terraform, enabling:

- The **DP Developer** to not worry about the details of provisioning, as computing resources are managed independently as needed, or of the DP life cycle, as build and deploy are automated through Argo workflows.
- The **DP User** to easily discover the data she needs through DataHub and consume it through BI tools and notebooks. The DP does not have to be "loaded", rather it is ready to use.

6.1.4 Federated Data Governance

This fourth principle has only been partially realized and will be the subject of future development. The presence of APIs encourages Standard as Code, making Data Products federated members of the Mesh. Metrics will be provided to tools such as Grafana and Prometheus when they are implemented, which will take care of the overall infrastructure situation, automatic tests, and monitoring. The biggest problem remains data access policies, now partially covered by the role definition made available by Trino, but this is an interim measure, as it does not easily scale.

6.2 A view from the big companies

Because the conception of the Data Mesh is very recent, there are no architectures prepared for this approach from start to finish, partially because technologies may be needed to exploit the full potential of the Mesh, particularly at the federation level. Since its conception in 2018, the Data Mesh has attracted a lot of attention, particularly within the online discussion, on sites such as Medium [46], has been filled with articles discussing the benefits of adopting this architecture or discussing implementations and suitable tools. In the past two years, large companies have shown interest in this architecture, creating implementations for customers with special data management needs. For example, Zalando already in 2020 realized that accessibility and availability at scale can only be guaranteed when moving more responsibilities to those who pick up the data and have the respective domain knowledge – the data owners – while keeping only data governance and metadata information central [38]. With the company Databricks they, implemented an architecture non-technically called Bring Your Own Bucket, which features a central platform playing the role of a self-serve data platform with:

- A Data Lake Storage, where teams can share buckets (Amazon S3), with refined data from their (sub)domain.
- A Processing Platform, where to run queries on data.
- A Metadata Layer, playing the role of the catalog.

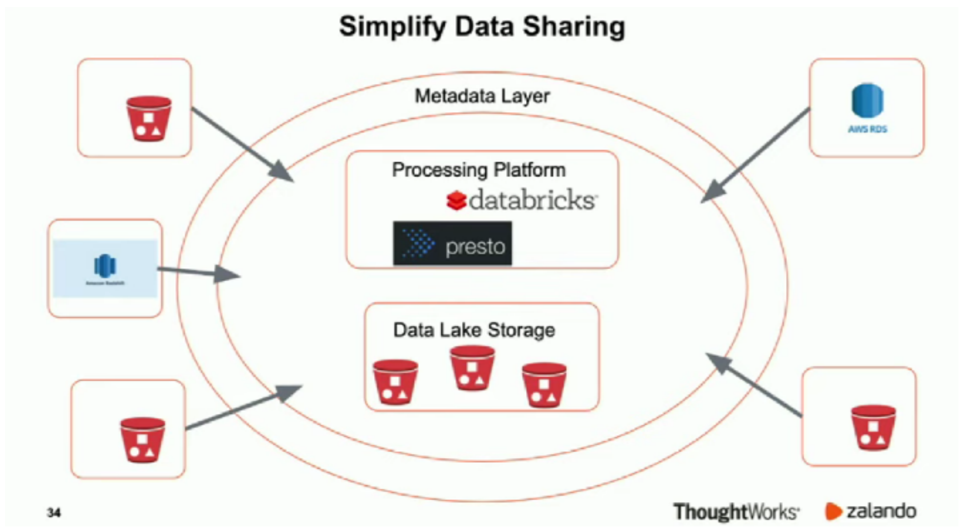


Figure 6-1: Bring Your Own Bucket, Zalando.

Data Users can search and use the data in the way they prefer, be a Relational Database, Redshift or other. There are rules for organizing the buckets to be shared: it is necessary to choose the data to be included, following a ranking of the most used datasets. In 2021 the three big companies offering Cloud-as-a-Service, started providing guides for implementing the Data Mesh architecture on their platforms and offering new services to exploit its potential.

6.2.1 Amazon Web Services

Amazon published an article [39] on how to approach an implementation of Data Mesh using their Web Services enabling lines of business to operate autonomously (like Domains) and providing a central data discovery, governance, and auditing for the organization at large (Self-Serve Data Platform and Federation). Amazon propose to use replication of the Lake House Architectures for implementing Data Domains and Products in a scalable way: the producers can store raw and transformed data into Amazon Simple Storage Service (S3), and with AWS Glue do the ETL processes to prepare the data products that can be cataloged into a Lake Formation Data Catalog. Data Domains then must expose a set of interfaces that make data consumable through analytics and machine learning services like Athena, Redshift and SageMaker. The Lake Formation service also offers the ability to enforce data governance, granting security. It is important to note that sharing is done through metadata linking alone. Data is not copied to the central account, and ownership remains with the producer [39].

