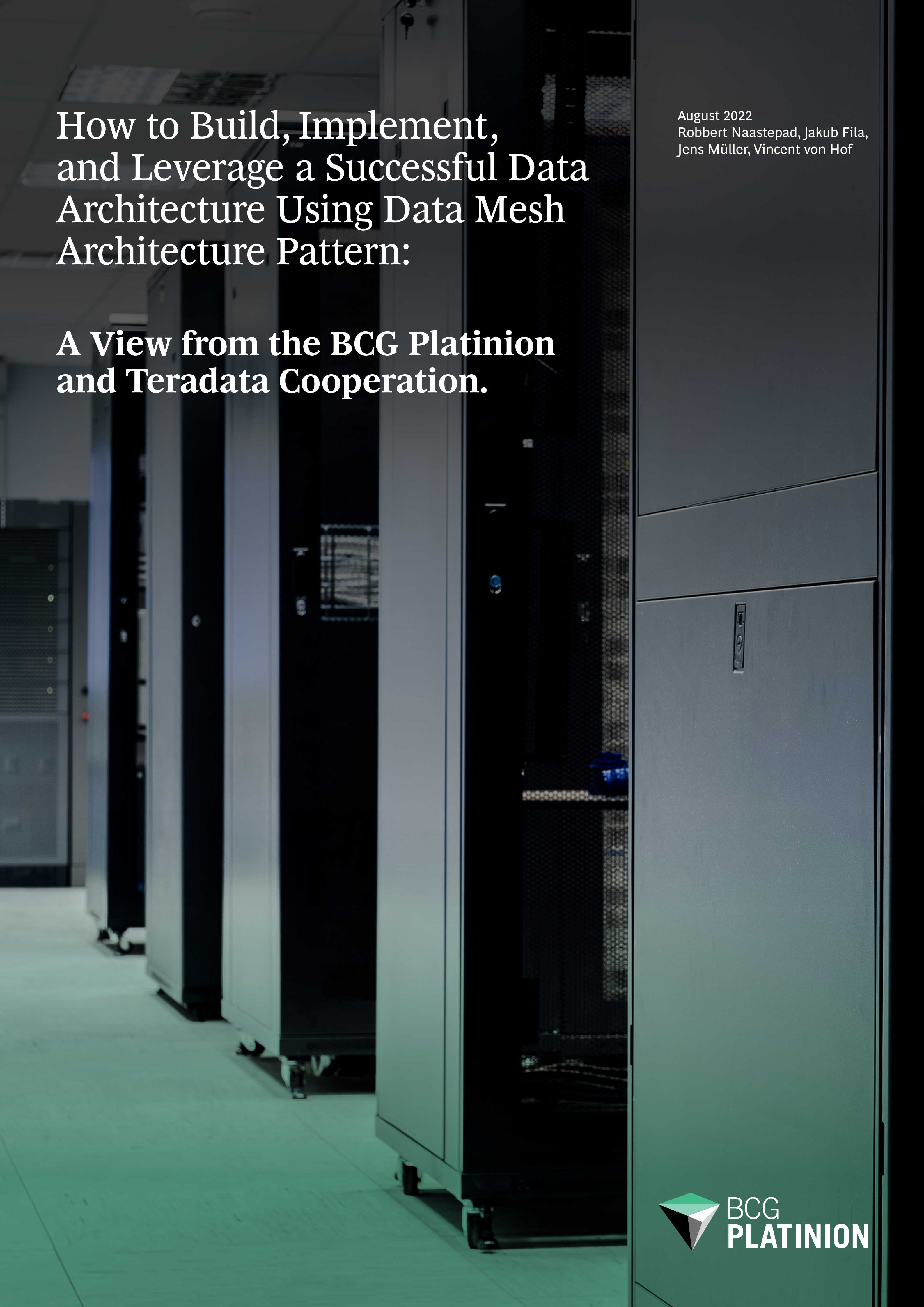


How to Build, Implement, and Leverage a Successful Data Architecture Using Data Mesh Architecture Pattern:

A View from the BCG Platinion and Teradata Cooperation.

August 2022
Robbert Naastepad, Jakub Fila,
Jens Müller, Vincent von Hof

BCG
PLATINION

As a part of Boston Consulting Group (BCG), BCG Platinion provides consulting & engineering services in human centric design, IT architecture, development, and implementation of advanced technology solutions that fuel critical transformation and creation of new-generation business models. Today, our presence spans the globe with offices in Europe, North and South America, South Africa, and Asia Pacific.

1. Data mesh: A new approach building on existing technology foundations

Data mesh is currently a heavily discussed topic, as it is perceived as game changer in data architectures. However, it is often misunderstood as a concept, as are its consequences.

What is data mesh, then? It is a shift in architectural and business usage paradigm that reorients data architecture from general towards business-specific to a company.

It is, however, not a remedy for every issue, nor a silver bullet to solve any fundamental data problem. Let's start the article with an honest look at what data mesh really is, and what it is not.

Data mesh IS: A client-centric way of organizing data and data interfaces for efficient and business-aligned consumption and monetization of the data. It serves business domains with relevant, timely, and high-quality data views and perspectives packaged as business-relevant data products or services. It shifts the paradigm of designing data architecture to business derived and business oriented. It is also a way of deconstructing a highly entangled or coupled, monolithic data architecture into a domain-oriented architecture, delivering on the premise of so-called self-service. Decentralization and federated governance are concepts at the center of this architectural style, enabling its usability and adaptability.

Data mesh IS NOT: A replacement for a data warehouse, data lake, or data lakehouse, nor is it a substitute for a properly designed data integration or a remedy for poor data management at the back end or in core systems. It is not a magic key for data to organize itself in an organization that has no proper data organization and lineage between its systems and repositories.

Data Mesh SHOULD NOT: Be confused with Data Fabric. Although both are data management architectures, a data mesh produces data products specific to (business) domains. A data fabric produces many data artifacts of which a data product can be one. Where a data fabric is a more technical focused architecture, data mesh tends to lean more towards the organization of data management and is an information architecture. The data fabric center is metadata, that of a data mesh is the domain.

1.1 Principles of data mesh

In a canonical design of data mesh, five principles need to be highlighted:

1. **Domain orientation**
2. **Data as a product**
3. **Self-service**
4. **Federated governance**
5. **Agility**

Each of these plays a vital role in building the value of a data mesh solution, as data mesh is more a business philosophy than a pure technical concept.

Domain orientation

Data is organized across domains, with clear domain ownership from operational sources and (optional) centralized data to dispositive data consumption. The examples might be party/client/partner domain or deposits domain in the banking area. Each defines its main business objects, data describing them, and ways the data is served, also involving direct owners in the process of defining the domain.

Data as a product

Data cataloging should allow discovery of the data by its consumers and stakeholders. APIs that operate on a self-service premise ensure interoperability of the data products. Data quality is declared in data delivery agreements. Data products are simply a concept, and the implementation of data entities and the services enabling them. They typically form a group of business-meaningful data objects that are served and operated. As an example of a product: Contracts in the service provider company would cover all business-relevant information on contractual agreements with the company's clients, providing insight into the data as well as potentially manipulating the data. The products can be built as raw business semantics on the data, or they can build on other, more fundamental products.

The products are typically delivered and developed further using an agile approach, with data ownership and a product team established for each of them.

Self-service

Data product teams typically develop the technical components needed to extract, load, transform, store, publish, and disclose the data. The data is accessible by means of data or service contracts, and the services providing them operate at scale. Data consumers only need to know the contract and endpoint to be able to identify the client.

In existing data architectures, technical components are often set in stone and managed by a centralized team. The interaction with that team is a quintessential requirement. Data owners must await their support before they can proceed. In self-service architectures, the data owners can shape the products themselves, deciding how the data will be served and consumed.

Federated governance

The data is harmonized in a centralized fashion using data catalogs, along with metadata management solutions for common discovery, consumption, and interoperability. Data product teams have significant freedom (within the framework of central principles) to build data assets allocated in domains in the most efficient way, matching the specificity of the domain. Computational governance is in place to enforce the data delivery agreements.

Agility

Canonical agile methodology and organization, combined with a set of data-specific capabilities, provide the tools needed to build a data mesh. DataOps provides a framework for the automated and scalable dev-test-deploy of various data assets. Obviously, agile methodologies have proved useful for other, older styles; however, data mesh benefits directly from them, as self-service and domain orientation are best aligned with them.

