GlobalLogic®

Distributed Data Mesh

Authored by:
Viktor Klymenko and Volodymyr Trishyn

Contents

Executive Summary 1

Overview 2

- Current Data Platform Challenges
- Domain-Driven Design for Data

An Introduction to the Data Mesh Approach 5

- Domain-Driven Data Ownership
- Data as a Product
- Self-Serve Infrastructure as a Platform
- Federated Computational Governance

Use Cases 9

Technology 9

- Data Discovery
- Data Governance
  - Data Observability
  - Data Quality
  - Data Security and Compliance
- ETL Enablers
- Polyglot Data Storage
- Self-Serve Infrastructure Tooling

Solutions 14

- Example of Cloud Solution
  - Data Storage and ETL
  - Data Governance and Discovery
  - -Serve Tooling
- Example of On-Premises Solution
  - Data Storage and ETL
  - Data Governance and Discovery
  - Self-Serve Tooling

References 18

Executive Summary

This paper covers the Distributed Data Mesh architecture, a proposed strategy to overcome the current analytical data platform challenges. Those challenges include low-quality analytical insights and reports, extended timeframes to receive new analytical use cases, and high development and maintenance costs for an analytical data infrastructure.

The Distributed Data Mesh architecture attempts to use the Domain-Driven Design (DDD) approach and techniques to transform the analytical data landscape, which is the same as DDD general Software Engineering practices, giving rise to the microservices architecture. Furthermore, it proposes to move the ownership of analytical data from the centralized data engineering team to the decentralized operational domain teams. In this scenario, each domain team will be responsible for the operational aspect of their product and the analytical data product obtained from the domain operational data.

The paper also includes the description of technologies required to enable Distributed Data Mesh, supplemented with a couple of example solutions - to clarify the concept better.

Contributors

Gene Leybzon	North America	Review, Master Template, Approver
Orkhan Gasimov	CEE	Coordination
Viktor Klymenko	CEE	Coordination
Volodymyr Trishyn	CEE	Coordination

Overview

Current Data Platform Challenges

In the current data landscape for data-driven companies and enterprises, we can see a clear pattern in which analytical processing is separate from operational processing, creating a need for an intermediary. The data engineering team takes that place and manages the analytical data platform. Below are the steps for this process:

1. Analysts and data scientists look for the required data sources to implement new analytical or machine learning use cases (they create a new report or dashboard and build a model to find anomalies in some processes, etc.). Since not all required data sources are available (or can be found), and those that are should be modified (transformed and enriched) dramatically, they ask the data platform team to create the required data assets.

2. After the analysis process, data engineers from the data platform team discover that they need to get new data sources from the operational teams.

3. Then after negotiations with all required operational teams, the integration process begins for the different operational domains to onboard new data sources into the data platform.

4. After operational teams onboard new data sources, the data platform team creates the data sources the analysts and data scientists requested.

5. Then the analysts and data scientists implement the required analytical products and make them available to customers or management.

For steps 1 and 2, there are the following questions and challenges:

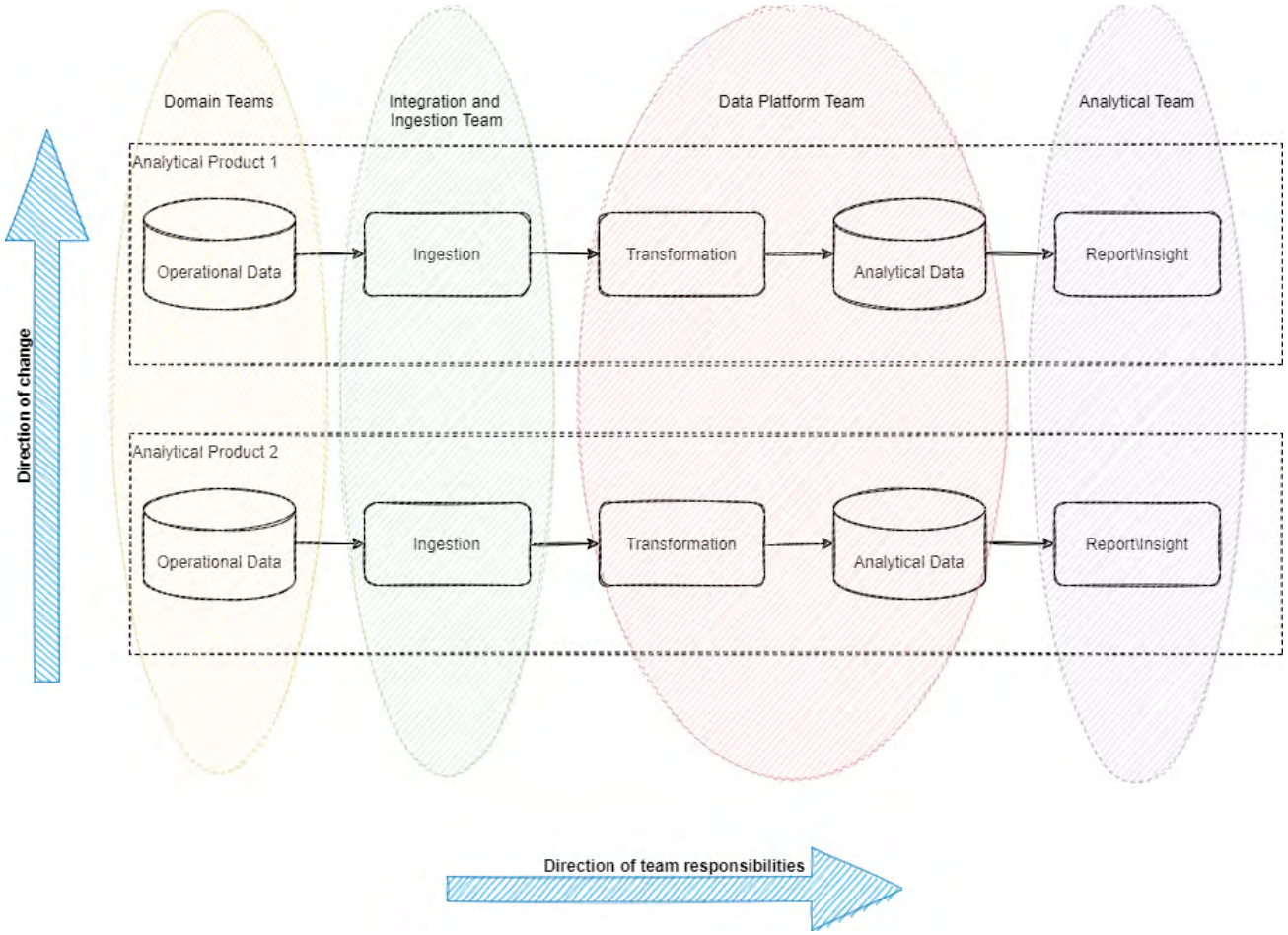
- Where do you find data about a business process or entity?
- Where do you find data about a specific product or source?
- Who is the owner of the data asset, and who can grant access to it?
- What is the business meaning for the data assets and individual fields? For example, can a specific field be used to join another data asset?
- Do other systems use this data? What systems use it and how?

