DATA MESH ARCHITECTURE

MINI-PROJECT REPORT

DATA ENGINEERING SPECIALIZATION

RESIDENT: Tạ Ngọc Minh
Sophomore in Data Science & AI @ HUST

SUPERVISOR: Mr. Nguyễn Chí Thanh, MSc.

Hanoi — 2023

PREFACE

Dear the board of directors of Viettel Group, the mentors, and all of my friends,

This document is a part of my research mini-project as a data engineer intern for the first period of Viettel Digital Talent 2023 Program. The completion of this research report does not mean the end of my duties as learners. In fact, this report is a first step towards consistently studying what I have reviewed and written here.

Data has grown up so quickly recently. The term 'big data' has become familiar with every tech-lovers is the clearest evidence of the explosion of data. This leads to many requirements of upgrading the current system and management, along with finding new architecture to store, process and analyze data. Once the central data lake has been overloaded, we need to design a decentralized architecture. This report is written under the research about the Data Mesh architecture, a new design that was first defined by Zhamak Dehghani in 2019.

To write this document, I would like to give many thanks to Viettel Group, and Viettel Digital Talent program for giving me chance to do my research. I want to send a big thanks to Mr. Nguyễn Chí Thanh, and all the mentors of the first period of the program, who gave me many useful lectures and guidance. Also, I want to say thanks to professor Huỳnh Thị Thanh Bình, professor Đỗ Tuấn Anh for their initial supports, and many other professors at the International Research Center for AI and the School of Information and Communication Technology, Hanoi University of Science and Technology about their public documents about data engineering. Last, I would like to say thanks to all of my friends, who always read my documents, comment about them and support me whenever I need.

Because this document is written in such a short time and some of the concepts are so new and hard to understand for a fresher like me, it is so hard to make sure it is completely correct, including the grammar errors and formatting errors in LaTeX. I would love to receive your contributions via **ngocminhta.nmt@gmail.com**.

Thank you for reading this report and looking forward to getting many comments from you.

Ngoc-Minh Ta
Viettel Digital Talent Residency 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

PROBLEMS AND INITIAL APPROACH

1.1 REAL-LIFE PROBLEM

1.1.1 A problem from history

Many organizations have invested in building a central data lake. To make data-driven business, there is a central data administration team take the responsibility for this.

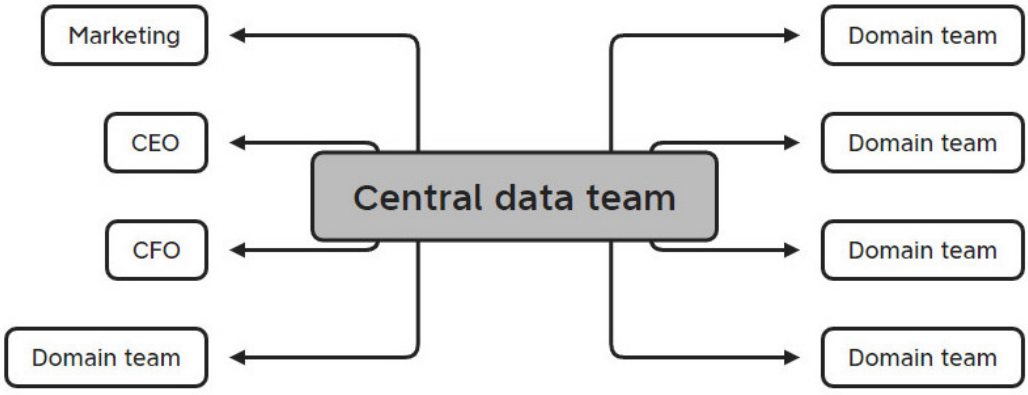

Figure 1.1: Central data team in a data-driven business

This model works quite well at the first time, however, after a while, the team became overloaded with tasks and requests, leading to a decrease in competitiveness due to slow analysis. Their slow response to changes in the operational database is due to the need to fix broken data pipelines and get domain expertise. This is a daunting task. [2, 3]

In some firm, applying domain-driven design strategies involve independent domain teams and a decentralized micro-service architecture to relieve burden on the central data team. However, these teams must contact the overloaded central data team to obtain essential data-driven insights. [2]

1.1.2 Demands from recent developments

Let's consider the scale-up of the software development over time. When one task grow bigger and bigger, we have to decentralize it into many smaller tasks, otherwise, all the organization will overload and lost of control. Take into consideration what we have done for the scale-up of software development:

- Decentralize business into domains;
- Decentralize engineering into autonomous teams;
- Decentralize monolith into micro-services;
- Decentralize operations into DevOps teams.

Hence, the next thing we need to do is scaling up data analytics by decentralizing data lake into data mesh.[3]

1.2 INITIAL APPROACH

1.2.1 What is Data Mesh?

The term data mesh was first stated by Zhamak Dehghani in 2019 and is based on four fundamental principles, which will be discussed in depth later:

- Principle of Domain Ownership;
- Principle of Data as a Product;
- Principle of the Self-Serve Data Platform;
- Principle of Federated Computational Governance.

1.2.2 What will changes after data mesh?

Data mesh is a fresh method for business intelligence, data analysis and management that is based on an distributed architecture. Along with that, data lake and data warehouse do not disappear, they just become nodes in the mesh. [1, 4]

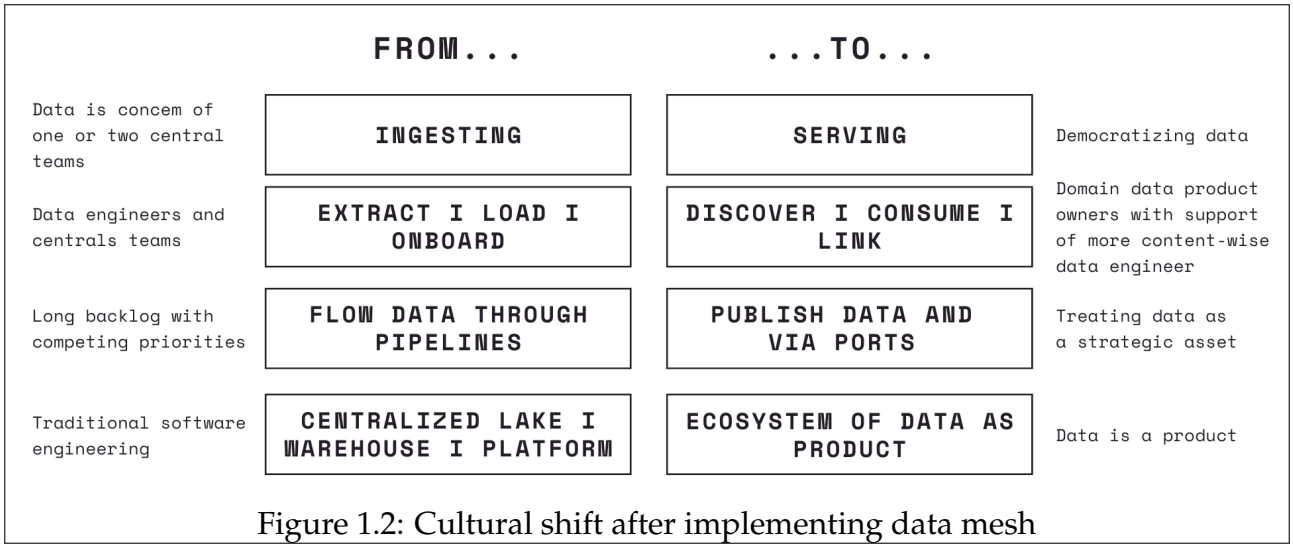

Figure 1.2: Cultural shift after implementing data mesh

Data mesh ensures organizations to continue to apply some data lake principles, and data

lake tooling for internal implementation of data products, but it would not be the centerpiece anymore.

It is an implementation detail sub-serving the idea of domain data product as the first-class concern. The same applies to data warehouse in terms of business reporting and visualization [4]. It also cause a cultural shift in the organization, also, a fundamental shift in the assumptions, architecture, technical solutions, and social structure of our organizations. [5]

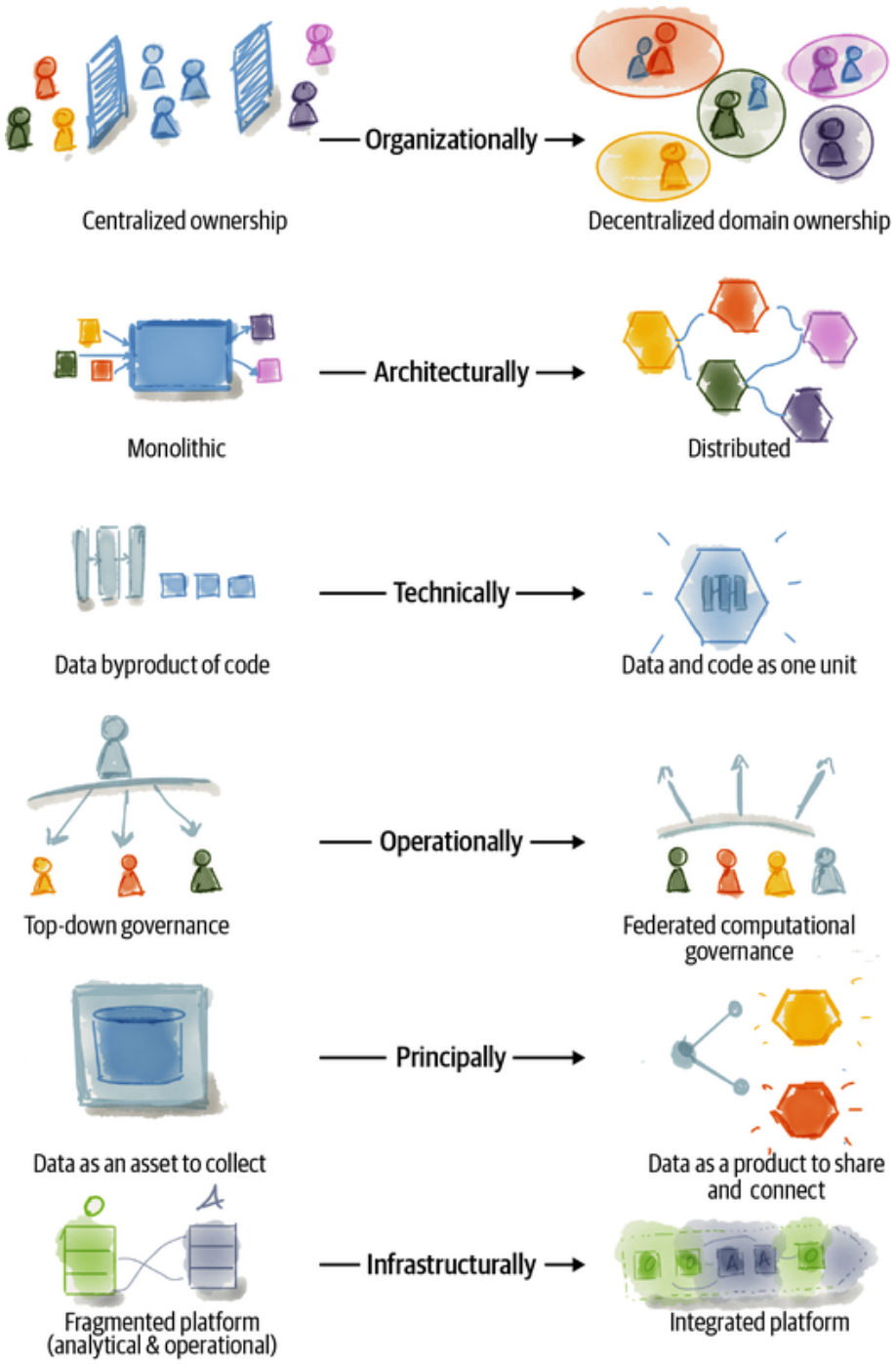

Figure 1.3: Data mesh dimensions of organizational changes

In its most basic form, it can be characterized by four interacting principles, which will be discussed in depth in section 2.1.

