WHITE PAPER

Build a Data Mesh Factory with DataOps

datakitchen.io info@datakitchen.io

DataKitchen

The data mesh design pattern breaks giant, monolithic enterprise data architectures into subsystems or domains, each managed by a dedicated team. With an architecture comprised of numerous domains, enterprises need to manage order-of-operations issues, inter-domain communication, and shared services like environment creation and meta-orchestration. A DataOps superstructure provides the foundation to address the many challenges inherent in operating a group of interdependent domains. DataOps helps the data mesh deliver greater business agility by enabling decentralized domains to work in concert. In this white paper, we'll explore the benefits and challenges of implementing a data mesh and review lessons learned from a pharmaceutical industry data mesh example.

Thought leaders in the data analytics industry are coming to the conclusion that large, complicated centralized systems too often fail. Why is that, and what can be done about it?

Thoughtworks describes the evolution of the enterprise data platform architecture, which we will rephrase as follows:

First-generation – expensive, proprietary enterprise data warehouse and business intelligence platforms maintained by a specialized team drowning in technical debt.

Second-generation – gigantic, complex data lake maintained by a specialized team drowning in technical debt.

Third-generation – more or less like the previous generation but with streaming data, cloud, machine learning and other (fill-in-the-blank) fancy tools. And you guessed it, managed by a specialized team drowning in technical debt.

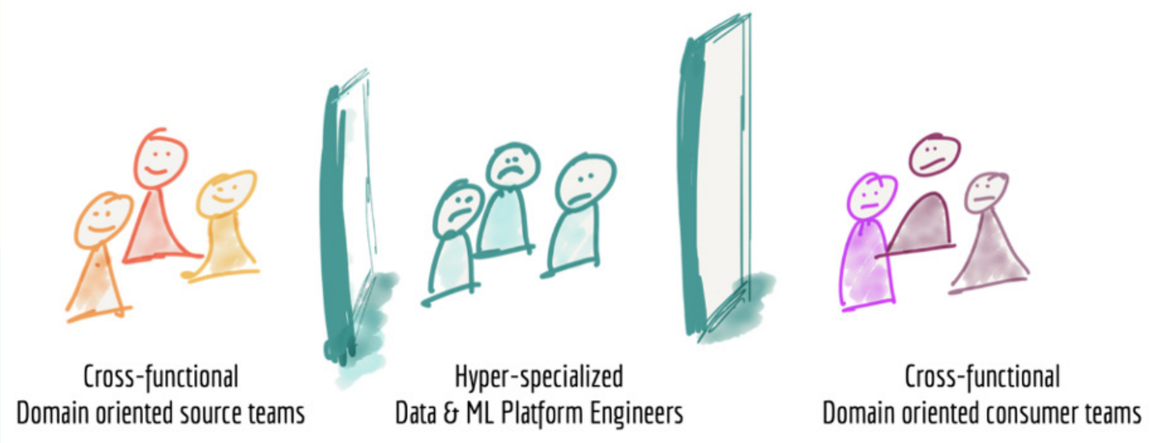

Cross-functional
Domain oriented source teams
Hyper-specialized
Data & ML Platform Engineers
Cross-functional
Domain oriented consumer teams

FIGURE 1: The organization structure builds walls and barriers to change. Source: Thoughtworks

See the pattern? It's no fun working in data analytics/science when you are the bottleneck in your company's business processes. A barrage of errors, missed deadlines, and slow response time can overshadow the contributions of even the most brilliant data scientist or engineer. The problem is not "you." It lies somewhere in between your enterprise data platform architecture and the enterprise's business processes. Below are some reflections upon the failure of modern enterprise architectures to deliver on data analytics agility and reliability:

- Centralized Systems Fail – Lots of inputs and outputs; numerous fragile pipelines; hard to understand, modify, monitor, govern; billions of dollars have been (and will be) invested in vain trying to cope with this vast complexity.
- Skill-based roles cannot rapidly respond to customer requests – Imagine a project where different parts are written in Java, Scala, and Python. Not everyone has all these skills, so there are bound to be bottlenecks centering on certain people. Data professionals are not perfectly interchangeable. Further, the teams of specialized data engineers who build and maintain enterprise data platforms operate as a centralized unit divorced from the business units that create and consume the data. The communication between business units and data professionals is usually incomplete and inconsistent. Centralized enterprise data architectures are not built to support Agile development. Data teams have great difficulty responding to user requests in reasonable time frames. Figure 1 above shows the barriers between data source teams, the platform team and data consumers.
- Data domain knowledge matters – The data team translates high-level requirements from users and stakeholders into a data architecture that produces meaningful and accurate analytics. This is much easier to do when the data team has intimate knowledge of the data being consumed and how it applies to specific business use cases. A schema designer who moves from project to project may not understand the nuances behind the requirements of each data consumer.
- Universal, one size fits all patterns fail – Data teams may fall into the trap of assuming that one overarching pattern can cover every use case. We've seen many data and analytic projects, and mistaken assumptions like this one are a common cause of project underperformance or failure.

INTRODUCTION TO DATA MESH

The data mesh addresses the problems characteristic of large, complex, monolithic data architectures by dividing the system into discrete domains that are managed by smaller, cross functional teams. Data mesh proponents borrow the term "domain" from the software engineering concept of "domain-driven design (DDD)," a term coined by Eric Evans. DDD divides a system or model into smaller subsystems called domains. Each domain is an independently deployable cluster of related microservices which communicate with users or other domains through modular interfaces. Each domain has an important job to do and a dedicated team – five to nine members – who develop an intimate knowledge of data sources, data consumers and functional nuances. The domain includes data, code, workflows, a team, and a technical environment. Data mesh applies DDD principles, proven in software development, to data analytics. If you've been following DataKitchen at all, you know we are *all about* transferring software development methods to data analytics. The concept of data mesh, proposed by Zhamak Dehghani, has taken the industry by storm. We'd like to add a discussion of how data mesh lifecycle management requires DataOps.

Below we'll look at an example of a data architecture partitioned into domains to illustrate data mesh.

You may have used a media streaming application such as Spotify or SoundCloud. A streaming service has to handle a variety of activities that logically partition into different areas:

- Artists – onboard, pay, manage, ...
- Podcasts – create, release, play, ...
- Users – register, manage profiles, manage access, ...