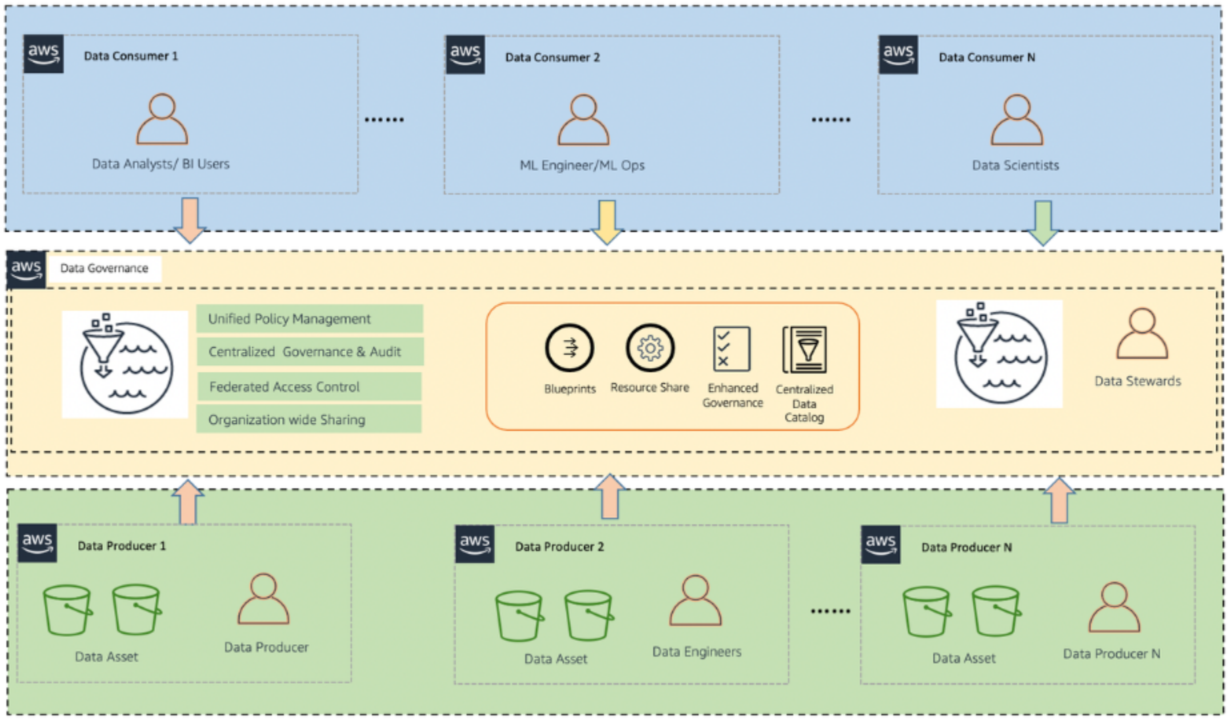


Figure 6-2: High level view of the Data Mesh implementation through AWS.

The company JPMorgan adopted this pattern to build its own Data Mesh architecture [40].

6.2.2 Microsoft Azure

Microsoft has not only published some articles on the benefits of the Data Mesh and how to implement it through its services, but in the first quarter of 2022 released Cloud-scale analytics: a scalable, repeatable framework that meets the organizations' unique need for building modern data platforms

[41]. The goals this framework aims to achieve are the ones of the Data Mesh. Its architecture is built upon Azure Landing Zones, whose scope is to bring data closer to users, enabling self-service and guaranteeing scalability through replication – they embody the first three principles of Data Mesh. Inside this architecture a central core is also present, called Data Management Landing Zone, that maintains common management and governance – it embodies the Federated Computational Governance principle of Data Mesh.

