DATA MESH ARCHITECTURE

MINI-PROJECT REPORT

DATA ENGINEERING SPECIALIZATION

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

SUPERVISOR: Mr. Nguyễn Chí Thanh

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.

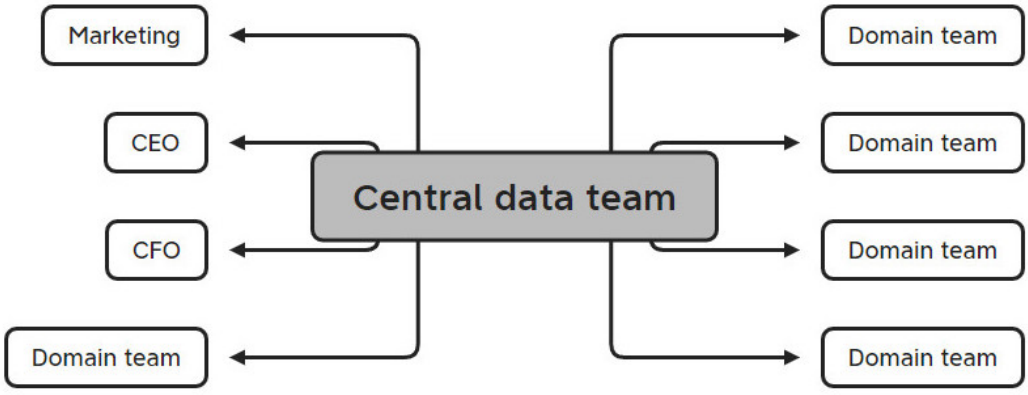

Figure 1.1: Central data team in a data-driven business

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]

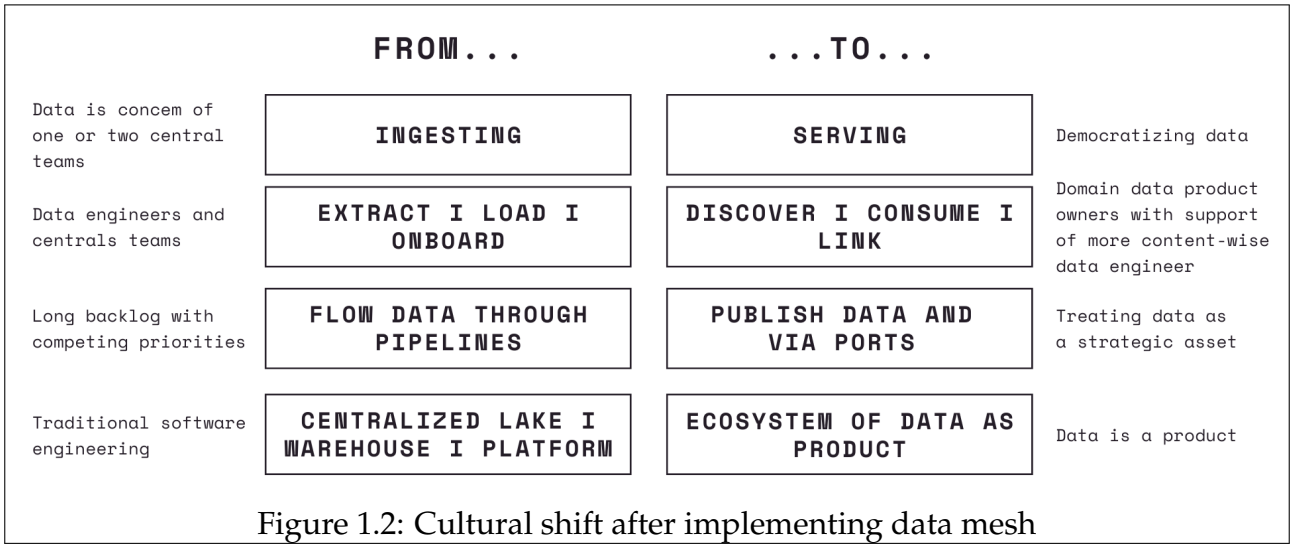

Figure 1.2: Cultural shift after implementing data mesh

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]

