dbt Labs™

A guide to data mesh

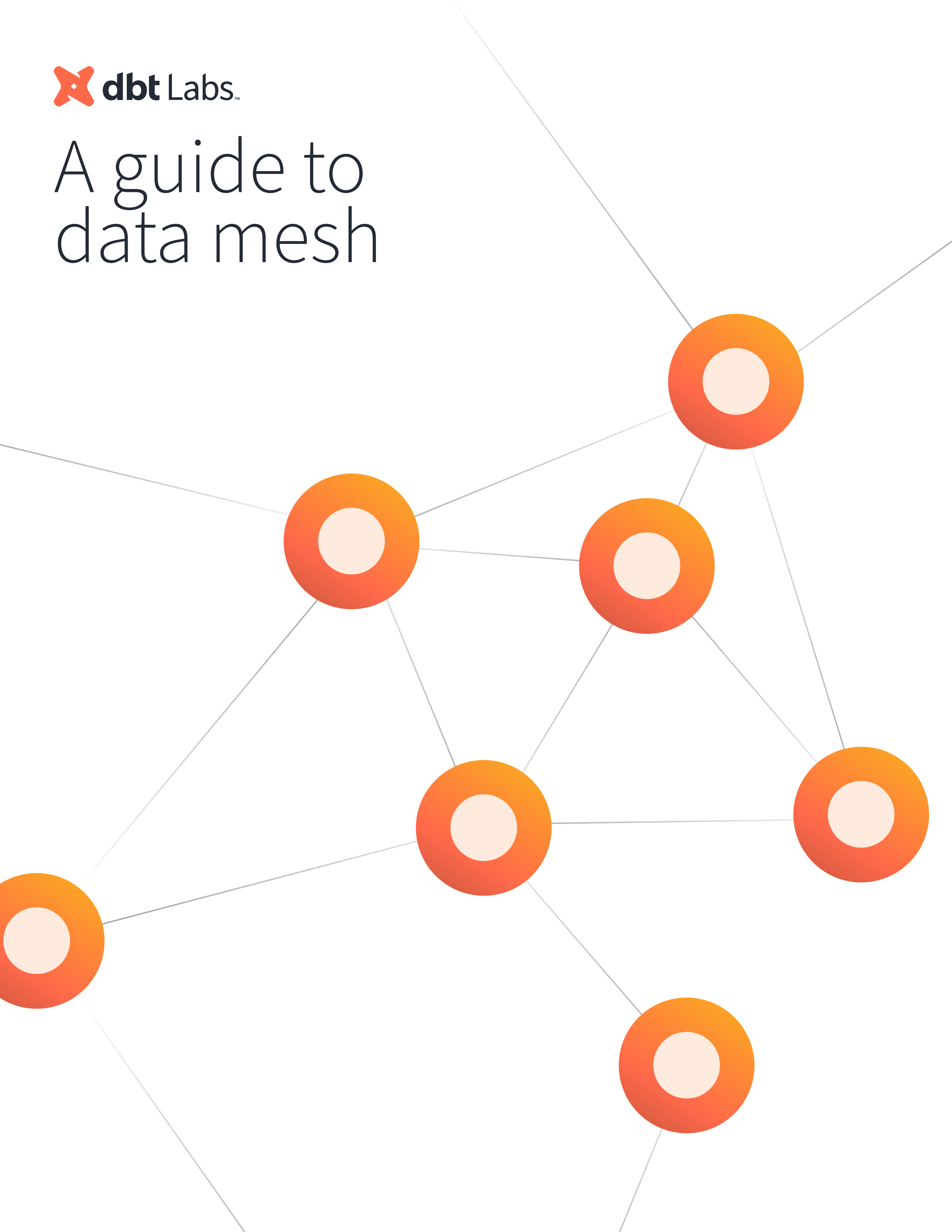


Table of Contents

Introduction

A guide to data mesh

There are many parallels between data analytics workflows and software engineering processes. Both have had to find ways to deal with mounting scale and complexity, larger networks of collaborators, and more expectations as their crafts have matured.

Software engineering has dealt with this complexity by discouraging hero mentality and embracing a services-oriented approach. Many in the industry realized that creating monolithic applications with massive teams was a recipe for increased costs and decreased quality. As a result, companies focused on creating small teams building well-defined components in a service-oriented architecture.