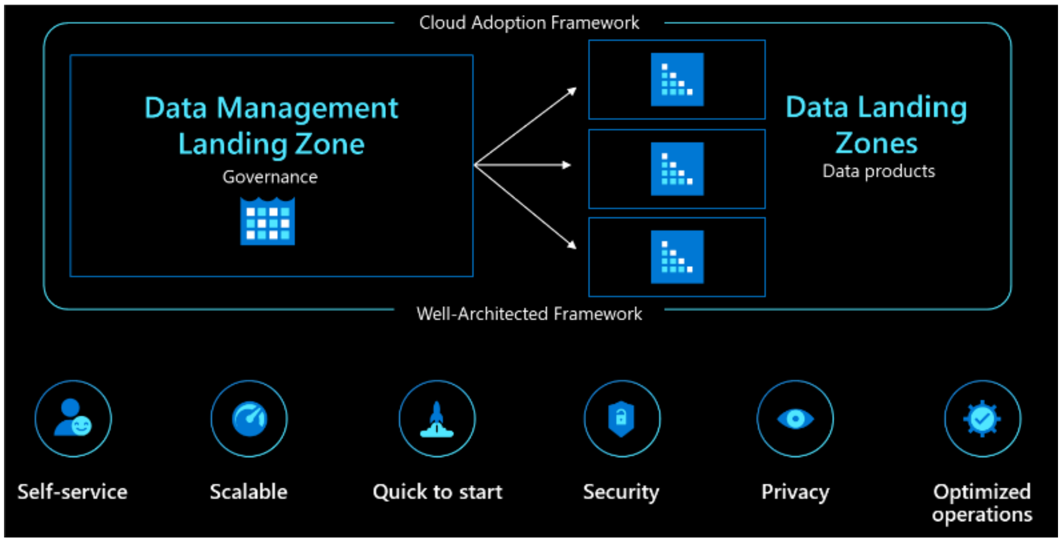


Figure 6-3: High level architecture of Azure Cloud-scale analytics.

Data Landing Zones are composed of several layers:

- **Core Services**: including all the services that allow to ingest, store, analyze, and transform data, plus monitoring and connectivity.
- **Data Application** that builds Data Products.
- **Visualization** for consuming data.

Data Management Landing Zone is responsible for the governance of the entire platform, via crawlers, which connect to data lakes and polyglot storage in data landing zones. It contains services like: Data Catalog, Data quality management, Data modeling repository, API catalog, Data sharing and contracts. Azure, therefore, allows one to create her own architecture through the replication of these Data Landing Zones, that already contain all the services to implement the Data Mesh principles.

6.2.3 Google Cloud Platform

Like Microsoft, Google has also shown interest in Data Mesh by proposing a new service called Dataplex, which allows data from Google Cloud Storage and Big Query buckets to be organized together. Dataplex presents a hierarchical organization that interprets the principle of Domain Ownership:

- **Lakes**: represent a company's domains.
- **Zones** represent subdomains, for example a logical group of data managed by a team. They can be: Raw with ingested and unstructured data; Curated with cleaned, formatted, ready for analytics data.
- **Assets**: can be GCS buckets or BQ tables.

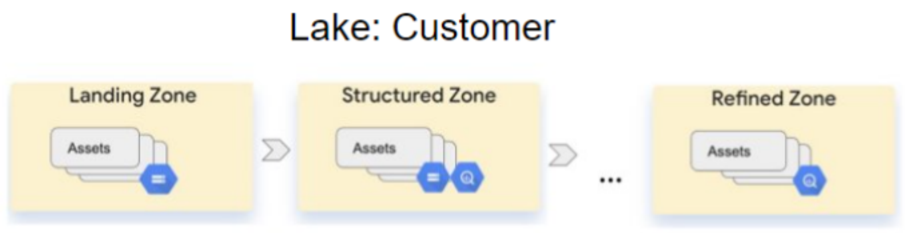


Figure 6-4: Dataplex hierarchical organization.

An example of how it can be used in a Data Mesh perspective is to integrate data from the various sources into untended zones that are "privately" managed by the team concerned. The goal is to transfer as little data as possible, through transformations that keep them within curated zones, and then publish them to the metastore provided by GCP (Data Proc) so that different teams in the company can access the Data Products and consume them through notebooks, graphs, and BI tools, some of which have been implemented within Dataplex's Explore.