Challenges for steps 3 and 4:

- The data platform team doesn't usually have domain knowledge for data they will onboard and transform, which leads to communication issues and many assumptions.

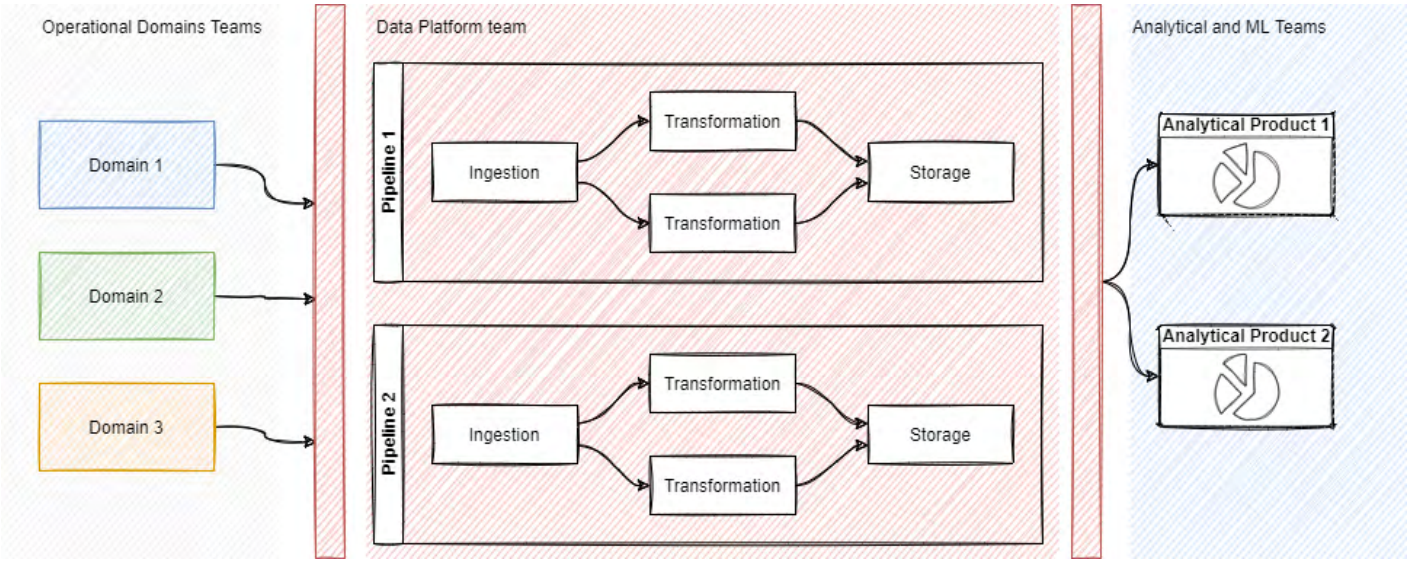
- As a consequence of the previous point, the quality of the data produced may be lower than expected.

- The teams sometimes build the analytical data architecture on top of data pipelines as an architectural quantum (minimum deployable component). However, historically, technical stacks split the teams' responsibilities. Because of this, to create a single data asset, several teams need to engage in the process (operational domain, data integration and cleansing, and data transformation) with dependencies on each other. This leads to a significant amount of time and resources spent. In other words, the existing logical data architecture is orthogonal to the direction of the required changes to build a new data asset:



As for step 5, after receiving the final version of the requested data assets, the data analysts often discover that some data characteristics do not meet the requirements (due to all the challenges above), and the whole iteration begins again.

In this case, the centralized data infrastructure team becomes a bottleneck, with domain knowledge concentrated on operational teams, not interested in the results of an analytical use case from one side, and impatient analysts - from another side:



As a result of all the points mentioned above, creating a new analytical data product takes a significant amount of time and resources, and the quality of the final result is often lower than expected.

Domain-Driven Design for Data

Several years ago, software development was similar to Domain-Driven Design, although it had comparatively minor changes to some enterprise applications that took months to accomplish. One main reason they were similar was that centralized teams of engineers managed and developed large monolithic applications without simple access to business knowledge.

Domain-driven design changed this process when its ideas and methodology gained popularity. As a result, there were corresponding changes in the teams' organizational structures and software architecture. Management organized teams according to business domains with mandatory inclusion of business or product owners into each team - to guarantee that the team had enough business (or domain) knowledge to be self-sufficient and effective.

The software architecture evolved from centralized monolithic applications to decentralized ones, such as microservices, where each domain team was responsible for some relatively independent part of an application, called the domain product.

Similarly, the domain-driven design can help teams overcome data processing and analysis challenges (described in the previous section). This is achieved by replacing the centralized data engineering approach and monolithic platform with decentralized data ownership and the "mesh" of data products.