1.2 Architectural concepts that support data mesh

Data mesh as a concept is fresh, but elements of the idea have existed long before now, as have the tools. It is the assembly of the concepts that reorients the data architecture towards business services and provides business domains with the most relevant support. The applicable concepts, which will be explained in the following, are:

- **Domain-driven design**
- **Microservice architecture**
- **API-driven integration**
- **Resource-oriented architecture**

Domain-driven design

Domain-driven design is the most fundamental underlying paradigm. It was formulated by Eric Evans to support modeling and implementation of business services grouped by specific business areas referred to as domains. For the data mesh, the data domain concept is often used to reflect the data specificity of the architecture; nevertheless, it is useful to maintain a holistic view on the domain as such.

Microservice architecture

As a way of deconstructing IT systems into loosely coupled parts that support very specific business services and can also be extended and scaled independently of the other chunks, microservice architecture allows service implementation to be specialized towards defined business services and to optimize its design.

Although data mesh patterns would typically benefit from the use of a microservice architecture, it is not a must and other application patterns can be considered that would work equally well. The main issue is to make them work with data mesh.

API-driven integration

This is an integration pattern that connects the application using high-level APIs that are typically implemented as RESTful services using an http protocol. Quality APIs are usually self-descriptive, cover a specific business area, and offer it as a service. They are manageable and allow for loose couplings between the applications and systems.

In fact, various types of APIs and API usage are conceivable for data mesh (see below); the one best fit for the specific purpose should always be used. However, RESTful API is one of one of the most versatile ways of communicating with the data or service platform.

Resource-oriented architecture

Resource-oriented architecture presents resources rather than behavioral APIs or services. A single resource (e.g., current account) exposes all the operations permitted on it, such as read, modify, create a new one, etc. The resource is typically well defined and implements a clear and relevant business term.

Combining the four paradigms mentioned above allows for defining and creating data domains, supporting them with properly designed and implemented microservices, and publishing/consuming data through well-defined, business-meaningful services. All of the paradigms have been known for some time, while the data mesh concept is just a glue that allows them to cooperate to provide business view of the data.

Obviously other choices such as event publishing are also viable and can be implemented depending on the needs and chosen patterns.

2. Data mesh versus data warehouse versus data lake

Data mesh is a functional and behavioral paradigm that puts data consumers at the center of the architecture. It can be implemented in several ways, using different concepts for organizing source data as well as various integration and enablement patterns. Each of the three architecture styles address different business problems:

Data warehouses have been here for many years, serving mostly as a structured source of (largely but not exclusively) structural data, as well as data for reporting, consolidating history, sometimes keeping general ledgers, etc. Their premise is to centralize the data in a single source of truth for the organization.

Data lakes were created to enable capabilities to mass process, store, and categorize the unstructured high volume, velocity, and veracity of data. If the data warehouse pattern is typically schema on write, the data lake pattern is schema on read, making high data volumes available quickly.

Data warehouses and lakes are often combined in a so-called lakehouse that offers the capabilities of both. Although seasoned, they are not obsolete.

Data mesh, on the other hand, is all about self-service and decoupling enterprise data management and consumption into simpler and less entangled data products.

Data mesh as such is not a direct replacement of a centralized data source such as a data warehouse or data lake. There are patterns in place that allow seamless collaboration between data mesh and centralized data stores, representing a single source of truth. In such a case, data mesh would typically be produced by a consolidated storage layer. Moreover, there are architectural patterns and data models that allow a single source of truth to be organized so that it automatically organizes the data into domains and enables easy exposure of data mesh as a prevalent consumption pattern.

There are 3 types of domains:

Source-oriented domain (source domain):

- Sourced from enterprise core applications
- Facts and reality of business
- Immutable timed events / Historical snapshots
- Change less frequently
- Permanently captured

Consumer oriented domain (consumer domain):

- Sourced from data products from source-oriented domain(s) or integration domain(s)
- Fit for consumer purpose
- Aggregation / Projection / Transformation
- Change often
- Can be recreated

Integration domain:

- Sourced from data products from source-oriented domain(s) or consumer domain(s)
- Integration over domains
- Granularity specific to consumer domains requirements
- Change specific to consumer domains requirements
- Can be recreated per source domain

3. Ways of organizing and sourcing distributed data mesh

Distributed data mesh is not a fixed pattern that can be implemented in one canonical way. There are at least two dimensions in which we can make decisions on its sourcing:

- Ways of organizing data (schemas) and allocating/colocating it
- Ways of sourcing the data for distributed data mesh domains

3.1 Ways of organizing schemas for the distributed data mesh

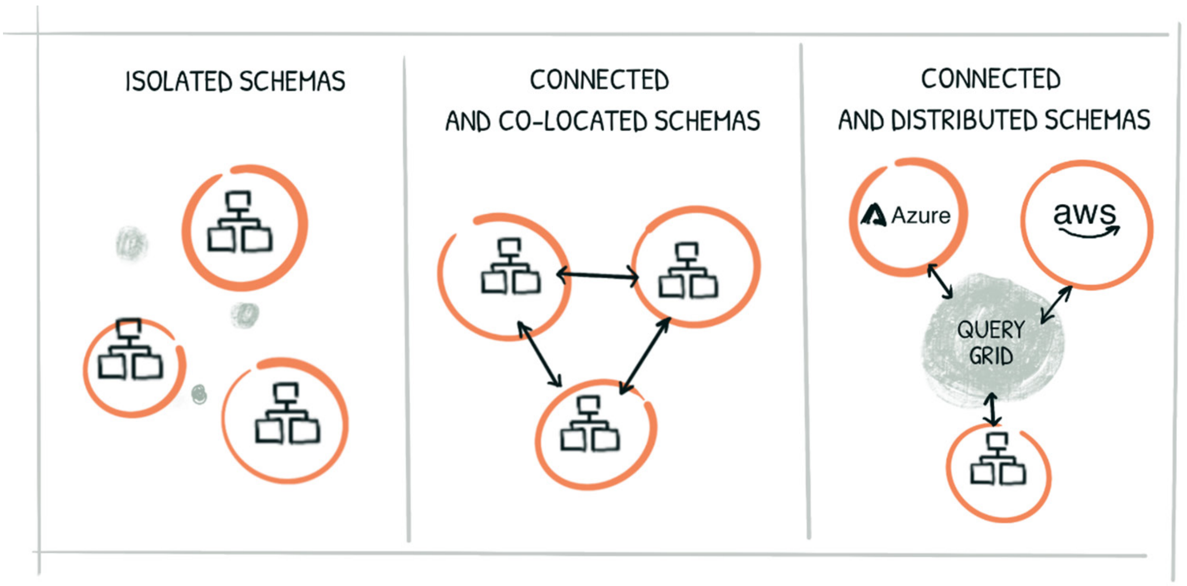
Federating the development of complex data products does not automatically imply federating their deployment. In fact, a range of deployment options are available to organizations deploying data mesh solutions. Different strategies are associated with fundamentally different engineering trade-offs, so it is important that organizations frame these choices correctly and are intentional about their decisions.

In general terms, there are three different strategies for deploying schemas within a data mesh, as defined by vendors such as Teradata:

1. **Isolation**
2. **Colocation**
3. **Connection**

These are not mutually exclusive, and many real-world implementations use a combination of these approaches.

When using environments such as Teradata Vantage, the play is between deployment of centralized image(s) to host collocated domains, host individual domains, or use Vantage as, e.g., a data platform gateway to virtualized data from other platforms.