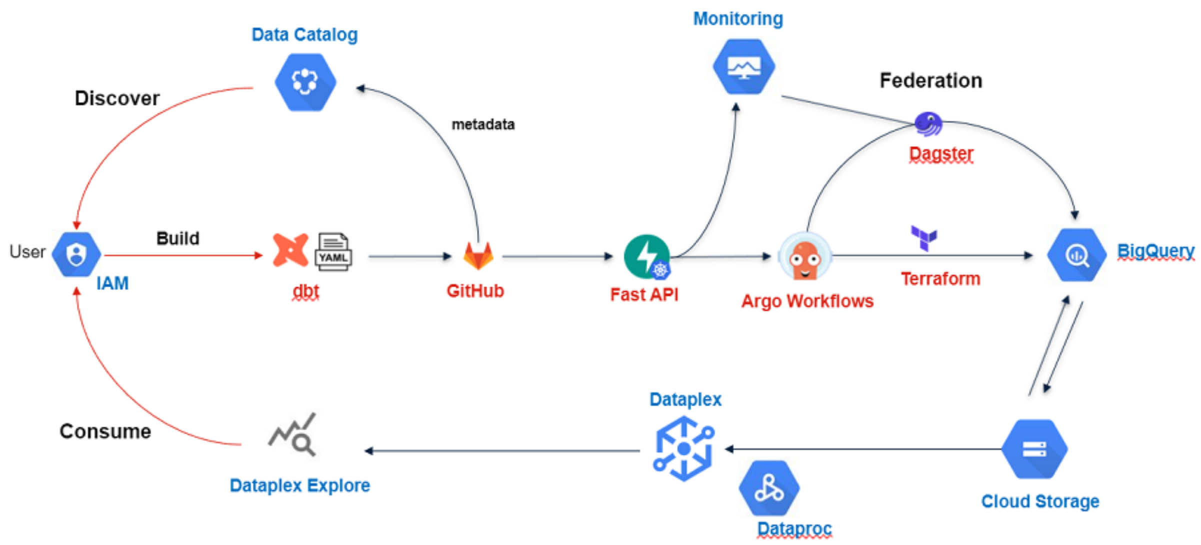
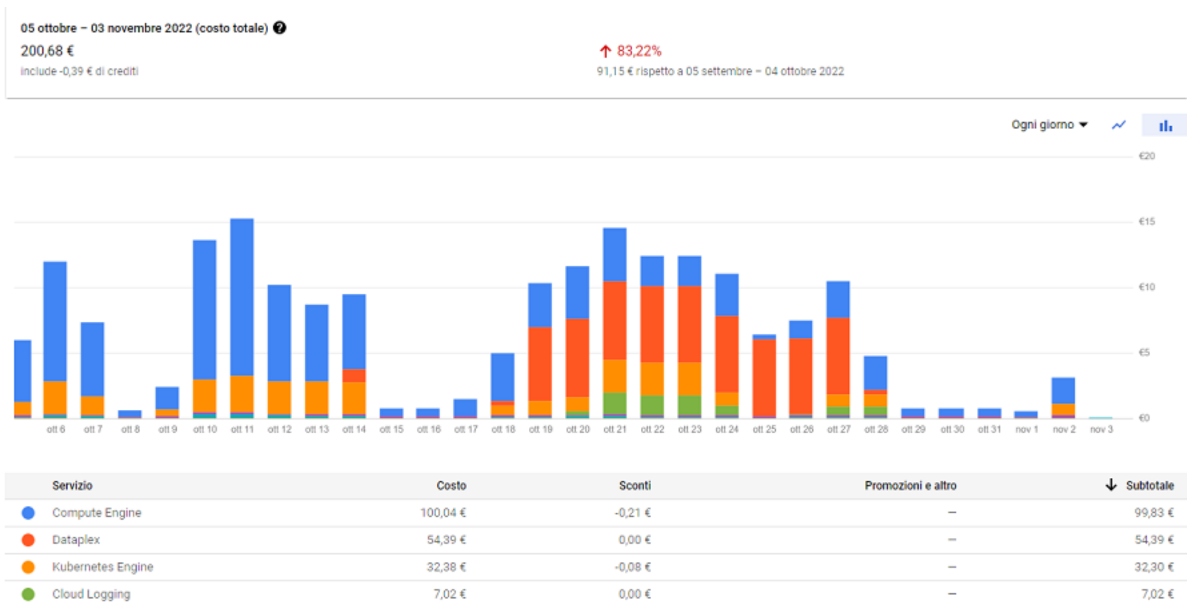


Figure 6-5: GCP Data Mesh Implementation.

Since the Bip implementation has been designed on GCP services, the first candidate chosen for the Data Consumption part has been Dataplex, but the problem with this service is that it does not make possible to build a cross-cloud architecture, failing the challenge of abstracting the backend of any Data Product. Using Dataplex, one will also have to rely on the other tools to implement the functionality needed for the Mesh, such as the Data Catalog for Data Product discovery. Such services have, however, a much higher cost than would be achieved with an identical structure built with open-source tools, such as those presented in previous chapters. Figure 6-6 represents the cost of the Data Mesh cluster during October 2022: Dataplex was tested on four data products for a total of ten days and costed about €50, half of the bill for the Compute Engine, in charge of keeping up the infrastructure for the entire month. Rather than implementing expensive GCP services such as Dataplex, it is more cost-effective to implement open-source tools with similar functionality.


05 ottobre – 03 novembre 2022 (costo totale)

200,68 €

include -0,39 € di crediti

↑ 83,22%

91,15 € rispetto a 05 settembre – 04 ottobre 2022

	Servizio	Costo	Sconti	Promozioni e altro	↓ Subtotale
●	Compute Engine	100,04 €	-0,21 €	–	99,83 €
●	Dataplex	54,39 €	0,00 €	–	54,39 €
●	Kubernetes Engine	32,38 €	-0,08 €	–	32,30 €
●	Cloud Logging	7,02 €	0,00 €	–	7,02 €

Figure 6-6: GCP Data Mesh Implementation.

Building a Data Mesh architecture using GCP services, however, solves the federation problem, as the Identity and Access Management (IAM) service allows data access policies to be managed through the definition of roles and contexts.

