Production services are no longer just generating data as a byproduct. Instead, data **is** the product and should be treated as such. This is the argument that data contracts make.

They have been a hot topic recently, with [Chad Sanderson](https://dataproducts.substack.com/p/the-rise-of-data-contracts) of Convoy and [Andrew Jones](https://medium.com/gocardless-tech/improving-data-quality-with-data-contracts-238041e35698) of GoCardless both writing lengthy blog posts cheerleading their usage. But are they actually worth building? In this blog post, we explore what they are, how they can be implemented, and their pros and cons.

### What is a data contract?

Data Contracts are API-like agreements between data producers and data consumers with the goal of exporting high quality data that is resilient to change.

In the data contract paradigm, instead of dumping all of the data generated by production services into data warehouses, service owners are expected to decide which data to expose to consumers, and then to expose it in an agreed-upon, structured fashion, similar to an API endpoint.

One consequence of this is that **responsibility for data quality shifts from the data scientist/analyst to the software engineer**.

### Example

Imagine we have a rideshare application. A bunch of production microservices might be writing into the `rides`, `payments`, `customers`, and `trip request` tables in the database. Over time, these schemas evolve, as the rideshare business runs promos or expands into different markets.

In the current way of doing things, these production tables eventually end up in a data warehouse; subsequently, any machine learning engineer or data engineer that is consuming the analogous tables in the data warehouse has to rewrite their data transformations upon schema changes.

In the data contracts way of doing things, data analysts and scientists don’t consume near-raw tables in data warehouses. Instead, they consume from an API that has already munged the data and produced a human readable event, say, a “trip request”, with the trip request metadata attached (pricing, yes/no surge pricing, promo, payment details, reviews).

### Pros:

1. **Consumers of data don’t have to worry about recreating the business logic that generated it**

The current ELT model, where data is dumped into data warehouses and then transformed in massive joins across different tables, is essentially replicating the business logic of the production services that generated the data in the first place.

Data contracts, on the other hand, are about exposing semantic events that are not tied to the transactional database and should remain compatible as the transaction database evolves. Downstream users no longer need to maintain matching logic and data models.

1. **Since it’s a strongly defined schema, you can document it, version it, and have CI/CD on it**

The schemas aren’t just items on Google docs – they’re usually defined in JSON or Protos or some other type of templating language that can be checked in on Github, code reviewed, and gate-kept with CI/CD. This brings a level of transparency and standardization that was previously impossible to maintain.

1. **Much easier to do root-case analysis when there is a data quality issue**

One complaint about recent data quality efforts that focus on monitoring the data warehouse, is that even if it tells you that there’s a problem in your data, you don’t necessarily know why. While you can certainly monitor the lineage of tables to get a sense of where the problem is coming from (Bigeye provides this as a feature), data contracts mean that data quality issues (in theory) should never have the opportunity to even get downstream. Instead, the only possibility is that the problem occurred somewhere in the production service itself.

### Cons:

1. **Difficult to get buy-in from software engineers**

Since the burden of data quality/data transformation now falls onto software engineers instead of data engineers, implementing data contracts requires a process change. This can be a tricky sell. Even if software engineers are willing, they may be unfamiliar with data modeling.

1. **Difficult to enforce data contract**

While in theory data contract enforcement is a matter of good CI/CD – if it doesn’t pass , it doesn’t merge – in practice many tables within organizations are often not created through proper CI/CD. Instead, many tables originate during prototyping/exploration, and somehow over time, end up referenced by downstream services.

1. **Data consumer needs might change**

While in theory, the data contracts should be designed in a backwards-compatible way, in practice, they probably still need to be occasionally modified. For instance, using the rideshare example from above, the data contract can easily handle changes in the metadata of trip requests – new pricing algorithms, for example, or name displays. But what if the machine learning team suddenly needs information about food orders? That’s a new/different entity which would need a separate data contract established.

### Implementing data contracts

While Sanderson and Jones agreed on the broad strokes of what data contracts mean and why people should use them, they outlined slightly different implementations at their employers.

#### Convoy Case study

1. Come up with enterprise data model
2. Teams that own production services define entities and events using Protobufs
3. Events that occur to these entities are published to Kafka (pub-sub service)
4. Consuming teams consume from Kafka?

#### GoCardless Case Study

* Self-serve data contract process
* Producing team uses Jsonnet to define the schemas for the data they want to make available; they categorize the data and choose their service needs
* Once the Json file is merged into Github, dedicated BigQuery and PubSub resources are automatically deployed and populated with the requested data via a Kubernetes cluster
* Consuming team gets their desired data from their dedicated BigQuery??

As you can see, both GoCardless and Convoy make use of the same basic ingredients in creating:

* Definition of entities and events
* Contract defined with some templating language
* A pub-sub system to handle events

### What’s the difference between data contracts and data SLA’s?

Here at Bigeye, we’ve talked a lot about data SLA’s, and you might be wondering what the difference is between data SLA’s and data contracts.

As a reminder, SLAs are agreements between the producers and consumers of a service that set performance expectations for that service. Data SLA’s are agreements between the producers and consumers of data that set certain metadata expectations for that data, e.g. freshness and accuracy.

**Data contracts complement data SLA’s.** While data SLA’s guarantee meta-properties about the data, data contracts guarantee what the data actually *is*.