A centralized enterprise application would combine all of these functions and services into one monolithic architecture. A data mesh tackles each functional domain as a separate set of services. In figure 2 below, three domains are shown. A dedicated team handles artists, podcasts and users respectively, but each domain depends on data and services from other domains and operational systems. The domains in the figure are shown receiving data from input data ports (IDP) and transmitting data to output data ports (ODP). The act of onboarding a new artist activates an artist onboarding service that updates the artists domain. Perhaps when a user plays a podcast (within the podcast domain) it triggers an input into the artist's domain, which activates the artist payment service.

Application scalability improves as each group independently manages its domain. For example, the users domain team could add a new data set or new microservices related to users data with complete autonomy.

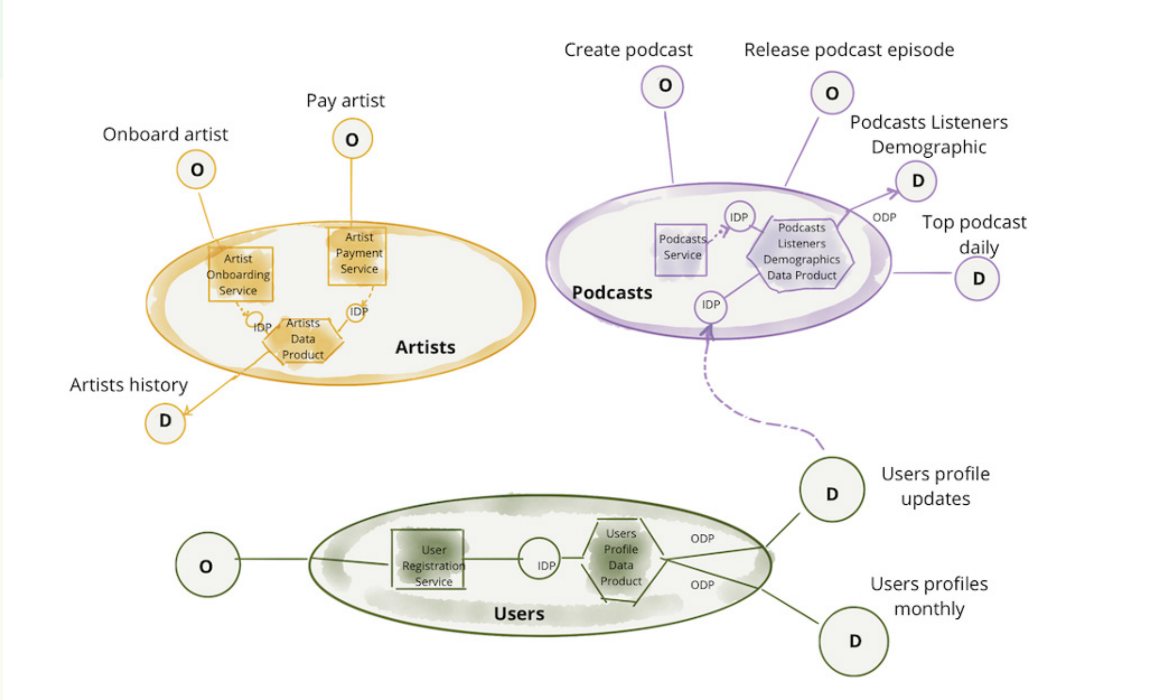


FIGURE 2: The streaming application organized into artists, podcasts and users domains. Source: Thoughtworks

DATA MESH VERSUS DATA FABRIC

In computer architecture, a fabric can be a mesh, so naturally some industry experts have had trouble wrapping their heads around the difference between a data mesh and a data fabric. We've written about data fabrics at length and can testify that data fabrics and data mesh are very different concepts. Like data virtualization, data fabrics focus on harmonizing the diverse technologies and tools that comprise an enterprise data architecture. A data fabric endeavors to unify an enterprise's technical environment into a coherent data platform. That capability can be quite useful for a data mesh.

A data mesh focuses on restructuring the data organization utilizing small teams that each assume ownership and responsibility for a subsystem or discrete domain of the enterprise data architecture. With greater autonomy and focus, teams can leverage composable infrastructure to operate in a more Agile and decentralized fashion. For a lengthy discussion of how system designs tend to mirror an organization's structure, see our piece on "Conway's Law." As an organizing principle for teams and architectures, the data mesh doesn't necessarily require specific tools, but the domain team will face tremendous challenges unless backed by DataOps automation. In summary, data mesh and data fabric are distinct design patterns. You can have one without the other.

CENTRALIZATION VERSUS FREEDOM IN ANALYTICS

In data analytics, there is a constant tension between centralization and decentralization. We like data democratization and self-service analytics because it empowers innovation, but the enterprise must adhere to strict governance standards. We need one version of reality, but centralizing analytics creates bottlenecks. DataOps resolves this struggle between centralization and freedom by organizing and automating workflows and pipelines. A DataOps superstructure can play a foundational role in enabling a data mesh to mitigate issues related to the over-centralization or over-decentralization of data analytics.

IMPROVE AGILITY WITH A DATA MESH

One difficulty of large, centralized enterprise architectures is that they don't lend themselves to using Agile methods. The centralized architecture consists of myriad pipelines that ingest, process and serve data. Everyone on the data team works on their specialized part of the larger system, and they rely upon emails, documents and meetings to stay organized. Figure 3 shows the strained lines of communications that cross functional boundaries. Roles tend to be more specialized on a big team, so there are bottlenecks. With heavyweight processes, it becomes much more bureaucratic to make a change to any part of the architecture. With a steady stream of requests for new data sources and new analytics, the centralized team managing the platform can quickly exceed their capacity to keep up. Customers are on a journey to get insight, and they may not know exactly what they want until they see it. With large systems, it's much harder to iterate toward a solution that addresses a user's latent requirements.