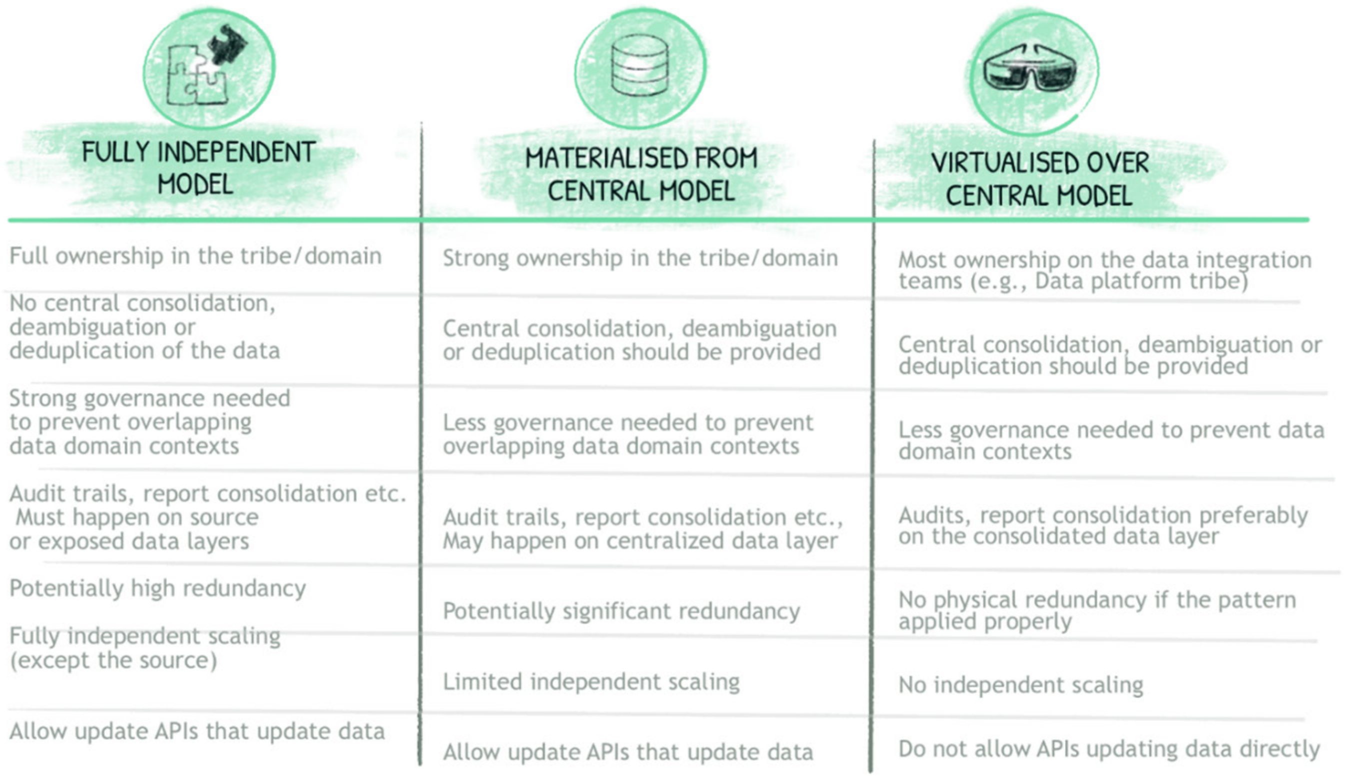

ISOLATED SCHEMAS
CONNECTED AND CO-LOCATED SCHEMAS
CONNECTED AND DISTRIBUTED SCHEMAS
Azure
aws
QUERY GRID

3.2 Ways of sourcing or building the data mesh

There are multiple patterns for building a data mesh, but three in particular deserve a closer look:

- A fully independent model in which each domain is sourced independently
- Centralized sourcing with materialization of the domain
- Centralized sourcing with virtualization of the domain

Of course, variations that mix the models can and certainly are built and used.

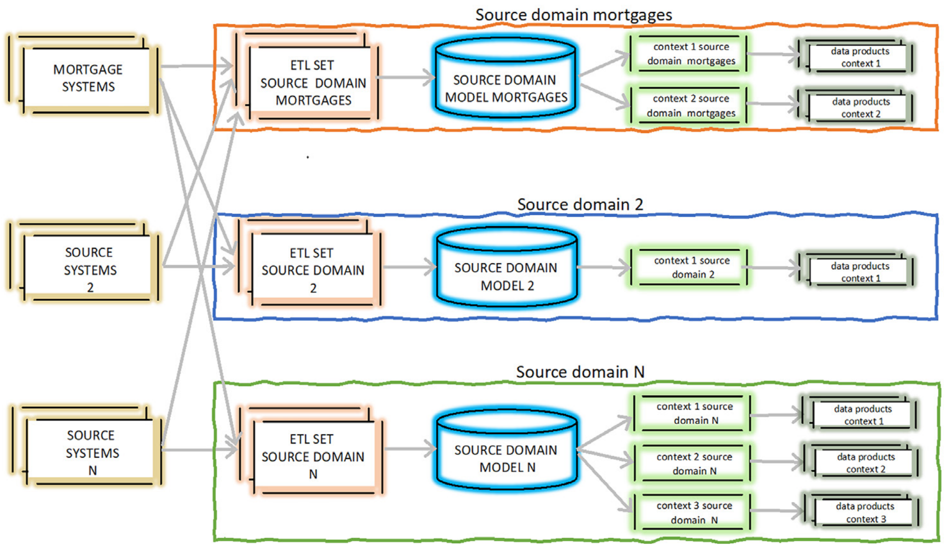


FULLY INDEPENDENT MODEL	MATERIALISED FROM CENTRAL MODEL	VIRTUALISED OVER CENTRAL MODEL
Full ownership in the tribe/domain	Strong ownership in the tribe/domain	Most ownership on the data integration teams (e.g., Data platform tribe)
No central consolidation, deambiguation or deduplication of the data	Central consolidation, deambiguation or deduplication should be provided	Central consolidation, deambiguation or deduplication should be provided
Strong governance needed to prevent overlapping data domain contexts	Less governance needed to prevent overlapping data domain contexts	Less governance needed to prevent data domain contexts
Audit trails, report consolidation etc. Must happen on source or exposed data layers	Audit trails, report consolidation etc., May happen on centralized data layer	Audits, report consolidation preferably on the consolidated data layer
Potentially high redundancy	Potentially significant redundancy	No physical redundancy if the pattern applied properly
Fully independent scaling (except the source)	Limited independent scaling	No independent scaling
Allow update APIs that update data	Allow update APIs that update data	Do not allow APIs updating data directly

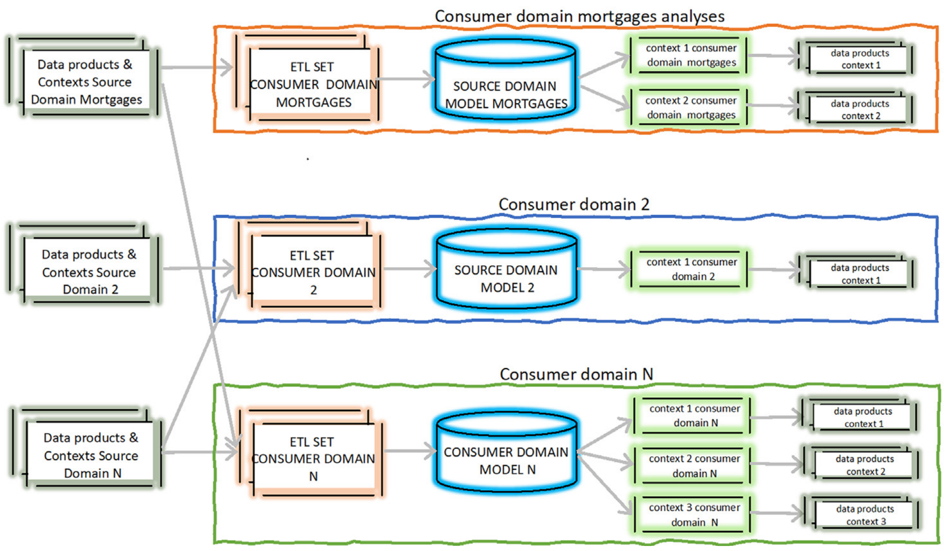
Of course, the models can be combined to maximize the benefits of data mesh. The condition for a successful combination of the patterns is consistent federated governance spanning the entire mesh. The Teradata Vantage platform can be used as a technology to serve any data architecture and is not only capable of implementing each of these 3 models, but also works in its various parts providing sources, serve as centralized repository as well as implement domain repositories delivering data within its bounded context.

3.3 Fully independent model

Each of the domains is logically separated, from its sourcing to consumption; however, the interoperability is satisfied by coherent data delivery agreements.



