Engineer

Data mesh

domain teams serving their analytical data as products

What it is

A data mesh is a way of organising analytical data in a large organisation so that the teams who run each part of the business also own, publish and answer for the data that part produces. Zhamak Dehghani, then a principal technology consultant at Thoughtworks, set the idea out on Martin Fowler's site in a series of instalments in May 2019, and restated it in December 2020 as four principles: domain-oriented ownership of data and its architecture, data as a product, self-serve data infrastructure as a platform, and federated computational governance. She meant the four to be necessary and sufficient together, and expected the practices and the tools to change while the principles stay put.

The mesh is about analytical data, not the operational kind. Dehghani draws the line like this. Operational data sits in the databases behind the systems that run the business, is transactional and holds the current state. Analytical data is the history of the business over time, aggregated and modelled, and it is what reports are built on and models are trained with. She leaves operational data where it is, inside each service, and aims the mesh at the analytical side, which is where she sees the friction at scale.

Domain ownership. The mesh is cut along lines the organisation already has. Dehghani's running example is a digital media company with domains such as podcasts and artists. The team that runs podcasts, and offers the interfaces for publishing them, should also provide the history of released podcasts and how many people listened over time. The boundary comes from domain-driven design, where each domain's bounded context is the place in which change stays local, so it is a good place to put ownership of the data as well.

Data as a product. Spreading data across many teams makes it harder to find and to trust, so each team has to treat the data it shares as a product and the people who use it as customers. The 2019 article lists what that takes: a data product should be discoverable, addressable, trustworthy, self-describing, interoperable under global standards, and secured by global access control. A domain data product owner answers for measures such as quality, how long consumers wait for data and how satisfied they are. Responsibility for quality moves upstream, as close to the source as it can get. In the architecture, a data product is the smallest unit that can be deployed on its own, and it carries three parts: the code that transforms and serves the data and enforces its policies, the data together with its metadata, and the infrastructure that runs it.

A self-serve platform. Building, running and watching a data product takes specialised infrastructure skills that cannot be copied into every domain. A platform team therefore offers tools at a higher level, covering things like storage, product schemas, declared pipelines and lineage, so that a generalist developer in a domain can build a data product without that specialist knowledge.

Federated computational governance. The products must still fit together, or nobody can combine them. A group of domain product owners and platform owners decides which rules apply everywhere and which are left to each domain, and the platform enforces the global ones automatically. Dehghani's example is listening figures: how the podcast domain models them is its own business, but how a listener is identified is a global matter, because the same person turns up as a user in other domains too.

Use the concept in a decision

Put "What is a data mesh?" to work in a real decision. We can help relate the concept to the system you are building or buying. Sales, operations and finance figures brought together, so two departments stop bringing two answers to the same meeting.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Made a geographical data pipeline 50x faster — 50x faster geographical data pipelines

Made a geographical data pipeline 50x faster Get in touch →

Why it matters

Dehghani's case begins with what goes wrong when a large company runs a central data lake or warehouse. Data from every domain is pulled into one platform that owns all of it, run by a team of specialists in data tooling who sit apart from the people who create the data and from the people who use it. She names three failure modes. The platform is centralised and monolithic, so it struggles as sources and consumers multiply. It is split into pipeline stages such as ingestion, cleansing, aggregation and serving, so a single new figure, like how often podcasts are played, needs a change in every stage and a release coordinated across teams. And its ownership is siloed and hyper-specialised: the engineers who must make sense of the data know little of the domains it comes from, while the teams that produce it have no incentive to get it right.

The mesh changes who does the cleaning. In the old model, data arrives with its errors and a central team cleanses it after ingestion. In a mesh, a domain's dataset becomes a data product only after the domain itself has checked it against the agreed quality measures and the global rules. The people who know what a field ought to mean are the ones who vouch for it.

Governance changes shape as well. The 2020 article sets the two models side by side in a table. A central team is itself responsible for quality, security and compliance; a federated team defines how quality is modelled and which security and regulatory rules the platform should build in and monitor. Success used to be counted in governed tables; in a mesh it is counted in how much the products are actually consumed. Manual review that tries to prevent errors gives way to automated processes that detect them and recover.

None of this is offered as a universal answer. In the 2019 article Dehghani grants that the central model can work for organisations with a simpler domain and fewer kinds of consumer, and says it fails for enterprises with rich domains, many sources and a diverse set of consumers. Whether a mesh is worth its overhead depends on which of the two an organisation is.

What it looks like in practice

A real mesh spans many teams and many stores. The sketch below keeps to one PostgreSQL database, so that each statement can be checked against the manual, and shows the two smallest moves: a domain publishing a product, and the platform enforcing one global rule.