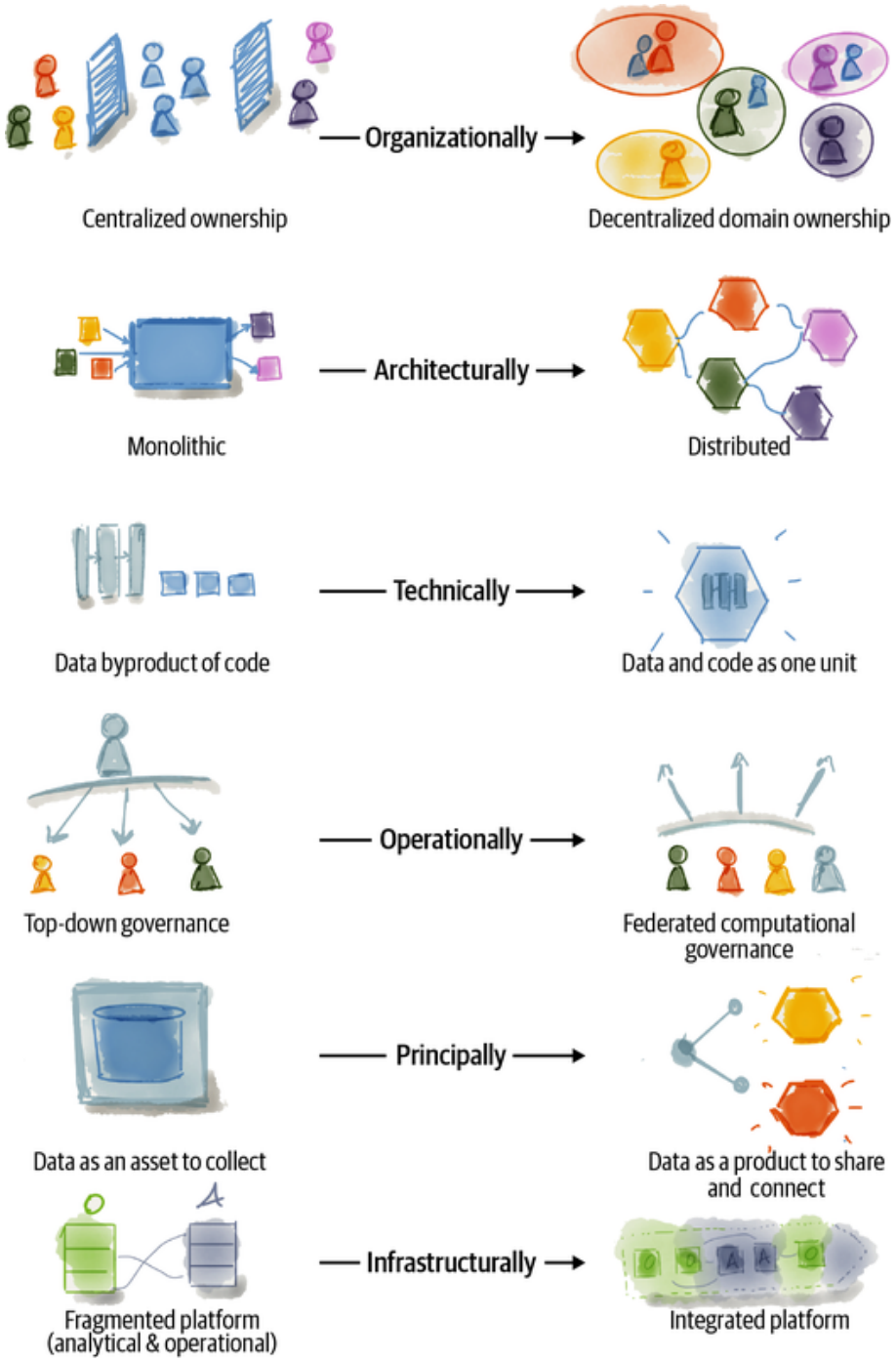

Figure 1.3: Data mesh dimensions of organizational changes

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:

[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]

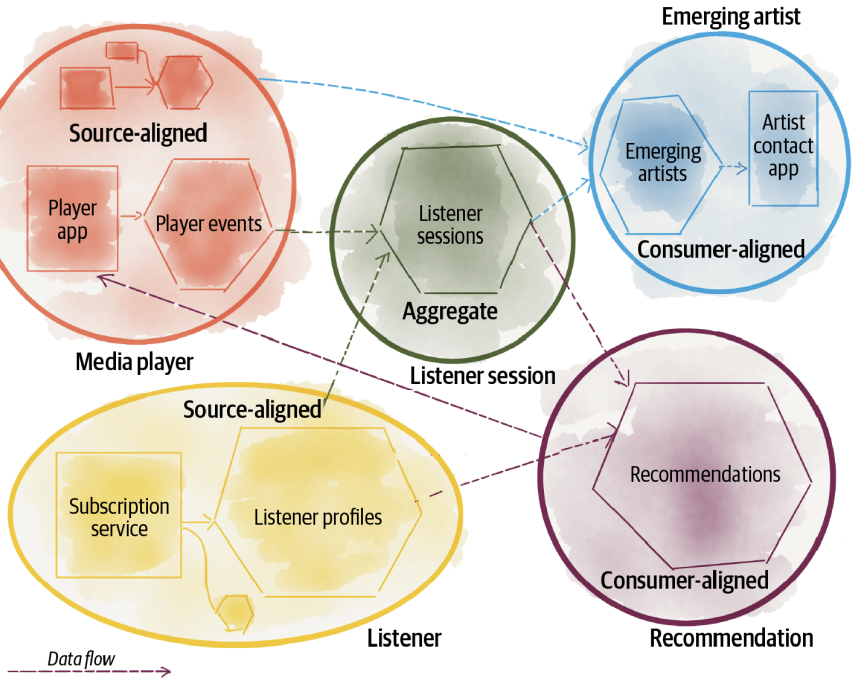

Figure 2.1: Decomposing the analytical data ownership and architecture, aligned with business domains, along with their data archetypes.

- Push Data Ownership Upstream.
- 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]

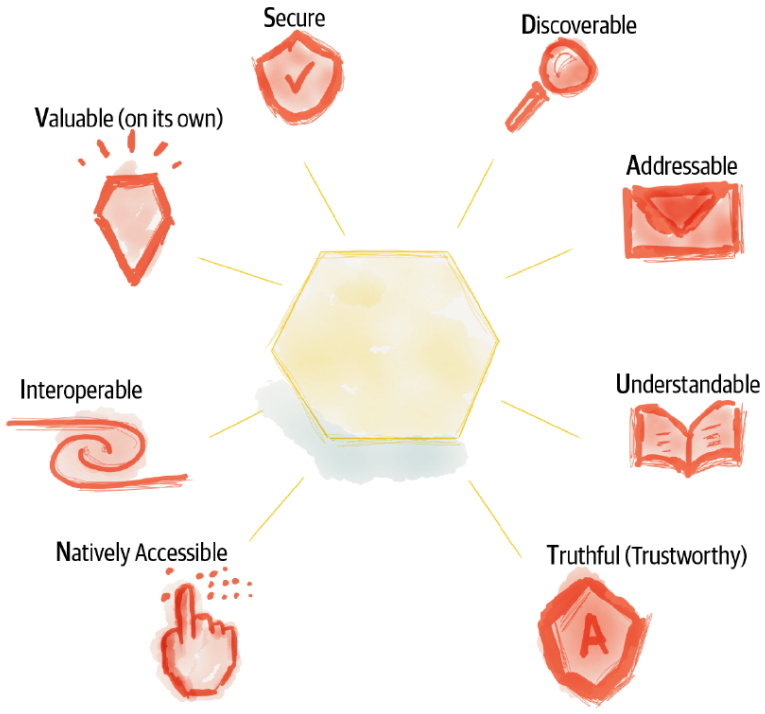

Figure 2.2: The baseline usability attributes of data products (DAUTNIVS)

2.1.3 Principle of the Self-Serve Data Platform

The principle aims to empower domain teams by hiding low-level complexity behind simpler abstractions and minimizing friction from their journeys to exchanging data products as a unit of value, using decentralized, interoperable methods.