But the same thing hasn't happened with data. Data analytics, for the most part, still centers on creating monolithic stores managed by single data engineering teams. This results in overworked teams, shipping delays, and a decline in data quality.

How do we bring the hard-won lessons of software engineering into the data realm? In this eBook, we take an in-depth look at how a data mesh architecture turns the monolithic paradigm on its head—and helps you deliver data-driven projects more quickly at higher reliability in the process. We'll cover:

- Why we need data mesh
- The four principles of data mesh
- How to use the four principles of data mesh to build your own mesh architecture

Why we need data mesh

Slowdowns and silos in the data monolith

The architectural choice to use a data monolith has numerous knock-on effects. Monolithic approaches break down a data processing pipeline into several stages—ingestion, processing, and serving.

A single team often handles all of these stages. This approach can work at first but breaks down with scale. As more and more requests come in, the data engineering team finds itself unable to respond to them promptly. This leads to an ever-growing backlog of feature requests and bug fixes. This slows down the pace of innovation and also leads to the system becoming more brittle over time.

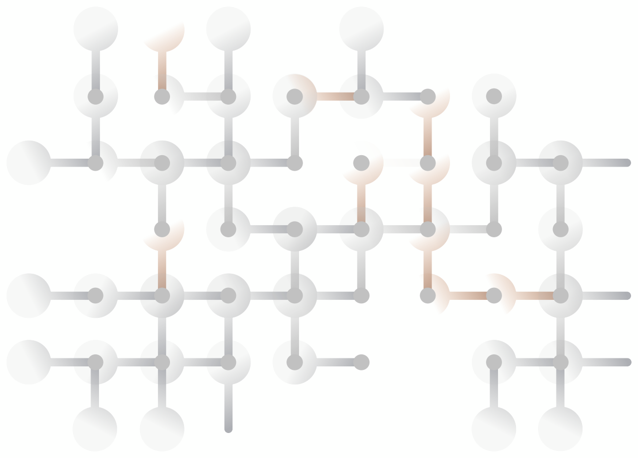
In this approach, data engineering teams often can't gain the full context behind the underlying data in this model. Since they're responsible for maintaining data sets from multiple disparate teams, they often don't fully understand the business rationale behind the data.

This can lead them to make uninformed—and, sometimes, harmful—decisions that impact business decision-making. For example, a data engineering team may format data in a way that the sales department doesn't expect. This can lead to broken reports or even lost data.

Complex and brittle systems

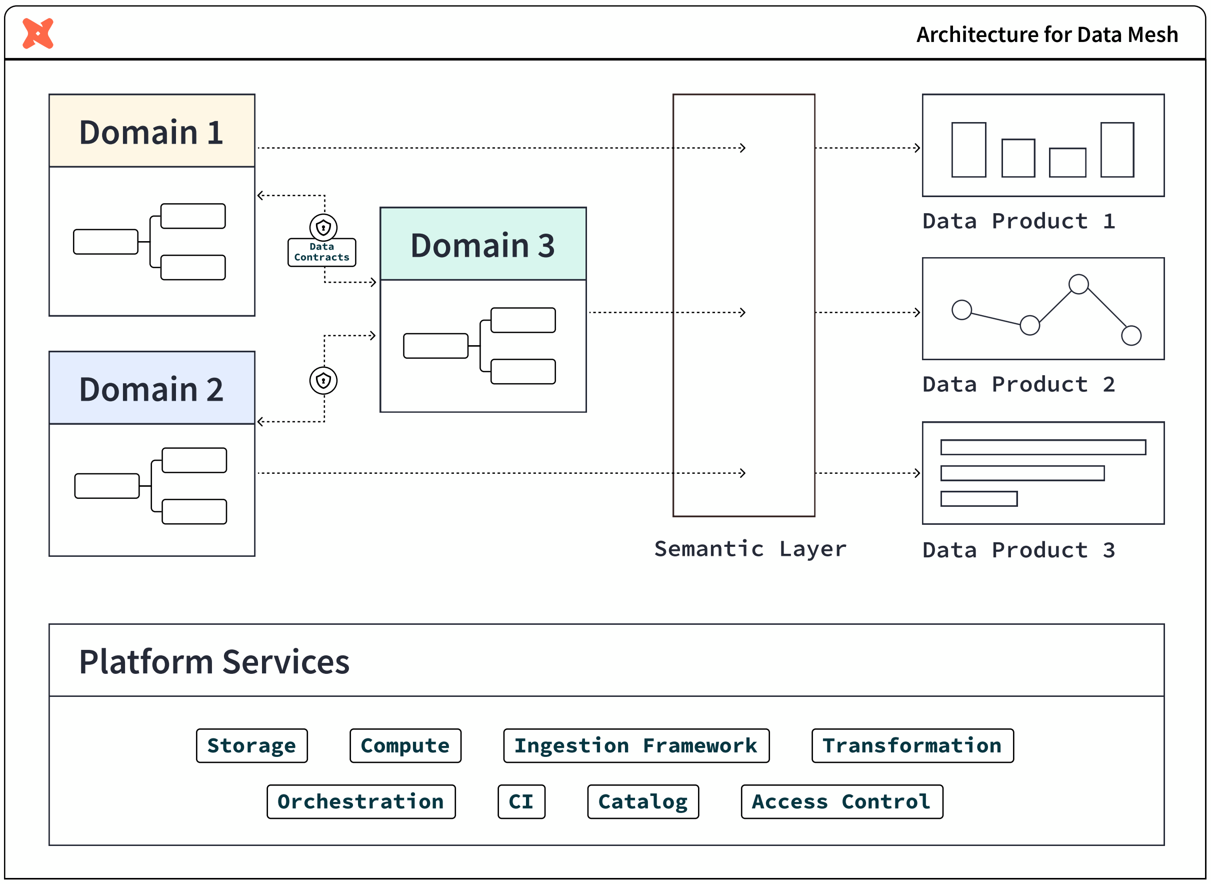
Monolithic systems rarely have clear contracts or boundaries. This means that data formatting changes upstream can break an untold number of downstream consumers. The result? This can cause teams to avoid making necessary changes for fear of breaking everything. This leads to monolithic systems gradually becoming outdated, brittle, and hard to maintain.

