

Data Mesh with Google BigQuery

A data platform is one of the cornerstones of any modern IT architecture, enabling analytics, reporting and machine learning capabilities. One of the challenges when constructing a data platform is the divide between operational and analytical data where operational data today typically lives in our (micro) services, e.g. stored in a PostgreSQL database, while analytical data resides in what we would call our data lake or data platform.

Operational data represents the current state of an object while the same object would be fully historized in our data platform where history is important. The divide between operational and analytical data often creates problems for organizations when enabling their analytical and machine learning capabilities.



Grow your Data Platform

Data proliferation

Traditional services would often be responsible for several key business objects which risked leading to unclear boundaries in our object design. If the database holds both order and customer,

how do we make sure what attributes are related to which entity?

Data silos

The introduction of micro services provided a clearer responsibility regarding our key domain objects. Data in our micro service architecture was often siloed and encapsulated, i.e. while there are methods in our services to create and update the state of a customer, the data is still encapsulated in the system database.

Brittle pipelines

One solution to data silos has been to connect an ETL tool to the application database and simply extract the data on a regular basis, perhaps once every night. Our data teams would then end up with ingested data in a format that would be very difficult to understand. Table names and columns can be hard to understand and data might not be normalized as expected.

These pipelines would also be inherently brittle and often fail; the internal design of a micro service is under the ownership of the team that owns the service and they expect to be autonomous. Any change in database layout risks breaking the pipeline.

As the number of pipelines grows, our data team finds themselves in constant firefighting mode just trying to keep all pipelines running.

Limited resources

The final problem, probably the one most companies most easily recognize, is the time it takes to make improvements or changes in their reports. Since we had developed an architecture of extracting data from different sources and then made this the responsibility of one data team, all change requests would have to go through this one team and their backlog was often very long, often

months.

As the number of pipelines grows, our data team finds themselves in constant firefighting mode just trying to keep all pipelines running.

Data Strategy, not Data Platform

To overcome the challenges described above, we need a new way of thinking about data and data ownership. This is where Data Mesh comes into play as one of the best strategies. Instead of talking about how to design and build a data platform, Data Mesh starts with the fundamental need to have scalability in our organization when it comes to working with data.

Domains & Domain Objects

Having clear boundaries of responsibility and ownership coupled with easy to understand representations of an organization's domain objects is one of the key benefits of introducing a micro service architecture. Working with our business objects at this abstraction level is much easier than having to read and understand database models.

Since our service architecture quickly becomes highly connected between services, we understand that lifecycle management of our domain objects is necessary. This forces us to think about our domain object representation carefully since changing it later is hard and costly and it forces us to have routines for handling it such as field deprecation, expand and contract etc.

In a Data Mesh strategy, this is the correct level of abstraction to start building our data platform.

Ownership and Responsibility

Having established that working with Domain Objects is the correct level of abstraction for our future data platform, it follows that we also need to think about the responsibilities of the teams that own these domain objects.

Traditional application teams are responsible for managing their domain objects and for communicating these changes to other teams that are dependent on them, hopefully remembering the data team. If the domain objects are fast moving and we keep introducing new objects in our business processes, our data team will be swamped with handling these changes and setting up new ingest pipelines of the new domain objects.

This setup of responsibility does not scale in any organization. Instead we should put the responsibility of creating and maintaining a data product for each domain object in the team that governs the lifecycle of each domain object. A data product is a representation of our domain objects and our teams must treat it with the same responsibility as they treat their services including making sure that the data product is updated and represents the latest model. They must also adhere to the same principles of model stability as they do in their APIs.

An Event Driven architecture based on a scalable messaging platform is a great way of creating an ecosystem of services...

Our data team or any other data consumer can now find and use these data products knowing that any specific data product is maintained both when it comes to domain object model and that the data is up to date.

By moving the responsibility of managing our data products to our domain object owners, we create the preconditions for faster data innovation and we increase the quality of our data platform.

Event Driven Architecture

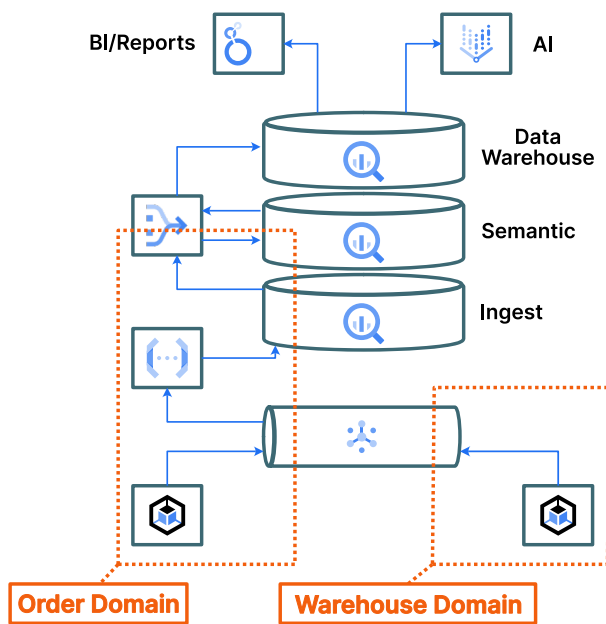
An Event Driven architecture based on a scalable messaging platform is a great way of creating an ecosystem of services that together combine to run our business processes. It gives us the ability to easily adopt our business processes by introducing intermediaries between a

producer and consumer of a service, or to change the business logic of one service without affecting any of the downstream services.

