Data Engineering Best Practices

Richard J. Schiller, David Larochelle

Published by Packt Publishing

4 Architecture Principles

Thus far we have written about the importance of establishing your organization's mission, vision, best practices, and patterns, and general corporate principles; however, there are specific principles related to the architecture when developing a future-proof data engineering solution that will have to be addressed as we move forward. These principles carry with them the underlying corporate and IT values that will help keep a solution effective in the future. Tools, vendors, and various cloud offerings will morph to embrace new machine intelligence capabilities.

In this chapter, you will be provided with the foundation for developing principles. You will also be provided with a number of principles that are relevant to the task of building a future-proof data factory. Lastly, you will be provided with an abundant number of references for additional reading needed to flesh out a common understanding of challenges that will be presented as you form an architecture that will withstand the test of time.

We begin with an overview of the technology landscape and the way businesses jumpstart their efforts with research firm findings. These non-academic sources are highly compensated for the information they provide, and that information often defines how the herd is moving. We use the term *herd* because those who wish to lead the field of their peers are those who have the qualifications, knowledge, and experience to advance cutting edge technologies without excess cost or outright failure.

Architecture principles overview

The principles to be outlined next will give professionals a guiding light to make wise choices now that define future success. Research firms can be a good source for guidance, such as the following:

- Gartner {**https://packt-debp.link/lXFfHo**}
- Forrester {**https://packt-debp.link/gEVSOD**}
- Bain & Company {**https://packt-debp.link/pAnKim**}
- Frost & Sullivan {**https://packt-debp.link/JrUuR0**}
- International Data Corporation (IDC) {**https://packt-debp.link/D4PrBA**}
- 451 Research (now owned by S&P) {**https://packt-debp.link/KF6rkc**}

- Economist Intelligence Unit (EIU) {https://packt-debp.link/zy1quO}
- Experts Exchange {https://packt-debp.link/GvVItH}
- Omdia (was Ovum) {https://packt-debp.link/ZhkiPT}
- Nucleus Research {https://packt-debp.link/Wsg8SG}
- GigaOM {https://packt-debp.link/q1GLgK}
- Lexis Nexis Research {https://packt-debp.link/30GcOM}
- Olive Tree Insights {https://packt-debp.link/WX88AP}
- Aberdeen Strategy & Research {https://packt-debp.link/NQJFAu}
- Outsell {https://packt-debp.link/mmlLot}
- CEB (now owned by Gartner) {https://packt-debp.link/3mfY5h}

All the aforementioned produce valuable current insights for professionals trying to navigate the rough waters of our current technology journey. Sometimes the future is charted by riding the various *hype cycles* through their ups and downs. But is there a better way? Can the elusive future be made clear and plainly communicated with minimal cost?

Every development journey, including the development of the data factory's data pipelines, begins with preparation and knowing what to pack. The choices are driven by principles. Following the herd will not get you to where you want to go; rather, it will get you only to where the herd is headed. These principles will become the constitution governing your architectural design choices. They will be anchors for conversations and settling disputes that *will* arise. Some of the most successful people, companies, and project efforts are those that have clear guiding directions for what they want and do not let babble (or babblers) get in the way of success.

In order to remain grounded as you make progress, you will want to develop the organization's IT principles and remain securely anchored in them as you develop the architecture and the data engineering designs that are derived from this architecture. Be practical about the principles you develop from what will appear in this chapter. Be careful with the wording of the principles because each carries huge implications for the downstream development effort. Begin with a primer, and do not assume everyone is on the same level of knowledge as you formalize *your* principles. Your constituents, stakeholders, and engineers may have been drinking a vendor's Kool-Aid for years and can't talk about technology if it threatens their limited knowledge. Expect them to be defensive! The following sections were written to provide you with the foundation that will be used to level-set others with the knowledge necessary to gain acceptance for your solution. Many links are provided for you to deep dive into the subject matter. If you remain open to innovation and closed to compromises to quality, fragile design, short-sightedness, and waste, clear principles will arise that withstand the test of time.

Architecture foundation

When developing a modern data-engineered solution, you will want to be level set on the foundational technologies required for success. There will be many themes and disrupting technology cycles to be dealt with. One such technical area is the concept of the **data lake** and the **data mesh**.