Finally, as dbt Labs CEO and Founder Tristan Handy notes, collaboration also becomes more difficult in a monolithic system. Since no one is familiar with the entire codebase, it takes more people and more time to complete data-related tasks. This affects time to market for new products and features—which impacts the company's bottom line.



Read dbt Labs CEO Tristan Handy's
The next big step forward for analytics engineering

What is data mesh?



A data mesh is a decentralized data management architecture comprising domain-specific data. Instead of having a single centralized data platform, teams own the processes around their own data.

In a data mesh framework, teams own not only their own data, but also the data pipelines and processes associated with transforming it. A central data engineering team maintains both key data sets and a suite of self-service tools to enable individual data ownership. Domain-specific data teams then exchange data via well-defined and versioned contracts. A semantic layer ensures organizations can centrally define business metrics so they are universally consistent by every user and team.

Data mesh architecture aims to solve the lingering issues in data systems by adopting the same approach to data systems that software engineering teams take to software systems. It does so by enacting the following four principles:

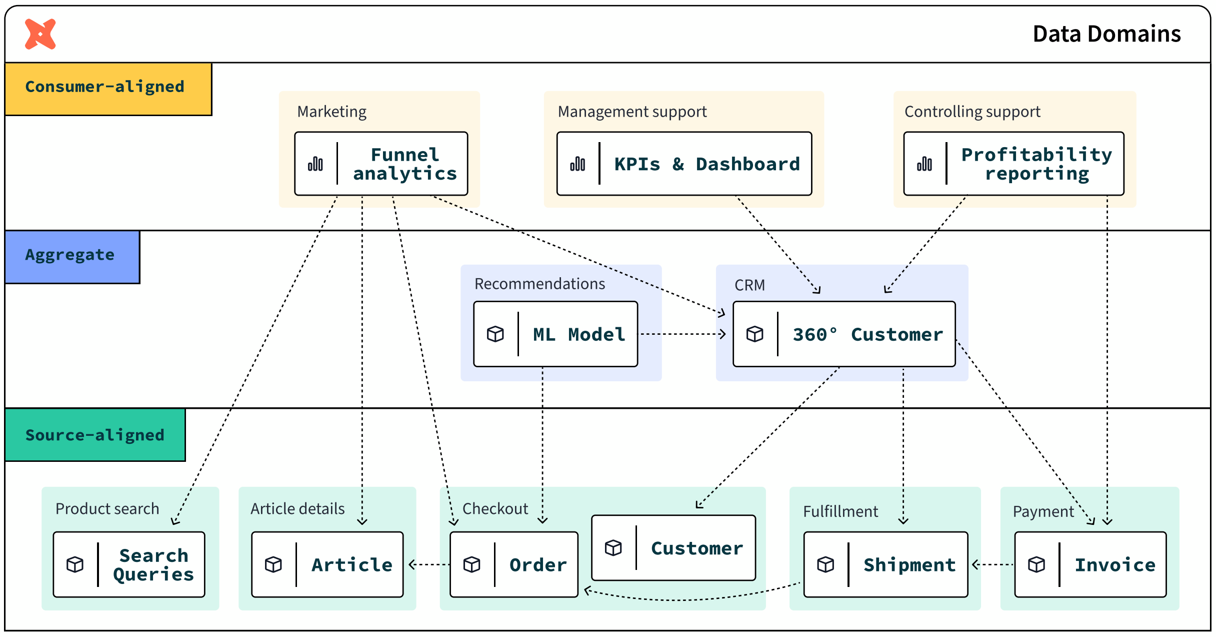
- Domain-oriented data
- Data as a product
- Self-service data products
- Federated computational governance

