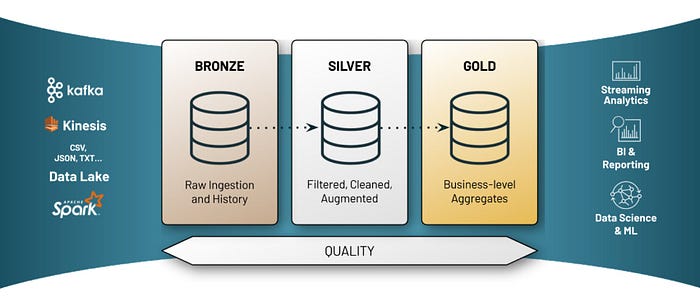
**JFO 28/03/2023  
From** [**https://piethein.medium.com/medallion-architecture-best-practices-for-managing-bronze-silver-and-gold-486de7c90055**](https://piethein.medium.com/medallion-architecture-best-practices-for-managing-bronze-silver-and-gold-486de7c90055)

**Medallion architecture: best practices for managing Bronze, Silver and Gold**

Numerous customers I work with use a [Medallion architecture](https://www.databricks.com/glossary/medallion-architecture) in which they logically organize data in a Lakehouse. As data flows in, they process data through several stages or layers. The most well-known design, as seen below, uses a Bronze, Silver, and Gold layer. Hence, the word “medallion”.



Although the 3-layered design is common and well-known, I have witnessed many discussions on the scope, purpose, and best practices on each of these layers. I also observe that there’s a huge difference between theory and practice. So, let me share my personal reflection on how the layering of your data architecture should be implemented.

**Data platform strategy**

The first and most important consideration for layering your architecture is determining how your data platform is used. A centralized and shared data platform is expected to have quite a different structure than a federated multi-platform structure that is used by many domains. The layering also varies based on whether you align platform(s) with the source-system side or consuming side of your architecture. A source-system aligned platform is usually easier to standardize in terms of layering and structure than a consumer-aligned platform given the more diverse data usage characteristics on the consumption side.

With these considerations in mind, let’s explore each layer after each layer. For each layer, I first provide some abstract and high-level objectives. After that, I’ll make the layering more specific with observations from the field.

**Landing area**

An optional layer that is often seen at organizations implementing a data platform is a landing area or landing zone. A landing area is an intermediate location in which data from various source systems will be stored before moving it into the Bronze layer. This layer is often needed in situations in which it’s hard to extract data from the target source system. For example, when working with external customers or SaaS vendors. In such scenarios, you have a dependency or sometimes receive data in an unpreferred (file) format or structure.

The solution design of a landing zone varies between organizations. Often, it’s a simple Blob storage account. In other cases, the landing zone is part of the data lake services, for example, container, bucket, or specific folder in which the data is ingested. Data within landing zones is often highly diverse. File formats could be CSV, JSON, XML, Parquet, Delta, and so on.

**Bronze layer**

The bronze layer is usually a reservoir that stores data in its natural and original state. It contains unvalidated data (without having to first define schemas). In this layer you either get data using full loads or delta loads. Data that is stored in bronze has usually the following characteristics:

* Maintains the raw state of the data source in the structure “as-is”.
* Data is immutable (read-only).
* Managed using interval partitioned tables, for example, using a YYYYMMDD or datetime folder structure.
* Retains the full (unprocessed) history of each dataset in an efficient storage format, for example, Parquet or Delta.
* For transactional data: Can be appended incrementally and grow over time.
* Provides the ability to recreate any state of a given data system.
* Can be any combination of streaming and batch transactions.
* May include extra metadata, such as schema information, source file names or recording the time data was processed.

The question that I often hear is “What is the best file format? Should I use Delta or Parquet?” Delta is faster, but since data is already versioned or historized using a folder structure, I don’t see any compelling benefits for maintaining a transaction log or apply versioning. Data in Bronze is generally new data or being appended. So, if you would like to go for Parquet, that’s fine. Alternatively, you use Delta in line with all other layers.

Some people say that data in Bronze is useful for business users for querying or ad-hoc analysis. In my experience, while working with customers, I rarely see raw data being used as input for running queries or ad-hoc analysis. Raw data is hard to work with. It requires in-depth understanding of how the source system has been designed. It requires you to work out complex business logic that has been encapsulated with the data itself. It often has many small tables, hence it’s impossible to secure. To conclude: Bronze is a staging layer and input for other layers. It’s mainly accessed by technical accounts.

**Silver layer**

The Silver layer provides a refined structure over data that has been ingested. It represents a validated, enriched version of our data that can be trusted for downstream workloads, both operational and analytical. In addition to that, Silver may have the following characteristics:

* Uses data quality rules for validating and processing data.
* Typically contains only functional data. So, technical data or irrelevant data from Bronze is filtered out.
* Historization is usually applied by merging all data. Data is processed using slowly changing dimensions (SCD), either type 2 or type 4. This means additional columns are added, such as start, end and current columns.
* Data is stored in an efficient storage format; preferably Delta, alternatively Parquet.
* Uses versioning for rolling back processing errors.
* Handles missing data, standardizes clean or empty fields.
* Data is usually enriched with reference and/or master data.
* Data is often cluttered around certain subject areas.
* Data is often still source-system aligned and organized.