Data lake, mesh, and fabric

So, what are some examples of modern data organizations that need to be understood as you roll out a modern data engineering solution? Consider the data lake {**https://packt-debp.link/PoLMQS**} concept, which was coined in 2011 by James Dixon {**https://packt-debp.link/oO3q9r**}. A data lake is a system/repository of data stored in raw format (for example, object blobs/files). So, what has happened since 2011? The data lake became the dumping ground for everything! Unfortunately, over time, data lakes have led to a loss of value. A way of properly understanding data was also lost when the context was not captured in a time series-sensitive metadata store. All this can quickly become a **data swamp**.

What other problems can appear with data lakes? If a data lake holds poorly organized data or too much data without the correct metadata with its required data governance, important data quickly becomes non-discoverable. Information retrieval time, insight discovery, and analytical misdirections (including **machine learning** (**ML**) hallucinations) increased since semantics was put into disassociated algorithms sprinkled like sugar across cloud vendors' **platform-as-a-service** (**PaaS**) services.

New data being added over time loses applicability since its processing rules drift! The data journey is not journaled, and the software and data life cycle pollute the data lake with misinformation that – worse yet – *looks* credible. Data begins to lose its usefulness and starts to incur retention costs with no measurable **return on investment** (**ROI**). Third-party vendors and cloud providers will attempt to fix issues by making the data lake perform faster and be more scalable, with increased caching and data mapping, but can miss the mark on the true need. That is to *provide more relevant data over the long haul*.

Even the proper representation of the *time context* for data can become lost in the data lake, and a stagnant data swamp results; this must be avoided. The data lake is a great idea, but as with all ideas, it needs to be created with thought regarding how to provide important metadata, governance, normalizing transforms, and numerous semantic contexts to data so that the data lake remains useful over time. The need to move beyond the data lake and toward the data mesh or its superset, referred to by Microsoft as the *data fabric blueprint*, becomes evident in the work by Zhamak Dehghani {**https://packt-debp.link/MJdwA6**}.

Note

The data lake is no longer the centerpiece of the overall architecture. Patterns for the data mesh, enhanced **extract, transform, load** *(***ETL***) pipelines, and analytic use cases for dataset class patterns need to be created.*

According to Zhamak Dehghani, the data mesh has four pillars:

- *Domain ownership*
- *Data as a product*
- *Self-serve data platform*
- *Federated computational governance*

It's worth reading through Zhamak's work because it comes at a time when the concept of the data lake has taken hold and organizations have begun experiencing data swamp issues. What the **artificial intelligence** (**AI**) and analytics communities point out is that there is a pressing need to go even further with the data as a product and self-serve concepts and provide commoditized, profitable software and data services to implement federated and domain ownership aspects of the data mesh blueprint. Microsoft calls this the *data fabric* since it adds several cloud-specific services that make the difficult job easier with, of course, the inevitable cloud vendor lock-in that is required to gain those benefits. To sum it up, the data mesh concept is the future but only with a data services and software services framework that makes the implementation something other than a theoretical discussion. Microsoft architects will admit that this is an evolving area. Expect rapid change but remain anchored in the principles.

Data immutability

Data lake datasets should be isolated into zones of clear value: *raw*, *bronze*, *silver*, *gold*, and *consumable zones* have been suggested in an earlier chapter. Cloud technology vendors have not built solutions that allow all these to be placed in a single physical data lake; so, expect your data factory-built data flows to be needed and migration of similar and enriched datasets via lineage-tracked transforms to be created. This means that operational costs will be higher than when the monolithic data lake pattern was envisioned. This will be the case until data fabric offerings mature in the future.

Beginning with the rawest form of any normalized dataset that can be catalogable, one populates the Raw zone. For example, when building out a retail analytics system, it became clear that data being too raw is just not acceptable – it would not be analytically consumable and incur too much cost to reprocess if not first normalized. It is useless until that ingested data is first verified as not being garbage data. For example, you do not want to retain the exact same logical resubmission of a raw dataset due to a technical glitch on the incoming FTP server (aka same data with different timestamps) because it would double the rolled-up data totals. You also do not want retail gray market sales items in a **point-of-sale** (**POS**) end-of-day summary file to be included with qualified sales numbers. Likewise, you do not need to miss a POS summary dataset for a set of stores in a region due to a power outage in that supplying region because the total store summary metrics will be way off. Raw data needs to be raw but not too raw to make it useful.