Let's look at each of these principles in depth.

Principle 1: Domain-oriented data

In data mesh, a data domain is a logical grouping of data, often source-aligned or consumer-aligned, along with all of the operations that its objects support.

The diagram below shows an example of how data domains might function in practice. The domains themselves can be loosely categorized into whether they are source-aligned, customer-aligned, or aggregates of data. Each domain itself is aligned not with a set of technologies, but with the part of the business that it supports.



Source: www.datamesh-architecture.com

In the absence of clearly demarcated data domains, many organizations use a single, central data team to store corporate data. That team "owns" the data from a technical standpoint. Any changes to its structure or format require that business owners file change requests with the engineering team. This model works for many companies…up to a point. Over time, these large, monolithic data structures become overly complex and harder to navigate.

The single-team approach also doesn't scale well. As more and more requests come in, the data engineering team becomes a bottleneck and falls further and further behind.

Data domains address this by parceling out data schemas into self-contained definitions owned and maintained by the business team that owns the data. The team owns the data storage and all processes—generation, collection, data pipeline transformations, APIs, reporting—that accompany it. Its output is a data product—a data container or a unit of data that directly solves a customer or business problem.

You can think of a data domain as a "service-oriented architecture" for data. Each data domain team is responsible for maintaining its data boundaries and the operations that support them.

Principle 2: Data products

A data product is a data container or a unit of data that directly solves a customer or business problem.

As a deliverable, a data product can be as simple as a report or as complex as a new Machine Learning model. Data products will also contain any metadata required for consumption, such as API contracts and documentation.

Data products have a core set of attributes that set them apart from bare-bones data. A data product should be:

- Discoverable
- Addressable
- Trustworthy and truthful
- Self-describing
- Interoperable
- Secure and governed

Discoverable
A data product should be easy to find—e.g., via central registration in a data catalog that tracks data products across the company.

Discoverability solves the chronic problem of finding relevant, high-quality data within a company. A survey by Coveo found that employees spend up to 3.6 hours per day searching for relevant information. The stats are even worse for IT professionals, who spend 4.2 hours per day.

Improving data discoverability reduces or eliminates this wasteful overhead and ensures that teams can start creating new data products promptly. For example, a team that wants to build a product recommendation engine could use discoverability tools—such as dbt's native documentation and lineage functionality—to find where the organization keeps an anonymized dataset of past customer orders. Once discovered, they can request permission to the dataset, then create their own data pipelines to transform it into the format they need.

Addressable
Addressable means that a data product has a unique, labeled location from which data teams can retrieve the asset.

The addressing format will differ based on the asset. For a database table, this may consist of a server name, port number, and schema/table path. For data exported by a partner, it might be a Parquet or CSV file stored in an Amazon S3 bucket. The only requirements are that the address uniquely identifies the asset and that it can be retrieved on demand by anyone with the proper permissions.

Trustworthy and truthful
In one survey of 220 data governance professionals, 46% said concern over data quality impedes use. Even if employees can find data, they won't use it if they can't trust it.

Data products can help improve people's confidence in data by providing self-service answers to fundamental questions about data:

- Who owns the dataset?
- How often is it updated (e.g., near-real-time, x times/day, every 24 hours, weekly) and when was the last update?
- What procedures do data owners take to cleanse and validate data?
- Has the data been tested?