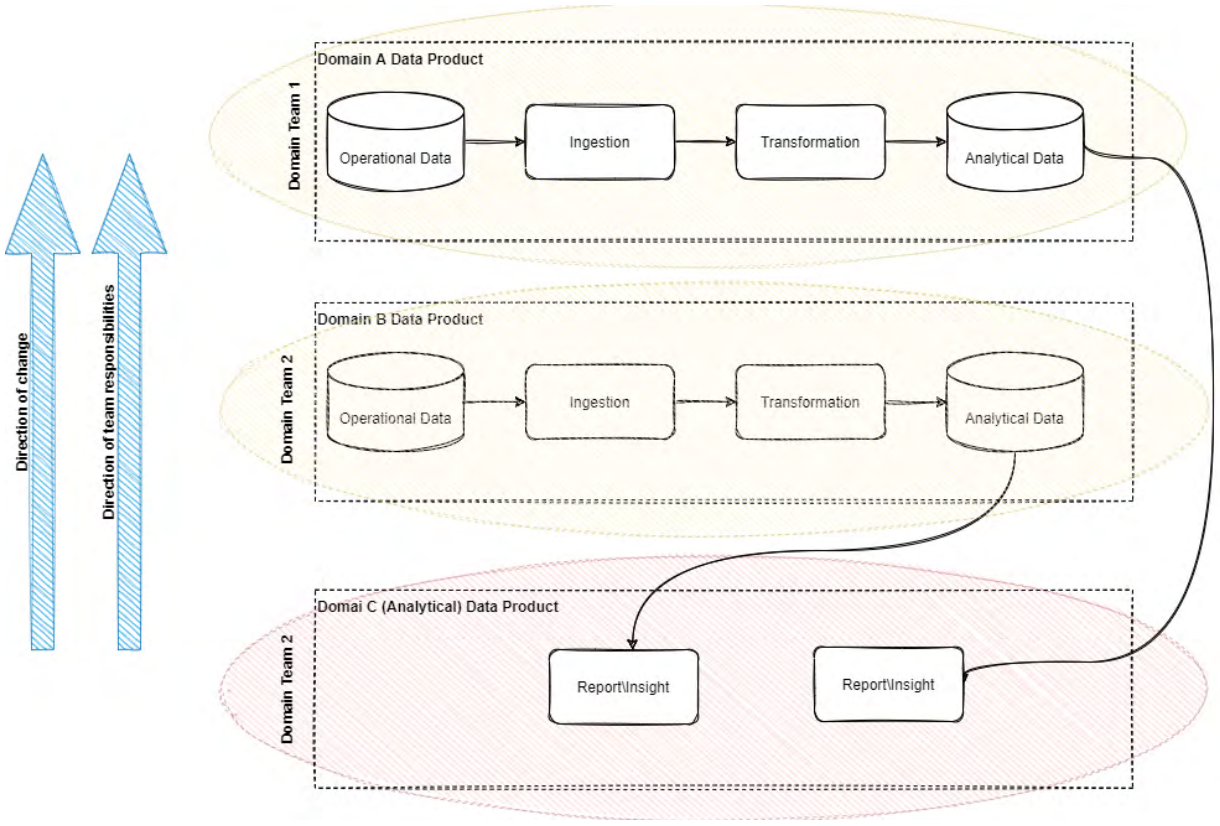
An Introduction to the Data Mesh Approach

Data Mesh is more of an organizational approach than a technical one. To implement Data Mesh, an organization should restructure the team's structure and responsibilities. Then management can select and apply the technology and concrete implementations. To build the Data Mesh, four pillars are mandatory. Let's discuss each of those pillars one by one.

Domain-Driven Data Ownership

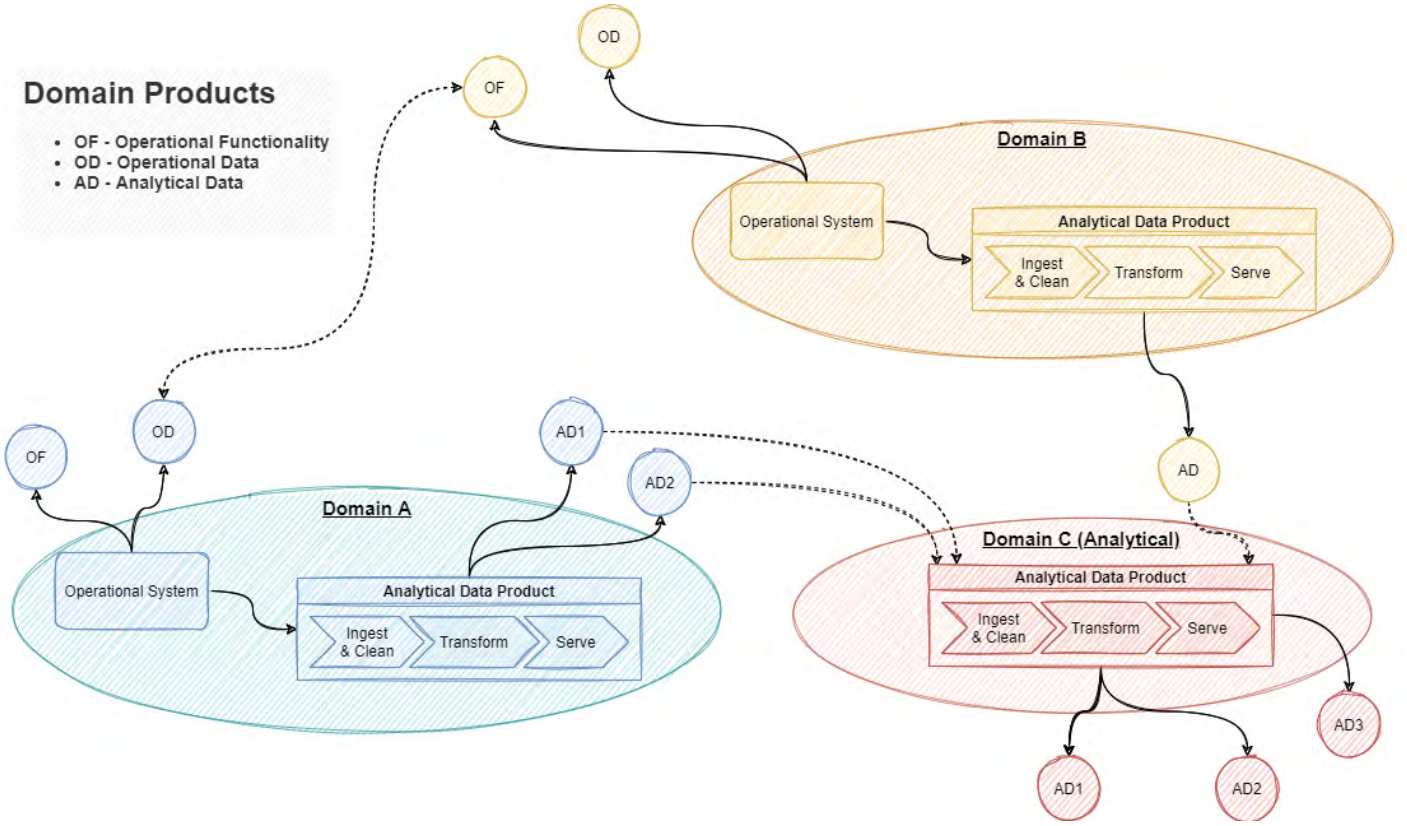
The main idea is to move data ownership closer to the data producers. So instead of a centralized data platform team without domain knowledge, the domain of the operational team separates the responsibility to provide the analytical data. This will remove the bottleneck from the data platform team and guarantee that domain knowledge about analytical data is within the same team that produces this data.