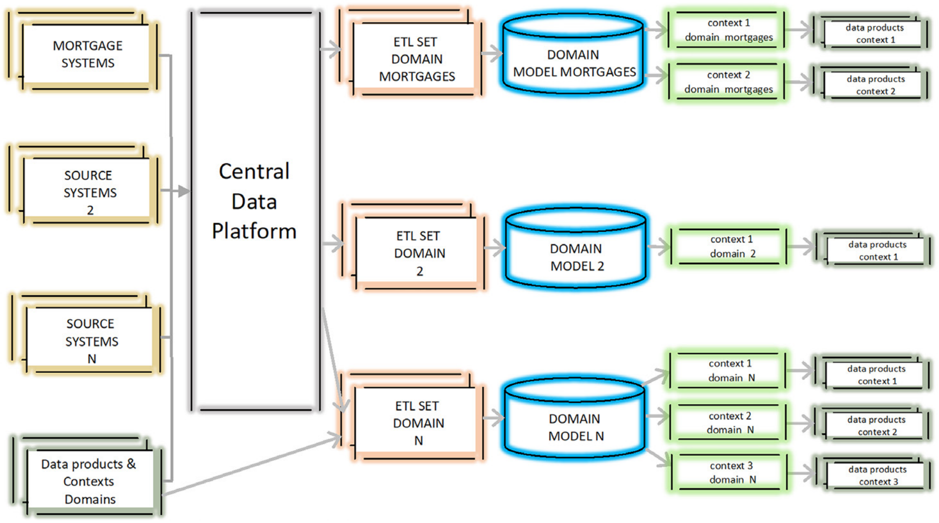
The source domain exposes data products to the downstream domains in the form of API's, databases and streams that are immutable for everything outside of the domain. The source domain can produce other data products too.



Consumer domain mortgages analyses
Data products & Contexts Source Domain Mortgages
ETL SET CONSUMER DOMAIN MORTGAGES
SOURCE DOMAIN MODEL MORTGAGES
context 1 consumer domain mortgages
context 2 consumer domain mortgages
data products context 1
data products context 2
Consumer domain 2
Data products & Contexts Source Domain 2
ETL SET CONSUMER DOMAIN 2
SOURCE DOMAIN MODEL 2
context 1 consumer domain 2
data products context 1
Consumer domain N
Data products & Contexts Source Domain N
ETL SET CONSUMER DOMAIN N
CONSUMER DOMAIN MODEL N
context 1 consumer domain N
context 2 consumer domain N
context 3 consumer domain N
data products context 1
data products context 2
data products context 3

3.4 Sourcing from the central store with materialization of the domain

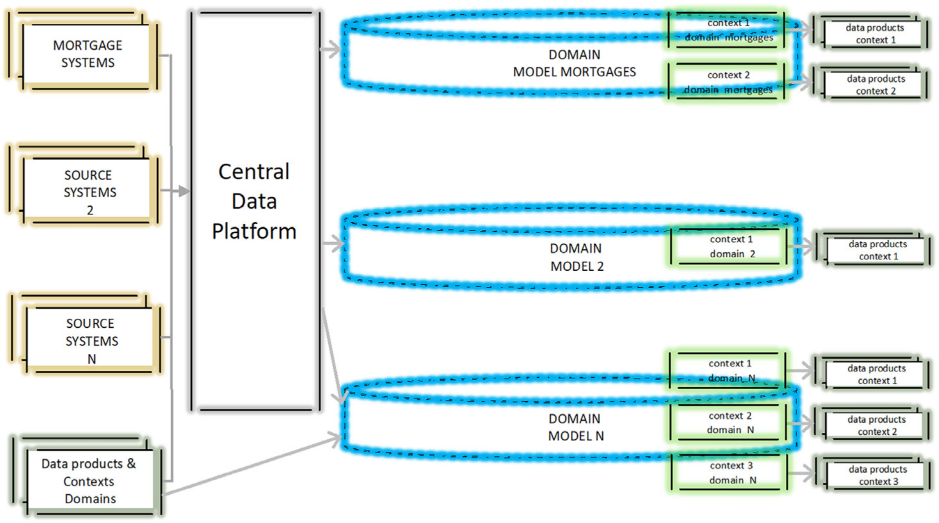
Domains are sourced from the central data store and are typically materialized. Less command is left in the hands of the data product team assigned to the domain. The central domain platform uses the data lake (nowadays mostly stored on native objects stores), the lakehouse and data warehouse patterns, whichever is fit for purpose to store data. The data in it will be aligned on a domain basis, for example in dedicated buckets/accounts/schemas. Domains get a slice of the technology stack. The concepts of source-, integration- and consumer domains are not strong here; however it is still possible for domains to consume data products directly from other domains. Integration domains could exist, but most of the time integration is done on the central data platform. Creating a separate integration domain must be done using good common sense.



3.5 Sourcing from the central store with virtualization of the domain

Domains are sourced from the central data store mainly on a virtual basis. The amount of command and control in the data product team's hands is similar to that of the previous pattern. A strong understanding of the centralized repository is needed. What goes for sourcing from the central data platform with materialization of the domain goes for this way of sourcing building a data mesh too. The central domain platform uses the data lake (nowadays mostly stored on native objects stores), the lakehouse and data warehouse patterns, whichever is fit for purpose to store data. The data in it will be aligned on a domain basis, for example in dedicated buckets/accounts/schemas. Domains get a slice of the technology stack. Virtualization tooling and/or database views support the domain model and its contexts.

The concepts of source-, integration- and consumer domains are not strong here; however it is still possible for domains to consume data products directly from other domains. The concepts of source-, integration- and consumer domains are not strong here; however it is still possible for domains to consume data products directly from other domains. Integration domains could exist, but most of the time integration is done on the central data platform. Creating a separate integration domain must be done using good common sense.



4. How to organize the service landscape to implement data mesh

Data mesh is always implemented using domain-driven design (DDD). The business areas are divided into domains, and common functional and model chunks are closed by bounded contexts as a way to enable data mesh.

Microservices and modern integration and service enablement techniques usually follow, but it is DDD that lays the foundation for constructing a successful data mesh implementation.

There are several techniques for subdividing business capabilities into the domains as well as for construction those; however, it is fairly easy to single out a stereotype of the data domain.

4.1 General organization of architecture layers for data mesh

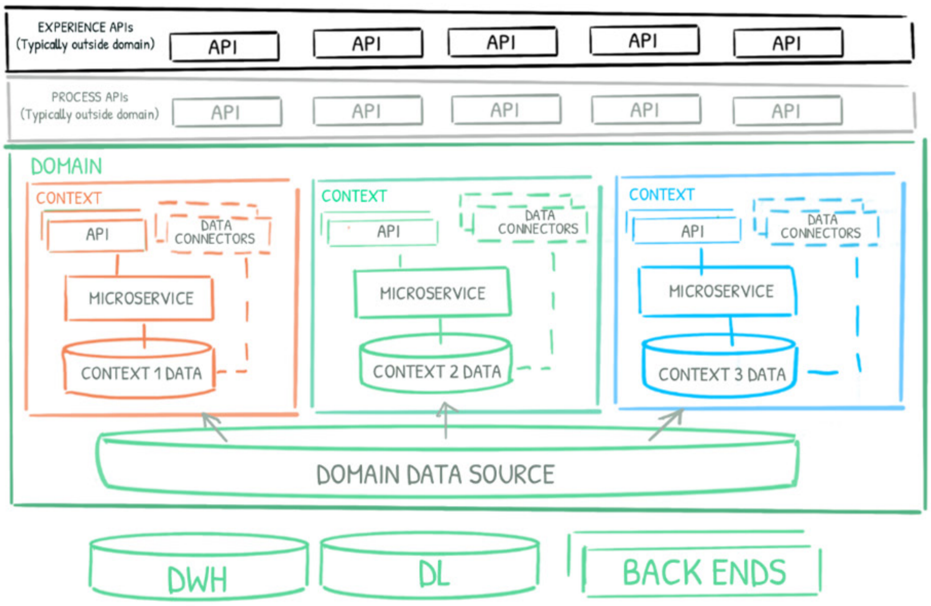
The data domain usually consists of several layers:

- Deep sources or operational systems that provide the transactional/operational data
- Domain data repositories — typically responsible for so-called data liberalization
- (Micro)services to provide the main functionalities enabling and operating the data
- The data service layer serving data as products

The domain can contain one or more bounded contexts. The contexts are usually self-contained but might overlap with other domains and subdomains.