Self-describing
A limitation of data without accompanying metadata is that it can be difficult to figure out what it means or why it exists. By contrast, data products provide mechanisms to describe the data they make available, its format, and its intended business purpose.

A dbt data model and a data model contract are good examples of self-describing data. Models describe their data and how it relates to other models in the company. A contract specifies a set of constraints to which a data model adheres.

Interoperable
Data products within a company should be interoperable—i.e., provide mechanisms for working seamlessly with other data products. Part of this work involves standardization of concepts (e.g., "customer", "product") common to the organization. Another part involves defining Application Programming Interfaces (APIs) and data formats to enable data exchange across teams and divisions.

Secure and governed
Finally, data products should leverage federated computational governance to ensure security and compliance.All data should be stored encrypted at rest and in transit, and protected via a strong "zero-trust" permissions model.

Principle 3: Self-service data platform

A self-serve data platform is a data platform that supports creating new data products without the need for custom tooling or specialized knowledge. It's the technical heart of data mesh, enabling data domain teams to take ownership of their data without unnecessary bottlenecks.

The self-serve data platform fulfills three critical functions:

1. Support developing data products. In a data mesh architecture, data domain teams take control of their data and all related artifacts and processes. The end product of their work is a data product—a self-contained unit of data that directly solves a customer or business problem.
2. In a data mesh approach, a centralized data platform team creates the infrastructure and tooling that supports creating data products. This ensures that each team is creating data products in a uniform and consistent manner.
3. Enable data discovery. The distributed nature of data mesh means there needs to be a systematic way for teams to discover and use the data products that other teams have developed. As part of the self-serve data platform, the data platform team provides tools such as a data catalog for registering, discovering, and enriching data products.

Institute data governance policies. Another potential challenge with the distributed nature of data mesh is maintaining a consistent quality bar across data domains. Without a way to enforce data quality and compliance standards, a data mesh can quickly turn into a data anarchy. To address this, the self-serve data platform implements core support for federated computational governance (discussed below).

Capabilities of a self-serve data platform

Basic data storage capabilities. If you have data, you need some place to put it. A self-serve data platform administers and provides secure access to storage in multiple forms—RDBMS, NoSQL, object storage, data warehouses, data lakes. It also standardizes transfer and storage protocols (e.g., encryption at rest and in transit).

Data product creation. The self-serve data platform also offers all the tooling necessary to turn data storage into data products. This can include:

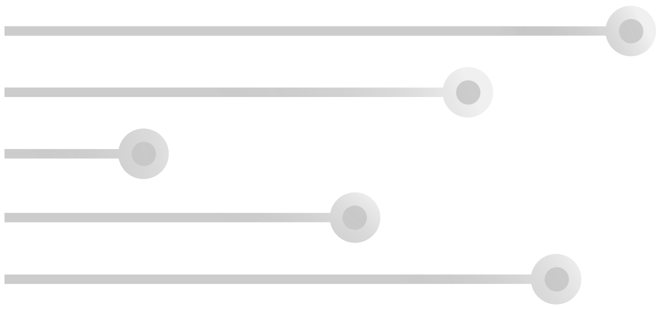
- A format for defining and testing model contracts, which define the commitments the data domain team makes to its consumers about the data it emits.
- A data transformation layer, such as dbt models, for converting source data into the formats required by the data domain team.
- Tools for orchestrating and running data pipeline jobs to transform new, incoming data periodically.
- Source control (i.e., Git support) for tracking changes to contracts, transformations, and other project assets.
- Support for data caching, data access API creation, and other related functionality.

Data discovery. The data catalog and accompanying data lineage provide basic data discoverability capabilities. A data catalog provides visibility into how data flows through the entire company, which enables advanced data debugging techniques such as root cause analysis and change impact analysis.

Security and data governance. A self-serve data platform must also provide methods for restricting data access to authorized individuals. We go into more details around this below, where we address federated computational governance.

Reporting and monitoring. As part of its base-level support for data products, the self-serve data platform should also provide basic support for logging and monitoring (e.g., auditing data changes, logging access requests). It can also provide useful statistics about data across the company, including:

- Which data assets are the most/least used
- Metrics on overall data quality and corporate compliance
- Alerts on important changes—e.g., alerting downstream consumers of a data product when a new version of the data contract has been published
- Reporting on costs and usage of the platform



Principle 4: Federated computational governance

So we have independent data domains using a self-serve data platform to self-publish data products. The question arises: How do you maintain consistency and oversight in such a distributed, highly-scalable data network?