2.1.4 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 2.3 shows an example of this 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.

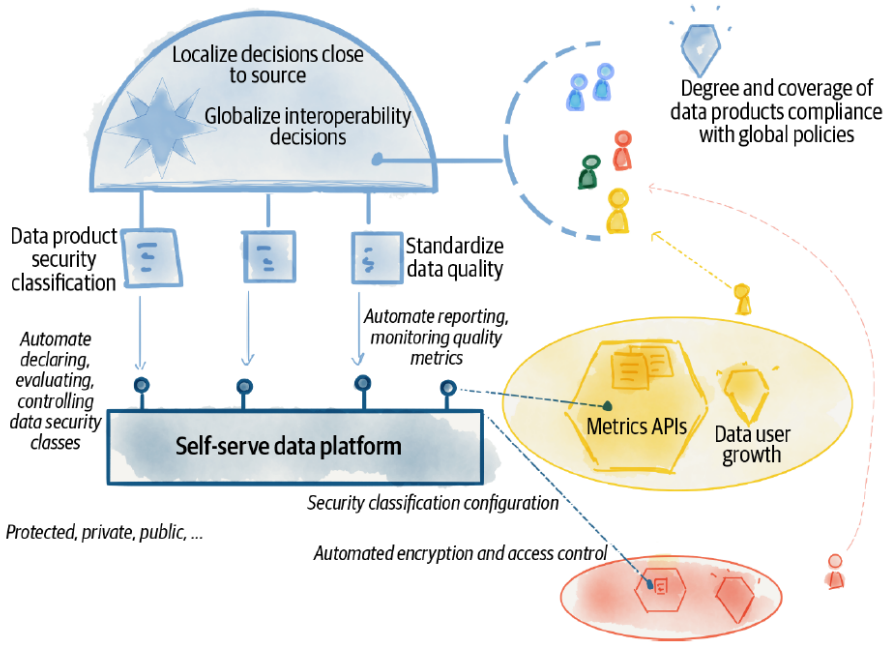

Figure 2.3: Example of data mesh governance operating model

2.2 WHY WE NEED DATA MESH?

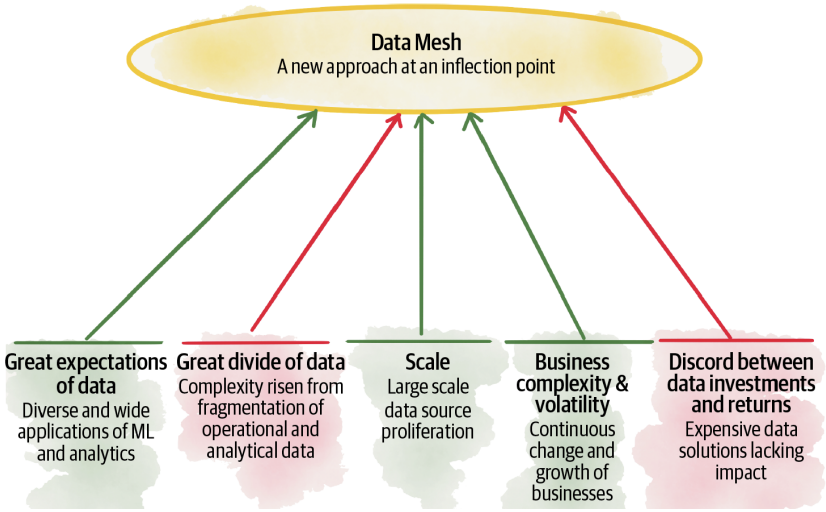

Figure 2.4: Macro drivers for the creation of data mesh

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.

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: [10]

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

Goal	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. *Principle of domain data ownership*
	Close the gap between the operational and analytical data planes	Dumb pipes should be used to integrate applications and data products. *Principle of data as a product*
	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 *Principle of data as a product*
	Reduce the accidental complexity of pipelines and copying of data	Break down pipelines, move transformation logic into data products. *Principle of data as a product* *Data product quantum architectural component*
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 *Principle of domain ownership* *Principle of data as a product*
	Reduce the coordination of data pipelines	Decomposition of pipeline architecture from functional to domain-oriented. Introduce explicit data contracts between domain-oriented data products. *Principle of domain ownership* *Principle of data as a product*
	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 *Principle of federated computational governance*
	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. *Principle of federated computational governance* *Principle of the self-serve data platform*

continued on next page –

Table 2.1 – *continued from previous page*

Goal	What to do	How to do it
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 *Principle of data as a product* *Principle of the self-serve data platform*
	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 *Principle of the self-serve data platform* *Principle of data 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 *Principle of data as a product* *Principle of the self-serve data platform*

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

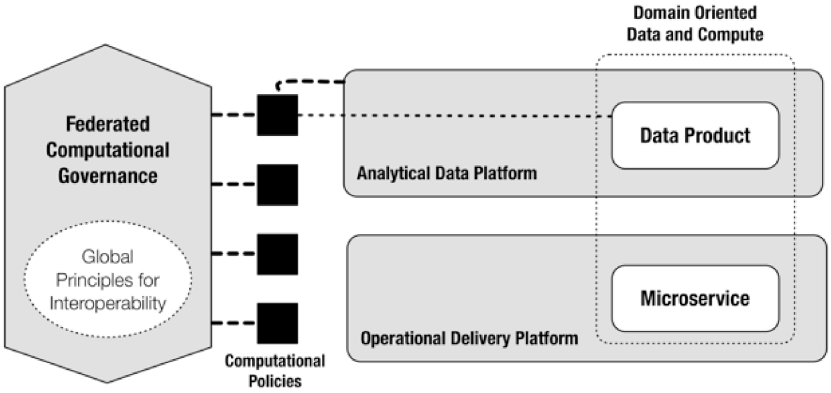

Figure 2.5: Data Mesh Logical Architecture [1]