As a result, teams treat the analytical data as a "first-class citizen" of the domain when the concrete implementation of storage (data lake, data warehouse, etc.), ETL or ELT pipelines, and other related tooling are internal details of the domain. Typically, other stakeholders don't care about implementation details as long as teams provide analytical data as an output of the domain. So, then the team aligns the final logical data architecture with the directional changes to onboard a new analytical data source or create a new analytical data product. In this situation, the process only needs one team, which significantly reduces cost and time to market:



Data as a Product

As described earlier, the data platform challenges are the discoverability and quality of the data assets. After decentralizing ownership of the data assets to the domain teams, the next logical step is to treat these assets as a product of the domain and traditional, operational products. Each domain provides access to its operational data and functionality plus exposes the analytical data relevant to the domain. The following is the logical view of the business domains after implementing such an architecture:



As seen in the picture above, the data product is just another mesh aspect. For the data product to be self-sufficient, the product must contain three main components:

- Code for ETL pipelines, data management, and data access.
- Data and metadata.
- Infrastructure pieces to support the previous two components.

Like any conventional domain product, a data product should maintain a certain level of quality, usability, and usefulness for "customers" such as analysts or ML engineers. To achieve this standard, the data product should have the following characteristics:

Discoverability and Addressability

Each data product should be registered and available in a registry or catalog. This allows users to search data assets and obtain some meta-information about them, such as schema description and business meaning of attributes, frequency of updates (and expected staleness), owner of the product, etc. Once found, the product should be easily and uniformly addressable to programmatically access it.

It Is Explorable

Somewhat related to the discovery concept, which means that the data provenance and data lineage should be readily available so that the consumer can make an informed decision about the usefulness of the data for their particular needs.

Trustworthiness

For each data product, the teams must define and maintain the service level objective. Additionally, teams must perform data cleansing and data integration verifications to meet the defined SLO, and they should describe the resulting data quality level for all product consumers.

Security

Teams must establish and maintain the control access procedures, so they only give access to the data product users who need it and control the access according to the sensitivity of the data.

To enable support for all characteristics listed above, the data product operation needs a new team role. The data product owner will be responsible for the following:

- communications with all data product consumers, so they understand their requirements;
- maintenance of the data product key characteristics and SLOs;
- and building a road-map for the data product evolution.

Self-Serve Infrastructure as a Platform

The two previous pillars provide a way for a conceptual and organizational shift towards data mesh to cover all existing problems within data platforms. However, forcing each domain team to maintain their own infrastructure for the big data platform (as well as the technical expertise required) is not an option.

Therefore, to make the first two pillars possible, we need to provide a centralized big data infrastructure as a self-serve platform. Each domain team will be using this infrastructure platform in a self-serve manner, while the platform's support shoulders the specialized infrastructure team.

In order to make this possible, the platform should have the following key characteristics:

- **Domain-agnostic** to allow the infrastructure team to be independent of any domains-specific concerns;
- **Hides all internal complexity,** providing the level of abstraction for domain developers to allow them to build and evolve data pipelines without deep knowledge of underlying platform internals;
- **Has the ability to register information** about new data products and maintain information about existing products, including product ownership, data schemas, data provenance and lineage, description of attributes, etc.

Based on the concrete use cases and organizational needs, such a self-serve platform may have different capabilities. For example, it may need to support multilingual storage, data encryption, schema versioning, federated identity management, etc. We will take a closer look at the platform-enabling components in the technology section.

Federated Computational Governance

So far, we have decentralized analytical data products distributed across business domains. However, with such decentralization, there is a risk that each data product will become an isolated data silo without any ability to correlate data between domains. In this case, most analytical use cases won't be possible.

Therefore, we need the final piece of the puzzle to avoid it - global governance. Instead of a traditional, fully centralized approach, we need the federated governance team, combined with representatives of all the domains for data mesh.

This team has the difficult task of balancing:

- what's governed by global policies,
- what's decided on the domain level,
- which are cross-domain concerns;
- what should be centrally managed;
- and the internal domain details.

The following are examples of global concerns:

- Security and global access control.
- Global standards of data quality metrics and rules.
- Interoperability of the data across domains (eg., there should be a way to join data from the order and customer's domains by a customer ID).
- Compliance (GDPR, PII, etc.).