Data lakes' immutable data is to be available for explorations. Raw data zone data should be locked down when accepted and normalized. It is subject to various anomaly detection profile checks and lineage transforms after arrival. The dataset will produce raw second-order profile data such as its *trends*. This is derived data that is subject to adjustment. Additionally, time series tagging and summarization data should create new data and not adjust the immutable raw data in the raw zone. So, does the raw zone only contain raw data? Let's say it contains data derived from primitive data, as well as the original primitive data itself, with semantic context in a form that can be useful for analytics. The first analytics use cases leveraging raw zone data are data profiling use cases.

Data lakes' immutable data is to be available for analytical usage. In the past, Trifacta {**https://packt-debp.link/UJPS9u**} was a very useful and scalable tool to create raw zone data for downstream uses, particularly for the curation of bronze zone data before entering the data pipelines feeding the silver and gold zones. Whenever the data pipeline results in a transform, the metadata lineage tracking capabilities of the data factory are applied to enable reporting and backward traceability for forensic discovery of errors. Data cataloging of the original data entity, its transform, and its result are all to be retained for analysis. Data is transformed, but immutable sources are retained with this approach.

Data lakes' source domain data will be discoverable. Since data, data context, and data pipeline lineage are all retained in the proposed blueprint architecture, the framework that implements the blueprint will need to have strong capabilities to discover data in the catalog and its recorded lineage. Data catalog reflection and lineage observability patterns are necessary to allow the raw data zone's content to be iterated on until quality metrics are attained. Preserving the quality of the data and its service components is essential as you put data into various zone-to-zone transformations.

Third party tool, cloud platform-as-a-service (PaaS), and framework integrations

Data lake tool selections and the cloud vendor PaaS must not contradict the architecture framework. Enabling data readiness for zone migration requires tool selections for **data quality management** (**DQM**) to be integrated carefully and fully support real-time iterative processing until a set quality metric is attained. The quality metric is a gating function that allows data to be propagated forward in the pipeline. Cloud vendors and third-party DQM providers will not easily integrate at this stage in the solutions designed by data factories, so be prepared for integration headaches. Set up the architecture that works for your organization and never compromise on data quality. Once bad data passes the zone-to-zone transfer gates, it is treated as truth, and that can have deleterious effects on analytics insights in downstream zone-hosted systems. At least if and when these are detected, the lineage and cataloging capabilities will enable datasets to be resubmitted for reprocessing and posting. You will have to decide how many of these stream aggregates form a reasonable release set supporting the organization's data release strategy.

Note

The data mesh pillar: Data as a product sounds good until it also must be released just as software is released, with versions, quality, history, time series, branches, and so on.

The data mesh blueprint requires domain-driven ownership of data. Various business teams comprise the domain, and they will take responsibility for data in their area. Any analytical data will be organized around these same business domains.

Data mesh principles

Data mesh defines domain-driven ownership of data. Just as software and systems are developed in domain-aligned team boundaries, so should the data be organized within the zone. Ownership of the distributed architecture, analytical data, and operational data belongs to the business domain team and not to a central IT team. Given our best practice of hosting data in zones, the domain owner will own a piece of each zone's data as cataloged in the data catalog with data lineage trails that are also owned by the domain team.

Data mesh defines data as a product. The data mesh blueprint also defines data, metadata, and analytical data as a product, and it should be curated and sold with it being a business product in mind. Consumers for the data mesh's datasets are not the programs and people for which the data may have been originally curated. A business domain team provides high-quality data, metadata, analytics, and contracts for service, just as with any other public service API. There is a lot more to be said about data mesh data as a product. But it comes down to the product's value and service for a given cost, which is maintained for a duration of time as a contract. The data has guarantees and warranties and even allows value to be added.

Data mesh data is fit for self-service. In traditional IT systems, the service wrapper for data enables it to meet the characteristics of *ready for self-service*. With the data mesh blueprint, the data can be clearly understood.