Chapter 2

DATA MESH ARCHITECTURE DESIGN

2.1 FOUR FUNDAMENTAL PRINCIPLES OF DATA MESH

Four simple principles can represent the logical architecture and operating model of data mesh. They are intended to move us closer to the goals of data mesh: increasing the value of data at scale, maintaining agility as an organization grows, and embracing change in a complex and turbulent business context.

2.1.1 Principle of Domain Ownership

Data mesh is a decentralized and distributed data architecture that follows the seams of organizational units, rather than technological partitions. It is founded on domain-driven design (DDD) strategies, which define a domain as "a sphere of knowledge, influence or activity." Data mesh gives the data sharing responsibility to each of the business domains, each domain is responsible for the data it is most familiar with. DDD's Strategic Design embraces modeling based on multiple models each contextualized to a particular domain, called a bounded context [1]. As Z. Dehghani's recommendation, data mesh adopts the boundary of bounded contexts to individual data products - data, its models, and its ownership.

Domain data ownership is the foundation of scale in a complex system like enterprises today. When we map the data mesh to an organization and its domains, we discover a few different archetypes of domain-oriented analytical data:

- Source-aligned domain data (native data product): Analytical data reflecting the business facts generated by the operational systems.
- Aggregate domain data: Analytical data that is an aggregate of multiple upstream domains.
- Consumer-aligned (fit-for-purpose) domain data: Analytical data transformed to fit the needs of one or multiple specific use cases.

The shift toward domain-oriented data ownership leads to accepting and working with real-world messiness of data, particularly in high-speed and scaled environments:

- Push Data Ownership Upstream.

[1] A bounded context is the delimited applicability of a particular model that gives team members a clear and shared understanding of what has to be consistent and what can develop independently. [6]

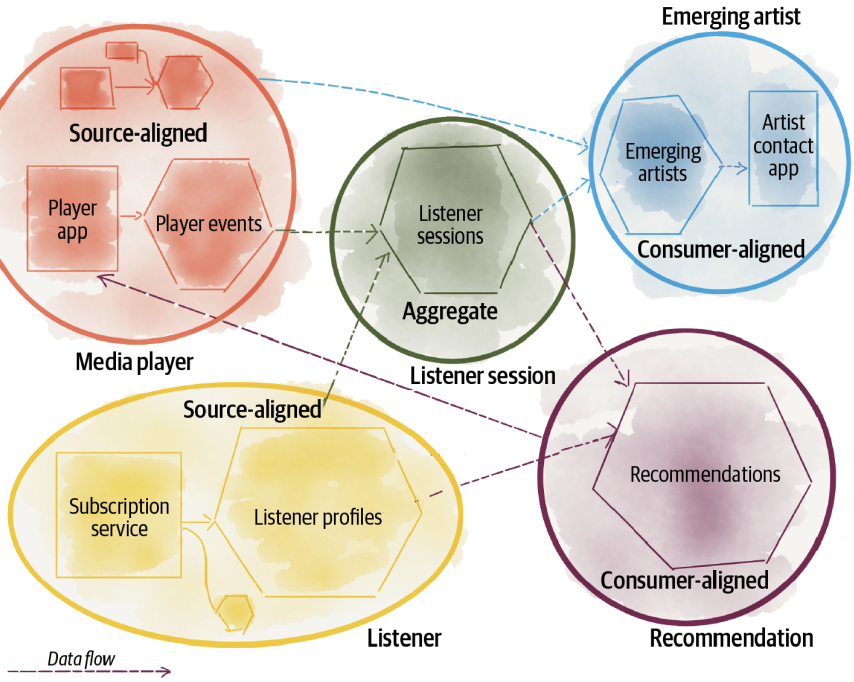

Figure 2.1: Decomposing the analytical data ownership and architecture, aligned with business domains, along with their data archetypes.

- Define Multiple Connected Models.
- Data mesh gives long-term domain ownership with obligation to exchange data.
- Hide the Data Pipelines.

Domains are taking up extra data responsibility with data mesh. To acquire agility and authenticity, responsibility and efforts transfer from a centralized data team to domains. [7]

2.1.2 Principle of Data as a Product

Data as a product is a response to data siloing and shifts data culture towards accountability and trust. It includes baseline data product usability attributes (see figure 2.2), which are an addition to FAIR data principles[2].

The introduction of analytical data as a product adds to the list of existing responsibilities of cross-functional domain teams [9] and expands their roles to Data product developer, and Data product owner.

Define these roles for each domain and allocate one or multiple people to the roles depending on the complexity of the domain and the number of its data products. Moreover, to make a data as a product, we need to satisfy these conditions:

- Reframe the Nomenclature.
- Think of Data as a Product, not a mere asset.
- Establish a Trust-But-Verify Data Culture.
- Join Data and Compute as One Logical Unit.

[2]**F**indability, **A**ccessibility, **I**nteroperability, and **R**eusability. [8]

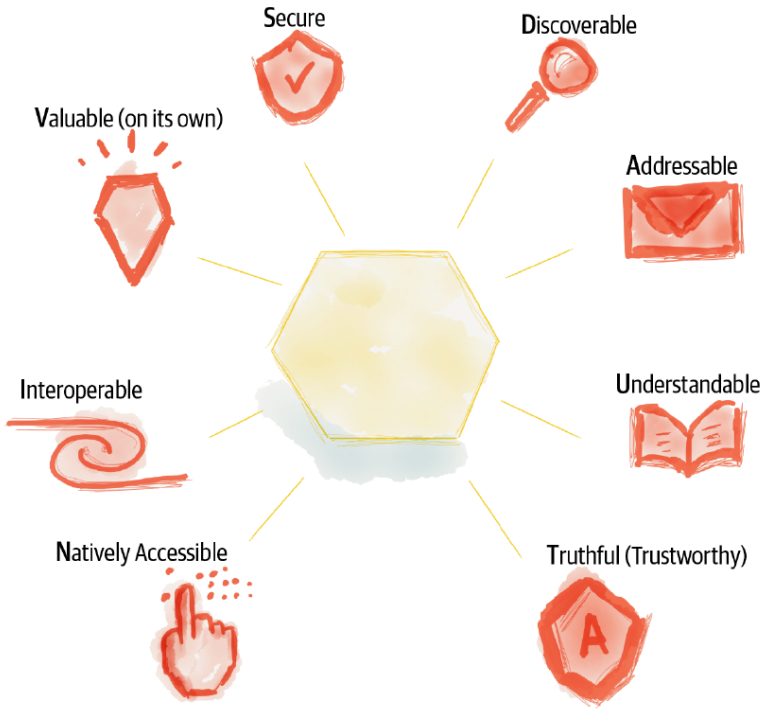

Figure 2.2: The baseline usability attributes of data products (DAUTNIVS)

2.1.3 Data contract

Data contract is an agreement between a service provider and data consumers. It is not in-depth legal documents, but a process to help data producers and data consumers get on the same page. It refers to the management and intended usage of data between different organizations to ensure reliable and high-quality data that can be trusted by all parties involved. [10]

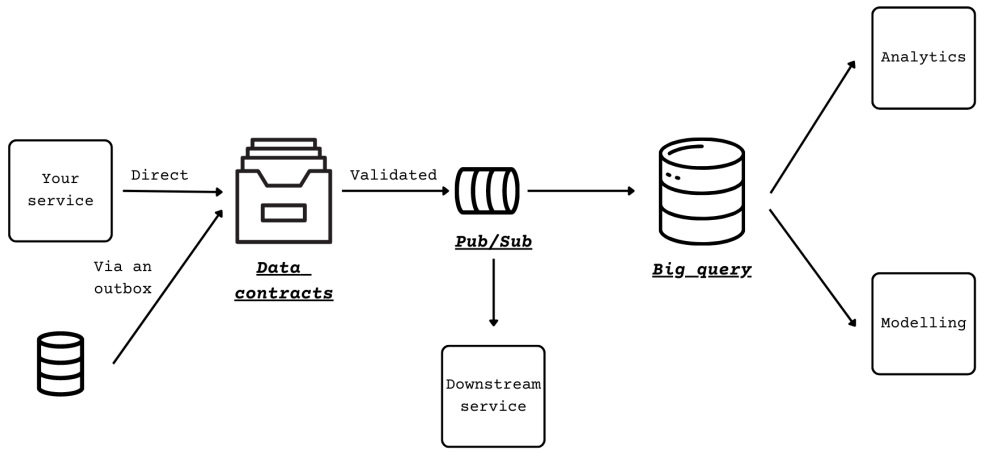

Figure 2.3: Data contract in organization

Data contract enforces specific formats, limitations, and semantic meanings to prevent data quality difficulties when it brought into data warehouse is unusable by data consumers.

The data mesh strategy is a critical notion for ensuring that data responsibilities are dis-

persed across domains. Contract is the method that helps producers and consumers communicate effectively, and they are especially critical when dealing with widely spread data.

Data contracts vary depending on data and organization, there is no standardization for them, but normally, we have:

- What data is being extracted.
- Type and frequency of consumption.
- Details of data ownership/ingestion.
- Levels of data access necessary.
- Information on security and governance (e.g., suppliers, users, starting & expiry time).
- How it affects any system(s) that intake may have an effect on.