In the properly constructed (and DDD-governed) data mesh solution, the rule is that only the master domain can change its data. Subordinate domains cannot; they can only read the master domain's data. Whenever a subordinate domain is the master of some data, that domain is the only authority to change it.

There is also a substantial difference between the concepts of data and business domains. The first is a provider and change originator for the data and data products and can potentially serve business domains as a data product provider. The second is usually a more behavioral and transactional concept exposing services for operations on the business processes and the state of business objects managed by the organization. Keeping that in mind, one can notice that the patterns/styles in fact complement each other.



Edge components related to thedomain
- Process APIs - typically belonging to the other domain
- Experience API typically belong to the channel
- Backend data sources

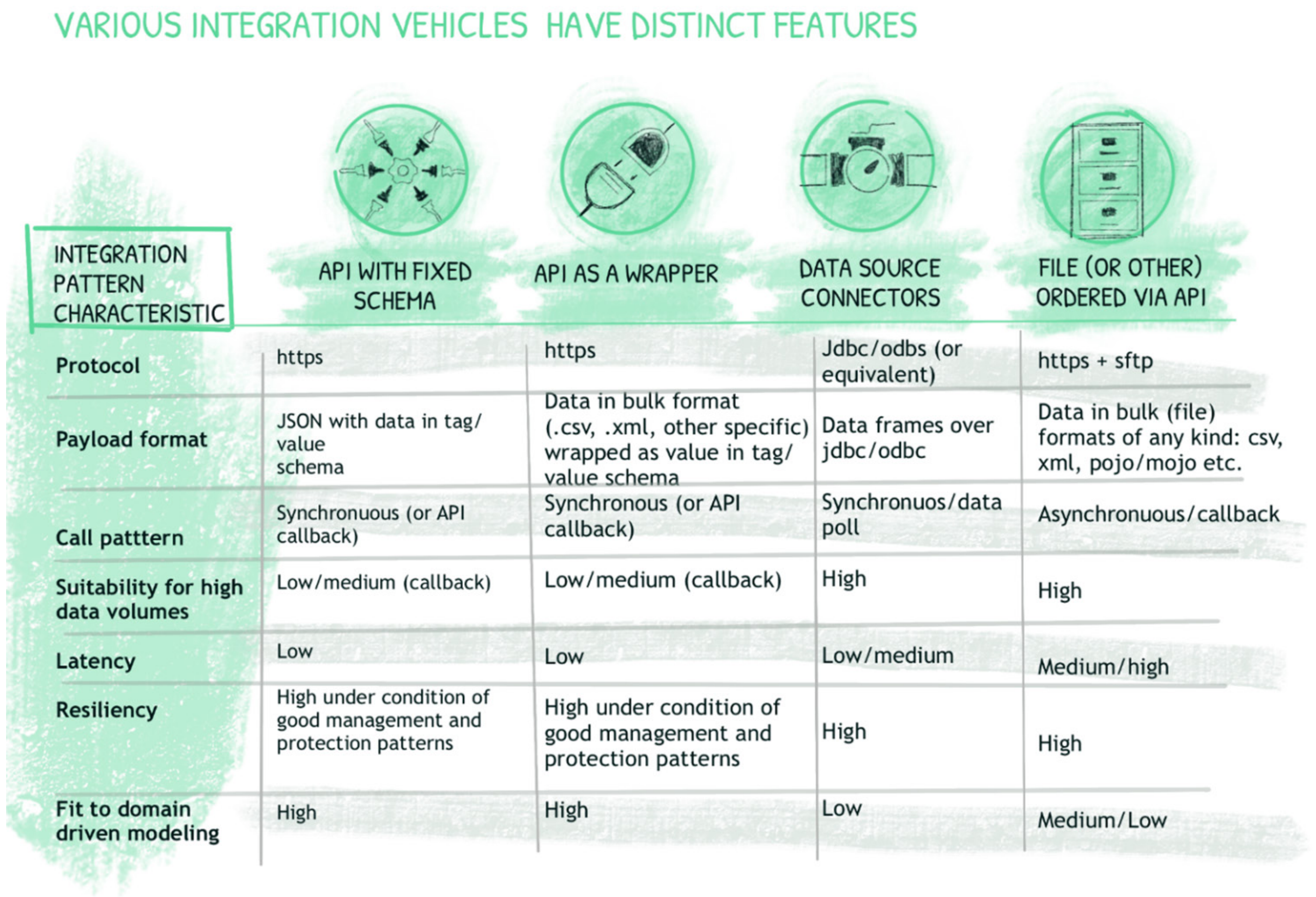
Domain implementation usually contains:
- Definition of bounded contexts and mapping of the data onto them
- Domain data store (material or virtual)
- Microservices (typically at least one microservice per bounded context):
- Domain APIs

To start domain implementation the following are needed to be defined:
- Division into contexts
- Domain data model
- List of microservices
- List of APIs
- Data entities mapped onto services

It takes a very business- and technology-savvy team to properly deconstruct and design the domain, from business services to implementation to a properly distributed data mesh. Once the culture and routine are there, however, adding the domains or deconstructing monoliths into the domains becomes increasingly easy.

4.2 Types of APIs used for data mesh

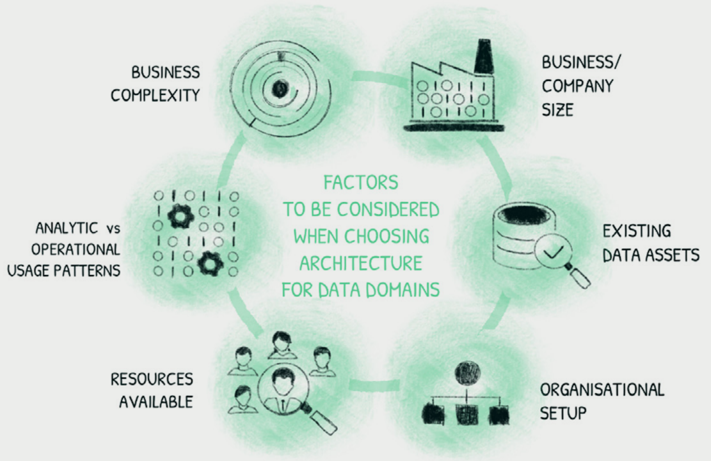
As stated above, API-driven connectivity is one of the most logical choices for offering data products. We consider a couple popular ways to technically share the data in the following. The volume of data to be exchanged determines the use of a particular API over the others. Acquisition of moderate data portions makes direct APIs advantageous, while querying mass data will always favor direct connections to the data or bulk file extracts. Those, however, benefit from being initiated and controlled by API calls.


VARIOUS INTEGRATION VEHICLES HAVE DISTINCT FEATURES

INTEGRATION PATTERN CHARACTERISTIC	API WITH FIXED SCHEMA	API AS A WRAPPER	DATA SOURCE CONNECTORS	FILE (OR OTHER) ORDERED VIA API
Protocol	https	https	Jdbc/odbs (or equivalent)	https + sftp
Payload format	JSON with data in tag/ value schema	Data in bulk format (.csv, .xml, other specific) wrapped as value in tag/ value schema	Data frames over jdbc/odbc	Data in bulk (file) formats of any kind: csv, xml, pojo/mojo etc.
Call patttern	Synchronous (or API callback)	Synchronous (or API callback)	Synchronuos/data poll	Asynchronuous/callback
Suitability for high data volumes	Low/medium (callback)	Low/medium (callback)	High	High
Latency	Low	Low	Low/medium	Medium/high
Resiliency	High under condition of good management and protection patterns	High under condition of good management and protection patterns	High	High
Fit to domain driven modeling	High	High	Low	Medium/Low

5. Ways of building data mesh are dependent on the scale and resources at hand

The decision to build a data mesh requires taking the most pragmatic approach. There is a list of factors to be considered ahead of deciding how to implement it:


FACTORS TO BE CONSIDERED WHEN CHOOSING ARCHITECTURE FOR DATA DOMAINS
BUSINESS COMPLEXITY
BUSINESS/ COMPANY SIZE
EXISTING DATA ASSETS
ORGANISATIONAL SETUP
RESOURCES AVAILABLE
ANALYTIC vs OPERATIONAL USAGE PATTERNS

If the company's IT department has sufficient scale, all the layers can be built or set up in almost every possible way, including building from scratch. Most companies are, however, constrained by the budget, resources, and learning curve needed to master the skills to prepare all the layers.