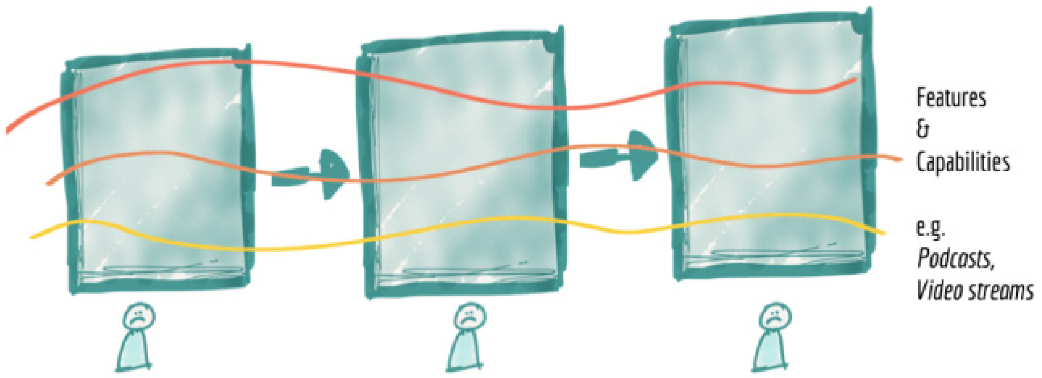


FIGURE 3: When you make a change to a centralized platform, you need to update each component and coordinate between several different teams. Source: Thoughtworks

Data mesh decouples the domain teams from each other. By partitioning the system into pieces, each domain team is able to work uninterrupted and at their natural iteration cadence. Over time, domain teams attain a much more intimate understanding of their data sources and uses, leading to more effective technical solutions. The data team addresses the backlog of tasks more quickly because task coordination is simpler with a smaller team. The data mesh team assumes total lifecycle ownership of the domain. The domain is their internal "product" and the domain product manager is their empowered mini-CEO. You may have heard the term "product thinking" with respect to a data mesh. Success means excellently serving the needs of internal customers. Team members cover all the roles, so there are more hybrid players enabling more flexibility in responding to surges in demand for a particular skill set. When something goes wrong, the domain team has all of the right incentives to iterate on a solution.

The organizational concepts behind data mesh that we have covered are summarized as follows.

- Five to nine-person team owns the dev, test, deployment, monitoring and maintenance of a domain.
- The team organizes around the domain, not the underlying toolchain or horizontal data pipelines.
- Domain data assets, artifacts and related services are viewed as the team's product. The product includes data and operations. Data consumers are the domain team's customers.
- Data Engineers must develop an intimate understanding of data sets to really add value.

TECHNICAL SIDE OF DATA MESH

We've talked about data mesh in organizational terms and how team structure supports agility. Let's take a deeper look at the technical side of data mesh before we provide an application example.

The data industry has a wide variety of approaches and philosophies for managing data: Inman data factory, Kimball methodology, star schema, or the data vault pattern, which can be a great way to store and organize raw data, and more. Data mesh does not replace or require any of these.

The data mesh is focused on building trust in data and promoting the use of data by business users who can benefit from it. In essence, a domain is an integrated data set and a set of views, reports, dashboards, and artifacts created from the data. The domain also includes code that acts upon the data including tools, pipelines, and other artifacts that drive analytics execution. The domain requires a team that creates/updates/runs the domain, and we can't forget metadata: catalogs, lineage, test results, processing history, etc, ...

Instead of having a giant, unwieldy data lake, the data mesh breaks up the data and workflow assets into controllable and composable domains with inherent interdependencies. Some domains are built from raw data. Other domains are built from raw data and the output of other domains. Figure 4 below shows a simplified diagram of a domain receiving input data from an upstream source like an operational system (O) and supplying data (D) to a customer or consumer.

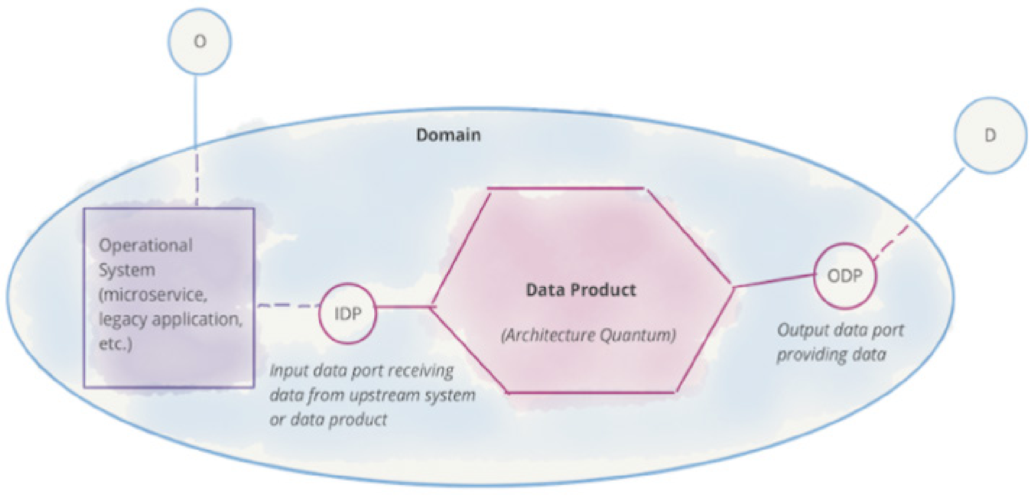

O
Domain
D
Operational System (microservice, legacy application, etc.)
IDP
Data Product
(Architecture Quantum)
ODP
Input data port receiving data from upstream system or data product
Output data port providing data

FIGURE 4: Domain components. Source: Thoughtworks

There's a clear order-of-operations challenge in building systems based on interdependent domains. For example, in the domain diagram in Figure 5 below, imagine that domain 1 has a list of mastered customers, which is utilized by domains 2 and 3. There's an implied producer-consumer, order of operations relationship between these two. It's also very important to make domains composable and controllable.