At the same time, the domain level concerns may look like this:

- Access control to intermediate domain data (inside domain team).
- Applicability of a specific set of globally defined data quality metrics and rules.
- Data assets schemas and granularity.

Use Cases

The current and most common problem for a centralized analytical data platform with a highly specialized technical team that collects, stores, and transforms analytical data from all the domains has a critical flaw: it doesn't scale.

As a result, the organization cannot scale the single platform and single team indefinitely. At some point, the cost of a new analytical data product becomes too high, both in cost and time. As with any computer system, the answer is to scale out instead, and the data mesh is the perfect solution.

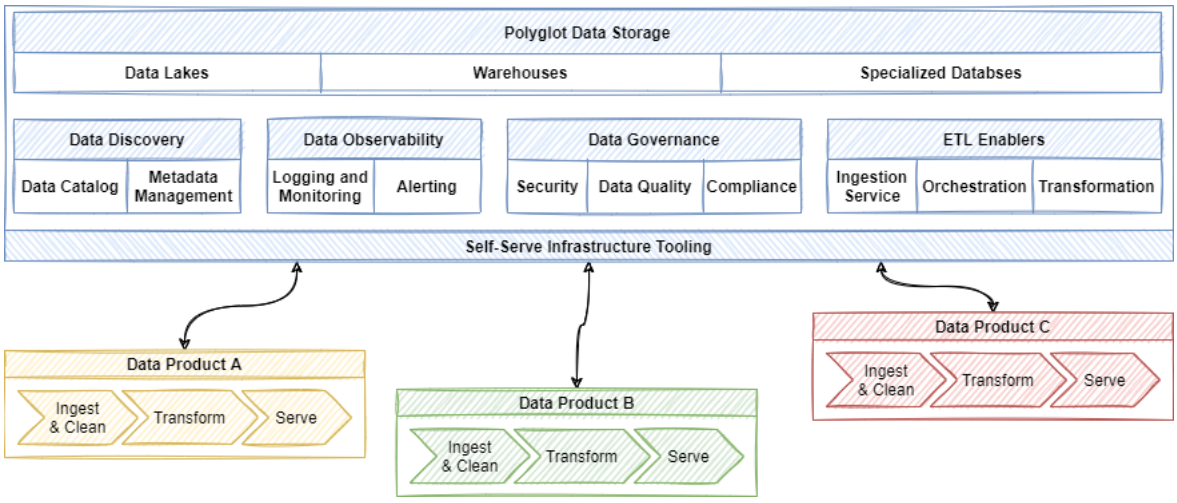
Compared to a traditional centralized data platform, the data mesh architecture provides the following benefits:

- Lower cost of discovering, understanding, and as a result, usage of the analytical data.
- Higher (and most importantly understandable) level of data quality, which leads to higher trust in data and analytical insights.
- Reduced time to market for analytical data products.
- Reduced cost of onboarding for new analytics and engineers.

Technology

Out of the four data mesh pillars, three are not about technology but rather organizational structure and rules, including the team's compositions and responsibilities. So, in this section, we will look deeper into the one pillar which stands out: self-serve data infrastructure as a platform.

This pillar enables data mesh on a technological level, providing the tooling required for data mesh functionality. The self-serve data platform may take different forms, depending on the organization's needs and capabilities. For some generic enterprises, it may look like the following:



Data Discovery

Both metadata management and data catalog form the data discovery capability, which is crucial for any data mesh as a tool that avoids data soiling. If it is not taken care of, it is inevitable in any decentralized solution. Therefore, the data discovery should support two main use cases:

1. Add a new data asset.
The data discovery solution should provide an easy way to register new data products with minimal friction. For example, instead of manually providing the schema, lineage, data samples, etc., teams should automate the gathering of this metadata. This data asset requires the user to provide just connection details and a business description for the data asset, automatically connecting all other information. The addition to this use case is an ability to keep the information about the data product up-to-date, again with minimal manual effort (ex. periodic scanning for updated schemas).t

2. Search for existing data assets.
The solution should have extensive search capabilities, including search by different metadata elements (table and column names, descriptions, etc.), including a fuzzy search. In addition, to make the search more efficient (ex., to narrow the search to a subset of business domains), the solution should provide a way to organize hierarchies, add tags, assign custom properties to the assets, and allow the use of all these attributes in the filtering and navigation.

Data Governance

Data Observability

The domain and infrastructure teams have the same data observability capabilities but different goals. The goal of domain teams is to have the ability to find the answers to the following questions:
Is my data up-to-date?

- Are any of my pipelines broken?
- Are there any data schema changes?
- Are there any failures in upstream dependencies?

At the same time, the infrastructure team's goal is to have the ability to monitor the overall performance and stability of the platform and quickly react to any changes.

The data observability tools provide the following main functions to make both goals possible:

1. **Default logging and monitoring for all the pipelines.** When a team adds a new pipeline to the platform on a self-serve basis, the platform starts collecting and storing the logs and metrics for this pipeline without any additional interventions.
2. **Alerting based on collected logs and metrics** to help quickly react to any dangerous changes in the data or environment.
3. **A searchable interface** to access the collected logs and monitors to allow root cause analysis and debugging of any data issues.

Data Quality

While federated computational governance sets global standards and rules for data quality, integrated data quality framework platforms create the actual implementation of these standards and rules execution.

In addition, the framework provides the ability to select a subset of global data quality metrics for each data product, calculate them regularly, and monitor SLO based on these metrics. The following are examples of these metrics:

- Referential integrity level.
- Deviations in frequency-based records count.
- Allowed range of values for specific columns.
- Custom business rules, eg., "the price of the line item in the order can't be greater than the order's total amount."

The key to successfully adopting the data quality framework by all domain teams is simplicity in the metrics selection process and the ease of new metrics introduction.

Data Security and Compliance

The data security and compliance aspect of data governance is one of the most important sets of functionalities. Since it is naturally domain-agnostic, it is implemented in a centralized manner on the platform level and applied transparently to all domain data products.