Data mesh defines a self-serve data platform. Self-service data and analytics platform thinking can be applied to the design of the data infrastructure. The data engineering platform team provides functionality, tools, and systems to build, execute, and maintain data products across all domains. A core data platform team will provide the patterns each business domain team needs to curate data products. With a data plane infrastructure, the data describing the data catalog, the data pipeline curated lineage, and the value-added analytic/measures/factors/metrics and installable value-added code segments, the **self-service data** concept may be realized. The issue will be what third-party tools exist that can be coerced into the patterns needed to implement this principle aligned with the overall framework.

The data mesh blueprint for domain-driven ownership of data, data as a product, and self-service data drives the need for **federated governance**. With the central governance group setting up practices for governance and the actual governance performed in a federated manner, the overall organizational rules and industry regulations can be adhered to. Compliance with federated governance standards is provided on an honor basis; however, verification via standard reporting is required to maintain data contracts.

Data mesh defines federated computational governance. Integration policies for federated data governance are created in a **guild** manner. Domain owners designate empowered members to the central guild. Any data standards and documentation sets are published and refreshed by the federated governance group (also known as the guild). Specific care is given to data: security, privacy, contracts, service levels, and legal aspects of the data mesh blueprint. Metadata management will include branches of data lineage for a dataset's pipelines, which make it possible to retain parallel branches of a dataset's lineage within and between zones. This aspect of data as a product will make it possible to curate data and its metadata iteratively till ready for release. Federated governance also makes it possible for data rights to be preserved by downstream consumers. All too often, a user changes the column name and resends that data after adjusting a small value. The rights to do this have to be identified in the governance metadata for data at rest – prohibiting such value-added adulteration of the curated dataset. Likewise, data redistribution rights and attribution of data ownership are to be clear in the federated governance policies in the metadata for a dataset.

Data mesh metadata

A data sets metadata shall be treated as first party data, since metadata is data! As such, it is linked to the data and linked to other metadata semantically for any data at rest (any pipeline stage) or its transforms defined, versioned, and grouped into a data lineage trail for any curated dataset. Parallel versions of data at rest metadata or lineage metadata can and do exist in a system. How many prior versions are to be retained is a governance issue associated with each federated domain owner. Value added datasets with at-rest and lineage metadata will be treated as first party data. Metadata is defined for data at rest and data in **pipeline transformation** (also known as **transitioning data**), and it exists for the system implementing the pipeline to give the forensic viewer the operational context of the system operating the transform at the time of the pipeline's operations on the dataset curated in the factory pipeline. It is essential that any value-added data or lineage operations also reflect the data mesh blueprints mechanism enabling value-dated contributions to the factory pipelines. Any class of data will be cataloged in a dictionary. The data mesh's data catalog is a logical data dictionary that enables reflection on all data and metadata of the solution. It is searchable, contains changesets, and can be operationalized for self-service analytics, data quality assessment, and forensic data analytics use cases. It may also be used to age out old data with its metadata. It is not good to retain expired datasets, datasets that have lost their worth, or when the multiplicity of dataset versions is too costly to retain. The data retention policies combined with federated governance capabilities drive data life cycle rules. The rules themselves are represented as metadata and retained with the datasets. The data catalog will also drive **disaster recovery** (**DR**) and archival activities of the data mesh. Since data is a product, you must consider how to stream changes to consumers and group these into intelligent release sets since data does not stand alone but always clumps together. It can be said that data has **gravity**, and the more that there is, the more it produces a **mass effect**, and this results in the creation of **implicit meaning** for datasets in a release set. Releasable datasets that are produced by one or more data pipelines establish the data contracts of the data mesh that are enforceable.

Data semantics in the data mesh

Tim Berners-Lee {**https://packt-debp.link/DBIXta**} raised awareness of the need for a **semantic web**. The follow-on discussions focused on this new awareness that required web documents to be rendered with a formal definition of the page's semantics. This all took place when he invented the concept of the World Wide Web in 1989. When at CERN {**https://packt-debp.link/NgzAKQ**}, he developed the semantic web concept by observing physicists meeting together for scientific discussions and discussing common interest areas. Over the years, the topic has risen over and over in different forms. Data needs to be self-explanatory, unambiguous, and clear.

Tim Berners-Lee

Berners-Lee stressed the decentralized form, allowing anything to link to anything. This form is mathematically a graph or web. It was designed to be global, of course.