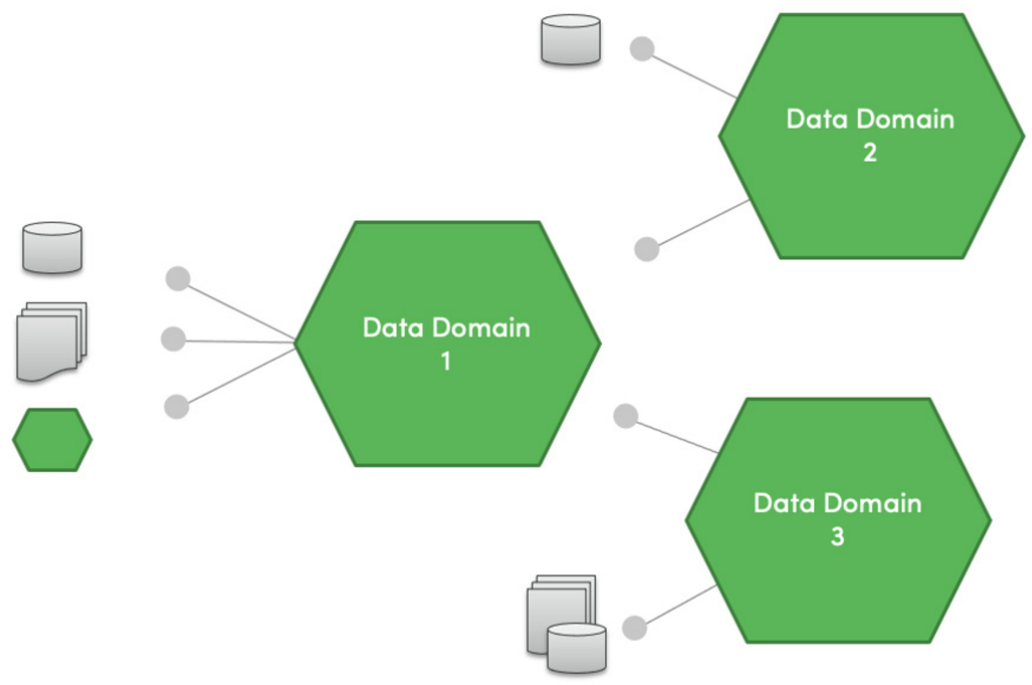


FIGURE 5: Managing composition is a first-order challenge – domains must be composable and controllable.

The consumers and customers of a domain can look at it as a black box. They don't have to think about the domain's internal complexity. The domain is accessed through its external interfaces which can be easily remembered as the four "W's" and an "H:"

- **What** is in the domain – description of the data, e.g., data catalog URL
- **Where** is the data – how to find and access the data securely, e.g., DB connect string
- **When** was the domain created/tested– data process lineage including artifacts like processing results, timing, test results, status, logs, etc., ...
- **With** what components was the domain created, set of raw data (or other data domain) used, hopefully immutable
- **How** was the domain created – steps, code/configuration, toolchain, and processing pipeline used to build the data

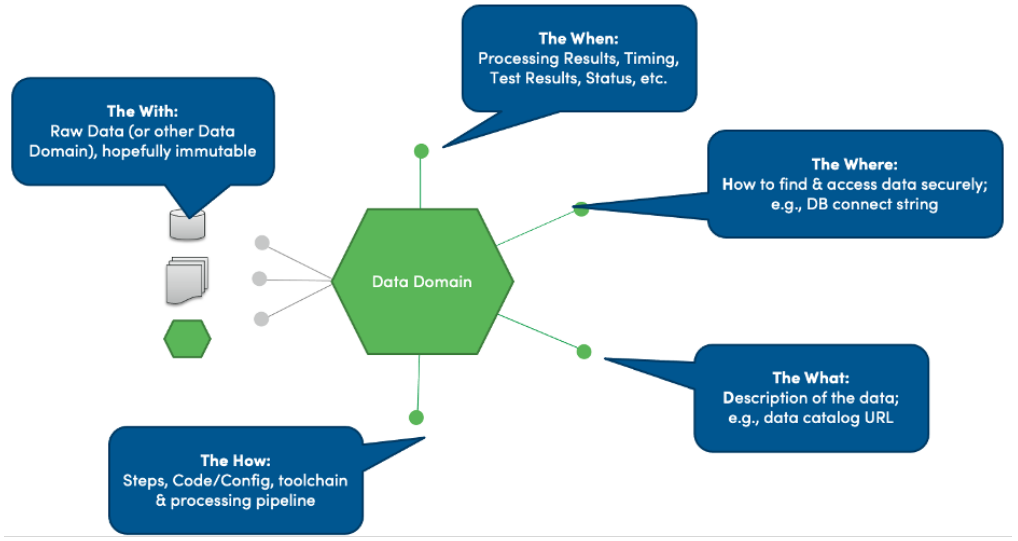

The When:
Processing Results, Timing, Test Results, Status, etc.
The With:
Raw Data (or other Data Domain), hopefully immutable
The Where:
How to find & access data securely; e.g., DB connect string
Data Domain
The What:
Description of the data; e.g., data catalog URL
The How:
Steps, Code/Config, toolchain & processing pipeline

FIGURE 6: illustrates how the four "W's" and an "H" relate to domains.

It's convenient to publish a set of URLs that provide access to domain-related data and services. How does one get access to a domain? How does one edit the catalog? How does a customer get status? Where is the source code that was used? A published set of URLs that access domains allow your team to better collaborate and coordinate. Figure 7 shows a domain with associated URLs that access key tools and orchestrations.

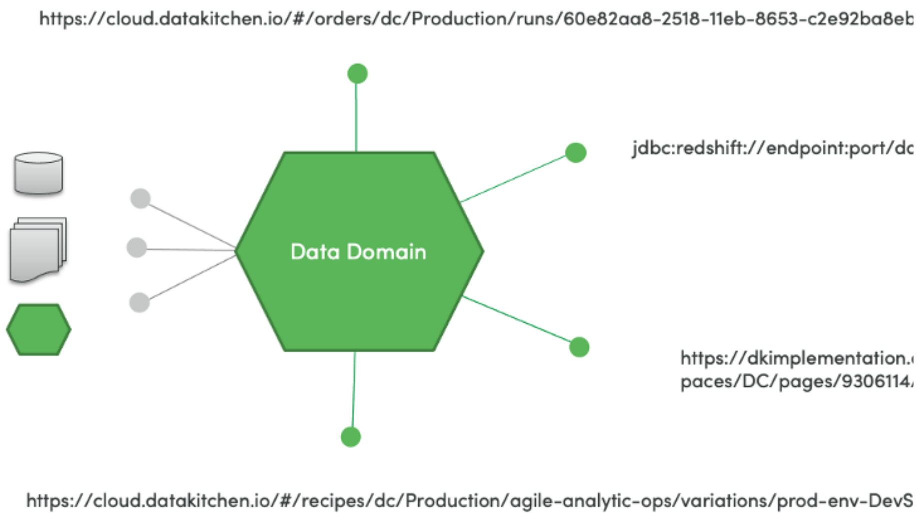

https://cloud.datakitchen.io/#/orders/dc/Production/runs/60e82aa8-2518-11eb-8653-c2e92ba8eb
jdbc:redshift://endpoint:port/dc
Data Domain
https://dkimplementation.
paces/DC/pages/9306114
https://cloud.datakitchen.io/#/recipes/dc/Production/agile-analytic-ops/variations/prod-env-DevS

FIGURE 7: Domain Interfaces as URLs

BENEFITS OF A DOMAIN

A well-implemented domain has certain attributes that offer benefits to the customers or domain users:

- Trusted – Users wish to be confident that the data and artifacts are correct. Trust must be earned, which is why it is so important for a domain to have interfaces that enable introspection and access. Users should be able to ask the domain questions and get answers.
- Usable by the teams' customers – Customer experience improves by virtue of having a dedicated team that attains intimate knowledge of the data and its use cases. Also, the domain must support the attributes that are part of every modern data architecture.
  - Discoverable – users have access to a catalog or metadata management tool which renders the domain discoverable and accessible.
  - Understandable and well-described – terms are defined in a dictionary and the domain has clean, well-designed interfaces.
  - Secure and permissioned – data is protected from unauthorized users.
  - Governed – designed with data quality and management workflows that empower data usage.
  - URL/API Driven – can easily interoperate with other domains through a hyperlink (URL) or application programming interface (API).
- Clear accountability – users interact with a responsive, dedicated team that is accountable to them.
  - Easy to report problems and receive updates on fixes.
  - Users may request new insights/improvements and get them into production quickly.