6.2.4 Summing up

Concluding this section, it is possible to state that, while Amazon's approach to Data Mesh is still quite immature, the architecture developed in Bip does not differ much from those designed by Microsoft and Google: they have in common the adoption of blueprints for creating scalable distributed systems hosting Data Products, whether they contain structured or unstructured data, and the presence of a central platform where Data Products can be shared, accessed, and consumed. The services of the two large companies work very well in their own ecosystem, but are not cross-cloud oriented, whereas the goal of the Bip project was to have an architecture ready to work with data from any system, universal. Obviously, this leads to accessibility and security issues because of the increased complexity.

Chapter 7

Conclusions

The internal project at Bip presented in this thesis has been quite articulate as it concerned the construction of a Data Mesh architecture from scratch, and even the consumption part alone required the study of different tools, services and architectures. This last chapter presents the difficulties, learnings from this project and future developments.

7.1 Challenges

Working on a Data Mesh project revealed a variety of challenges. The first one was to move from understanding the theory to putting it into practice: what a Data Product actually is, how to build it, how to implement a self-service data platform, which tools best embrace the concept of a Data Mesh among the many ones available.

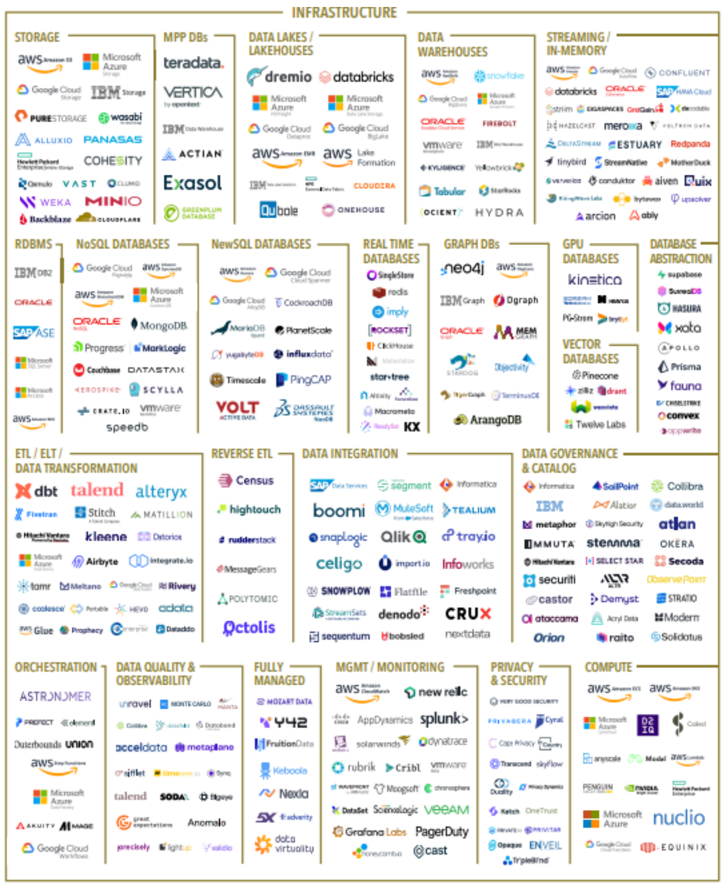


Figure 7-1: MAD landscape 2023, Infrastructure tools [42].

A second type of difficulty turned out to be the technical complexity of implementing this architecture, due to:

- Deploying an infrastructure via CI/CD and managing dependency issues – innovative products release new versions, very often causing conflicts;
- Dependencies over outdated services, such as Hive Metastore;
- DP management and consumption with data distributed across multiple external repositories;
- Terraforming and configuring all the tools is time-consuming, more

demanding than deploying services born integrated from one individual cloud provider, such as GCP.

A third type of difficulty involved the planning of migrations and transformations to the Data Mesh for client companies: this point proved to be the most complicated, because, unlike a technical challenge, the effort put in does not guarantee a result, since some compromises are necessary when dealing with companies: concretizing the Data Mesh means making changes at the level of personnel organization, introducing and modifying roles and responsibilities, fundamental levels in the internal relationships. Another issue emerges from the fact that there are still no native platforms for the Data Mesh, but only services – as seen in the last chapter – that help building it: the result is a high degree of system integration, which presents risks for large companies. Throughout this project, it has been possible to work with different types of companies, which highlighted the complexity of the approach:

- The first type of company is already well organized with teams of analysts divided by data domain, in which case a transition to a Data Mesh architecture could bring benefits very quickly;
- A second type of company does not have such a defined organization, but can leverage capabilities at the technical level, such as terraforming its infrastructure and using sandbox environments;
- A third type of company had a path that did not include integration of data and cloud platforms, so moving to a Data Mesh architecture would be premature.
- A fourth type of company, with a good know-how on data, has expanded greatly over the years, but no strategy has been developed to integrate the data analysis of acquired companies; it would be necessary to create a data strategy context, before migrating to Data Mesh.