Data Contract is a crucial Data Mesh idea for managing data products and their usage. Data contracts improve data quality and reliability, also, administrators' jobs get easier.

2.1.4 Principle of the Self-Serve Data Platform

A platform is required to link products with customers due to Data Mesh's decentralized nature. Furthermore, when each data management unit is independent of the others, there will be a lot of repetitive labor across units, which will impede the product's progress and efficiency. As a result, there is a need for a self-service infrastructure that provides the tools, interfaces, and resources necessary for domain units to interact with data on their own. This cuts development and operating costs dramatically.

Some essential features of Data Platform:

- Self-service and self-management: The data platform allows entities to work with data and serve output to other units or external systems. This helps organizations stay organized flexibly, work with processes that fit their style and goals, and automate the sharing and use of data products.
- Self-service and self-management: The data platform allows entities to work with data and serve output to other units or external systems. This helps organizations stay organized flexibly, work with processes that fit their style and goals, and automate the sharing and use of data products.
- Security and access management: Every shared platform needs to provide security and access management, and the platform is no exception. Units can easily change the access to their products, add or remove permissions, and The data platform must also provide security tools to ensure that the organization's security policy is enforced.
- Ensure integration between teams: Since each unit develops independently, the data platform must have a mechanism to integrate these products together. This mechanism can be in the form of regulations to be followed or adapters that the data platform provides.
- Provide data discovery and discovery services: The Data Platform must provide a mechanism for users to find out which data products they need to use. Usually, this is done via a data registry or data catalog. In addition, the Data Platform must provide documentation, metadata, and metrics about the Data Product so that users have the information they need to use it effectively.

2.1.5 Principle of Federated Computational Governance

In contrast to data lake and data warehouse's governance models, data mesh one shifts to decentralized, federated computational governance.

The data mesh governance model consists of three complementary pillars:

- **System thinking:** Manage the behavior of the mesh to create value by exchanging data products at scale.
- **Apply a federated operating model:** Create incentives to align domains' data products and ecosystem success.
- **Embed the governance policies** into each data product in an automated and computational fashion.

To bring these three pillars together, Figure 2.4 shows an example of this model.

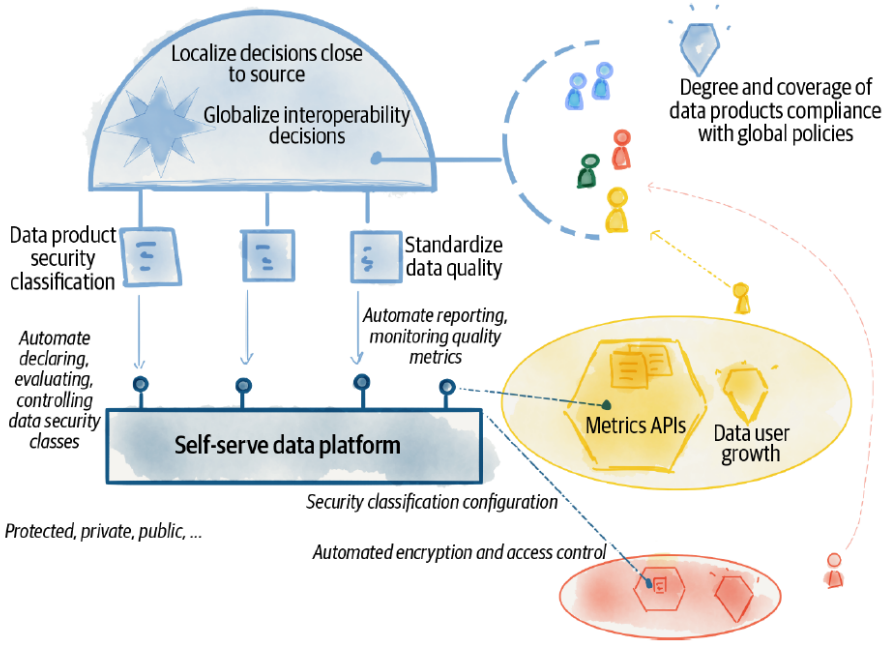

Figure 2.4: Example of data mesh governance operating model

Data mesh governance seeks to improve existing data governance by finding the dynamic equilibrium between localization and globalization of decisions. It requires trustworthiness and useful data across multiple domains to train ML-based solutions.

2.1.6 The correlations between the four principles

The correlations between the four principles are shown in figure 2.5.

2.2 WHY WE NEED DATA MESH?

Data mesh is what comes after an inflection point, shifting our approach, attitude, and technology toward data. Data mesh assumes a new default starting state: proliferation of data origins within and beyond organizations' boundaries, on one or across multiple cloud platforms.

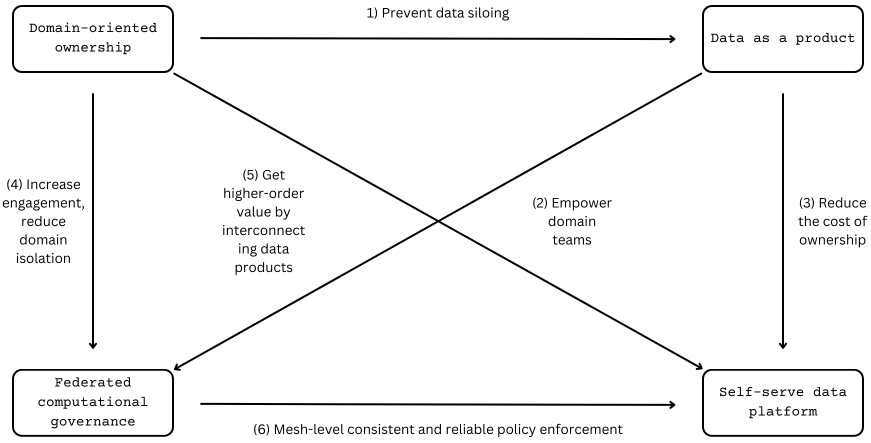

Figure 2.5: The correlations between the four principles

After the inflection point, data mesh make us reimagine data, specifically how to design solutions to manage it, how to govern it, and how to structure our teams: [11]

Table 2.1: Summary of after the inflection point with data mesh by the goals

What to do	**How to do it**
Manage changes to data gracefully in a complex, volatile, and uncertain business environment	
Align business, tech, and data	Create cross-functional teams responsible for long-term data ownership.
Close the gap between the operational and analytical data planes	Dumb pipes should be used to integrate applications and data products.
Localize data changes to business domains	Localize maintenance and ownership of data products in their specific domains Create clear contracts between domain-oriented data products to reduce impact of change.
Reduce the accidental complexity of pipelines and copying of data	Break down pipelines, move transformation logic into data products.
Sustain agility in the face of growth	
Remove centralized architectural bottlenecks	Remove centralized data warehouses and data lakes Enable peer-to-peer data sharing of data products through their data interfaces.
Reduce the coordination of data pipelines	Decomposition of pipeline architecture from functional to domain-oriented. Introduce explicit data contracts between domain-oriented data products.
Reduce coordination of data governance	Delegate governance responsibilities to autonomous domains and their data product owners Automate governance policies as code embedded and verified by each data product quantum.

continued on next page –

Table 2.1 – *continued from previous page*

What to do	How to do it
Enable team autonomy	Give domain teams autonomy in moving fast independently. Balance team autonomy with computational standards to create global consistency. Provide domain-agnostic infrastructure capabilities to empower domain teams.
Increase value from data over cost	
Abstract complexity with a data platform	Create a data-developer-centric and a data-user-centric infrastructure to remove friction and hidden costs in data development and use journeys Define open and standard interfaces for data products to reduce vendor integration complexity.
Embed product thinking everywhere	Focus and measure success based on data user and developer happiness Treat both data and the data platform as a product.
Go beyond the boundaries of an organization	Share data across physical and logical boundaries of platforms and organizations with standard and internet-based data sharing contracts across data products.

While the evolution of data architecture has been necessary and an improvement, all of the existing analytical data architectures share a common set of characteristics that inhibit them from scaling organizationally. They are all monolithic with centralized ownership and technically partitioned. It is no longer valid when data gets sources from hundreds of microservices and millions of devices from within and outside of enterprises.

2.3 Data Mesh Architecture Design

2.3.1 The Logical Architecture

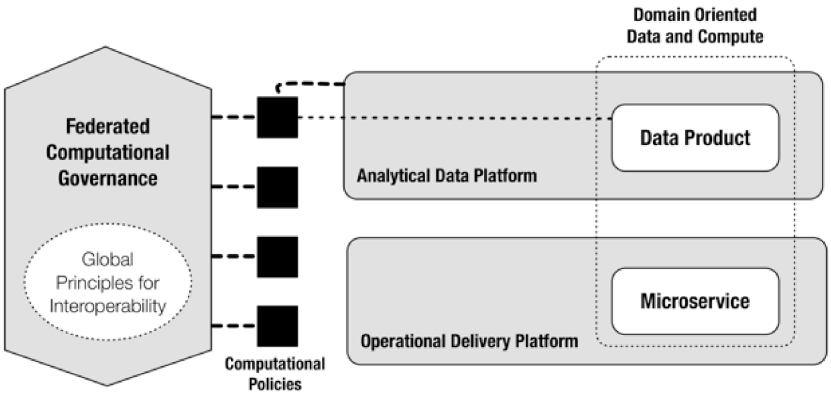

Figure 2.6: Data Mesh Logical Architecture [1]

Table 2.2: Data mesh logical architectural components

Architectural component	Description
Domain- Oriented Data Sharing Interfaces	
Domain	Systems, data products, and a cross-functional team aligned to serve a business domain function and outcomes and share its analytical and operational capabilities with the wider business and customers.
Domain analytical data interfaces	Standardized interfaces that discover, access, and share domain-oriented data products. The proprietary platforms need to offer open interfaces to make data sharing more convenient and interoperable with other hosting platforms.
Domain operational interfaces	APIs and applications through which a business domain shares its transactional capabilities and state with the wider organization.
Data product quantum	
Data product structural components	Data product implemented as an architecture quantum that encapsulates all the structural components it needs to do its job - code, data, infrastructure specifications, and policies. It is referred to in architectural discussions. It is used interchangeably with data products.
Data Product Data Sharing Interactions	There are many technologies for implementing a data product's I/O data port of data mesh, but it shares the common properties: • **Input data port:** A data product's mechanisms to continuously receive data from one or multiple upstream sources. • **Output data port:** A data product's standardized APIs to continuously share data.
Discovery and observability APIs	A data product's standard APIs to provide discoverability information - to find, address, learn, and explore a data product - and observability information such as lineage, metrics, logs, etc.
The Multiplane Data Platform	
Platform plane	A group of self-serve platform capabilities with high functional cohesion surfaced through APIs.
Data infrastructure (utility) plane	Platform plane providing low-level infrastructure resource management - compute, storage, identity, etc.
Data product experience plane	Platform plan providing operations on a data product.
Mesh experience	Platform plane providing operations on the mesh of connected data products.
Embedded Computational Policies	
Data product container	A mechanism to bundle all the structural components of a data product, deployed and run as a single unit with its sidecar.
Data product sidecar	The accompanying process to the data product implements cross-functional and standardized behaviors.
Control port	A data product's standard APIs to configure policies or perform highly privileged governance operations.