There are several reasons organizations need to exert some measure of verification and oversight over the data that domain teams manage:

- Ensure data quality. Poor quality data -malformatted dates, missing values, corrupt structured data, etc. - can lead to poor decision-making and broken data pipelines. According to some estimates, poor data quality costs organizations up to $12.9 million a year.
- Verify regulatory compliance. Regulations such as the General Data Protection Regulation (GDPR), the Health Insurance Portability & Accountability Act (HIPAA), and many others worldwide specify strict rules around handling consumer data—and stiff penalties for violating them.
- Simplify data interoperability. If different teams define similar concepts in different ways, it can complicate sharing data and building on one another's work. Companies that define standard data structures for common concepts such as "customer" or "product" help foster a data environment where interoperability is painless.
- Provide accountability and transparency. A clearly-documented set of data governance policies defines the rules every data product team must follow in an open and auditable format. This minimizes the risk of errors and intentional abuse.

This is where federated computational governance comes into play. Each data domain team continues to own its data (plus associated assets and processes). Each team is also required to register its data products with one or more data governance platforms. The data governance platforms, in turn, run automated data governance policies that ensure all data products conform to organizational standards for quality and compliance.

Hallmarks of federated computational governance

Governance isn't a new concept. Many of the tools used to implement federated computational governance - data catalogs and data governance platforms - pre-date the concept of data mesh.

Federated computation governance in a data mesh is unique in that it balances decentralized autonomy with global interoperability. It combines the benefits of data mesh's scalable data model and local ownership with automated implementation of policies and best practices. It accomplishes this by being:

Distributed (federated). Federation frees data domain teams up from the downsides of working with data monoliths and gives them greater control over their own data and its processes. But it does so without sacrificing oversight. Without data governance software, each data domain risks becoming its own data silo, enforcing (or, even worse, not enforcing) its own data governance and quality rules.

Automated (computational). Federated computational governance leverages data governance software to implement data governance, security, compliance, data quality, and more as code—e.g., as declarative policies, scripts, etc.

This isn't to say there are never human-verified quality gates, checks, or inspections in a data mesh architecture. However, most data governance and quality checks are automated to achieve greater scalability and faster delivery times for new data products.

Data governance policies are written down in both human- and machine-readable formats. That means they can be checked in to Git and version-controlled, code reviewed, tested, and read by others. This increases accountability, transparency, and accuracy in the data governance process.

Scalable. Federation and automation mean that data projects can scale faster. Automated policies can verify more data in a shorter time than a human could through manual inspection. They can also react to fast-changing data more quickly. This helps establish confidence that a new data product is high quality and fully compliant with all appropriate regulations, which means it can ship faster.

Automation can also increase scalability by performing some data governance actions automatically. For example, if a field is marked as sensitive, an automated process could use data lineage information to propagate this sensitivity tag to all of the data sources both upstream and downstream from the field's data store.

Community-focused. In many companies, compliance is often perceived as red tape—an obstacle. When imposed from the top-down with no transparency, it can generate resentment and frustration.

In a data mesh, federated computational governance is more of a community function in which data engineering, data governance, and data producers collaborate on data quality and governance. Since policies are clearly set forth in code kept under source control, teams can work together on refining data governance policies to cover any unexpected edge cases.

Implementing the four principles of data mesh

How do you implement these four principles to create a data mesh architecture? Follow the guidelines below to get your company thinking about how to embark on this new approach to managing data.

Implementation: Domain-oriented data

Initialize and develop the data domain
Teams requisition new data domains through the self-serve data platform. This enables data domain teams to remain focused on the domain-specific portions of their business instead of reinventing the wheel regarding data management. It also ensures consistent technical support and cross-team data governance.

Design the data domain
When designing a data domain, domain teams need to answer a few fundamental questions:

- Where are we sourcing our data?
- What does our data schema look like?
- What transformations must we apply to our source data to work with it efficiently?
- How do other teams access our data?
- Who has what rights to which data sets?
- Which data should be public and which data should be internal to our team?
- Which data needs to be tagged as sensitive—e.g., Personally Identifiable Information (PII)?