In 1990, the HTML standard {**https://packt-debp.link/BB77OC**} was produced, and in 1994, Tim Berners-Lee founded the **Open Data Institute** (**ODI**) {**https://packt-debp.link/kInTo5**} in London. The ultimate ambition of the semantic web, as Berners-Lee sees it, is to *enable computers to better manipulate information on our behalf*. He proposes that in a *semantic web*, the word *semantic* indicates *a web of data that can be processed directly and indirectly by machines* {**https://packt-debp.link/bC8ErQ**}.

The brilliance of this insight must not be lost on those seeking to engineer future-proof systems. You cannot build intelligence about this sum of human intelligence that exists in the World Wide Web of data if it is misunderstood, if it is not linked in order to preserve context, or if it is full of gaps and omissions. A web of data without semantic representation is missing the mark. To date, the vision of the semantic web has yet to be delivered. We are still hindered by large, centralized repositories of information. We are sheltered by the website logic that business folks wrap around data to provide secure data services. To practically minded data and software engineers, the semantic web is a utopian dream.

The semantic web's basis is a new set of standards required to annotate web pages. Structured XML (or RDF) would have no effect on page rendering. These metadata structures would be readable by software to gather page meanings that could be visually, implicitly, and inferentially determined by a human reader. In all pragmatic engineering approaches, you tend to code to the requirement or simplify the solution. You do not build *more* than can satisfy immediate requirements. Machine-readable page semantics may not have been created for web pages, and as such, even if it were available, it would not be leveraged by many external entities. Even if that level of information were available, by publicly exposing it a business could be put at risk. You can see that the development of semantic web-enabled pages could be costly, may not be useful to a consumer, and even poses a risk.

The same argument existed for the development of the World Wide Web, yet it was built anyway. The key was that there is tangle, tacit, and direct value in collaboration. But there was a cost, and you had to *put it out there*! This does not mean you have to give all your data away; it only means that you need to be *smart* about the data.

Smart data is just that – data that can be shared and not subject to abuse just because it can't be contextualized or carry its meaning with it when consumed. It's just right to make sure what is curated in a released dataset has this correct meaning. Metadata is the key, but as pointed out, pragmatic software engineers will not take direction from a data engineer easily, nor will they in turn take direction from a *knowledge engineer*. When the data engineer takes up the mantra of the knowledge engineer, there will be a wider acceptance of the solution to structured metadata that Tim Berners-Lee envisioned.

The recent developments in generative artificial intelligence (AI), rather than the prior technology generations machine learning pattern matching, **deep learning** (**DL**), or predictive capabilities, bring the need for proper data engineering into focus and will drive semantic web concepts forward. Will it be the XML/RDF/OWL 2 {**https://packt-debp.link/MvV5CH**} standards of 2012? Maybe.

The development of operational ontologies as **knowledge bases** has been valuable in solving generative AI's hallucination problem {**https://packt-debp.link/vjX44A**} encountered with OpenAI, ChatGPT, and related generative solutions. We know that machine processing of data into information to gather key insights is a key capability in the future state of the IT industry. The technology will enable so much more for mankind's productivity; however, it is being hindered by current data organization, which was forced by pragmatic software engineering decisions of the past and further hindered by cloud providers solidifying those solutions as easily consumable platform services. This has been good and bad since dead-end architectures are used by those who are not thinking through the ramifications of their organization's architecture choices. Cloud providers are driven by a healthy desire to give the customer what they want today and do not always look out for the customer's future needs. This also produces a huge drag on the development success of the data engineer, because they are mandated to implement IT in ways that just do not make sense for the future. Think about the complexity required for cloud pipeline implementation and the lack of clear observability from end to end with huge gaps in metadata representation offered today.

So, how should we proceed? The architect and data engineer see what is possible but can't possibly build every aspect of the solution to execute on the vision. When building the first production car in America, the Model T, you can read the following quote from Henry Ford:

"I will build a motor car for the great multitude. It will be large enough for the family, but small enough for the individual to run and care for. It will be constructed of the best materials, by the best men to be hired, after the simplest designs that modern engineering can devise. But it will be so low in price that no man making a good salary will be unable to own one – and enjoy with his family the blessing of hours of pleasure in God's great open spaces."

The assembly line was invented by Ransom E. Olds when he rolled out the Oldsmobile Curved Dash in 1901, yet Ford's engineers made many efficient innovations for the Model T. Ford's vision required workers and engineers to assemble in a line and not have to craft each car from scratch.