If ready-made supporting platforms are used to limit the overhead needed to build the solution, the following areas of consideration should be addressed:

DATA VISUALISATION PLATFORM

DATA INTEGRATION PLATFORM

MICROSERVICE BUILD AND HOST PLATFORM

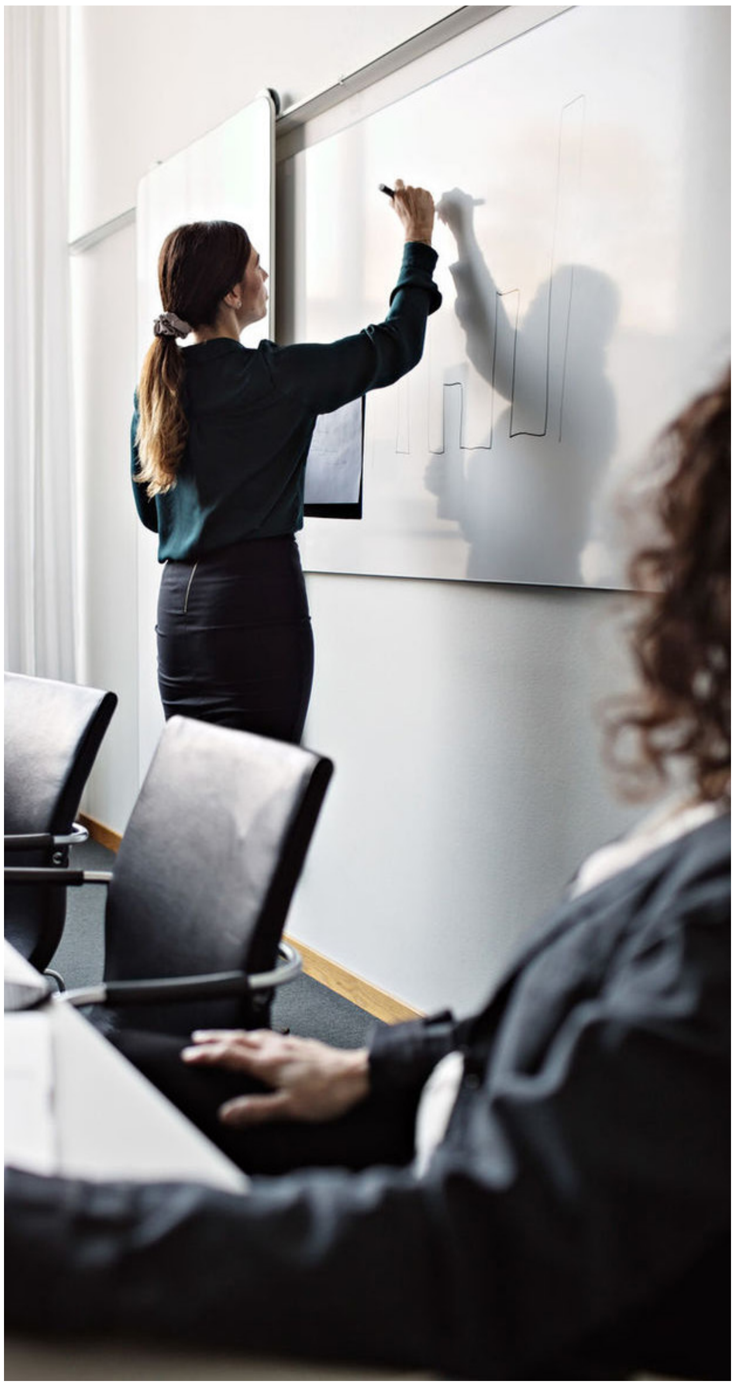
PLATFORM AND INFRA MANAGEMENT

INTEGRATION MIDDLEWARE

METADATA MANAGEMENT

Ready-made platforms are available for each of the areas mentioned. While they do somewhat fact limit the possibilities, they provide a quick starting point and platform that can be used directly to build and host the data mesh components.

The next big step that is likely to emerge or may even already be on the horizon is data mesh as a service offered by significant players. There are many we could mention, but the Teradata Vantage platform has native connections to sources like native object storage, and when using Teradata QueryGrid, this can be extended to Apache Hive and Apache Spark, Oracle, and Google BigQuery. The Starburst Presto connector makes it possible to further extend the connection to a myriad of data engines through Starburst Presto. This makes the platform a ready-to-use solution offering a data mesh skeleton that can be integrated, filled with the data, and provided in the cloud for data clients to use.



6. Dos and don'ts: When (not) to implement data mesh architecture

As with every pattern, there are some limitations and caveats that need to be taken into consideration when it is used contradictory to its objectives—and without considering its limitations it has no chance to deliver on its premises. Below is a list of major points to consider:

1. Do not treat data mesh as a golden hammer to solve all your data problems.
2. Do not try to replace your data warehouse or data lake with data mesh if they are properly fulfilling their function—think about data mesh as an evolutionary step.
3. If a data mesh style is planned for implementation, change or evolve both the technology and data models and the organization—federated governance, domain deconstruction, and strict domain ownership are important.
4. Do not start with technology—technology is an enabler, but objectives come from business definitions of the domains.
5. Leverage modern service and deployment patterns such as cloud, data virtualization, CI/CD, etc. to fully explore deconstructed and additive models.

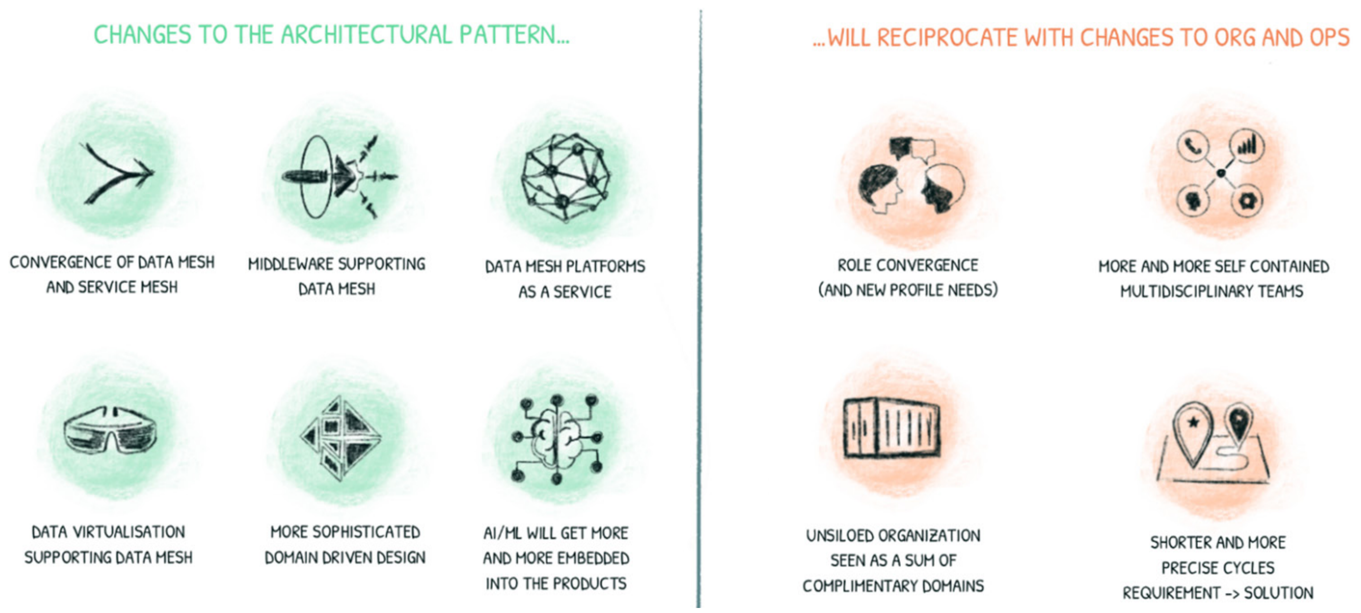
7. Teradata as a provider of core technology to enable data mesh architecture