Organizing a data architecture into domains is a first-order decision that drives organizational structure, incentives and workflows that influence how data consumers use data. Domains change the focus from the data itself to the use cases for the data. The customer use cases in turn drive the domain team to focus on services, service level agreements (SLA) and APIs. Domains promote data decentralization. Instead of relying on a centralized team, where the fungibility of human resources is an endless challenge, the domain team is dedicated and focused on a specific set of problems and data sets. Decentralization promotes creativity and empowerment. In the software industry, there's an adage, "you build it, you run it." Clear ownership and accountability keep the data teams focused on making their customers successful. As an organizing principle, domains favor a decentralized ecosystem of interdependent data products versus a centralized data lake or data warehouse with all its inherent bottlenecks.

DATA MESH IN A REAL-WORLD PHARMACEUTICAL APPLICATION

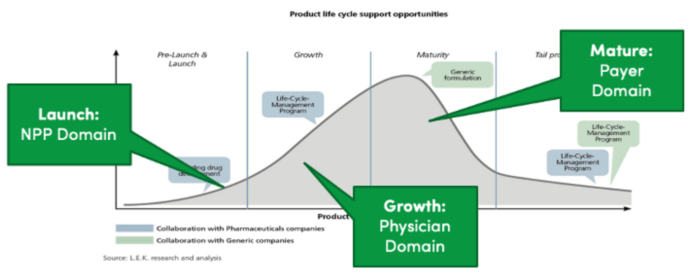
For those embarking on the data mesh journey, it may be helpful for us to discuss a real-world example and the lessons learned from an actual data mesh implementation. Some of our senior staff have extensive experience using the data mesh design pattern with pharmaceutical company data. In the United States, private manufacturers of pharmaceuticals receive a patent for a limited period of time – approximately 20 years. In figure 8 below, we see that the data requirements are quite different for each of three critical phases of a drug's lifecycle:

Lifecycle Phase	Marketing Target	Data Focus
Launch	Patients	Non-Personal Promotion: emails, website visits, even radio ads
Growth	Physicians	Sales, claims data, anonymized patient data
Mature	Payer / Insurance Co	Rebates, formulary

Table 1: Lifecycle phases of pharmaceutical product launch

Each distinct phase of the drug lifecycle requires a unique focus for analytics. During the launch phase, the focus is on marketing to patients through consumer channels. A successful launch lays the groundwork for the growth phase where physicians prescribe the drug (the manufacturer hopes). As generic alternatives become available, the market passes through the maturity phase where cost efficiency and margins become paramount. There are different teams within the pharmaceutical company that focus on the respective target markets.

Commercial Pharma Analytics

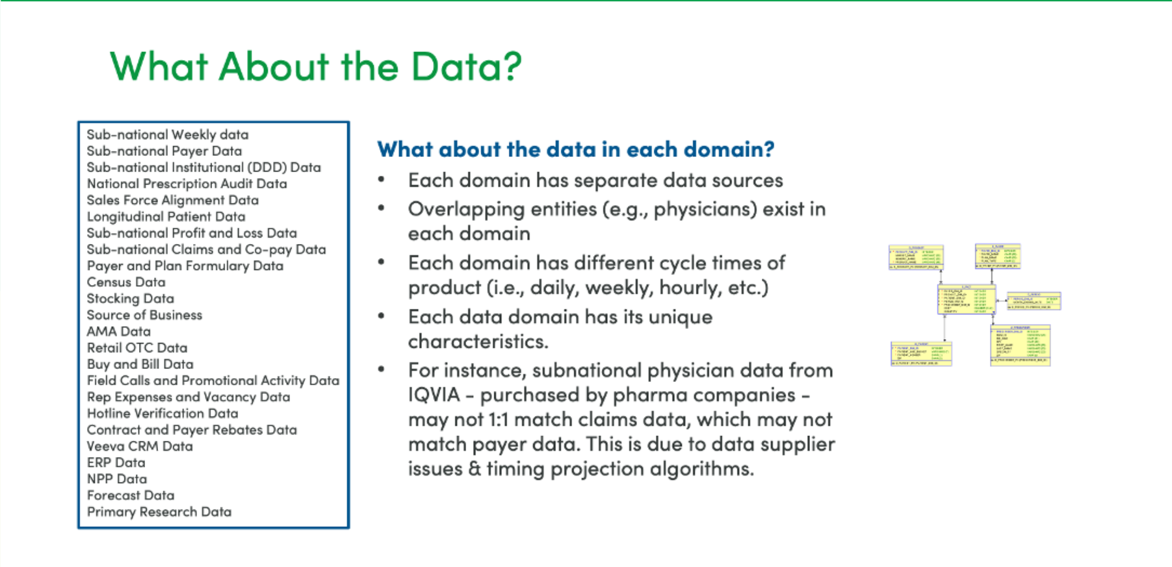

Product life cycle support opportunities
Pre-Launch & Launch
Growth
Maturity
Tail pro
Generic formulation
Life-Cycle-Management Program
Life-Cycle-Management Program
Life-Cycle-Management Program
Launch: NPP Domain
Growth: Physician Domain
Mature: Payer Domain
Product
Collaboration with Pharmaceuticals companies
Collaboration with Generic companies
Source: L.E.K. research and analysis

US Commercial Pharma Domains

- **NPP** (Non-Personal Promotion): emails, web site visits, even radio ads
- **Physician**: doctor (& other outlets) sales, claims data, anonymized patient data
- **Payer**: Payer/Plan, rebates, formulary

FIGURE 8: Data requirements for phases of the drug product lifecycle

Drug companies maintain internal data sets, but also rely heavily upon third-party data (a multi-billion dollar industry). Some data sets are used by multiple teams, but that introduces complexity. Figure 9 shows the wide array of data sets used by pharmaceutical product launch organizations.