It is intentionally that the technology evolves to the point that we can get the logical architecture and its physical implementation as close to each other as possible. [12]

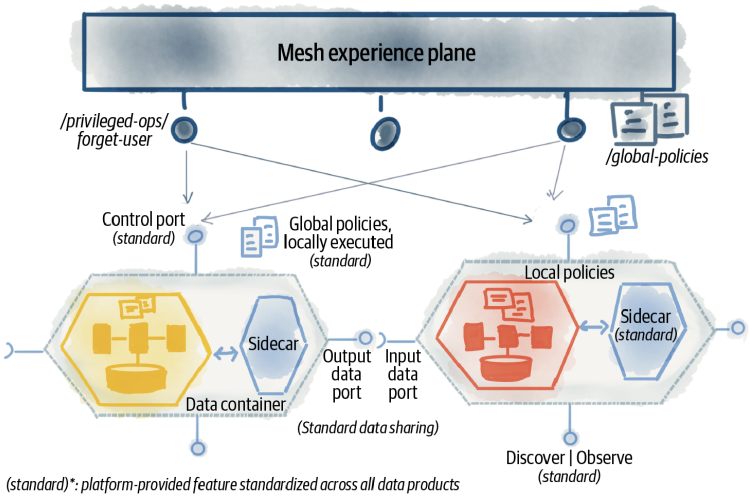

Figure 2.7: The logical architectural components of embedded computational policies

2.3.2 The Multiplane Data Platform Architecture

As we have discussed previously, the multiplane data platform consists of three planes, with the interactive diagram as figure 2.8.

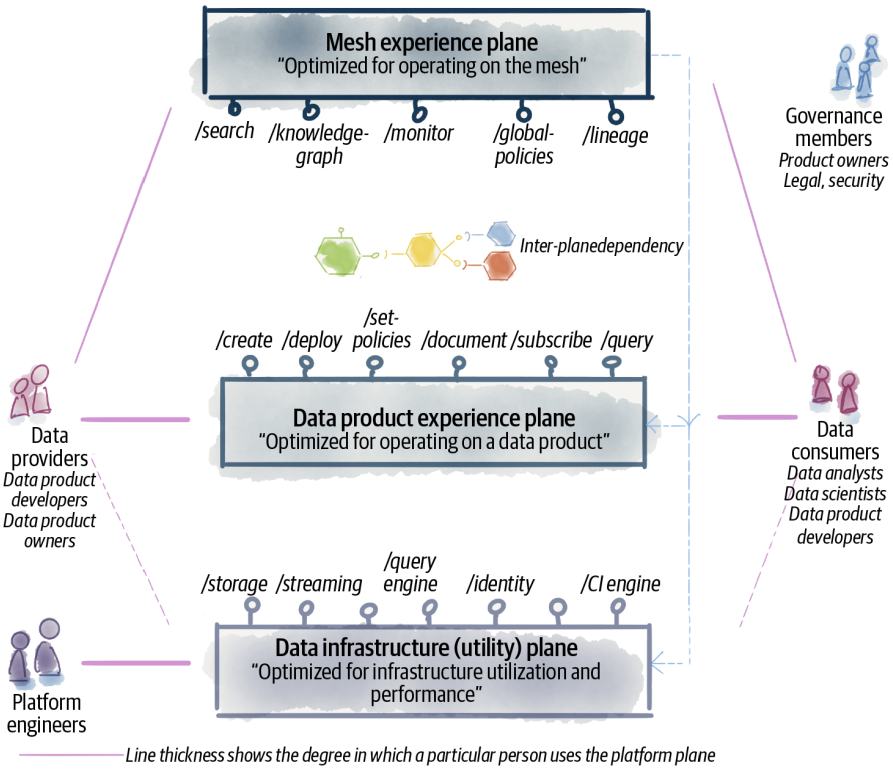

Figure 2.8: Multiplane self-serve platform and platform users

The data product experience plane provides operations on a data product and manages the complexity of provisioning its underlying infrastructure, while the infrastructure utility plane optimizes the resources' performance and utilization to get the best out of the underlying infrastructure providers. The most important idea is to understand the main journeys of platform users and evaluate how to make it easy for them to complete.

Data Product Developer Journey

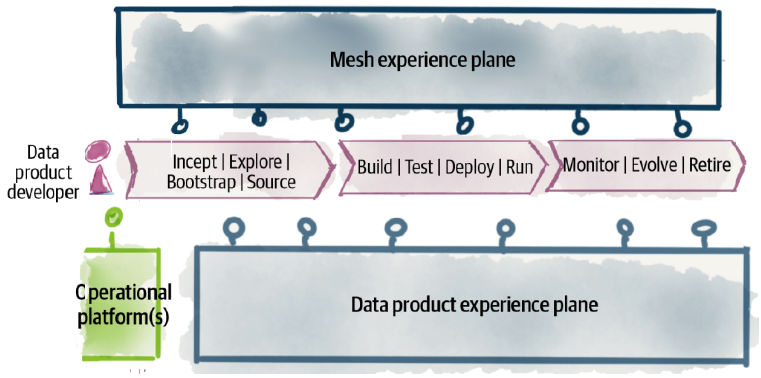

Figure 2.9: High-level example of data product development journey using the platform

Creating and operating a data product is one of the most important journeys of a data product developer. This is a comprehensive and long-term obligation that adheres to the continuous delivery premise of "everyone is responsible." A data product development journey interacts with other journeys. Let's look at the figure 2.9 for an example of high-level stages of data product development and how the platform interfaces are designed to support them.

We should go deeper to consider each stage of the platform planes and interfaces

Table 2.3: Example interfaces provided by the platform planes

Phase	**Platform plane**	**Platform interface**	**Platform interface description**
Data product inception			
Incept | Explore	Mesh experience	`/search`	Search existing data products based on parameters such as operational systems, domains, and types of data.
		`/knowledge-graph`	Browse the mesh of related data products' semantic models. Traverse their semantic relationship to identify the desired sources of data.
		`/lineage`	Traverse the lineage of input-output data to identify desired sources based on origin and transformations.
Bootstrap | Source	Data product experience	`/{dp}/discover`	Once a source is identified, access all the data product discoverability information such as documentation, data model, available output ports, etc.
		`/{dp}/observe`	Access data product guarantees and metrics in real-time, such as release frequency, last release date, and data quality metrics.
		`/init`	API bootstraps a barebones data product with infrastructure to connect to sources, access data, run transformations, and serve output in a single access mode.

continued on next page –

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
		`/connect`	The data product gets access to the source by connecting to it, validating access control policies and triggering a request for permission.
		`/{dp}/{output}/query` `/{dp}/{output}/subscribe`	Read data from a particular output port of the source data product, either in a pull-based querying model or subscribing to changes.
	Mesh experience	`/register`	Data products are registered with the mesh and given a unique global identifier and address, making them visible to the governance process.
Data product development			
Build	Data product experience	`/build`	Compile, validate, and compose components of a data product into a deployable artifact. See Table 2.3 for the data infrastructure plane interfaces used during this phase.
Test		`/test`	Testing capabilities are offered in different deployment environments to test various aspects of a data product, such as data transformation, integrity, versioning, data profile, and bias.
Deploy		`/deploy`	Deploy a data product to multiple environments, including local, development, pre-production, and production.
Run		`/start` `/stop`	Run or stop running the data product instance in a particular environment.
Build/ Test/ Deploy/ Run		`/local-policies`	The platform facilitates configuration and authoring of these policies locally, during the development of a data product, their validation during test, and their execution during access to data.
Build/ Test/ Deploy/ Run	Mesh experience	`/global-policies`	The platform enables authoring of the global policies and application of these policies by all data products.
Data product maintenance			
Maintain, Evolve, and Retire	Data product experience	`/{dp}/status`	Checking the status of a data product.
		`/{dp}/logs` `/{dp}/traces` `/{dp}/metrics`	Data products emit runtime observability information such as logs, traces, and metrics to be used by mesh layer monitoring services.

continued on next page –

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
		`/{dp}/accesses`	The logs of all accesses to the data product.
		`/{dp}/cost`	Tracking the operational cost of a data product. This can be calculated based on the resource allocation and usage.
		`/migrate`	Updating a data product revision is a simple process of build and deploy.
		`/{dp}/controls`	Data products are registered with a unique global identifier and address, making them visible to governance.
	Mesh experience	`/monitor`	Multiple monitoring abilities at the mesh level, logs, status, compliance, etc.
		`/notifications`	Notification and alerting in response to detected anomalies of the mesh.
		`/global-controls`	High-privilege administrative controls can be invoked on data products.

The data infrastructure plane is a key enabler in lowering the cost and effort required to code or configure components of a data product. Developers interact with the data product experience plane to build, deploy, and test a single data quantum. The data product experience plane delegates the implementation of the data product components to the data infrastructure plane.