Teradata has been around for more than 40 years now, providing unparallel capabilities in processing huge amounts of data. It is an original MPP design that stems from its shared-nothing architecture that has resolved problems with storing, selecting, and merging large-scale data sets. Over the years, many capabilities have been developed around Teradata:

- Efficient and scalable data integration tools (through Teradata Parallel Transporter, part of Teradata's tools and utilities that come standard with Teradata Vantage)
- TASM, a best-in-class workload management solution that allows workloads of various types (tactical, strategical, analytical, etc.) to be protected and makes it possible to meet your SLAs
- A scalable cloud offering (Vantage) that allows data processing solutions to be easily scaled to adapt to varying workloads (storage and computation intensity)
- A heterogenous data integration platform that allows various technologies to be combined into one data ecosystem managed from the Teradata Vantage platform
- Separate storage and computing, which makes Teradata Vantage highly suited to adapt cloud capabilities such as scalability, elasticity, agility, flexibility, and efficiency in resource usage
- Teradata connectors, among others the Teradata Kafka connector, which makes it possible to use Kafka to stream data into Vantage
- Teradata QueryGrid, which makes it possible to connect Vantage to a myriad of data engines and understand the statistics in those data engines. This allows Vantage to work together with those engines to determine the best path to your data, thus limiting resource usage
- Reading and writing to native object stores (NOS) like AWS S3, Azure Blob, and ADLS, Google cloud storage, or on-prem object stores that use the AWS S3 API, making it possible to use this cheap storage for archiving or event store purposes
- Bring Your Own Model, which makes it possible to score your models in the database and brings the processing to the data instead of bringing the data to the processing, which can be very expensive in the cloud
- Running R, Python, and Java in the database, again bringing the processing to the data. There is no longer a need to move your data to your R, Python, or Java clients
- A robust set of Teradata Vantage ecosystem management tools, including back and recovery, sandboxing, moving data, Vantage management, and business continuity management.
- Teradata Vantage supports the data mesh concept in all three strategies:

8. Outlook

The future is now, as numerous communities think and work on refining and preparing the new patterns. In the following, the authors speculate on the future evolution of the data mesh style. Some of the anticipated novelties are applicable to the other patterns as well and as such will probably be used widely.


CHANGES TO THE ARCHITECTURAL PATTERN...
CONVERGENCE OF DATA MESH AND SERVICE MESH
MIDDLEWARE SUPPORTING DATA MESH
DATA MESH PLATFORMS AS A SERVICE
DATA VIRTUALISATION SUPPORTING DATA MESH
MORE SOPHISTICATED DOMAIN DRIVEN DESIGN
AI/ML WILL GET MORE AND MORE EMBEDDED INTO THE PRODUCTS
...WILL RECIPROCATE WITH CHANGES TO ORG AND OPS
ROLE CONVERGENCE (AND NEW PROFILE NEEDS)
MORE AND MORE SELF CONTAINED MULTIDISCIPLINARY TEAMS
UNSILOED ORGANIZATION SEEN AS A SUM OF COMPLIMENTARY DOMAINS
SHORTER AND MORE PRECISE CYCLES REQUIREMENT -> SOLUTION

The main predictions concern architecture/technology and organization/operations. The changes will be reciprocal, as business changes will create new requirements while, in turn, advancements in technology and architecture will enable business and operational advancements.

8.1 Technology

Data mesh and service mesh will converge

The two patterns will converge as data mesh will increasingly support transactional and operational activities, while analytical data will more and more often be enabled through services defined as APIs or similar.

Middleware will support data mesh

As mentioned in the previous chapters, the enablement teams will be more and more fully supported by dedicated middleware or including data mesh pattern. Services, integrations, and provisioning of data domains will become more and more codeless.

Data mesh platforms as a service

Hyperscalers and traditional analytics repository providers will start building and advertising data mesh platforms offered as a service, in a similar turn of events that led to productized data lakes or lakehouses.

Data virtualization as an enabler of data mesh

Data mesh as a pattern is a natural candidate to be enabled by data virtualization tooling—for example, Teradata QueryGrid. It typically allows for rapid microservice-style deployment of data domains or domains sets. It seems logical that this trend will increasingly prevail.

Domain-driven design sophistication

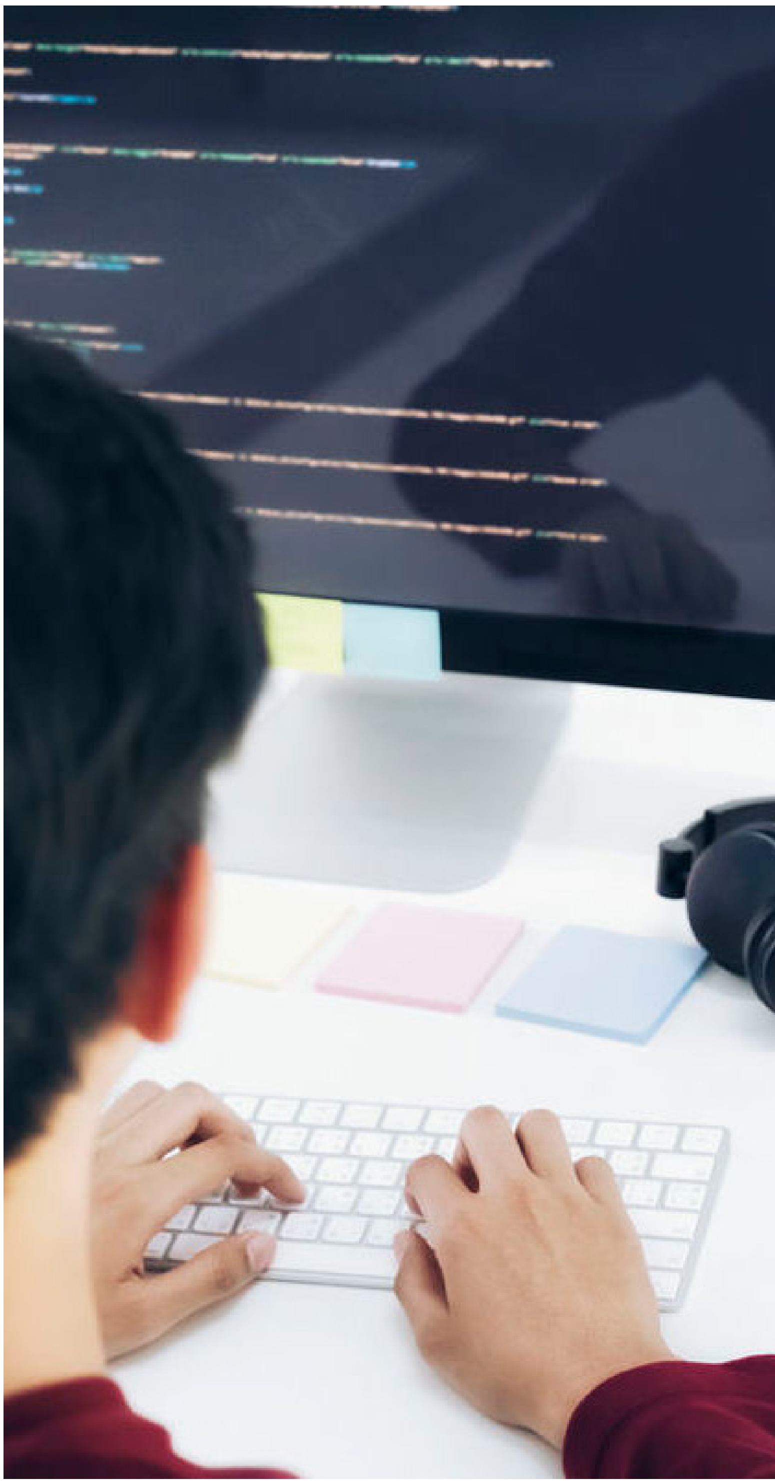
Domain-driven design is a great enabler of data mesh. New techniques of breaking business down into domains and designing the domain will gain ground and DDD will be automatically linked with the data mesh pattern.

AI/ML will be embedded in data mesh

A prediction that is relevant for both data mesh and lakehouse or similar patterns assumes that AI and ML will be embedded in the data platform to properly match and translate between business and technical semantics. Ultimately, this would enable an AI-supported fetch of the results for the queries requested in business language. Support in the organization of data, use of semantic graphs, self-organizing structures, and database management will also find their application.