What About the Data?

Sub-national Weekly data
Sub-national Payer Data
Sub-national Institutional (DDD) Data
National Prescription Audit Data
Sales Force Alignment Data
Longitudinal Patient Data
Sub-national Profit and Loss Data
Sub-national Claims and Co-pay Data
Payer and Plan Formulary Data
Census Data
Stocking Data
Source of Business
AMA Data
Retail OTC Data
Buy and Bill Data
Field Calls and Promotional Activity Data
Rep Expenses and Vacancy Data
Hotline Verification Data
Contract and Payer Rebates Data
Veeva CRM Data
ERP Data
NPP Data
Forecast Data
Primary Research Data

What about the data in each domain?

- Each domain has separate data sources
- Overlapping entities (e.g., physicians) exist in each domain
- Each domain has different cycle times of product (i.e., daily, weekly, hourly, etc.)
- Each data domain has its unique characteristics.
- For instance, subnational physician data from IQVIA – purchased by pharma companies – may not 1:1 match claims data, which may not match payer data. This is due to data supplier issues & timing projection algorithms.

FIGURE 9: Data feeding the drug product lifecycle domains

Some data sets are a mix of actual and projected data, complicating their use with other data sets that purport to be the same thing, but use a different algorithm to fill in gaps or projections. Two data sets of physicians may not match. They each tell a different story about the data.

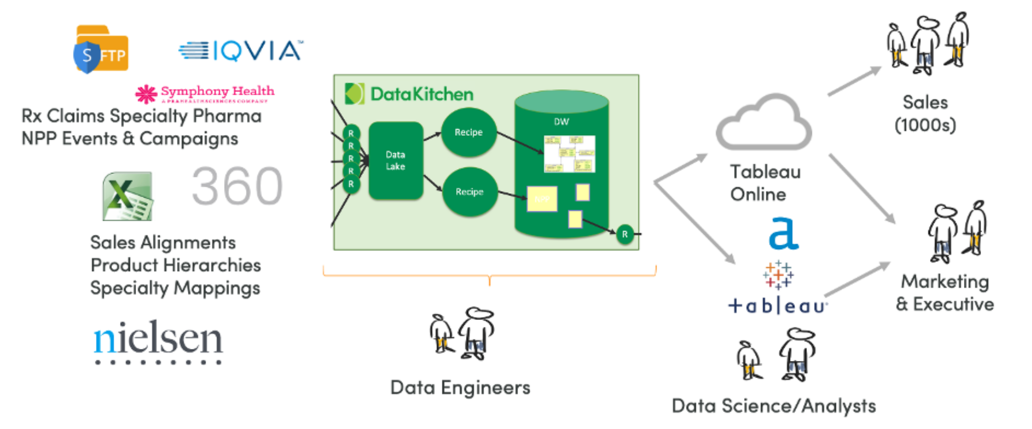


FIGURE 10: Example DataOps architecture based on the DataKitchen Platform

Figure 10 shows an example processing architecture with data flowing in from internal and external sources. Each data source is updated on its own schedule, for example daily, weekly or monthly. The DataKitchen Platform ingests data into a data lake and runs recipes to create a data warehouse that is leveraged by users and self-service data analysts. These users are very important to the company because their success greatly influences the strength of the revenue ramp in the drug's growth and maturity phases.

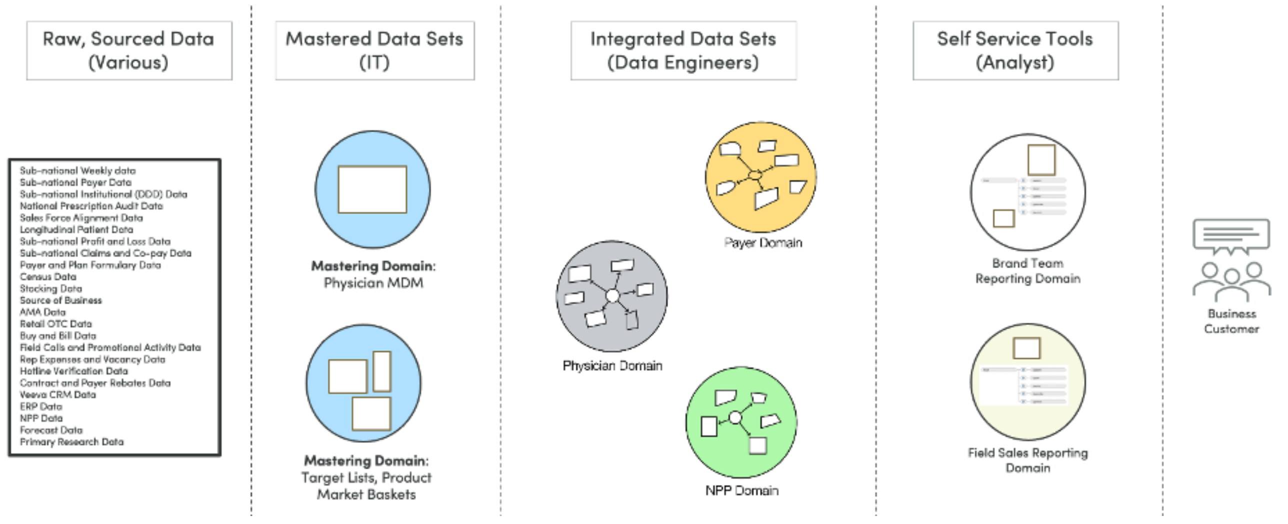

Raw, Sourced Data (Various)
Mastered Data Sets (IT)
Integrated Data Sets (Data Engineers)
Self Service Tools (Analyst)
Sub-national Weekly data
Sub-national Payer Data
Sub-national Institutional (DDD) Data
National Prescription Audit Data
Sales Force Alignment Data
Longitudinal Patient Data
Sub-national Profit and Loss Data
Sub-national Claims and Co-pay Data
Payer and Plan Formulary Data
Census Data
Stocking Data
Source of Business
AMA Data
Retail OTC Data
Buy and Bill Data
Field Calls and Promotional Activity Data
Rep Expenses and Vacancy Data
Hotline Verification Data
Contract and Payer Rebates Data
Veeva CRM Data
ERP Data
NPP Data
Forecast Data
Primary Research Data
Mastering Domain: Physician MDM
Mastering Domain: Target Lists, Product Market Baskets
Payer Domain
Physician Domain
NPP Domain
Brand Team Reporting Domain
Field Sales Reporting Domain
Business Customer

FIGURE 11: Example domain partitioning

Let's consider how to break up our architecture into data mesh domains. In figure 11 we see our raw data shown on the left. First, the data is mastered, usually by a centralized data engineering team or IT. Each of the mastered data sets could be a domain. For example, there may be one million physicians in the US, but for a given drug, perhaps only 40,000 are important. Getting this standardized is important because it affects sales compensation.