Table 2.2: Data mesh logical architectural components and their maturity

Aspect	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. *This is a well-established concept.*
	Domain analytical data interfaces	Standardized interfaces that discover, access, and share domain-oriented data products. *Up to now, the implementation of these APIs is custom or platform specific.* 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. *This concept has mature implementations.* It is supported by de facto standards such as REST, GraphQL, gRPC, etc.
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. *At the time of writing this is an experimental concept with custom implementations.*
	Data Product Data Sharing Interactions	There are many technologies for implementing a data product's input and output 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. *At the time of writing this has custom implementations with existing event streaming and pipeline management technologies.* • **Output data port:** A data product's standardized APIs to continuously share data. *At the time of writing this has vendor-specific custom implementations.* *A mature implementation of the concept requires open data sharing standards with support for multiple modes of access to temporal data.*

continued on next page –

Table 2.2 – *continued from previous page*

Aspect	Architectural component	Description
	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. *At the time of writing custom implementations of these APIs have been built.* *A mature implementation requires open standards for discoverability and observabilityinformation modeling and sharing. Some standards[3] are currently under development.*
The Multiplane Data Platform	Platform plane	A group of self-serve platform capabilities with high functional cohesion surfaced through APIs. *This is a general concept and well established.*
	Data infrastructure (utility) plane	Platform plane providing low-level infrastructure resource management - compute, storage, identity, etc. *The infrastructure plane is mature and supported by many vendors with automated provisioning.*
	Data product experience plane	Platform plan providing operations on a data product. *Custom implementation of services for data product experience plane, no reference implementation available.*
	Mesh experience	Platform plane providing operations on the mesh of connected data products. *Custom implementations of discovery and search services have been implemented, but no reference implementation exists.*
Embedded Computational Policies [4]	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. *At the time of writing this is an experimental concept with custom implementations.*
	Data product sidecar	The accompanying process to the data product implements cross-functional and standardized behaviors. This still is an experimental concept with custom implementations.
	Control port	A data product's standard APIs to configure policies or perform highly privileged governance operations. *At the time of writing this concept is experimental.*

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. [11]

[3] OpenLineage is an example of an observability open standard that data products can adopt.
[4] See figure 2.6

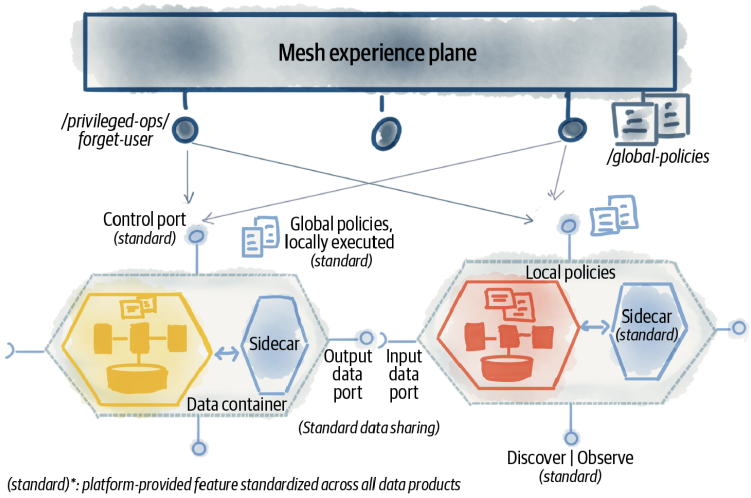

Figure 2.6: The logical architectural components of embedded computational policies

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 2.7.

The data product experience plane provides operations on a data product and manages the complexity of provisioning its underlying infrastructure, while the data 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 the platform users and evaluate how to make it easy for them to complete their journeys.

Data Product Developer Journey

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 2.8 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.

continued on next page –

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
		`/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.
		`/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.

continued on next page –

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
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.
		`/{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.
	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.
	Data product experience plane	`/{dp}/controls`	Data products are registered with a unique global identifier and address, making them visible to governance.

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.