So, the takeaways from the preceding Ford example focus on the general guidance, which is to follow his pattern of success:

- Build data curation pipelines in an assembly manner, buy the assimilable components, and integrate the whole
- Never let the vision and mission be compromised
- Don't sacrifice data quality!

Remember – Ford first developed Model A through Model S before releasing the Model T. Spend the time to build the support scaffolding, and then be ready to take it all down or reuse it on the next iteration of the pipeline data flow.

Okay, enough of the analogies! What are the data factory assembly components that we *must* have to make data pipelines work correctly in the data mesh? Let's explore these:

- **Knowledge graph** {https://packt-debp.link/M8lcwd} is the ontology conformed to the formal definition of the domain model
- **Knowledge base** {https://packt-debp.link/rOuDFy} is the operationalized ontology defined by the knowledge graph
- **Semantic validation** of ingested data leading to correct incorporation of data into the knowledge base
- **Data lineage** semantics of datasets as they transit data pipelines prior to release and consumption
- **Performant and scalable data store** cloud storage classes with in-memory down to glacial access contracts
- **Data contract validation** capabilities to *observe* data lineage and data quality of the data rather than just the correct software functionality of the solution

From the preceding list, you can take a more in-depth dive into the specific selection process for this core assembled solution components of the data pipeline assembly. What is very important is the focus on quality that is itself ill defined if not standardized and then measured.

So, what does it mean to have quality data? I go back to my applied math days – an item is correct if it is true, always true, and never wrong.

The ability to make sure ingested data is correct is to check it, build assertions, and through a lot of tests, if no errors are found, then it's good enough! Right? The answer is "No!" Data needs to be correct, and that means it needs to be associated with assertions that can be checked by the contact of the data as it is being loaded.

You need not throw tests at data; data needs to throw tests at itself and tell the program it has encountered a semantic error if data is being applied in such a way as to violate its context constraints. Imagine in a retail setting, having a shoe's *item ID* put into an attribute field of a kitchen appliance. It's lunacy!

Today, we often encode meaning within the *item ID* as alphanumeric characters. We abuse the pure intent of it being a numeric *item ID* by overloading it with some encoded semantic meaning. There are hundreds of constraints on the semantics of data (aka knowledge) within a domain, and this needs to be codified in the data's associated metadata. Semantic metadata is defined in the knowledge base to make it possible for data to be fit for purpose and right! Semantic data validation is required to define metadata for data at rest. This metadata should also be put under life cycle management. Semantic validation of knowledge base items is performed using **Shapes Constraint Language** (**SHACL**) {**https://packt-debp.link/2Z9cj3**} (or its competing standard, **Shape Expressions Language 2.1** (**ShEx**) {**https://packt-debp.link/UFNZvJ**}). Leveraging these tools and approaches is not for the faint of heart. It is the lack of clear, easy to use third-party tools in this area that leads software engineers to raise developmental red flags to the data engineer. Some common objections heard are the following:

- This is not taught in undergraduate schools!
- I only know RDBMS and SQL!
- How is this to work or perform in production?
- Bring in the architect – let's hang that individual!

My answer to the faint of heart is: *Suck it up, buttercup* {**https://packt-debp.link/f9CZmY**}! It is the right thing to do, yet making the case and bringing others along will be a difficult journey, and you will need some solid vendor help from sources such as these:

- TopQuadrant {**https://packt-debp.link/xjFYBF**}
- Stardog {**https://packt-debp.link/gxG2os**}
- Ontotext {**https://packt-debp.link/yKIsHK**}
- Cambridge Semantics {**https://packt-debp.link/rToggK**}

The preceding vendor tools may be used to assist in the development of the model, the semantic validation, or the operationalization of the knowledge base vision.