The second set of domains are the integrated data sets created by data engineers in the form of data warehouses or analytic data marts. There are facts and dimensions, along with multiple tables, used to answer business questions that come up. In many cases, these are star schemas.

The third set of domains are cached data sets (e.g., tableau extract) or small data sets that self-service analysts can mix with the central data in Alteryx or other tools. Self-service data science teams may require their own segmentation models for building reports, views, and PowerPoints.

Each circle in the figure above could be its own data mesh domain. The separation of these data sets enables the teams to decouple their timing from each other.
In figure 12 below, we see that each domain has its own domain update processing (recipes or data analytics pipelines), represented by a directed-acyclic graph (DAG). For each domain, one would want to know that a build was completed, that tests were applied and passed, and that data flowing through the system is correct.

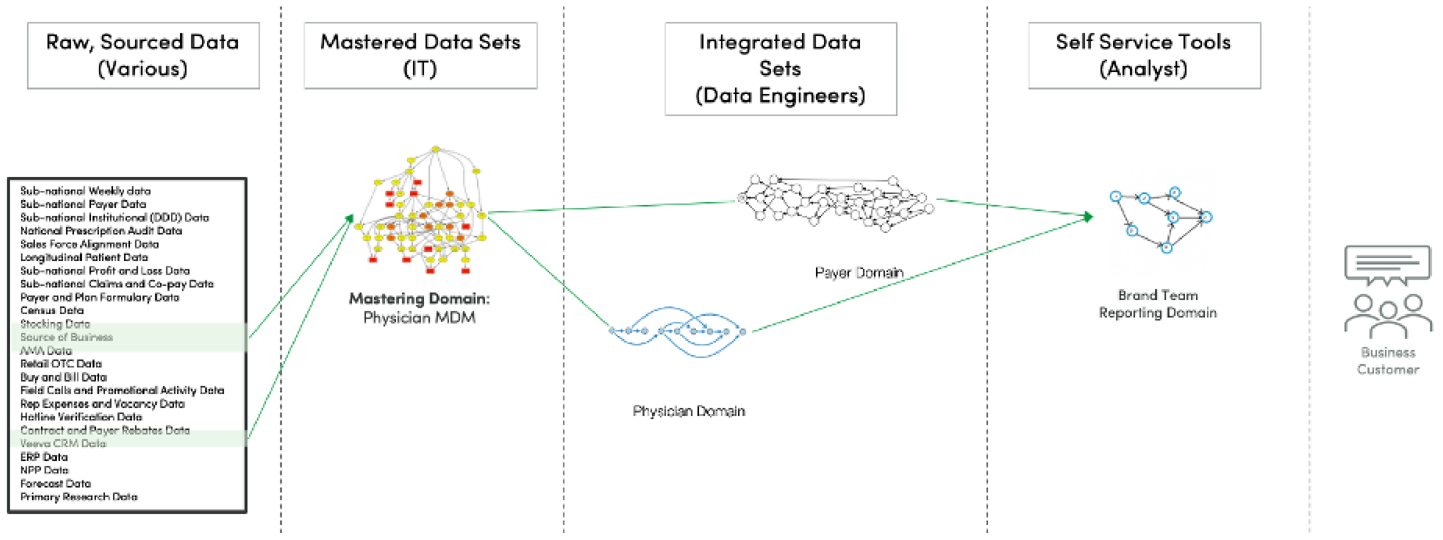

Raw, Sourced Data (Various)
Mastered Data Sets (IT)
Integrated Data Sets (Data Engineers)
Self Service Tools (Analyst)
Sub-national Weekly data
Sub-national Payer Data
Sub-national Institutional (DDD) Data
National Prescription Audit Data
Sales Force Alignment Data
Longitudinal Patient Data
Sub-national Profit and Loss Data
Sub-national Claims and Co-pay Data
Payer and Plan Formulary Data
Census Data
Stocking Data
Source of Business
AMA Data
Retail OTC Data
Buy and Bill Data
Field Calls and Promotional Activity Data
Rep Expenses and Vacancy Data
Hotline Verification Data
Contract and Payer Rebates Data
Veeva CRM Data
ERP Data
NPP Data
Forecast Data
Primary Research Data
Mastering Domain: Physician MDM
Payer Domain
Physician Domain
Brand Team Reporting Domain
Business Customer

FIGURE 12: Domain layer processing steps

There's a top-level DAG relationship between the raw data, arriving asynchronously, the mastered domain, the integrated data sets and the self-service domain. One challenge is that each domain team can choose a different toolset that complicates multi-level orchestration, testing and monitoring. A DataOps superstructure that supports connectors to data ecosystem tools addresses this issue.

Another challenge is how to manage ordered data dependencies. The American Medical Association (AMA) may update its data set of physicians which flows into the physician mastered domain. It then gets used by the physician and payer data warehouses which are eventually used by the self-service teams. We don't want the physician data set in the physician domain and the payer domain to drift apart or get "out of sync," which might happen if they are updated on different iteration cadences. While the goal of a data mesh is to empower teams through decentralization, the overall architecture has to consider the order-of-operations dependencies between the domains. If not intuitively obvious, this is incredibly hard and a major cause of unhappy users.

To manage complexity, the system requires inter-domain communication shown in the table below. A domain query provides information about builds, data, artifacts, and test results. Process linkage aids in multi-level DAG orchestration. Some designs perform process linkage with an event bus that perhaps marks the completion of a DAG by putting an event on a Kafka queue and using a publish/subscribe model. Data linkage refers to the sharing of common tables or the output of data from one domain being fed into another. Finally, there is development linkage. Can development environments with all related domains be created easily? Can they be modified? Is there a seamless path to production?

The many ports or access points of inter-domain communication further illustrate why it's so helpful to support URL-based queries. It enables the data engineer to ask questions, parse responses and set up automated orchestrations, instead of maintaining checklists and manual procedures that someone must execute.