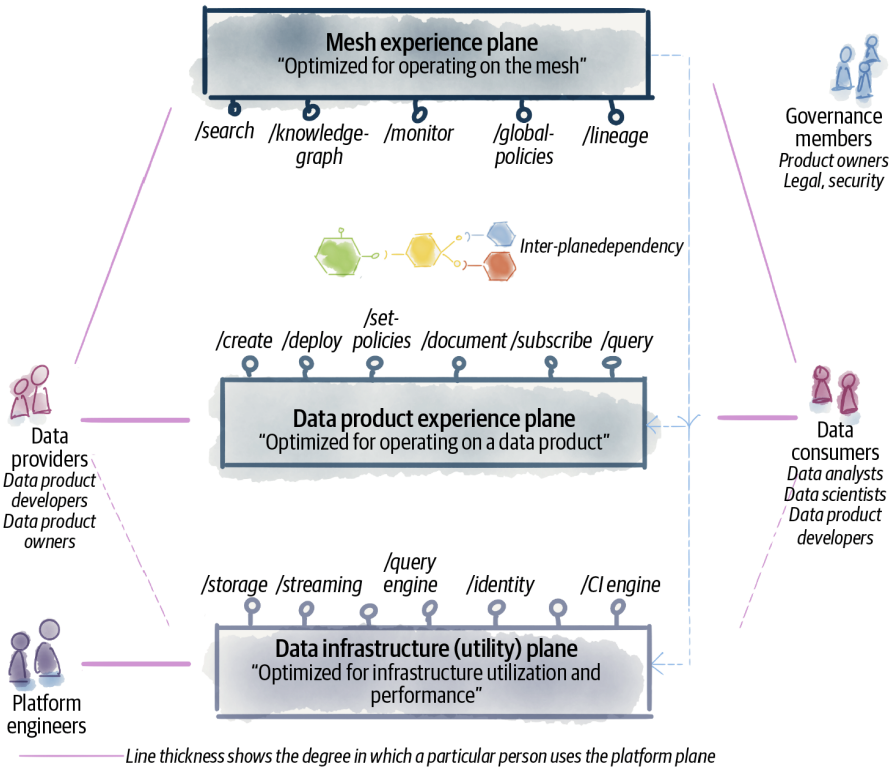

Figure 2.7: Multiplane self-serve platform and platform users

Table 2.4: Data platform interfaces that support data product experience plane APIs

Platform plane	Platform interface	Platform interface description
Data product experience	`/{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.
	`/{dp}/status`	Checking the status of a data product.
	`/{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`	Ability to migrate a data product to a new environment. Updating the data product revision is simply a function of build and deploy.
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`	Ability to invoke high-privileged administrative controls such as right to be forgotten on a collection of data products on the mesh.
Data product experience plane	`/{dp}/controls`	Data products are registered with the mesh and given a unique global identifier and address, making them visible to the governance process.

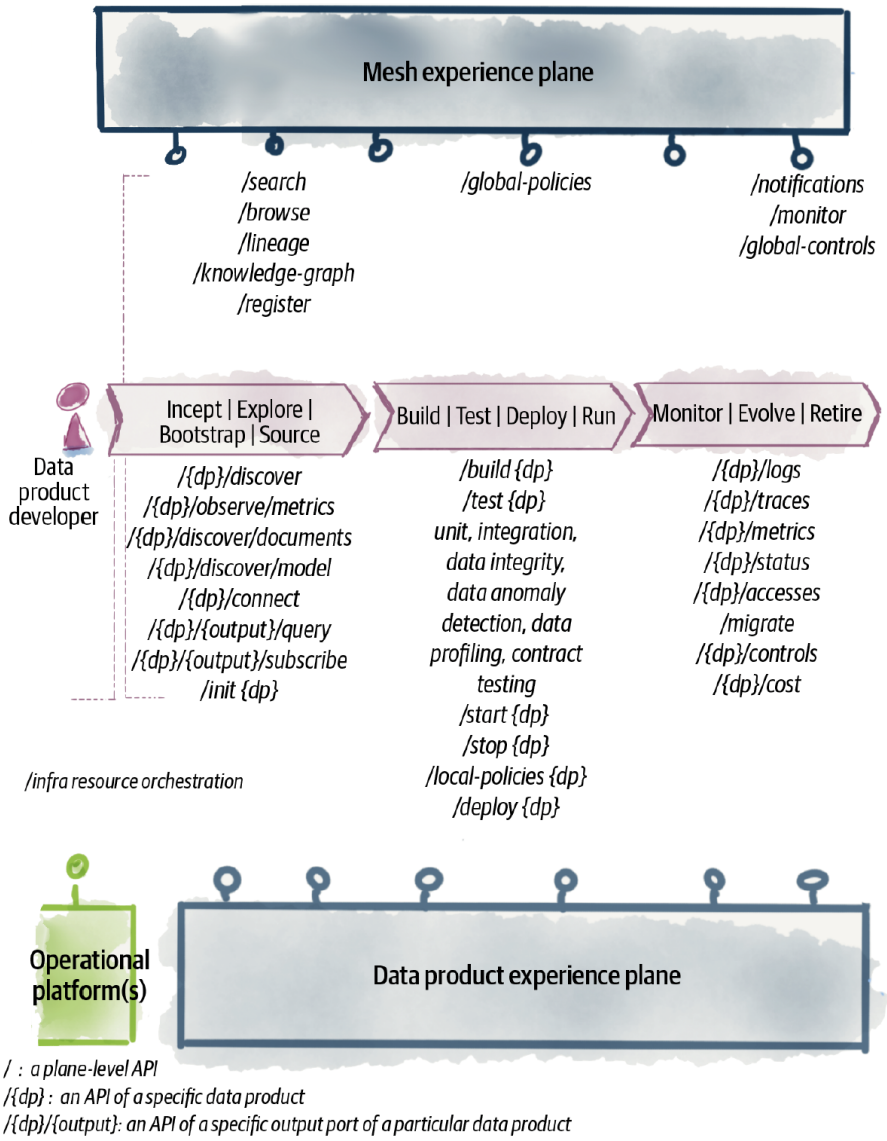

Figure 2.8: High-level example of data product development journey using the platform

Data Product Consumer Journey

A data consumer is a user with various skill sets and responsibilities, such as consuming existing data products to train an ML model and then deploying it as a data product. ML models can belong to either plane. For example, an ML model can be deployed as a microservice to make inferences online as the end user makes requests. Alternatively, an ML model can be deployed as the trans-formation 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 machine learning (CD4ML[5]) to continuously hypothesize, train, deploy, monitor, and improve the model in rapid feedback loops. The figure 2.10 illustrate this journey.

In conclusion, there is no single entity such as a platform but rather a set of well-integrated services, such as APIs, services, SDKs, and libraries that provide a seamless experience for

[5]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.