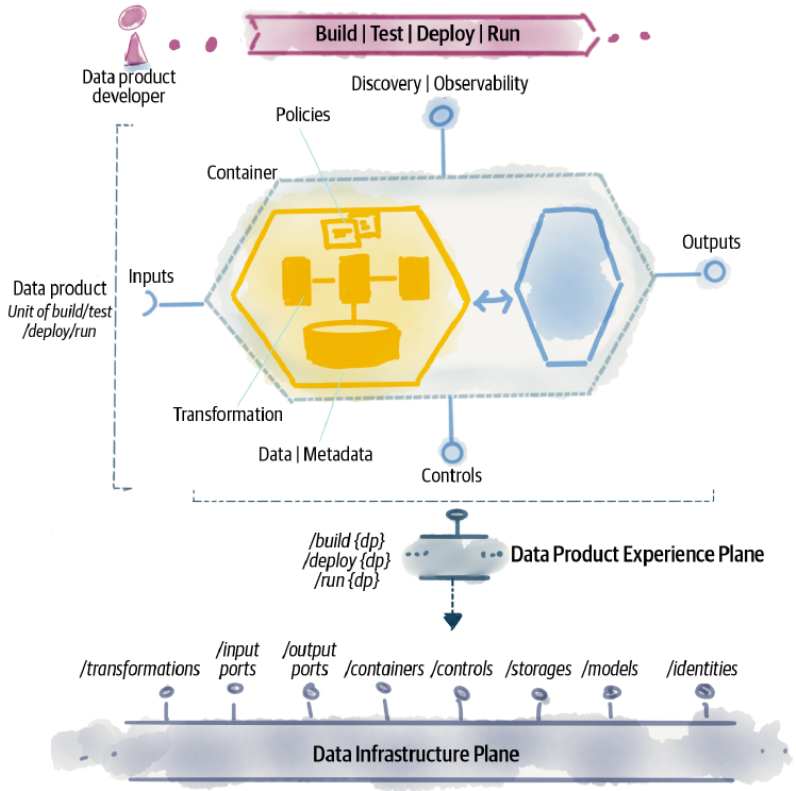

Figure 2.10: Example of data infrastructure plane to support data product delivery

Data Product Consumer Journey

A data consumer is a user with various skills and responsibilities, such as consuming existing data products to train an ML model and then deploying it as a data product. ML models can be deployed as a microservice or as the transformation logic of a data product.

The data scientist's journey to continuously deliver an ML model as a data product follows the practice of continuous delivery for ML (CD4ML[3]) to continuously hypothesize, train, deploy, monitor, and improve the model in rapid feedback loops. The figure 2.11 illustrate this journey.

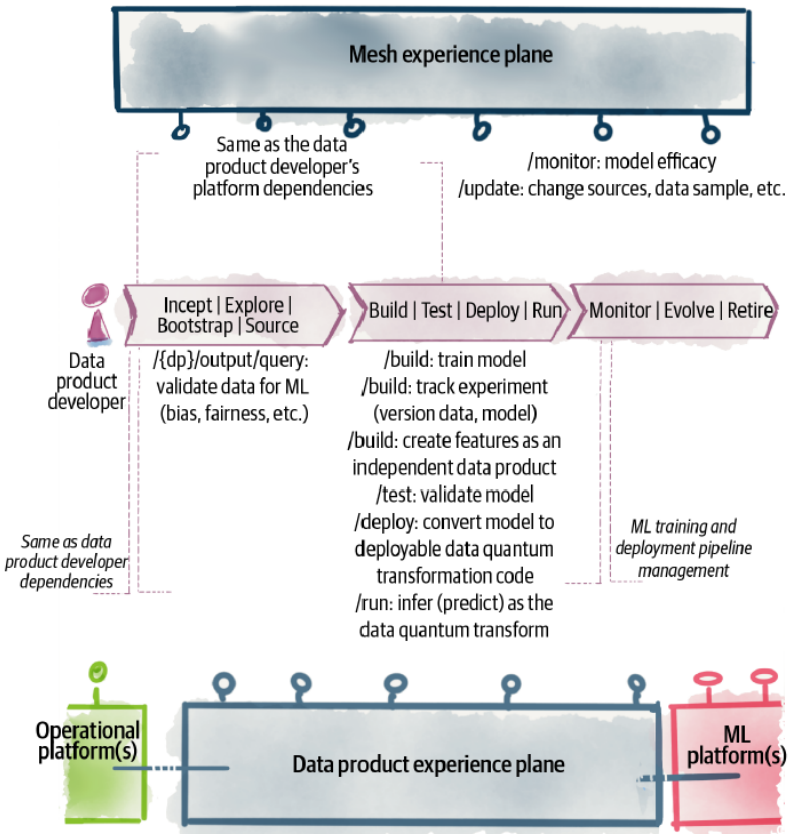

Figure 2.11: Example of ML model development journey

In conclusion, there is no single entity such as a platform but rather a set of well-integrated services, such as APIs, SDKs, and libraries that provide a seamless experience for users. Additionally, the separation of planes should be respected to optimize both for the human experience and machine efficiency.

[3]This process is also known as MLOps; however, I use the CD4ML notation since it overlays and integrates closely with continuous delivery of software and data products.

Chapter 3

DATA PRODUCT DESIGN & IMPLEMENTATION

3.1 WHAT IS DATA PRODUCT AND ITS CANVAS

Data product is a core component in data mesh architecture. Conceptually, a mesh is a graph, or network, composed of nodes and connecting edges. Each node in a data mesh is referred to as a data product. Every data product resides within a confined context, and a single bounded context may contain multiple data products.

Data product can be everything that has value for consumers (e.g., a database table, raw unstructured files, dashboard, data stream, etc.). It contains all data, code, and interfaces to serve analytical data needs. Data product connects to sources, such as operational systems or other data products and perform data transformation. It serves data sets in one or many output ports. [3]

A Data Product Canvas is a visual framework that guides a team through the data product specification. In total, the canvas consists of ten building blocks:

- **Domain:** Data products should belong to one domain team, with accountability, requirements, questions, and fixes.
- **Data Product Name:** This should follow a common naming strategy.
- **Consumer and Use Case:** Data product design follows "Product Thinking" to identify consumer needs.
- **Output Port:** Output ports define the format and consumption protocol for data.
- **Metadata:** This includes data product ownership, schema, semantics, security.
- **Input Ports:** Input ports define data format and protocol for reading.
- **Data Product Design:** Design a data product on a conceptual level by specifying data ingestion, storage, transport, etc.
- **Observability:** This includes quality, operational metrics, and service level objectives to build trustworthiness in data products.
- **Ubiquitous Language:** Common language shared for operational systems and data products.
- **Classification:** Classifying data as source-aligned, aggregated, or consumer-aligned.

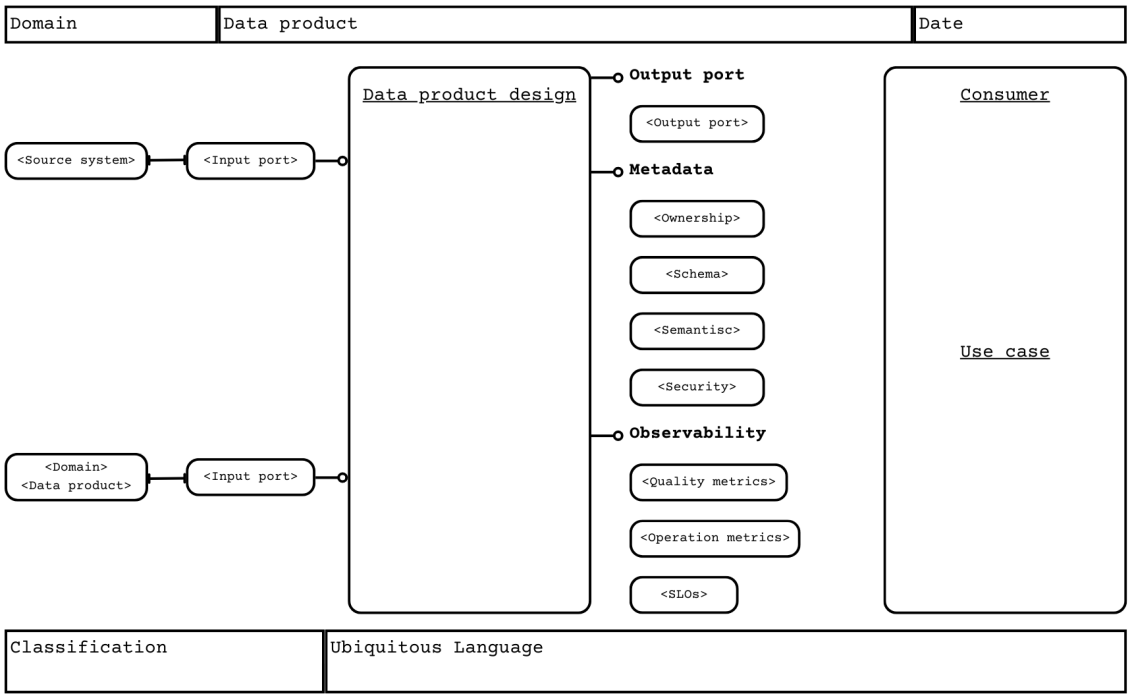

Figure 3.1: Data Product Simple Canvas

3.2 Design a Data Product by Affordances

Data mesh is a distributed scale-out architecture that designs data products as self-contained, autonomous, and coequal nodes, providing users and developers with the capabilities they need to do their job: discover, understand, use, build, govern, and debug.

In addition to four common constituents of architecture[1], we introduce and focus on designing a data product by another element: affordances [13].

Table 3.1 lists affordances for a data product to autonomously transform and serve meaningful, understandable, trustworthy data that can be connected to other data in the ecosystem. It has all the structural components to do its job, and its dependencies to other data products or platform services are through APIs with explicit contracts, creating loose coupling.

Table 3.1: Data product affordances

Affordances	**Description**
Serve Data	The data product shares immutable and bitemporal data, allowing users to access it for multiple purposes, but not for transactional and operational applications.
Consume Data	The data product consumes data from upstream sources, but only from sources it identifies and configures through the platform, not from sources it does not identify and configure.
Transform Data	The data product transforms input data into new data, which can be program code, a machine learning model, or a complex query. It can generate new data or improve the quality of input data.

continued on next page –

[1]Structure, Characteristics, Decisions, Principles

Table 3.1 – *continued from previous page*

Affordances	Description
Discover | Understand |Explore | Trust	The data product serves APIs and information that affords data product users to discover, explore, understand, and trust it.
Compose Data	The data product allows users to compose, correlate, and join data with other data products, while the data quantum provides programmatic data composability by performing set operations.
Manage Life Cycle	Data products provide a set of configuration and code to enable developers to build, provision, and maintain them.
Observe | Debug | Audit	The data product provides programmatic APIs to enable users to monitor its behavior, debug issues, and audit it.
Govern	The data product provides data users and the mesh experience plane with APIs and policies to self-govern their data, enabling build-time configuration and runtime execution of policies. It also maintains data security and protects privacy and confidentiality through encryption.

Data products must be designed to meet the data mesh objective of "getting value from data at scale", that means, we have to satisfy:

- Design for change: Design for change is essential for data products to respond gracefully to change, such as fronting aspects with APIs or adding time.
- Design for scale: A data product must scale out while maintaining agility and speed of access.
- Design for value: Data product interfaces should focus on sharing easily understandable, trustworthy data with minimal friction.

The final thing is design simple and local rules and behaviors in data products to create mesh-level knowledge and intelligence.