The key functions are:

1. Transparent Data Encryption in rest and in motion.
2. Globally managed access control, including row and column level - ex: attribute-based access control.
3. Global compliance and conformance to the regulatory requirements (PII, GDPR, etc.). It partly covers the attribute-based access control (ex: tables or columns marked with a "PII" attribute which are governed by centralized policy) but also includes additional functionality, such as the ability to remove all information for a specific user (the "right to be forgotten").

ETL Enablers

The extract, transform, and load (ETL or ELT) part of the platform can be logically divided into three components (whereas physical implementations may cover more than one component):

1. Ingestion (or, in some cases, integration) services.
This component is responsible for integrating different data providers such as OLTP or OLAP databases, third-party APIs, various file systems, or object storage. The goal is to allow the domain developers to quickly ingest new data sources into the platform and make them available inside the domain data product.

Usually, it includes some predefined set of generalized connection "templates" for different company data storage. The domain developer is responsible for providing implementation details for the concrete data asset to be ingested from the specific data storage.

2. Transformation pipelines.
This component's primary goal is to provide an ability to map and transform data to build a data product. The standard set of capabilities offered is very similar to SQL functionality. Often, the way to define the transformations even looks like SQL - to simplify the tool adoption by engineers. However, the distinctive characteristic of the data transformation component for the analytical data products is the ability to do data transformations in a distributed manner to handle a large amount of data that needs processing.

3. Jobs orchestration component.
Both Ingestion and Transformation pipelines (or jobs) need arrangements:

- Most jobs should be executed periodically (e.g., daily or hourly), so there should be a scheduler.
- There needs to be a way to define dependencies between jobs.
- The domain developer should have an easy way to define a new job (or set of jobs) and monitor existing jobs executions and dependencies; some sort of UI is required.

Polyglot Data Storage

Polyglot Data Storage is a broad term covering all solutions used to store data. Depending on the use cases and organizational needs, it may vary from one OLAP database to a set of different solutions to meet the requirement of each domain data product.

Then we may extract the three generic types of storage solutions below.

Data Lake

Usually based on HDFS (in case of on-premises hosting) or Cloud Object Storage (e.g., ADLS, GCS, S3) and consists of a hierarchically organized set of folders and data files. This works with data implemented in files via the metadata management tool (to discover data location and schema) and the data transformation component (to process and transform the data)

Data Warehouse

Distributed Database, optimized for OLAP workload - batch insertion of the data, queries that scan a significant number of records, out-of-the-box support for data partitioning (sharding), etc.

Specialized Databases

The databases (usually distributed) are optimized for particular workloads, such as:

- time-series databases
- graph databases
- document-oriented databases

Self-Serve Infrastructure Tooling

The Self-Serve Tooling is a component (or rather a layer) whose primary goal is to glue disparate platform components together from one side. From another side, the goal is to provide for the domain product engineers abstraction that hides away the internal platform implementation and simplifies all interactions with the platform. As the infrastructure platforms usually consist of different components for different companies, it is unlikely that any existing tools can cover all the needs for self-serve tooling. So, this component is the one where most teams require customization.

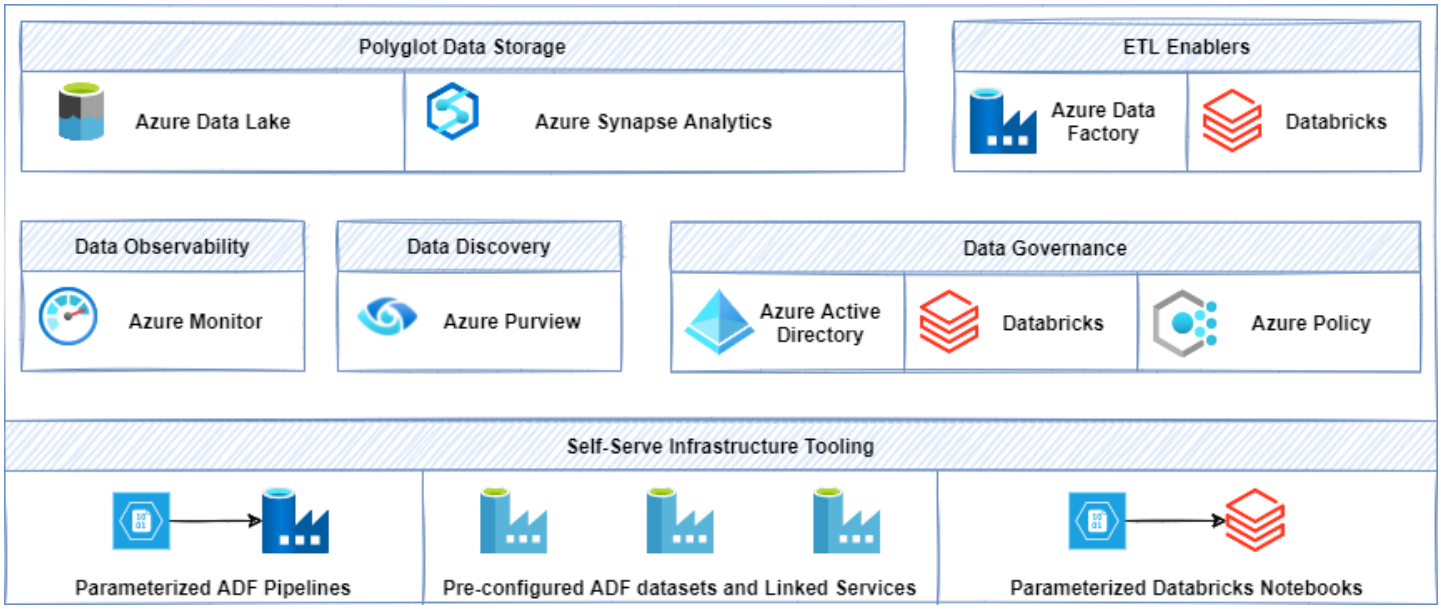
Since it is a front-end for engineers to work with the platform, it should provide at least some of the following self-serve capabilities (ideally, in a declarative manner):

- Adding (on-boarding) new data source (ingestion).
- Adding intermediate data assets and pipelines for their creation.
- Adding new Data Product and registering it in the Data Catalog.
- Editing of already existing pipelines and data assets.