The orders team owns the operational tables in the `orders` schema. It publishes its analytical data in a schema of its own, as a versioned view with a description and an explicit grant:

```
-- the orders team owns this schema and everything in it
CREATE SCHEMA order_data;

CREATE VIEW order_data.orders_by_day_v1 AS
SELECT order_date, currency, count(*) AS orders, sum(total) AS revenue
FROM orders.orders
WHERE status = 'paid'
GROUP BY order_date, currency;

COMMENT ON VIEW order_data.orders_by_day_v1 IS
  'Paid orders per day and currency. Owner: orders team.';

GRANT USAGE ON SCHEMA order_data TO analysts;
GRANT SELECT ON order_data.orders_by_day_v1 TO analysts;
```

The PostgreSQL manual explains why this is enough on the consumer's side: whoever uses a view needs the USAGE privilege only on the schema that holds the view, not on the schemas the view reads from. The analysts can query the product while the operational tables behind it stay closed to them. Evans, for his part, lists database schemas among the physical forms a bounded context's boundary can take.

The platform's side is a rule that holds for every domain, written as a query and run on every release: no product schema may contain a view without a description. `obj_description` returns the comment stored on an object, and a `relkind` of `v` marks a view in `pg_class`:

```
-- global rule: every published view carries a description
SELECT n.nspname, c.relname
FROM pg_class AS c
JOIN pg_namespace AS n ON n.oid = c.relnamespace
WHERE c.relkind = 'v'
  AND n.nspname ~ '_data$'
  AND obj_description(c.oid, 'pg_class') IS NULL;
-- any row returned stops the release
```

That split is the federated part in miniature. What counts as a paid order is the orders team's decision. That every product describes itself, and that product schemas end in `_data`, are global conventions the platform checks without anybody having to remember them.

The service page Data Governance & Quality promises that it will be plain who owns each figure and what its source is, which is exactly what a data product owner must be able to say. Among its delivered work it lists a function-first PostgreSQL data layer, designed so that reads and writes pass through functions the database can grant access to, never through the tables themselves. It is the instinct behind the view above: consumers get an interface with a grant on it, not the tables underneath.

The portfolio case One infrastructure plan for 14 departments does not call itself a data mesh, but it runs into the tension a mesh's governance group lives with. At a research institute, 14 research groups worked with different data sources, formats, scales and tools, and their technical skills and IT resources differed widely: a few had built their own solutions, while many struggled. The plan set out to let researchers concentrate on research, and it balanced flexibility against standardisation, with a modular architecture, ways in for very different data sources, interfaces suited to different levels of skill, scalable storage and processing, and strategies for onboarding, support and governance. Deciding what each group keeps for itself and what everyone shares is the balance Dehghani gives to federated governance, and lowering the skill a group needs is what she asks of a self-serve platform.

Where it goes wrong

A central team with a new name. Calling the data platform team a mesh changes nothing if the same team still takes in every domain's raw data, cleans it and answers for its quality. Dehghani's governance table makes a usable checklist: who is responsible for quality, who decides the security rules, and whether success is still counted in governed tables. Move ownership of each data product into the domain that produces the data, and keep the central team for the platform and the rules.

Products nobody stands behind. A domain that exports tables without an owner, a description or a promise about freshness has only moved the old problem into its own corner. Dehghani asks each data product to state its quality as service level objectives, such as timeliness and error rates, and notes that one

domain may offer two products of the same events: one close to real time that can miss or repeat events, and one that arrives later and is more accurate. Name an owner for every product, publish its objectives, and test quality where the data is produced.

No shared identifiers. If each domain identifies the same things its own way, no two products can be joined. In the 2019 example an artist looks different in the domain that records plays and in the domain that pays artists. Agree globally on how such shared entities, which Dehghani calls polysemes, are identified, and leave the rest of each model to its domain. Field formats, address conventions and common metadata fields belong on the same global list.

Every domain rebuilds the stack. Without a self-serve platform, each domain has to provision its own storage, pipelines and monitoring, and the specialised skills that were scarce in one central team are now needed in every team. Build the platform before asking domains to publish, and judge it by how little specialist knowledge a domain needs to ship a product.

Changing a product under its consumers. A data product is an interface, and changing it in place breaks the reports built on it. In PostgreSQL, `CREATE OR REPLACE VIEW` refuses part of this mistake: the new query must produce the same columns, with the same names, order and data types, and may only add columns at the end. It does not catch a change of meaning, because the calculation behind a column may be completely different. Put any change that renames, drops or redefines a column into a new view, `orders_by_day_v2`, beside the old one, and retire the old one once its consumers have moved.

A mesh where none is needed. Four principles, a platform team, a federated governance group and a product owner in every domain are a lot of machinery. Near its end, Dehghani's 2019 article admits that "enterprise scale adoption still has a long way to go". If a handful of sources and one team cover the needs, keep a well-run central store with a named owner for each table, and borrow the parts that pay at any size: owners, described datasets and rules that the machine checks.

Further reading

- Data Mesh Principles and Logical Architecture — Zhamak Dehghani, 2020: the four principles, the parts of a data product and the table that contrasts central with federated governance.

- How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh — the original article from May 2019, with the three failure modes of a central data platform and the qualities a data product needs.
- Domain-Driven Design Reference — Eric Evans's summary of the patterns, including the bounded context along whose edges a mesh divides ownership.
- PostgreSQL: CREATE VIEW — the rules for replacing a view and the privileges its users need, as used in the example.
- PostgreSQL: COMMENT — how a description is stored on a database object, which `obj_description` then reads back.

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is a data mesh?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/data-mesh/