By implementing an Event Driven Architecture and making our Domain Objects part of the payload published, we can easily extend to ingest each instance of our domain objects to our data platform as one side effect in parallel with any other business processes that are triggered by the event.

With clear responsibilities and ownership of our domain objects defined, implementing this data platform ingestion is the responsibility of the same team that owns the domain object and publishes the event. For every change to a domain object, the new state should be published.

This design will have the added benefit of having close to real time data accessible in our data platform.



Data Mesh Architecture

Self Service Data Platform

The final piece to the puzzle of adopting a Data Mesh strategy is a Self Service Data Platform. Typically we would not want other data consumers to read data directly from the ingested state in our data platform. Before we can properly

provide a data product, care must be taken to remove potential duplicates of events, validate entries or perform format conversion.

Our service teams, which are usually specialized in application development, need easily accessible tools and patterns so that they can publish their domain objects as properly maintained data products.

From Strategy to Operational Capability

Once the decision is made to adopt Data Mesh as a strategy, we need to make it part of the daily lives of our domain teams. This includes training to create an understanding of what Data Mesh means in our business context, why we have chosen to adopt it and learning the necessary skills in each team. It also includes selecting the right tools to make it easy and low cost to enable the strategy.

In this article, I will assume that our teams will be responsible for two different sets of their data, ingest and semantic. Ingest represents the raw, unvalidated, version of our domain objects and it should be considered as an append only log. Semantic represents a validated and deduplicated version of each domain object from our ingest layer and this is where the first instance of our data products appear.

Data Platform

The central decision when adopting a Data Mesh strategy is what Data Platform that fits our needs. Given the possibility, going for a cloud alternative such as Google BigQuery is a great option since it provides great scalability and performance. It also helps to keep development costs in your organization down since it works with SQL, something most developers and data analysts are very familiar with.

Google BigQuery provides a rich set of API:s and libraries, making it easily accessible regardless of what technical

stack that comprises our service architecture.

With Dataform, developers can rapidly build complex ETL pipelines using one SQL based tool.

Job Scheduling

With an established Data Platform for storing and processing our data, we need to add a tool to run our jobs on schedule or when dependent data is modified. Google Dataform is an easy to use tool that solves this problem for us.

With Dataform, developers can rapidly build complex ETL pipelines using one SQL based tool. The tool will automatically recognize all dependencies in each project and make sure that downstream dependencies are updated on changes upstream in your data pipeline. It is also very easy to collaborate on code with shared workspaces and source code in Git, making it easy to adopt continuous integration and deployment in your data organization.

Using Dataform, traditional micro services teams can use their existing knowledge of SQL and build ETL pipelines when preparing their data for consumption without having to worry about maintaining any infrastructure. This greatly reduces the cost of adopting a Data Mesh strategy.

Data Ingestion

The final piece to the puzzle is to ingest data to our data platform. One of the most successful strategies for handling this that I have experienced is to have one common pattern for running our business processes as well as ingesting our domain objects.

Let's assume a scenario where a customer places an order. Both the Customer and Order are Domain Objects, i.e. key entities in our business. By adopting a domain based micro service architecture, we would have a clear ownership of both of these entities in two separate services

and possibly two different teams.

As a side effect of an order being placed, we want the order to be sent to our warehouse management system so that it can be delivered to the customer. We can achieve this by publishing the entire Order using a Publish & Subscribe pattern along with relevant metadata about the order such as event type, event timestamp etc.

Google Pub/Sub is a highly available, scalable, fully managed service...

This signal is picked up by the service responsible for integrating with the warehouse management system and the interaction between these two services makes up our business process. The same service would subsequently publish the domain object Warehouse Order to other internal services using the same Publish & Subscribe strategy.

What this enables with little extra cost, is to set up a second consumer of our Order domain object that would ingest the object to our data platform. Google Pub/Sub is a highly available, scalable, fully managed service that provides this capability for us.

Adopting this architecture gives the Customer and Order teams the freedom to choose the technology that best suits their skills for ingesting their domain objects to BigQuery. Google Pub/Sub API:s are publicly available so consuming a domain object message from Pub/Sub and writing to BigQuery can be done from Cloud Functions, from a micro service in GKE or even from a service running in another cloud provider.