8.2 Organization and operations

Organizational and operational changes will enable development of architectural standards and patterns. The process of agile methodology-driven convergence of roles and skillsets will continue. Analysts and data scientists will acquire technology proficiency, while IT-oriented individuals will gain more awareness and excellence in the use of requirements and business process analysis. The teams are and will continue to become multidisciplinary, typically organized in tribes or similar structures. Cooperation between topic- or domain-oriented groups should remove siloes in organizations, which is the main prerequisite for federated governance over the data and the domains.



9. Conclusion

Data mesh is a promising but already widely adopted pattern that allows some significant shortfalls of patterns used so far to be overcome. It moves the development closer to the owners and users of the data while retaining their overall business alignment through federated —preferably computational—governance.

The market adopts the data mesh concept while the vendors of IT solutions and service providers develop data mesh as a service or product.

Data mesh is a style likely to coexist and integrate with patterns used so far such as data lakes or lakehouses. It is also typically a driver of organizational and operational changes in large organizations, leading to more efficient handling and processing of data.

As a company with 40 years of experience, Teradata was able to build a concept and products for implementing large-scale data mesh. Teradata Vantage and Teradata QueryGrid allow for every data mesh flavor to be implemented.

BCG Platinion and Teradata partner on projects involving data mesh (and other patterns), shaping and delivering support to data-intensive business organizations.

About the Authors

Robbert Naastepad

Teradata

Robbert started his career as a Cobol programmer on the IBM MVS operating system, using the IMS-DB/DC database management system. After using dBase II/III/III+ and Foxpro, he was introduced to Oracle 6.1.7 RDBMS in 1994.

From that moment on, he got more and more involved in executive information systems, data warehousing, and business intelligence. Robbert evolved from developer and analyst on several projects to technical architect at Oracle into an enterprise data, and now to BI and analytics architect at Teradata. He has been a Teradata team member since January 2017 as he felt Teradata had the right business strategy and products to become a leading company in data management for business intelligence and analytics. Nowadays, Robbert supports Teradata's customers and partners in developing and implementing data architectures using the Teradata Vantage data platform.

Jakub Fila

BCG Platinion

Jakub has a professional background as an aerospace engineer, first in the turbine engine industry and then in the nuclear industry. He is a graduate of aerospace engineering and physics faculties, where he also learned software engineering and started his interest in data processing and parallel programming. He has been working as an IT architect of various levels of seniority for companies such as Accenture, IBM, or Teradata. Jakub is now a Principal at BCG Platinion, helping clients to make strategic decisions and implement the right technologies to fulfill their strategy. He specializes in integration architecture, enterprise architecture, massive data processing, and software engineering and is an enthusiast in MPP platforms and efficient parallel programming techniques. Jakub leads the Application Architecture chapter at BCG Platinion in the EMESA region. His private interests lie in aerospace and triathlons.

About the Authors

Jens Mueller

BCG Platinion

Jens is an engineer to the core. He started by coding J2EE solutions in financial industries and evolved into a domain/enterprise architect over the years through performance measurement and optimization cases, which helped him to build a thorough understanding of traditional relational DBMS and their inner workings (Oracle and DB2). Moving on to strategic IT consulting in 2006, he explored all aspects of IT management. The path to data platforms was laid out in pilot cases on SAS around 2010 and then, from 2012 on, he took over the role of head of design authority capital markets at a large German bank, where he substantially helped set up a data hub based on data lake and streaming technologies that was later moved to a cloud native setup. After rejoining BCG Platinion in 2019, Jens took over the Data Architecture chapter to extend our competences further and work on (cloud) data platform cases in financial industries.

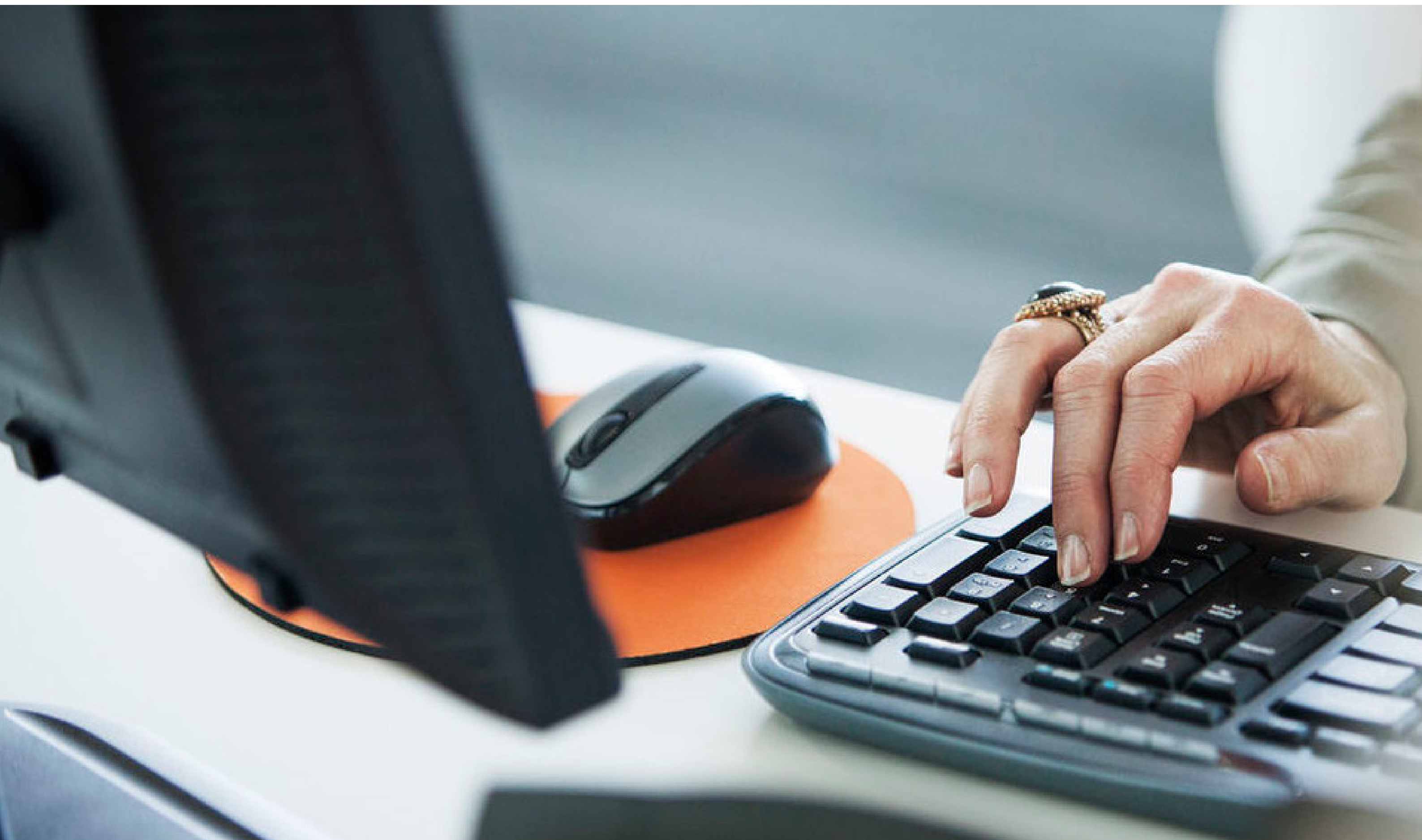
Vincent von Hof

BCG Platinion

Vincent is a Senior IT Architect at BCG Platinion and an engineer by his background and passion. Vincentis a key member of BCG Platinion’s Data chapter.

Vincent started his career as a software developer focused on Java and C#, solving data challenges in the middleware and backend for large-scale data processing and creating data ontologies for the German government, as well as working on solving data challenges on the opposite scale in constrained environments on Android, where he led a development team at a startup for multiple years. He holds PhD in software engineering, majoring in automated test case generation. With over 10 years of development experience, he joined BCG Platinion as an Architect.

His main interests include massive data processing, software engineering, test automation, and cloud architecture. Vincent typically leads large data management projects delivering content and architecture concepts, and implementing and aligning business with IT.



bcgplatinion.com