Solutions

There is an endless amount of possible solutions for this process. So we'll consider two high-level primers of self-serve infrastructure as a platform, one implemented on top of Azure Cloud, using the cloud-provided services. And the other is an on-premises solution mainly built with open source tools, with one notable exception.

Example of Cloud Solution



Data Storage and ETL

The core of the solution is as follows:

The data sources are ingested using Azure Data Factory with the help of Datasets and Linked Services (internal ADF abstraction for working with external diverse data services).

The ingested data is landed into the Azure Data Lake Storage - we may say that this is Azure Cloud implementation of a distributed file system designed specifically for Data Lakes.

The landed data is then transformed through the chain of pipelines using Databricks - managed data transformation and analytics service based on Spark data processing engine. The Azure Data Factory is used in this case as a jobs orchestrator.

If there is a need for using Data Marts - the Azure Synapse Analytics is a perfect choice since it's a fully managed cloud data warehouse explicitly designed for OLAP use cases. However, the usage of Synapse Analytics is not mandatory; the Databricks may be used for this use case as well (namely, the Databricks SQL environments).

Data Governance and Discovery

The teams register all data assets in Azure Purview, a metadata management solution with a data catalog at its core. In addition, Azure Purview provides an Apache Atlas API, which may simplify migration from on-prem Hadoop-based data platforms.

There isn't a specific data security solution in Azure, so a combination of more generic tools handles the security and access control:

- Azure Active Directory for managing users, groups, and their accesses based on the RBAC model.
- The Databricks is integrated with AAD to reflect the managed users and groups and create and maintain views, aimed to implement the more granular access control - on row and column levels.
- Azure Policy applies and enforces all centralized data policies defined by the federated governance team.

Similar to the security component, there is no specialized native data observability service in the Azure Cloud. Instead, the more generic Azure Monitor service generally covers this part with a solid set of customizations. For example, one of the customizations requires a data quality framework to continuously assert and monitor ingested and transformed data for quality issues. The teams can implement this framework using Databricks and build the data quality checks into the transformation notebooks.

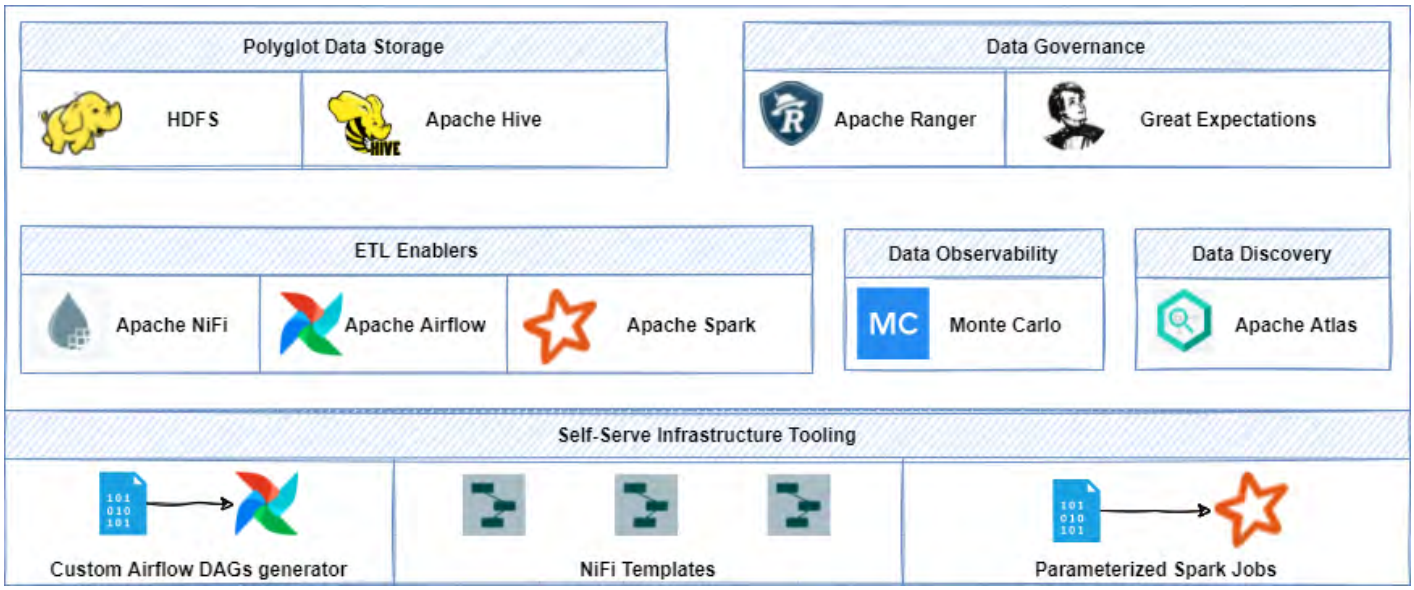
Self-Serve Tooling

To allow the self-serve usage of the platform for each domain team and to minimize the amount of specialized data engineering and data operations knowledge, each team should use the following techniques:

- **Pre-configured ADF Datasets and Linked Services,** which hide the concrete implementations of data sources and provide a user a unified set of abstractions with similar usage rules.
- **Parameterized ADF Pipelines.** Instead of forcing each domain team to create their own pipelines to ingest and transform data, the infrastructure team provides a set of generic pipelines that can parameterize to meet their specific needs (the parameterization occurs in run-time using Triggers functionality).
- **Parameterized Databricks notebook.** Similar to the ADF Pipelines, domain teams parameterize notebooks (Databricks executable components with Scala, Python, or SQL code), giving the ability to execute the same notebook with different contexts against different data.

Even with pre-configured assets, teams can't accomplish all use cases, so the domain team should have some expertise in ADF, Databricks, Spark, etc. But given that there is a pre-implemented and pre-configured framework, enriched with a set of examples and "how-to" guidelines, this data-related expertise level is much lower than usually required to implement, maintain, and use the data platform "from scratch."