A company's readiness for a Data Mesh architecture therefore depends on many factors, and it is therefore difficult to convey to a customer the advantages of a transition to Data Mesh despite the costs and time involved.

7.2 Learnings

The project was important to explore concepts such as:

- Employ declarative logic to build infrastructure and data products is very powerful;
- Make this foundation transparent, hiding its complexity from the user via API;
- Decoupling storage and computing, via a virtualization layer;
- Make the data product valuable, easy to discover and use.

Particularly, it was important to gain a deeper understanding of the integration level of a modern data stack. The use of modern tools has led us to positively value a best-of-breed approach, that is, to choose to implement the instrument, among the many in the ecosystem, that best performs a specific task. The problem of this approach is the lack of standards: new protocols and interfaces are needed to make compatible the countless tools composing the Machine Learning, AI and Data ecosystem.

7.3 What's Next

Building a cross-cloud architecture has brought some complexity to data governance and federation management, so future developments will focus on these aspects, both with the implementation of monitoring and a policy management model.

7.3.1 Service Mesh

As seen in previous chapters, the problem of security, access to data products and policy management remains an open issue: a possible solution is represented by service mesh, a dedicated infrastructure layer, that could be very useful to implement routing, security, resilience, and other policies on a service's inbound and outbound calls. A service mesh consists of network proxies paired with each service in an application and a set of task management processes. The proxies are called the data plane and the management processes are called the control plane. The data plane intercepts calls between different services and "processes" them; the control plane is the brain of the mesh that coordinates the behavior of proxies and provides APIs for operations and maintenance personnel to manipulate and observe the entire network [43]. For example, Istio, an open-source implementation of a service mesh, embeds the configuration of traffic routing policies in each endpoint of every single service and executes them locally right at the time of making a request [4].

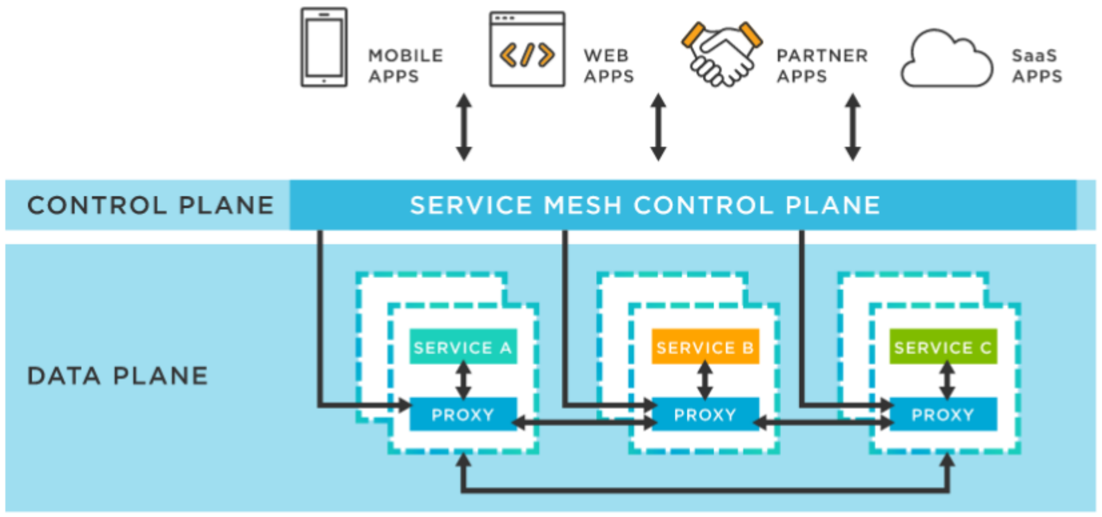


Figure 7-2: Service Mesh high level structure.

7.3.2 Arrow Flight SQL

On the virtualization side, Trino may not be the perfect solution as it has a high computational cost regarding the data discovery in the various backends; in particular, it could be a bottleneck for aggregation and join operations, which would be quite frequent in a decentralized architecture such as Data Mesh. A better alternative might be the Arrow Flight SQL, a protocol for interacting with SQL databases using the Arrow in-memory format and the Flight RPC framework [44]. This recently released protocol (February 2022) makes data collection faster because it can transport data without having to convert it into a row-based form and allowing the data to be transferred from server to client without the delay of serialization and deserialization.