We see a DataOps superstructure like the DataKitchen Platform playing a central supporting role in the successful implementation of a data mesh. DataOps excels at the type of workflow automation that is able to coordinate interdependent domains, manage order-of-operations issues and handle inter-domain communication. The next section will explore the DataOps-enabled data mesh in more depth.

Inter-Domain Communication	Question / Steps Asked
Domain Query	• "When was the last time you were updated?" Successful or failure? Warnings? • "Is the data or artifacts in your domain good? • "Can you prove it with some test results?"
Process Linkage	• "Ok, you start. I am done." • "Ok, you start. I am done AND here are a bunch of parameters you need to keep going."
Event Linkage	"Here is an event: e.g., processing completed, error, warnings, etc."
Data Linkage	"We share a common table (e.g., a dimension table) in our domain."
Development Linkage	• "Can I re-create your domain in development?" • "Can I see the code you used to create it?" • "Can I modify that code in development?" • "Is there a path to production?"

Table 2: Inter-Domain Communication

DATAOPS AND THE DATA MESH

DataOps focuses on automating data analytics workflows to enable rapid innovation with low error rates. It also engenders collaboration across complex sets of people, technology, and environments. DataOps produces clear measurement and monitoring of the end-to-end analytics pipelines starting with data sources. Whereas data mesh talks about architecture and team alignment, DataOps automates workflows that simplify data mesh development and operations. A data mesh implemented on a DataOps superstructure, like the DataKitchen Platform, has a much greater chance of success. One can implement DataOps on its own without data mesh, but data mesh is a powerful organizing principle for architecture design and a great fit for DataOps-enabled organizations. DataOps is the scaffolding and connective tissue that helps you construct and connect a data mesh.

A discussion of DataOps moves the focus away from organization and domains and considers one of the most important questions facing data organizations – a question that almost never gets asked. "How do you build the factory that makes the machines?" The data factory takes inputs in the form of raw data and produces outputs in the form of charts, graphs and views. Before you can run your data factory, you have to build your data factory. Before you build your factory, you would do well to design mechanisms that create and manage your data factory.

Architecture, uptime, response time, key performance parameters – these are challenging problems and so they tend to take up all the oxygen in the room. Take a wider view. Architect your data factory so that your data scientists lead the industry in cycle time. Design your data analytics workflows with tests at every stage of processing so that errors are virtually zero in number. Doing so will give you the agility that your data organization needs in order to cope with new analytics requirements. Agile analytics will help your data teams realize the full benefits of an application and data architecture divided into domains.

Efficient workflows are an essential component of a successful data team initiative. One common problem is that code changes or new data sets may break existing code. Your domain DAG is ingesting data, and then transforming, modeling and visualizing it. It's hard enough to test within a single domain, but imagine testing in relation to other domains which use different teams and toolchains, managed in other locations. How do you allow a local change to a domain without sacrificing global governance and control? That's important to do, not only within a single domain but between a group of interdependent domains as well.

With a DataOps superstructure, like the DataKitchen Platform, you are testing each step in your DAG on whatever infrastructure and tools the domain is using. DataKitchen supports an intelligent, test-informed, system-wide production orchestration (meta-orchestration) that spans toolchains. Unlike orchestration tools like Airflow, Azure Data Factory or Control-M, DataKitchen can natively connect to the complex chain of data engineering, science, analytics, self-service, governance and database tools, and meta-orchestrates a hierarchy of DAGs. Meta-orchestration in a heterogeneous tools world is a critical component to successfully rolling out a data mesh.

The DataKitchen Platform natively provides URL access to nearly all the interfaces that are required for inter-domain communication.

Inter-Domain Communication	Question / Steps Asked	DataKitchen Support
Domain Query	• "When was the last time you were updated?" • "Successful or failure? Warnings?" • "Is the data or artifacts in your domain good? • "Can you prove it with some test results?"	✓
Process Linkage	• "Ok, you start. I am done." • "Ok, you start. I am done and here are a bunch of parameters you need to keep going."	✓
Event Linkage	"Here is an event: e.g., processing completed, error, warnings, etc."	✓
Data Linkage	"We share a common table (e.g., a dimension table) in our domain."	Link to 3rd Party Tools
Development Linkage	• "Can I re-create your domain in development?" • "Can I see the code you used to create it?" • "Can I modify that code in development?" • "Is there a path to production?"	✓

The capability to execute domain queries comes from DataKitchen order runs. Process queries come from calling an order run or running a recipe (orchestration pipeline). Event linkage is natively supported and development linkage stems from Kitchens (on-demand development sandboxes). DataKitchen supports data linkage by integrating with tools that access and store data – there are plenty of great ones.

DataKitchen groups recipes into components called ingredients. Ingredients are composable units that can enable domains to change independently. They can also be orchestrated together within Kitchens.

DataKitchen can help you address the important tasks of how to develop, deploy, and monitor analytics related to a domain.

- Recipes – Orchestrate the whole pipeline (i.e., ingest, process/curate, serve, etc.) inside of a domain. DataKitchen Recipes can serve as a master DAG as well as lower-level DAGs nested inside other DAGs.
- Monitoring Tests – Make sure the data (from suppliers and to customers) is trustworthy. This aids in diagnostics (i.e., detection and localization of an issue). In the DataKitchen context, monitoring and functional tests use the same code.

- Variations – Execute data pipelines with specific parameters. Deliver the correct data product for the consumer, because one size does not fit all. Variations enable the data team to create different versions of their domain to handle development, production, "canary" versions or any other change. Variations unlock a great degree of agility and enable people to be highly productive.
- Kitchen Wizard – Provide on demand infrastructure to the data teams to prevent delays. One concern related to domain teams is the potential duplication of effort with respect to horizontal infrastructure. DataKitchen can be used to automate shared services. For example, DataKitchen provides a mechanism to create self-service, development sandboxes so the individual teams do not have to create and support this capability.
- Kitchens/Recipes/Functional Tests - Iterate to support the "product-oriented" approach. Building the factory that creates analytics with minimal cycle time is a critical enabler for the customer focus that is essential for successful domain teams.

CONCLUSION

Data mesh is a powerful new paradigm which deals with the complexity in giant, monolithic data systems. As an organizing principle, it focuses on data, architecture and teams and less, the operational processes that are so important for agile, error-free analytics. As part of your data mesh strategy, DataOps assists with the process and workflow aspects of data mesh. DataOps automates shared services preventing duplication of effort among teams. DataOps also addresses some of the complexity associated with domain inter-dependencies and enables the data organization to strike the right balance between central control/governance and local domain independence.