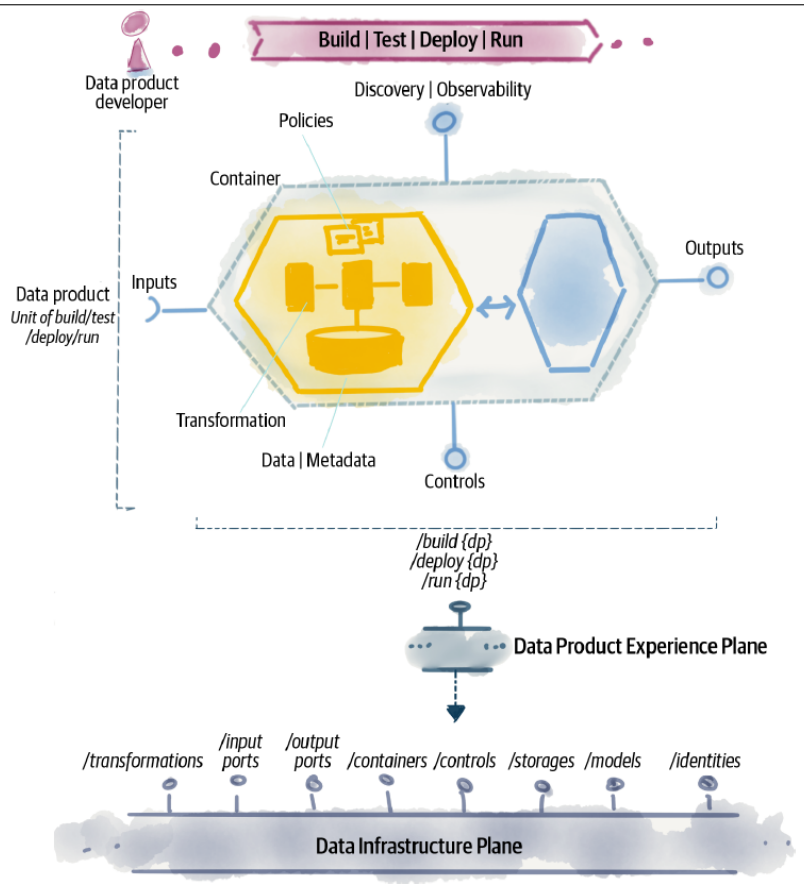

Figure 2.9: Example of data infrastructure plane to support data product delivery

users. Additionally, the separation of planes should be respected to optimize both for the human experience and machine efficiency. For example, the data product experience plane should optimize the user experience at the logical level, while the data infrastructure plane should optimize for the machine. Lower plane optimization decisions should not impair the developer experience, and vice versa.

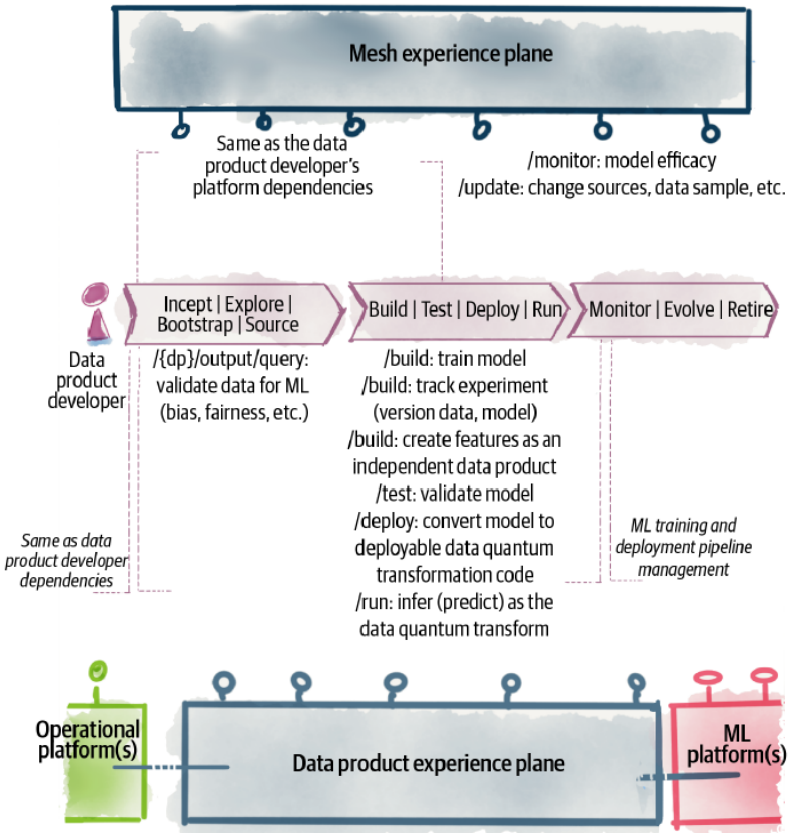

Figure 2.10: Example of data infrastructure plane to support data product delivery

Chapter 3

DATA PRODUCT DESIGN & IMPLEMENTATION

3.1 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.

Fundamentals of Software Architecture defines the constituents of architecture as: Structure, Characteristics, Decisions, Principles. In addition to these, we introduce and focus on another element: affordances - one of the five fundamental principles of interactions [12]. Hence, this approach to the design of data product architecture design by affordances.

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 training ML models, generating reports, data analysis, and building data-intensive applications, 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.
Discover | Understand | Explore | Trust	The data product serves APIs and information that affords data product users to discover, explore, understand, and trust it.

continued on next page –

Table 3.1 – *continued from previous page*

Affordances	Description
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.

3.2 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.

3.2.1 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.

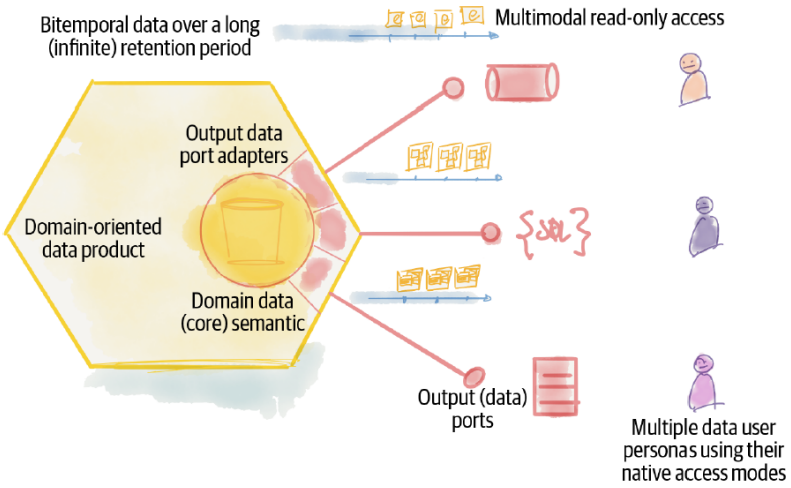

Figure 3.1: Data product's high-level components to design serve data

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.