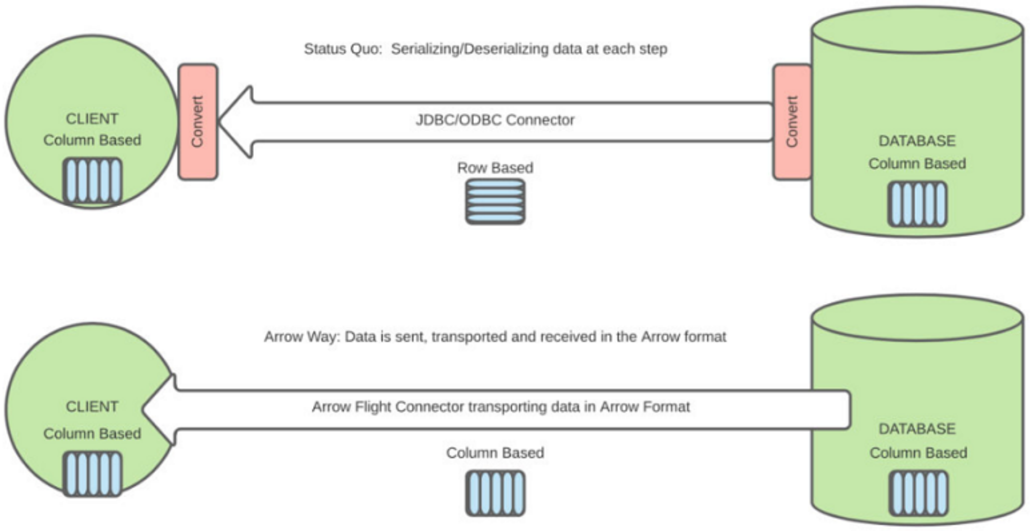

Status Quo: Serializing/Deserializing data at each step
CLIENT
Column Based
Convert
JDBC/ODBC Connector
Row Based
Convert
DATABASE
Column Based
Arrow Way: Data is sent, transported and received in the Arrow format
CLIENT
Column Based
Arrow Flight Connector transporting data in Arrow Format
Column Based
DATABASE
Column Based

Figure 7-3: Arrow way skips conversion steps, making the process faster.

An architecture that implements both technologies could involve a scalable container that adopts communication protocols (such as Arrow Flight) and hosts a proxy Service Mesh with policies. This would create an intermediate layer that would ensure interoperability between the tools and federation of the data products.

7.4 Data Mesh

"When I first developed the concept of data mesh in 2018, I understood the magnitude of the change I was proposing [...] the idea took on a life of its own. It's become the subject of countless conferences and articles. Hundreds of organizations from a variety of industries have attempting it." [45] - Zhamak Dehghani.

The great attention currently being reserved to Data Mesh finds a reason in how it solves the problem of data decentralization, which for decades has been tried to be contained by flowing data into centralized structures that were unable to handle the ever-increasing amount of information. As result of this interest a further tool innovation is expected in this 2023:

- At the Data Product level, something even more intuitive than dbt could emerge, perhaps with the ability to define DPs through no-code interfaces.
- At the data catalog level, we expect an evolution of metrics, observability and discovery made more effective by semantic search and NLP.
- A native Data Mesh platform that facilitates access management and interoperability among custom tools.

Regarding the latter point, on January 16, 2023, Zhamak Dehghani announced Nextdata, a company that will focus on decentralizing data. Their first project is Nextdata OS, “a data networking toolset” that will introduce the concept of Data Product Containers, a new unit of data value designed to be shared and used responsibly at scale [45].

We will continue to hear about Data Mesh for some time.

Chapter 8

Ringraziamenti

Ringrazio chi ha contribuito a questa tesi: al relatore Marco Patella per la disponibilità e l'interesse dimostrati, al tutor Riccardo per avermi seguito durante il tirocinio, a Martino, Matteo e Andrea per la fiducia e l'aiuto che mi hanno dato. Ringrazio tutti i colleghi di Bip che stanno contribuendo a creare un ambiente amichevole in ufficio.
La conclusione della laurea magistrale è anche un buon momento per guardarsi indietro ed essere grati delle persone e dei momenti che hanno reso questi anni così belli.
Un primo ringraziamento va a mia mamma e mio papà, su cui so di poter contare in qualsiasi momento e parte del mondo io sia.
Un ringraziamento a tutta la famiglia: ai nonni che hanno sempre una raccomandazione, agli zii che mi hanno trasmesso passioni e interessi, ai cugini con cui sono cresciuto.
Un ringraziamento va agli amici.
Ai cavalli dell'Ippodromo per i capodanni in montagna, le grigliate al lago, i palidani, i concerti, le nottate semplicemente a parlare: a quante ne abbiamo vissute e a quanto siamo cambiati, insieme.
Ai "modenesi" per le lezioni in ultima fila, lo studio in biblioteca, le birre e le serate: con la vostra spensieratezza mi ricordate quanto faccia bene non prendersi troppo sul serio.
To the Erasmus guyze (eccoli qua!), we made Idun great again with (after)parties, kasespatzle, carbonare, sunrises on the beach: you gave me so much in a short time.
Un ringraziamento particolare ai quei fratelli e sorelle che in questi anni mi hanno ascoltato, consigliato e condiviso con me il loro mondo.