This all has a goal of making data correct, and the one who makes this vision a reality for your data architecture is a winner. If the architecture is right, the subsequent data engineering will be as well. Note that there are alternatives to the use of model-based semantics:

- Neo4j {https://packt-debp.link/IeVzwA}
- JanusGraph {https://packt-debp.link/F2tw8U}
- TigerGraph {https://packt-debp.link/sHzkVP}
- Microsoft Graph {https://packt-debp.link/BPp1Kq}

After incorporating data into the knowledge base, you must make it perform. This is where, even in the consumption zone, data performance needs will outpace the technology available with on-premises tools or cloud solutions. Data within the zone will need to be performant. For this, you will need to replicate the data and keep it in sync when it needs to be kept in parallel physical forms, even though it is logically the same. Some examples of performing in-memory data stores follow:

- Snowflake {https://packt-debp.link/jTIKqT}
- ClickHouse {https://packt-debp.link/vqn4WW}
- StarRocks {https://packt-debp.link/tuRbdO}
- Rockset {https://packt-debp.link/7XgW9h}

And in-memory caching solutions exist, such as the following:

- Redis {https://packt-debp.link/bXS5ds}
- Gridgain {https://packt-debp.link/Y9Babe}
- SingleStore (formerly MemSQL) {https://packt-debp.link/ZwmBt9}

All offer solutions for in-memory data storage. Each may be integrated, but consider not allowing non-persistent third-party databases to take on a master data role unless parallel-hosted persistent storage engines are also leveraged.

It will be clear at this point that there are many choices to be made as the technology field matures and the data mesh capabilities become more commonly understood as essential. For this, you will need a clear logical and physical architecture to be produced and kept up to date at all times. The logical architecture and any engineering design choices reflected in the physical architecture need to focus on the following:

- The data with validation checking
- The factory as various datasets are curated, resulting in various transformations, and data lineage traceability
- The data flows that need to be monitored via *observability* capabilities guaranteed to operate within contracted tolerances

What becomes particularly difficult in the data architecture's definition will be the handling of time series data.

A logical and physical architecture for handling time series data as well as data freshness are business considerations to be addressed upfront in any design. The architecture and capabilities of data stores that effectively support time series data vary. You will need to look at the requirements for scale and performance before selecting one third-party product or another. I recommend you look at the **Time Series Benchmark Suite** (**TSBS**) {https://packt-debp.link/7Una8J} that currently supports many of these time series-capable databases:

- Akumuli {https://packt-debp.link/QtZRVQ}
- Cassandra {https://packt-debp.link/WSUvep}
- ClickHouse {https://packt-debp.link/E4xMkJ}
- CrateDB {https://packt-debp.link/QeMePV}
- Druid {https://packt-debp.link/41QVqY}
- Elasticsearch {https://packt-debp.link/nDiCwJ}
- Graphite {https://packt-debp.link/88E7jW}
- InfluxDB {https://packt-debp.link/ts5TO3}
    - InfluxDB 3.0 {https://packt-debp.link/yMbo8u} (not open source)
- KDB+ {https://packt-debp.link/OtFAtb}
- MongoDB {https://packt-debp.link/LVI2kQ}
- QuestDB {https://packt-debp.link/Dp3UX4}
- SiriDB {https://packt-debp.link/0AJo4H}
- TimescaleDB {https://packt-debp.link/y1H4Nt}
- Timestream {https://packt-debp.link/Yhc5vi}
- Victoria Metrics {https://packt-debp.link/fRzW1s}

Not yet officially supported with a TSBS benchmark are Apache Druid {https://packt-debp.link/41QVqY}, Graphite {https://packt-debp.link/88E7jW}, Elasticsearch {https://packt-debp.link/nDiCwJ}, and InfluxDB 3.0 {https://packt-debp.link/yMbo8u} (not InfluxDB OSS) time series data-enabled databases. Note that the TSBS benchmark may be extended for these or other time series databases. You may wish to take time to evaluate the performance results in the examples published by InfluxDB {https://packt-debp.link/Ekqclu} in its comparisons on the GitHub site to view the outcomes, or you may wish to review the footnotes for these products that contain links to sites with valuable information as you come up to speed on the competitive capabilities of each. You'll also want to look at how these products are best integrated by your chosen cloud provider where security and integration blueprints are available to facilitate solution development.

Remember that this is a hyper-competitive subject area, and you should expect vast changes in capabilities such as scale and performance as the field matures. Each vendor will seek an advantage over the other, and some of these areas will focus on the ease of cloud and analytic tool integration.

Data mesh, security, and tech stack considerations

When getting into the details of data mesh engineering, we need to step back and understand that the data mesh pattern is somewhat vague in regard to providing prescriptive integration directions. Consider security integration, for example. Should you implement a **zero trust** design?