Before shifting to a data domain framework, companies should settle on tools and conventions for codifying these design decisions. These tools should be a standard part of a self-service data infrastructure.

dbt data models and model contracts are examples of tools you can use here. dbt models use simple SQL or Python models to define how to transform data between source and destination formats. You can also define data access rules through simple declarative YAML markup. Model contracts further define the conditions that the domain team guarantees to its external consumers that its data fulfills.

Registering the components of a data domain
As part of a self-service data infrastructure, the centralized platform team will also offer a method for registering new data products, their sources, contracts, and their outputs from a data domain. Registration:

- Enables other teams to find a data domain team's data products and use them in their own data products
- Identifies the owners of a data domain so that other teams can easily find and work with them
- Enables securing and classifying all data in a company to ensure compliance with industry standards and legal regulations

Registration is usually managed by a data catalog, which tracks and manages all of the data in a company's data estate. The data domain team ensures that its assets—data sources and destinations, models, contracts, APIs, data products—are registered in the data catalog. The team is also responsible for tagging all of its data appropriately for compliance and for adhering to corporate standards around data quality.

Manage and revise the data domain
Data domain teams publish their data products with accompanying contracts. At a minimum, contracts specify an owner, a version, a description of the data that the product exposes, and the conditions that data meets (e.g., field is non-null, is maximum x characters long, contains a specially-formatted string).

Because data domain teams are small and lean, they can work agilely and release new versions of their data products iteratively. Teams can publish new versions of their contracts to the data catalog, which in turn will notify downstream teams that an update is available.

Implementation: Data products

Design the data product
Architecturally speaking, a data product consists of several components. The most important of these are the data specification and data contract.

Data specifications are one or more human- and machine-readable documents that define the format of data, data definitions, access policies, and data transformations.
Data contracts are guarantees of behavior for a given version of the data product. You can think of them like APIs in a service-oriented software architecture.

Contracts guarantee that the data product's output for that version will always return consistent results. That's because a contract is a machine-readable specification that can be used for testing and verification. For example, a centralized quality management system (e.g., a data catalog) can run a contract against a portion of a data domain team's data when it submits a new data product.

Other assets that comprise the data product include:
Tests: Code that verifies the validity of your models against representative data.
Version control: Leverage git to check in and track changes to data definitions, contracts, and data pipeline code. This serves to document changes and enable rollback to previous versions when required.
Data storage: Object file storage, RDMBS/NoSQL database tables, data warehouses, date lakes, etc. to hold raw and transformed data.
Orchestration pipeline: Computing processes that transform data, run tests, and deploy changes to one or more environments.
Additional deliverables: Any additional artifacts that comprise the data product, such as reports and metrics.

Provision, develop, and register the data product.
Once provisioned via the self-serve data platform, the data domain team develops the models, permissions, tests, ELT processes, reports, and other deliverables that comprise its data product.

After extensive testing, the team publishes its initial version to the data catalog. The registration includes information such as the data model, the current contract specification, the address of the data product, and any additional metadata required by the registry.

Revise the data product
After registration, the data domain team resolves any security and compliance issues detected by the registry. From there, other teams can discover and use the data product in their workflows.
When the data team needs to introduce a breaking change, it creates a new contract with a new version and publishes it to the registry. It also provides an "end of life" date for obsoleting the previous contract. The registry can use data lineage information to inform owners of downstream data products of the upcoming change

Implementation: Self-serve data platform

The good news is, you're probably already running some of the technological components—data warehouses, data catalogs, transformation tools, pipelines—that the new system will need. Beyond technology, however, you'll also need to determine how to structure your data and how different data domain teams will interface with each other.

Here are some general guidelines to get you started.

Obtain executive buy-in
A self-serve data platform isn't a minor undertaking. It'll require a significant, ongoing investment in data platform engineering resources.

The first step is creating a business plan showing how the platform will deliver a return on investment. You can calculate this by contrasting the estimated cost of the new system in terms of time, labor, and licensing with revenue and cost-saving factors such as:

- The business value of the data projects that the new approach will unblock
- The reduction in time/resources spent debugging data quality issues
- The reduction in redundant infrastructure effort and associated licensing costs

Create first cut of self-serve data platform architecture
Identify the technology you currently have to support a self-serve data platform, as well as any gaps (e.g., data contract authoring/validation tools). Design a rollout plan for the new platform that introduces key features over subsequent version releases. The platform itself is a product with releases, roadmaps, SLAs etc.