Chapter 9

Bibliography

[1] The Zettabyte Era Officially Begins (How Much is That?), Thomas Barnett Jr., https://blogs.cisco.com/sp/the-zettabyte-era-officially-begins-how-

[2] Data Never Sleeps 10.0, https://www.domo.com/data-never-sleeps

[3] Data Mesh — Thoughtworks, https://www.thoughtworks.com/what-we-do/data-and-ai/data-mesh

[4] Z. Dehghani, Data mesh: delivering data-driven value at Scale. Seabstopol, CA: O'Reilly Media, 2022

[5] Business integration partners - Wikipedia, https://it.wikipedia.org/wiki/Business_integration_partners

[6] xTech - Bip Consulting (bip-group.com), https://www.bip-group.com/it/who-we-are/practices/x-tech/

[7] Data - Wikipedia, https://en.wikipedia.org/wiki/Data

[8] What is Online Transaction Processing (OLTP) — Oracle , https://www.oracle.com/database/what-is-oltp/

[9] The Evolution Of Data (forbes.com), https://www.forbes.com/sites/forbestechcouncil/2018/07/17/the-evolution-of-data/?sh=2621d6f0c95f

[10] https://www.researchgate.net/publication/228765967_Business_Intelligence

[11] Data Warehouse Testing — QuerySurge, https://www.querysurge.com/solutions/data-warehouse-testing

[12] The Difference Between Operational and Analytical Data Systems (arkatechture.com), https://www.arkatechture.com/blog/the-difference-between-operatio

[13] Evolution to the Data Lakehouse - The Databricks Blog , https://www.databricks.com/blog/2021/05/19/evolution-to-the-data-lakehouse.html

[14] What Is a Data Lakehouse and Answers to Other Frequently Asked Questions - The Databricks Blog, https://www.databricks.com/blog/2021/08/30/frequently-asked-questions-about-the-data-lakehouse.html

[15] How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh (martinfowler.com), https://martinfowler.com/articles/data-monolith-to-mesh html

[16] Vernon, Vaughn (2013). Implementing Domain-Driven Design. Upper Sadle River, NJ: Addison-Wesley. p. 3. ISBN 978-0-321-83457-7

[17] Scott R. Ellis EnCE, RCA, in Network and System Security (Second Edition), 2014, https://www.sciencedirect.com/book/9780124166899/network-and-system-security

[18] What is a data mesh and how not to mesh, https://towardsdatascience.com/what-is-a-data-mesh-and-how-not-to-mesh-it-up-210710bb41e0

[19] https://cloud.google.com/kubernetes-engine?hl=en

[20] https://kubernetes.io/it/docs/concepts/overview/what-is-kubernetes/

[21] https://en.wikipedia.org/wiki/YAML

[22] https://en.wikipedia.org/wiki/Terraform_(software)

[23] https://docs.gitlab.com/ee/ci/introduction/index.html

[24] https://argoproj.github.io/argo-workflows/

[25] https://fastapi.tiangolo.com/

[26] https://en.wikipedia.org/wiki/API

[27] https://en.wikipedia.org/wiki/Data_build_tool

[28] https://docs.getdbt.com/docs/build/projects

[29] https://datahubproject.io/docs/introduction

[30] https://datahubproject.io/docs/features/

[31] https://dagster.io/platform

[32] https://en.wikipedia.org/wiki/Grafana

[33] Matt Fuller, Manfred Moser, Martin Traverso, Trino: The Definitive Guide. Sebastopol, CA: O'Reilly Media, 2021

[34] https://trino.io/blog/2020/10/20/intro-to-hive-connector.html

[35] https://helm.sh/docs/topics/charts/

[36] https://trino.io/episodes/21.html

[37] https://superset.apache.org/

[38] https://www.databricks.com/session_na20/data-mesh-in-practice-how-europes-lea

[39] https://aws.amazon.com/blogs/big-data/design-a-data-mesh-architecture-using-a

[40] https://aws.amazon.com/blogs/big-data/how-jpmorgan-chase-built-a-data-mesh-a

[41] https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/cloud-scale-analytics/

[42] https://mattturck.com/mad2023/

[43] Rahul Sharma; Avinash Singh (2019). Getting Started with Istio Service Mesh: Manage Microservices in Kubernetes. Apress. p. 103. ISBN 9781484254585.

[44] https://arrow.apache.org/docs/format/FlightSql.html

[45] https://medium.com/@zhamakd/why-we-started-nextdata-dd30b8528fca

[46] https://medium.com/