3.3 CONSUMING, TRANSFORMING & SERVING DATA

A data product's primary job is to consume data from upstream sources using its input data ports, transform it, and serve the result as permanently accessible data via its output data ports.

3.3.1 Serve data

Data products serve domain-oriented data to diverse consumers through output data ports and APIs. Serve data design has some properties:

- Multimodal data: Data products serve data of a domain with a specific and unique domain semantic. They can be served in different formats. When a user accesses a data product, they first learn about its semantic and then access one of its output APIs.
- Immutable data: Immutable data is a key axiom in functional programming. This is relevant to analytical use cases, as it allows users to rerun analytics in a repeatable way.
- Bitemporal data: Bitemporal data modeling records two timestamps: actual time and processing time, making it possible to serve data as immutable entities and perform

temporal analysis and time travel.

Table 3.2: High-level data product components to serve data

Component	Description
Output data port	APIs are used to serve data according to a specific mode of access for a particular spatial format.
Output (data) port adapter	The code responsible for presenting data for a particular output port is either a step in the data product's transformation code or a runtime gateway.
Core data semantic	Expression of the data semantic - agnostic to the modes of access or its spatial syntax.

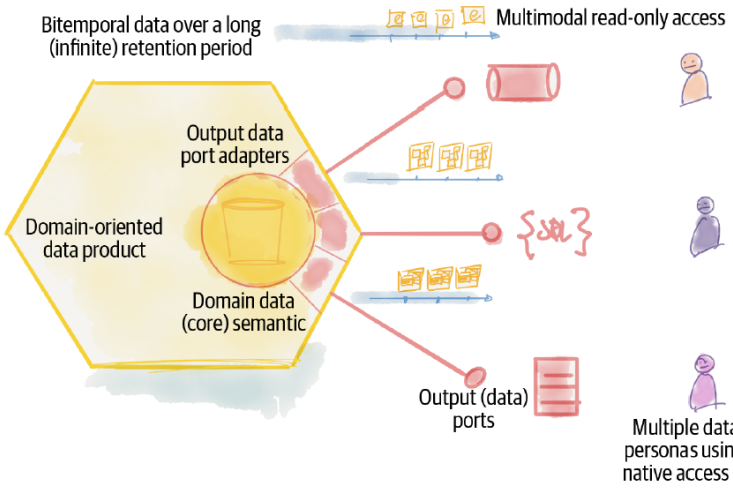

Figure 3.2: Data product's high-level components to design serve data

3.3.2 Consume Data

In most of cases, data in organizations originates from internal or external systems, and data products consume it from one or multiple sources. Input data ports are a logical architectural construct that allow a data product to connect to a source, execute queries, and receive data as a continuous stream or one-off payload. Consume data design has some notable characteristics that impact the design of a data product's input data:

- Archetypes of data sources: collaborate operational systems, data products or itself.
- Locality of Data Consumption.

Table 3.3: High-level components to design consume data

Component	Description
Input data port	The mechanism by which a data product receives its source data and makes it available for internal transformation.
Input data port specification	A declarative specification of input data ports that configures from where and how data is consumed.
Asynchronous input data port	Asynchronous input ports reactively execute transformation code when necessary source data is available.

continued on next page –

Table 3.3 – *continued from previous page*

Component	Description
Remote query	An input port specification can include a query executed on the source to receive desired data, reducing the amount of data fetched redundantly.
Input port synchronizer and temporary storage	Temporary storage is necessary to keep track of observed and unprocessed data until all observations are available for processing.

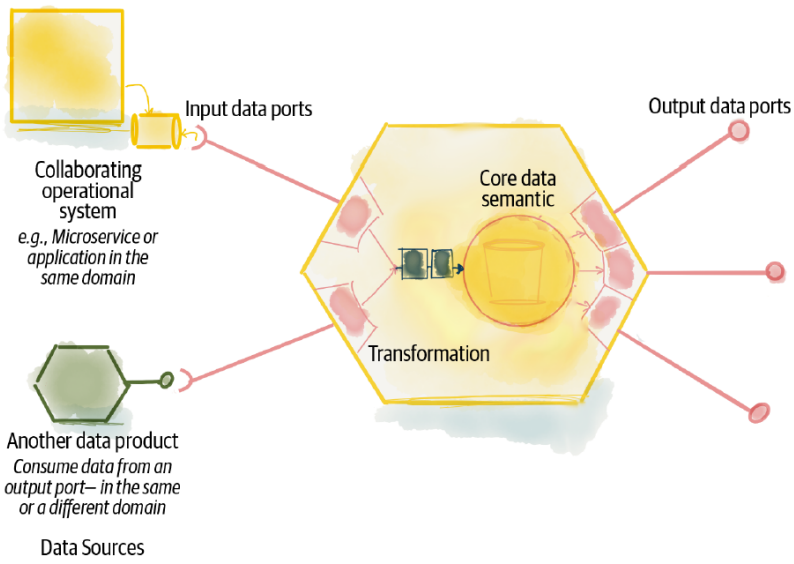

Figure 3.3: Data product's design to consume data

3.3.3 Transform Data

Data products are created to share a new analytical model of existing data. Transformation is an internal implementation of a data product and is controlled by it. It is up to the data product developer to choose how to implement the transformation, and it is helpful to look at a few different ways of implementation:

- Nonprogrammatic: Nonprogrammatic transformations use relational algebra or flow-based functions to capture the intent of how data is transformed, but are limited to the features of the statement.
- Programmatic: Programmatic data processing uses code logic, conditions, and statements to transform data, allowing it to be modularized and tested, making it more extensible.
- Dataflow-Based: Dataflow programming is a natural paradigm for implementing transformation code, as long as the pipeline stages don't extend beyond the boundary of a data product.
- ML Model: ML models can be deployed in many contexts to make predictions and store recommendations for playlist extension.
- Time-Variant: Transformations respect the axes of time, processing and actual, by keeping track of processing time and generating output with actual time.

Table 3.4: High-level components to transform data for a data product

Component	Description
Transformation artifact(s)	Code, configuration, statement, or model executes transformation on input data.
Transformation runtime environment	Transformations require a computational environment to execute, provided by the underlying platform.
Temporary storage	Platform provides temporary storage for transformation code steps.

3.4 Discovering, Understanding & Composing Data

3.4.1 Discover, Understand, Trust, and Explore

Data products are responsible for bridging the gap between the known and unknown, so users trust them. Interfaces and behaviors are designed to share information about data semantics, formats, usage documentations, statistical properties, expected quality, timeliness, and other metrics. ML-based discovery functions can be built to search, browse, examine, and get meaning from the mesh of connected data products.

To satisfy the above design, we need to follow these steps:

1. Begin discovery with self-registration,
2. Discover the global URI,
3. Understand semantic and syntax models,
4. Establish trust with data guarantees,
5. Learn with documentation.

The data product sidecar is a platform-provided agent that runs within the computational context of a data quantum and is responsible for cross-cutting concerns that need to be standardized across all data products. Figure demonstrates the data product sidecar's interactions in support of discoverability.

3.4.2 Compose Data

The design considerations for a system of distributed data models that enables data composability and quick evolution without building tightly connected or centralized models are the most essential details in this article. Ownership of a time-variant, sharable, and referenceable semantic data model, semantic model linking, a global identification system to map data across data products, and the responsibility of data product owners to continuously look for opportunities to discover and create meaningful relationships with other data products are among the design considerations. These design principles are required for individual data products to generate higher-order intelligence and understanding.

The design prioritizes loose coupling between data products and minimizing centralized synchronization points. Data composability across the mesh is based on a distributed type system (schemas) where each data product owns and controls its own life cycle.

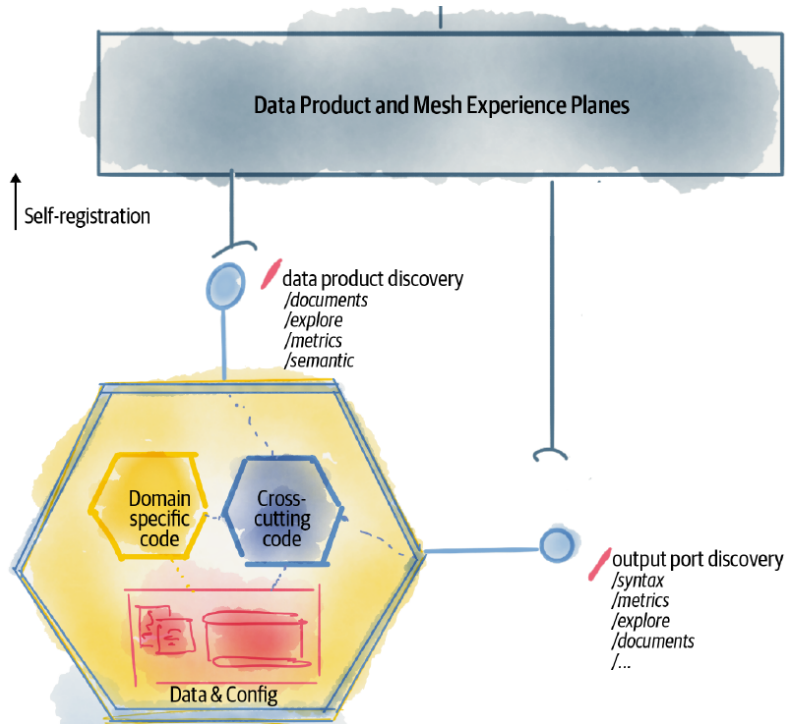

Figure 3.4: A data product sidecar provides a unified model of discoverability for all data products

3.5 MANAGING, GOVERNING & OBSERVING DATA

3.5.1 Manage the Life Cycle

A data product's manifest is the specification of its target state, which can change and evolve during its lifetime. Developers create two groups of artifacts: source artifacts and data quantum manifest, which are used to generate the build and runtime artifacts, provision necessary resources, and run the data product container. The declarative modeling of a data product hides complexity, allowing developers to communicate various facets and properties of a data product in a standard fashion.

Table 3.5: High-level components to transform data for a data product

Component	Description
Data product's URI	Data product's globally unique identifier.
Output ports	Platform declares output ports, access modes, and guarantees.
Output port SLOs	Ports must declare the service level agreements they guarantee.
Input ports	Declaration of input ports, data sources, and retrieval methods.
Local policies	Configuration of local policies such as locality, confidentiality, etc.
Source artifacts	Data product includes transformation code and input ports queries.