For Silver there are a couple of attention points:

Some people say that Silver can act as a temporary storage layer. Thus, older data can be wiped out or storage accounts can be created on the fly. Customers question me, do you see this as well? Well, it depends. If you aren’t planning on using data in the original context for operational reporting or operational analytics, then Silver can be a temporal layer. However, if you plan on retaining history and use this data for operational reporting and analytics, then, I encourage you to make Silver a persistent layer.

Data in Silver is queryable. From a data modeling standpoint, this means that for Silver it’s recommended to follow a more denormalized data model. Why? Because such a design better utilizes the distributed column-based storage that is separated from compute. Does this mean that you shouldn’t use a 3rd-normal form or Data Vault-like data model? Well, you could implement a design that is heavier normalized. On the flip side, I don’t see any compelling arguments. Delta already provides isolation and protection. For example, you can turn on merge schema for handling changes. Besides that, you have history in Silver and also history in your Bronze layer from which you can (re)load data again. Thus, flexibility is already provided. So, then why add increased complexity, slow down performance and add more expensive runtimes to your design? It’s a trade-off you should make. If you would like to learn more on this subject, then I recommend you to watch the [video from Simon Whiteley](https://www.youtube.com/watch?v=RNMoWnSWcTo) in which he provides many considerations.

Another discussion that I sometimes have is on whether you should already join or integrate data between applications and source systems. Here the situation is a bit more nuanced. I recommend, if possible, breaking things apart for better managing and isolating concerns. For facilitating scenarios in which you utilize Silver for operational reporting or operational analytics, this consequently means that it’s recommended to not already combine and integrate data between source systems. If you would do so, then you would create unnecessary coupling points between applications. For example, for another consumer or user that is only interested in data from a single source, there’s also coupling to other systems, because data first is combined in a harmonized layer, and then served out. So, such data consumers are more likely to see potential impact from other systems. If you aim for such an isolated design, then consequently the combination or integration of data between sources moves up one layer.

The same above argument holds true when aligning your lake houses with the source-system side of your architecture. If you plan on building data products and strongly want to align data ownership, then I wouldn’t encourage your engineers to already cross-join data between applications from other domains.

For the enrichments, such as calculations, there are also considerations you should make. If you plan to facilitate operational reporting and for that, enrichments are needed, then I recommend already enriching your data in the Silver layer. This potentially could result in some extra calibration when combining data at a later stage in Gold. Yes, this might be additional work, but it’s worth having the benefits of flexibility.

**Gold layer**

Data in the Gold layer, according to the principles of a Lakehouse architecture, is typically organized in consumption-ready “project-specific” databases. From from this perspective, you could argue that data ownership changes, because data is no longer source-system is aligned. Instead, it has been integrated and combined with other data.

For Gold, depending on your use cases, I recommend a more de-normalized and read-optimized data model with fewer joins. So, a Kimball-style star schema. In addition to that, expect the following characteristics:

* Gold tables represent data that has been transformed for consumption or use cases.
* Data is stored in an efficient storage format, preferably Delta.
* Gold uses versioning for rolling back processing errors.
* Historization is applied only for the set of use cases or consumers. So, Gold can be a selection or aggregation of data that’s found in Silver.
* In Gold you apply complex business rules. So, it uses many post-processing activities, calculations, enrichments, use-case specific optimizations, etc.
* Data is highly governed and well-documented.

Gold is often the most complex layer because its design varies on the scope of your architecture. In the simplest scenario, your Lakehouses are only aligned with the source-system side. If so, data that sits in Gold will represent “data product” data. Thus, data is generic and user-friendly for broad distribution to many other domains. Then, and after distribution, you expect data to land in another platform. This could be a Lakehouse again. If so, you would expect Gold in these platform to match the precise requires of analytical consumers. Thus, data has been modelled. Its shape and structure is highly specific for the use case you’re working on. The upper way of working could lead to principles in which you allow domains to only operate on internal data that is not directly served to other domains. So, there might be tables that are flexible and can be adjusted at any times. And there are other tables with a formal status that are being consumed by other parties.

If your Lakehouse scope is larger and targets both sides, then, extra layers are expected. Some people call these extra layers workspace or presentation layers. In such a design, data in Gold is more generic. It’s integrated and prepared for a set of use cases. Then, within these workspace or presentation layers you see subsets. Such a design is very similar to how you usually do data modeling within data warehousing. Gold behaves like a generic integration layer from which data marts or subsets can be populated.

Some companies also use these workspace or presentation layers for sharing data across to other platforms or teams. So, they make selections and/or prefilter data for specific use cases. Other customers apply tokenization, which means that they replace sensitive values with randomized data strings. Other use additional services for data anonymization. You could argue that this data in a way behaves as “data product” data.

I hope you found this a helpful article. And to all of you building a next generation data platform: enjoy the trip!