3.2.2 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.

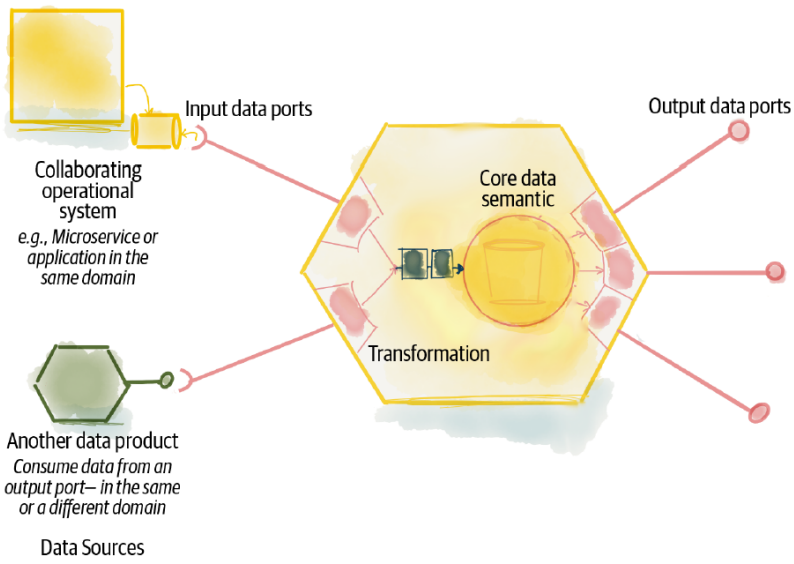

Figure 3.2: Data product's design to consume data

3.2.3 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.

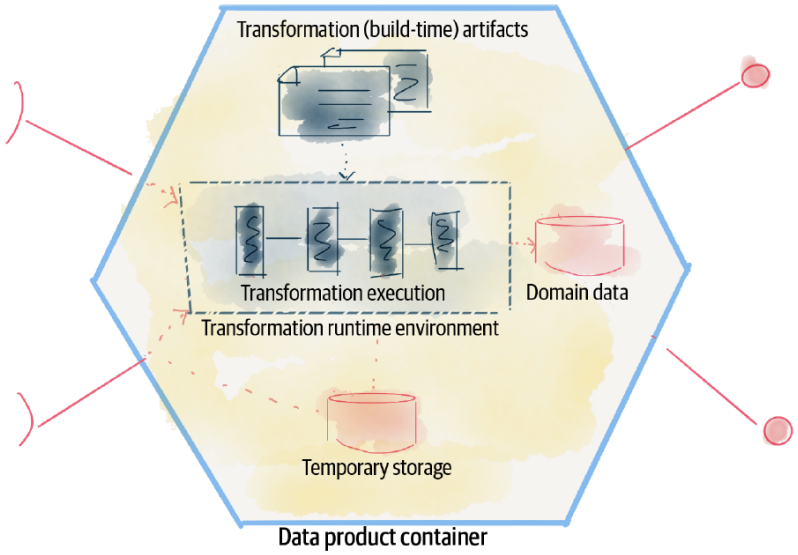

Figure 3.3: High-level components to design transformation for data products

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.

3.3 Discovering, Understanding & Composing Data

3.3.1 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 stan-

dardized across all data products. Figure demonstrates the data product sidecar's interactions in support of discoverability.

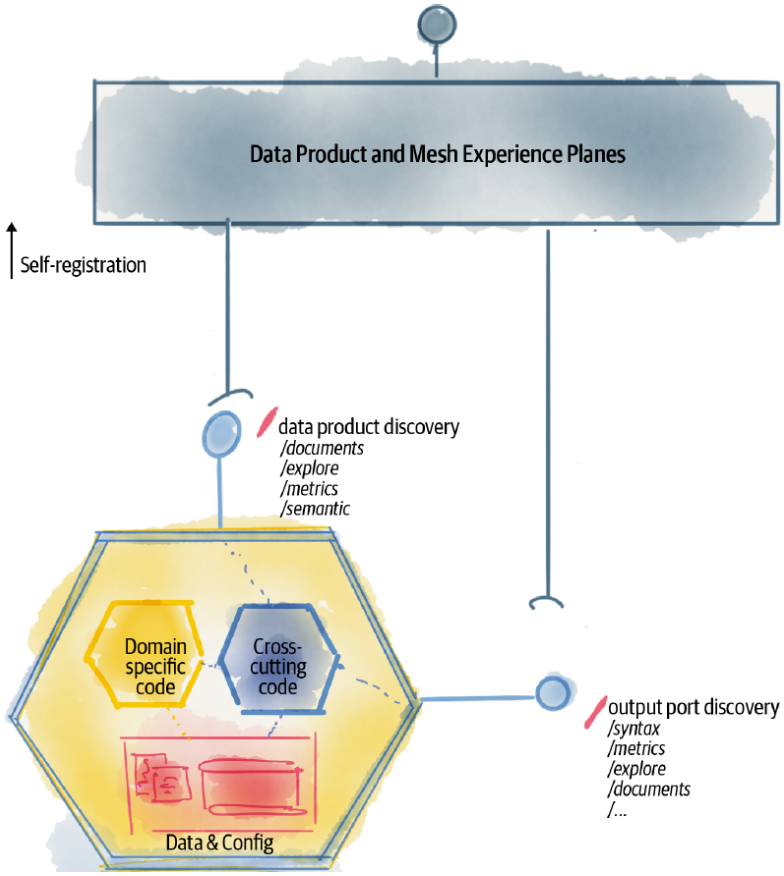

Figure 3.4: A data product sidecar provides a unified model of discoverability for all data products

3.3.2 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.