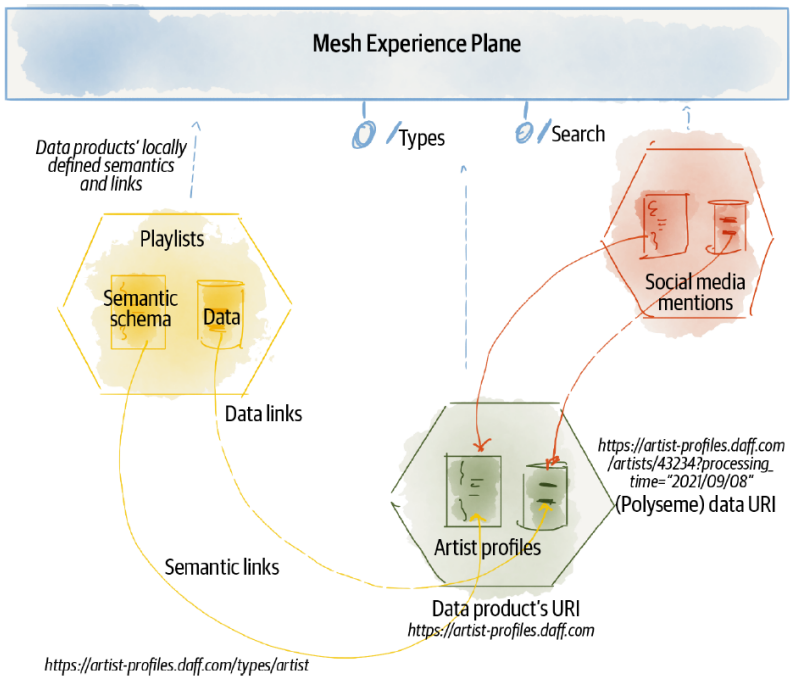

Figure 3.5: Data product's high-level components to design data composability

3.5.2 Govern Data

Data products are architectural components that enable configuration and execution of policies. These components include a control port, data product sidecar, and data product container. These components work in collaboration with the mesh experience plane, which provides capabilities for governance processes such as configuring policies and the right to be forgotten globally across the mesh. Governance of policies requires observability of their state of adoption, which can be domain-specific or agnostic. Successful implementation of data mesh governance requires design characteristics such as conformance to HIPAA and guaranteed data accuracy levels.

3.5.3 Observe, Debug, and Audit

Data product observability is the ability to infer the internal state of data products by observing their external outputs. The distributed architecture of data mesh creates complexity in observability, as there are many moving parts that can fail and failures may go unnoticed.

The use cases for data product observability can be summarized as follows:

- Monitor the operational health of the mesh,
- Debug and perform postmortem analysis,
- Perform audits,
- Understand data lineage.

Data mesh observability begins with each data product sharing its external output and reporting its status, allowing the experience plane to observe and monitor the mesh-level status. Some notable design characteristics of data products enable observability:

- Observable outputs: Distributed architectures use logs, traces, and metrics to enable observability.

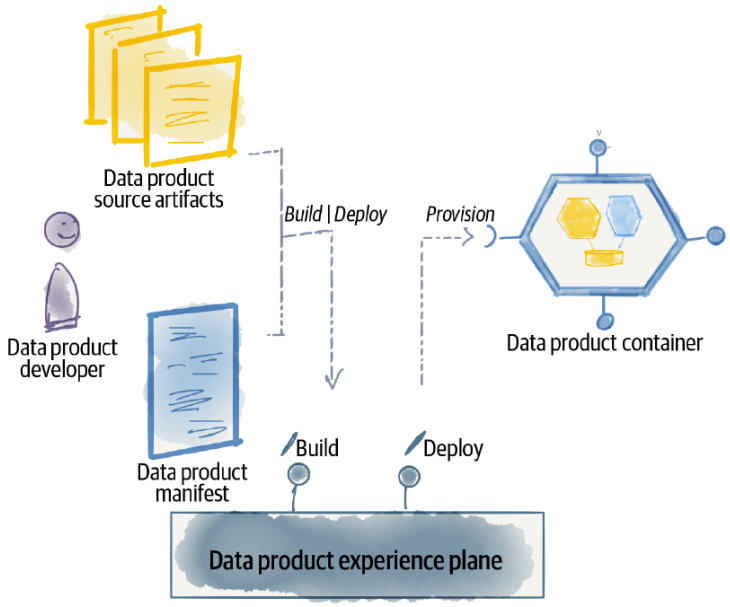

Figure 3.6: High-level interactions to manage the life cycle of a data product

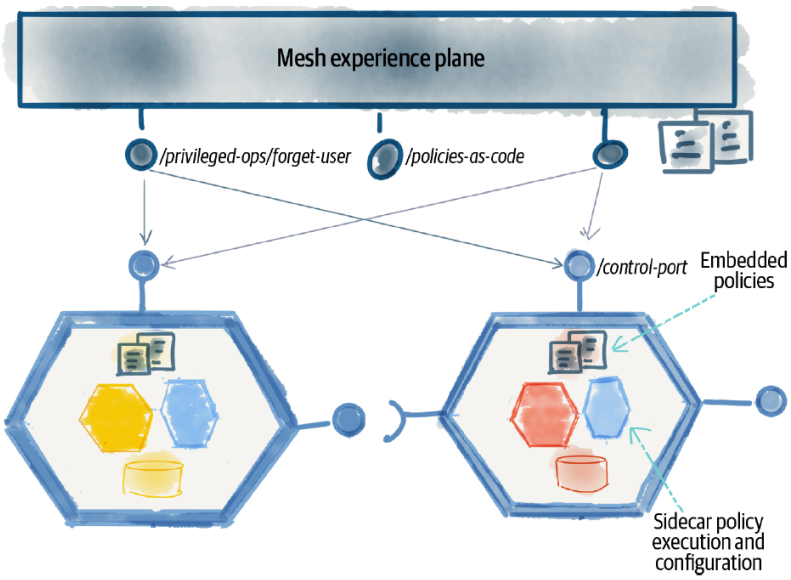

Figure 3.7: High-level design of embedded policies as code

- Traceability across operational and data planes: Observability of data must extend beyond data products to operational systems to provide a complete picture of data lineage and root cause analysis.
- Structured and standardized observability data: Data product metrics, logs, and traces must be structured and standardized to create higher-order intelligence and insights.
- Domain-oriented observability data: Domain-oriented design can scale and extend observability by creating domains of observability and managing them as data products.

Chapter 4

DATA MESH IN USE

4.1 FRAMEWORKS AND TECHNOLOGIES FOR DATA MESH

Data mesh is primarily an organizational approach, and that's why you can't buy a data mesh from a vendor. Technology, however, is important still as it acts as an enabler for data mesh, and only useful and easy to use solutions will lead to domain teams' acceptance. There are some tools and platforms for installing data mesh, such as Google Cloud Platform, AWS Platform and Data Build Tool (dbt) with Snowflake. [3] For an easy implementation and imagination, I suggest you to use Data Mesh Manager to build the concept of data mesh.

I have created a small comparison between these tools for an easy comprehension between them. Note that, all of them can connect with some current project of domain teams and extensions such as Apache Kafka, Apache Airflow.

Table 4.1: Comparison between tools for data mesh

Google Cloud Platform	**AWS Platform**	**dbt & Snowflake**
Use BigQuery, Dataplex, etc.	Use AWS S3 and AWS Athena.	Use dbt and Snowflake warehouse, stage, etc.
A common infrastructure, highly integrated platform, at least for analytical data.	The primary means to share and query data products.	dbt is the default framework to engineer analytical data, Snowflake is a data warehouse.
Every domain team typically gets their own GCP project under an organization resource.	Every domain team typically has their own AWS S3 buckets to store their own data products.	Every domain team manages their own dbt projects and CI pipelines to run the models.

Here, for the purpose of an enterprise, I will focus on implementing data mesh using Google Cloud Platform with supports from Apache Kafka, PostgreSQL, and other extensions. Other tools can be find a simple description in [3]

Dataplex is a tool that enables us to manage assets at scale and define data domains within lakes and zones virtually layered over the actual physical location of the asset. **BigQuery** is a data processing ecosystem that allows us to process terabyte- to petabyte-scale datasets in seconds. It also provides direct access to data residing in **GCS Buckets**. [14]

Google Cloud enables business users to create insights on the fly with self-service analytics

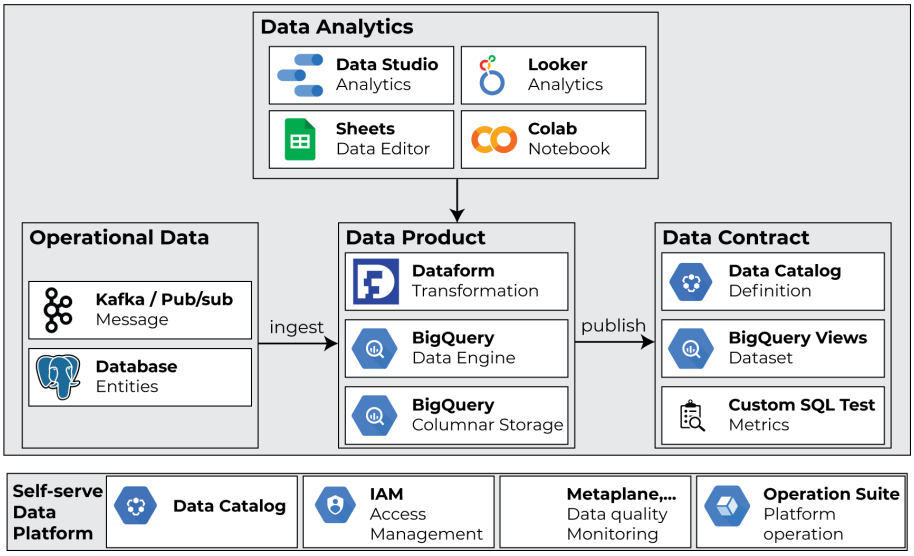

Figure 4.1: Data Mesh Architecture with Google Cloud Platform

using familiar tools such as SQL or BI. The flexible, on-demand compute power of BigQuery means that ad hoc analysis does not compete with scheduled reporting. We can focus valuable engineering bandwidth on developing data into strategic assets for our organization, in turn allowing our business users to focus on generating business value from this data. Figure 4.2 is one example of how Google Cloud might offer a serverless and integrated analytics architecture.