Example of On-Premises Solution



Data Storage and ETL

1. Apache NiFi handles the ingestion of data sources. Apache NiFi is a data processing system with a rich set of built-in connectors to different systems. Also, the NiFi has many capabilities to transform and govern the data, so NiFi can transform data (ex: initial cleansing of the data, encryption, throttling, etc.) on this level.

2. According to the predefined hierarchy, Hadoop Distributed File System completes the ingestion of partially transformed data.

3. Apache Spark jobs transform the landed data, coordinated by Apache Airflow orchestrator - both these tools work in conjunction with each other to build a powerful Data Processing Engine.

4. Finally, Apache Hive transforms and structures the data. Apache Hive is the data warehouse software built on top of the distributed file system, allowing working with file-based data using SQL and databases and tables. Intermediate data with fixed structure may also be registered in Hive to qualify analytical use cases against raw data (e.g., discovery, exploratory data analysis, etc.).

Data Governance and Discovery

The Apache Atlas data catalog is the entry point for all data consumers. It's a data discovery solution. It has an extensive set of metadata management and discovery functionality, including predefined types of various types of metadata, the ability to dynamically create classifications, lineage support, UI to search data assets by various attributes, etc.

The Apache Atlas integrates natively with Apache Range - an open-source data security solution, with the following features:

- Access control management to the data assets using the Attribute-Based Access Control model, which combines well with Role-Based AC.
- With the dynamic classification of the data by Atlas, the access control based on data sensitivity also becomes dynamic.
- Centralized audit of user accesses and administrative actions.

The final piece of this component is a Data Observability tool called Monte Carlo, the only non-open-source tool in the example. It allows proactively monitoring data quality issues in a centralized and standardized manner, automatically detecting anomalies in data based on historical trends, etc.

In addition, teams use the Great Expectations library to make assertions and validate the data to provide metrics for this tool.

Self-Serve Tooling

To enable the self-serve usage of the Infrastructure Platform by Domain Teams, use the following set of features:

Pre-configured NiFi Templates

These are used to incorporate the standard flows of data ingestion and transformation into blocks, which can be used in different places by Domain Engineers.

Custom Airflow DAGs (Directed Acyclic Graphs) Generator

This is a custom application that allows generating DAG (the Airflow term for job or pipeline) in run-time based on some configuration file. The infrastructure team creates, maintains, and extends such a generator while the domain teams develop and maintain the pipelines they need in the form of configuration files.

Parameterized Spark Jobs

This is the set of generic Spark applications for standard operations with extensive parametrization. Similar to DAG generators, the Spark applications are created and maintained by an infrastructure team while domain teams write configuration files.

Like the cloud example, there will be specific use cases that pre-configured assets and configurations can't cover. In addition, domain teams require expertise in data engineering. However, implementing the Self-Serve tooling, well-defined documentation, and examples of the platform usage is much easier and more efficient when implementing data products than a centralized data platform team.

References

Data Mesh Principles and Logical Architecture, accessed at https://martinfowler.com/articles/data-mesh-principles.html.

How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh, accessed at https://martinfowler.com/articles/data-monolith-to-mesh.html.

What is a Data Mesh — and How Not to Mesh it Up, accessed at https://www.montecarlodata.com/what-is-a-data-mesh-and-how-not-to-mesh-it-up.

What is a Data Mesh — and How Not to Mesh it Up, accessed at https://medium.com/intuit-engineering/intuits-data-mesh-strategy-778e3edaa017.

The Data Dichotomy: Rethinking the Way We Treat Data and Services, accessed at https://www.confluent.io/blog/data-dichotomy-rethinking-the-way-we-treat-data-and-services.

Azure Data Factory documentation, accessed at https://docs.microsoft.com/en-us/azure/data-factory/.

Azure Databricks documentation, accessed at https://docs.microsoft.com/en-us/azure/databricks/.

Azure Purview documentation, accessed at https://docs.microsoft.com/en-us/azure/purview/.

Azure Active Directory documentation, accessed at https://docs.microsoft.com/en-us/azure/active-directory/.

Azure Policy documentation, accessed at https://docs.microsoft.com/en-us/azure/governance/policy/.

Azure Monitor documentation accessed at https://docs.microsoft.com/en-us/azure/azure-monitor/.
Apache NiFi Features, accessed at https://nifi.apache.org/.

Apache Airflow homepage, accessed at https://airflow.apache.org/.

Apache Spark product page, accessed at https://spark.apache.org/.

Apache Hive page, accessed at https://hive.apache.org/.

Apache Atlas Overview, accessed at https://atlas.apache.org.

Apache Ranger overview, accessed at https://ranger.apache.org/.

Monte Carlo Data homepage, accessed at https://www.montecarlodata.com/.

Great Expectations homepage, accessed at https://greatexpectations.io/.

About the Authors

Viktor Klymenko

I've been working in the IT industry for more than 15 years in different roles - test engineer, software developer, data engineer, and data architect. Passionate about Big Data technologies and applications in various areas - from Cybersecurity analytics to diseases diagnostics.

Volodymyr Trishyn

23 years of experience of software designing, great professional experience of object–oriented designing for Windows, designing of database applications.

Strong skills in .NET, C#. Deep understanding of OOD. Experience with any RDBMS, preferably MS SQL Server.

Now very interesting for me - Microsoft Azure technology. Knowledge of DDD, TDD, SOA, Microservice Architecture. Knowledge of Event Sourcing. CQRS Experience with Windows Azure. Experience of working in a Continuous Deployment setting.

GlobalLogic®

GlobalLogic, a Hitachi Group Company, is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating our strategic design, complex engineering, and vertical industry expertise with Hitachi's Operating Technology and Information Technology capabilities, we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the automotive, communications, financial services, healthcare & life sciences, media and entertainment, manufacturing, semiconductor, and technology industries.



www.globallogic.com