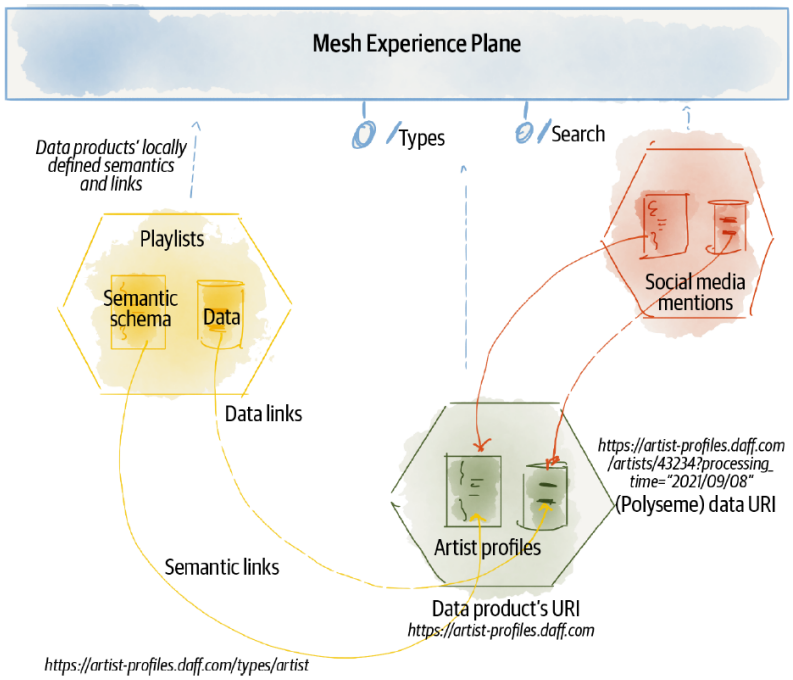

Figure 3.5: Data product's high-level components to design data composability

3.4 MANAGING, GOVERNING & OBSERVING DATA

3.4.1 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.

3.4.2 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

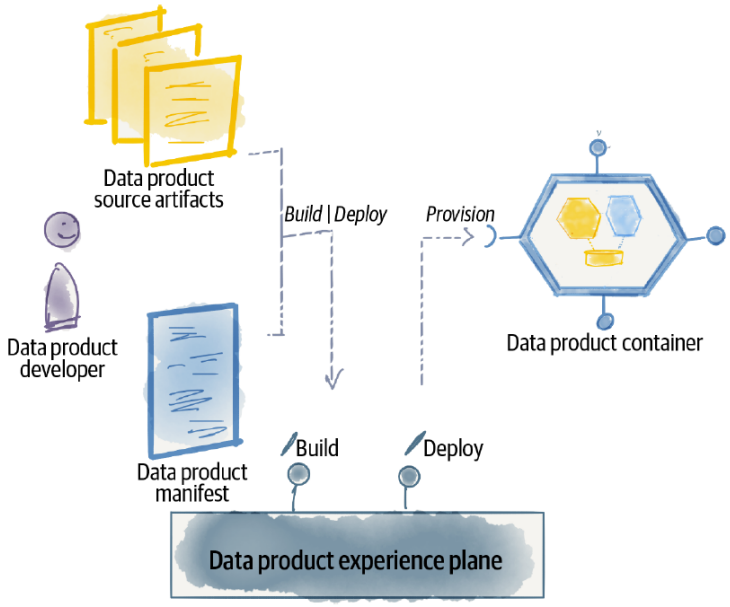

Figure 3.6: High-level interactions to manage the life cycle of a data product

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.

3.4.3 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.
- 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.

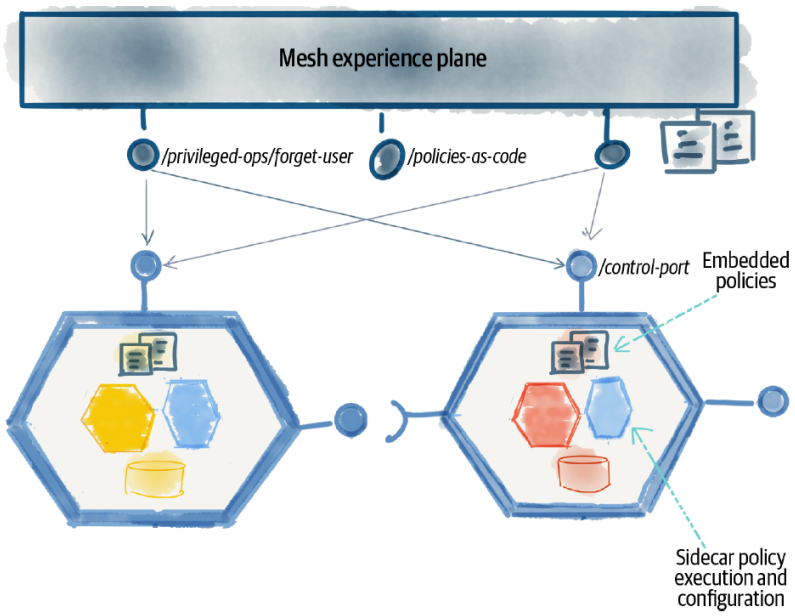

Figure 3.7: High-level design of embedded policies as code

- 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 DATA MESH IN COMBINATION WITH DATA LAKEHOUSE

4.2 FRAMEWORKS AND TECHNOLOGIES FOR DATA MESH

4.3 CASE STUDY AND DEMO

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

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

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