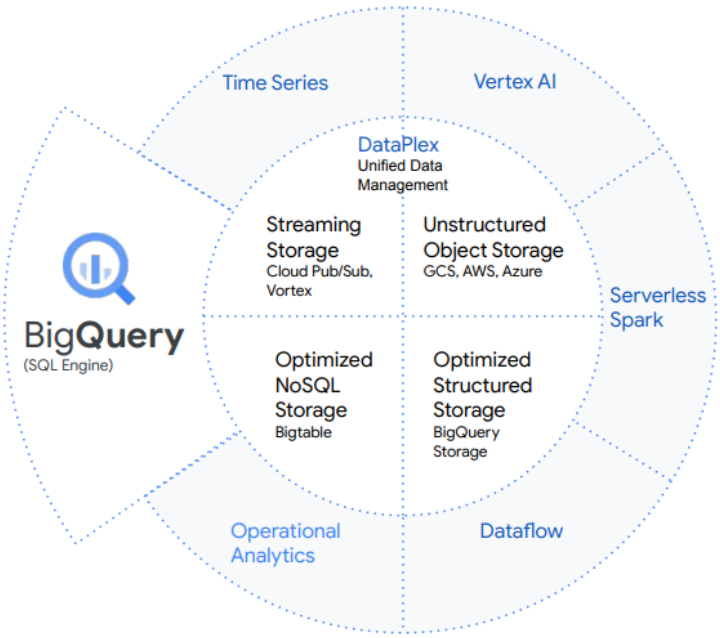

Figure 4.2: Google Cloud Platform supports for data mesh

Google Cloud allows using different execution engines for diverse workloads and user needs on shared data tiers. The separation of compute and storage enables this flexibility. Users can access data using SQL, Python, or GUI-based methods. Data Catalog, Dataproc Metastore, and BigQuery Metastore ensure data access from anywhere with any query en-

gine. The process involves onboarding data to a Dataplex lake, triggering discovery and publishing in BigQuery and Dataproc Metastore. Tools like Vertex AI notebooks, Spark on Dataproc, BigQuery, or Serverless Spark integrate seamlessly. With BigQuery and Serverless Spark, data engineers focus on code and logic without managing clusters. They submit SQL or PySpark jobs, automatically scaled as needed.

Dataplex is a managed service that groups data containers from Google Cloud, allowing hierarchical organization. It offers a unified platform to curate, catalog, secure, integrate, and explore data at any scale. With Dataplex, you can quickly build data lakes without resource concerns. It enhances automatic data discovery and schema inference across systems, seamlessly representing resources in the Data Mesh.

Dataplex enhances this process by automatically registering metadata as tables and filesets as metastores and Data Catalog. Additionally, it tightly integrates with our Cloud Data Loss Prevention API (DLP) and includes built-in data quality checks, making it easier to tag sensitive data. With Dataplex, implementing a Data Mesh with simplified security controls becomes possible, as shown in figure 4.3. It provides consistent security policies and enforcement for Cloud Storage and BigQuery, enabling central governance teams to audit the environment. Moreover, it offers managed Data Lake Storage with precise access control, ACID transactions on files, and a unified interface for managing BigQuery.

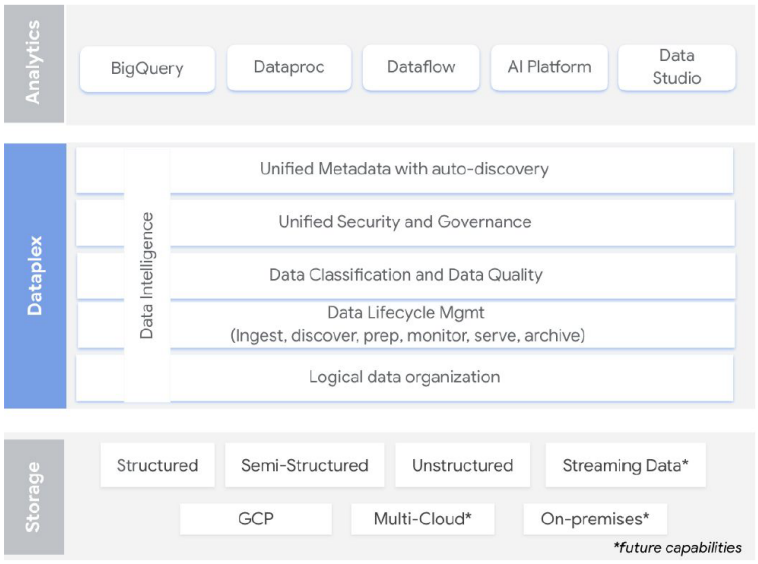

Figure 4.3: Single pane of glass - Dataplex

Dataplex provides a data management and governance layer for organizing data across BigQuery and Cloud Storage. It enables centralized governance, empowering data administrators to establish workspaces, manage access, and control costs.

For data scientists, Dataplex simplifies notebook access and discovery, making it easy to save and share notebooks alongside associated data. Data analysts benefit from the SQL Workspace, eliminating the need for multiple data processing environments.

Google Cloud offers an integrated experience across its data analytics services, providing virtual Lakehouse capabilities. Serverless Spark and a serverless notebook experience are designed for data science. Data stored in Cloud Storage is accessible for querying through OSS tools and BigQuery. Data Catalog enables search and discovery across the data landscape.

4.2 CASE STUDY AND DEMO

4.2.1 Data contracts in PayPal Inc.

As we can see in the figure 2.3, data contract includes six main components. In PayPal Holdings Inc., they have built their own data contract that have published the example and template based on YAML data contract. There are eight main sections in the contract, including a catch-all section. The pricing part is a surprising addition.

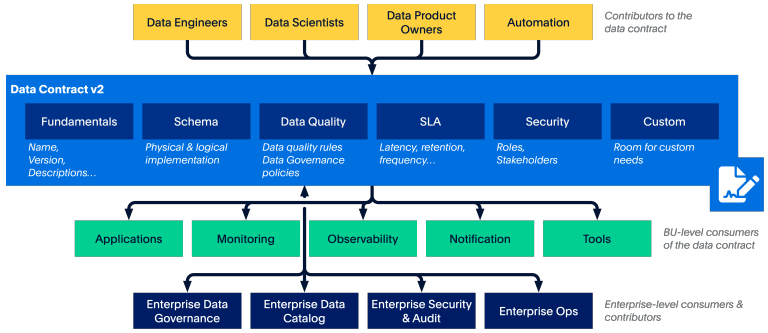

Figure 4.4: Data Contract example in PayPal Holdings Inc.,

This contract encompasses a dataset containing multiple tables, each with various details at the table and column levels. Data engineers can establish data quality (DQ) rules at different levels, which are not tied to any specific tool. They have the option to use either their own custom DQ tool or tools like Great Expectations. The pricing aspect of this version is still experimental.

Stakeholders have an important role in building trust in a particular data product. The contract provides information about the stakeholders and their project history. While tribal knowledge may still exist, the contract helps track its evolution.

I have reconstructed PayPal's data contract and uploaded it in my Github repository here.

4.2.2 Data mesh concept demo on Data Mesh Manager

This is a demo environment created on Data Mesh Manager with some data products, global policies, governance meetings, and teams. The data mesh is created for a stock company, with many domain team such as Marketing, Checkout, Product, Fulfillment, etc.

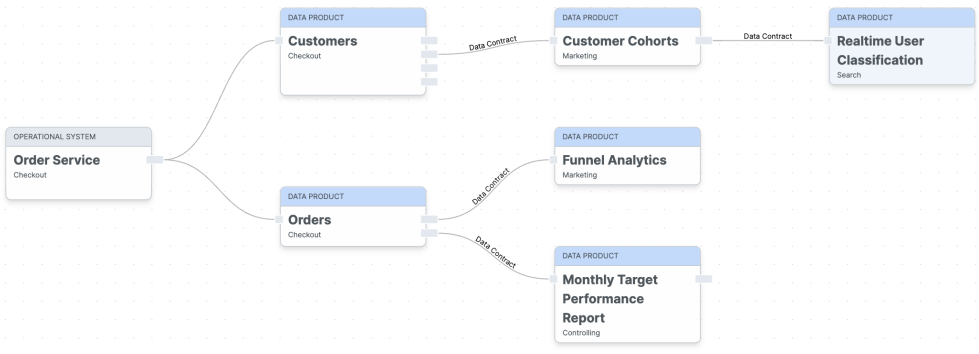

Figure 4.5: A part of data mesh demo in Data Mesh Manager

Chapter 5

CONCLUSION

This paper investigates and introduces the notion of "data mesh", a cutting-edge and game-changing architecture for data management and processing. We've dug deep into discovering the origins of data mesh, the main principles of data mesh, and how to properly build the data product, the fundamental component that makes up the data mesh. Data Mesh is a distributed and flexible data management approach in which domains are responsible for creating and distributing data in the form of data products. In addition, the paper recommends a few technologies to aid in the deployment of a data mesh architecture in the company.

Because data mesh is a relatively new concept, additional study and testing are required before it can be properly implemented. To change the way the entire organization runs, its culture, and its architecture, a significant amount of effort is required. Furthermore, the provision of infrastructure and the development of supporting technologies are important factors in being able to use Data Mesh. However, once the data mesh is in place, the benefits significantly surpass the initial challenges. With dedication and concentration, Data Mesh can be a significant step forward in data mining and application in the present digital era.

REFERENCES

[1] I. A. Machado, C. Costa, and M. Y. Santos, "Data mesh: Concepts and principles of a paradigm shift in data architectures," *Procedia Computer Science*, vol. 196, pp. 263–271, 2022.

[2] Z. Dehghani, "Prologue: Imagine data mesh," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. xxv–xxxviii.

[3] C. Jochen, V. Larysa, and S. Harrer. (2023) Data mesh from an engineering perspective. [Online]. Available: https://www.datamesh-architecture.com/

[4] D. Nicolas. (2023) Will the buzz around data mesh really help solve my data challenges? [Online]. Available: https://kpmg.com/be/en/home/insights/2023/03/lh-the-impact-of-data-mesh-on-organizational-data.html

[5] Z. Dehghani, "Data mesh in a nutshell," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 3–14.

[6] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[7] Z. Dehghani, "Principle of domain ownership," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 15–28.

[8] M. D. Wilkinson *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[9] Z. Dehghani, "Principle of data as a product," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 29–46.

[10] S. Michael. (2023) Data contracts – everything you need to know. [Online]. Available: https://www.montecarlodata.com/blog-data-contracts-explained/

[11] Z. Dehghani, "Why data mesh?" in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 93–137.

[12] ——, "How to design the data mesh architecture," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 139–190.

[13] D. A. Norman, *The Design of Everyday Things*. Basic Books, 2002.

[14] T. Firat, H. Thinh, P. Johan, C. Victor, and P. Susan, *Build a modern, distributed Data Mesh with Google Cloud*. Google Cloud, 2022.