For example, your initial release might contain the minimum features required to support self-service for new data products built on top of your data warehouse. This may include scoping permissions correctly for each team to access the storage and existing data they need, and providing a data catalog for all teams to discover and view data contracts for other teams' data products.

Onboard first key team and iterate
When you're ready, onboard your first key team to the new platform. It's important to move slowly and onboard a single team at a time. This enables you to vet your initial assumptions and make adjustments during onboarding. It also gives you a quick win you can use to demonstrate the value of the new systems to other stakeholders and senior leadership.

Once you've onboarded your first team, continue onboarding additional teams while also iterating on the capabilities of the self-serve data platform.

Implementation: Federated computational governance

As with the other components of a data mesh, federated computational governance isn't a one-and-done project. It's an ongoing effort involving trial-and-error that grows in both scope and quality with each iteration. It's also one that grows in tandem with the other components of your overall data mesh architecture.

Build data governance into the self-serve data platform
Federated computational governance is a critical part of the self-serve data platform. The platform usually provides all or most of the components below to support federated computational governance:

- ***Data catalog.*** A data catalog acts as a single source of truth for all data within a company. It stores information such as a data's sources, its metadata, and data lineage (the relationship between a data asset, its producers, and its consumers). It enables easy discovery of data, enrichment of metadata, and tagging and classification of sensitive information.
- ***Data transformation tools.*** A data mesh often also provides support for a data transformation layer. Data transformation tools, such as dbt Cloud, can act as a common intermediary layer for all of the data stores in your modern data stack. In particular, dbt Cloud provides support for data governance via data lineage generation, rich documentation and metadata support, tests, model contracts, and model versions.
- ***Data governance policy software.*** This is the toolset that interprets and runs automated policies on your data. It may be a built-in part of your data catalog or an extension of it.
- ***Access control framework.*** The platform will also specify a set of role-based rules for who can access what data, particularly data marked as sensitive—e.g., a user's Personally Identifiable Information, or PII. This may use one or more security mechanisms, including Microsoft Active Directory, OAuth, SAML-based identity frameworks, Amazon Web Services (IAM) Identity & Access Management (IAM), and others.

Design and implement data governance policies and access controls
The data governance and data engineering teams work together to define the policies that will govern all data in the organization. These become the rules to which all data domain teams must adhere when they publish new data products. The data engineering teams build these into the self-serve data platform as automations that run against the data described in the data catalog.

A company might implement numerous types of policies, including:

- Security policies: who can access what data, both at a tabular and columnar level.
- Compliance and privacy policies: which field formats indicate potentially sensitive data (e.g., address, national ID number, birth date) and should be labeled as such.
- Data quality policies: standards for the format of common fields, such as dates, as well as standards for accuracy (e.g., decimal-point precision in financial values; use of common measurement units), duplicate data, and freshness.
- Interoperability policies: policies that define standard representations for common data objects; policies requiring the use of data contracts and other assets that define data products and make it easier for one team to leverage another team's data.
- Documentation and metadata policies: which metadata should accompany all data assets and how teams should document assets to convey their business purpose and proper usage.

Register for data governance monitoring and respond to issues

With this framework in place, data domain teams register their data, data contracts, and other assets (such as dbt projects) with the data catalog. The central data governance policies will run periodically on the team's data looking for potential compliance violations.

If the data governance platform detects a potential violation, it can send a notification to the data owners on the data domain team. One or more team members responsible for compliance can navigate to a dashboard to see a list of all outstanding issues, along with a due-by date for resolution. The compliance point person can then work with their team to deploy fixes for these issues that will be picked up in the next scan.

Data domain teams may also have additional programmatic requirements for compliance that they must meet. As an example, a team may need to supply some method (e.g., a REST endpoint) to help the company comply with right to erasure requests under GDPR.

Conclusion

dbt Labs' goal is to empower teams to work both independently and collaboratively, without sacrificing security or autonomy. That's why we're big believers in the data mesh architecture.

dbt Mesh enables companies with complex transformation workflows to increase their flexibility and performance. By leveraging features built into dbt, you can manage any number of dbt projects as separate workflows with defined contracts and versions, cross-project data lineage, and granular security and access controls.

To learn more, check out the documentation on dbt Mesh, and talk to us about how you can use dbt Mesh to bring the scalability and security of data mesh to your teams.