Data security (zero trust) adds complexity to the data engineering design

Zero-trust security designs provide many rewards in regard to software solution simplicity.

Additionally, foundational capabilities such as **test-first design** (**TFD**), data profiling, metadata design, audit, and ML anomaly detection reduce costs and lead to a future-proof solution. When building out the physical architecture, there is a lot of room for innovation. You should study examples of data mesh implementation that exist. There are several different ways to implement a data mesh architecture. Here is a selection of typical tech stacks that we have gathered:

- Amazon Web Services Simple Storage Service (AWS S3) and Athena {**https://packt-debp.link/SWBKqs**}
- Azure Synapse (Azure Fabric) {**https://packt-debp.link/lI0tmc**}
- Databricks {**https://packt-debp.link/v0cmbU**}
- dbt/Snowflake {**https://packt-debp.link/3Ty7F7**}
- Google Cloud BigQuery {**https://packt-debp.link/pwulCh**}
- MinIO and Trino {**https://packt-debp.link/BNxAHg**}
- Starburst Enterprise {**https://packt-debp.link/77sMny**}

Some excellent guidance exists with the aforementioned integrations; however, as you can see, some cloud vendors have yet to formulate unified cloud PaaS offerings to simplify what the MinIO and Trino integration stack and the Starburst recommendations direct you to implement. The engineer must look at the big issues first and then fill in the details. There are many niche problems that can be cause for concern, but you can't become stuck on thorny issues when juggling boulders!

This puts big issues into the crosshairs of our target architecture. You want to identify and resolve these first. The biggest issues that arise are not simple technical ones; but rather compound issues made worse by ineffective and fragile integrations. The following areas will be addressed: security, test first design (TFD), data profiling, metadata design, audit, and machine learning (ML)-based anomaly detection.

Security

Federated domain ownership is a primary principle of the data mesh architecture pattern. How do you enforce security as a contract as you get the identification, authentication, authorization, access, nonrepudiation, and rights management features correct? These need to be easily configured, provisioned, run, and administered in a managed way in order for data and metadata to flow through the data factory solution. You must also consider the effect on time series instantiations where the security context will change over time.

Third-party and cloud tools vendor offerings will conflict with the data mesh pattern's goals, but they are necessary because, on day one, the engineered solution design can't reinvent all aspects of the system. There is a lot of potential for fragile integration. The amount of glue holding together **identity and access management** (**IAM**) for **role-or attribute-based control** (**RBAC**/**ABAC**) compared to group/user level row-level access and entitlement in the data store will be significant. Compromises must be made, and they require cloud provider and third-party security vendor tool integration.

Plan this carefully and completely. Provide for ethical hacking tests, regression testing and lockdown drills with testing scenarios, security response playbooks, release/change procedures, and **incident response** (**IR**)/escalation processes. Think about the creation of the runbook for IT operations of the DevOps/DataOps solution first, and what gets engineered will be robustly handled. Even when there are failures, they will be managed and still look like success to the business owner.

There are numerous data mesh security hurdles to overcome. Federated domain ownership/governance will make the goal of obtaining 100% security effectiveness challenging. Making sure that security policies are uniformly implemented according to their contracts with policies that back up those contracts can be an integration headache. This is especially true when trying to get a data factory's pipelines performant and scalable in the cloud.

It is very important that there is a clear line of sight to the logical security fence around the datasets being protected. Setting the boundaries for data defense by establishing a zero-trust approach that's affordable and scalable in the cloud is a goal. The organization's processes enforcing global policies must be controlled with feedback provided to the **chief information security officer** (**CISO**) and clearly made identifiable in the organization's logical security dashboard.

If you lose trust, it is hard to get it back. This is especially the case in the federated and distributed data mesh architecture. Clear proactive and reactive security testing along with **incident response** (**IR**) scenario results give the security **incident response team** (**IRT**) something to work with when battling issues. There will always be issues to fight even if, on day one, the system is certified as secure, and there will always be new evolving threats to contend with. Trust in your process enables an adaptive response to new threats.

As part of a zero-trust implementation and least-privilege access to datasets, industrial, government and commonly accepted cloud standard blueprints require audit capabilities. Identity and access controls will be audible, and that is a very important aspect of security implementation. Making sure data security is correct for a given user is vital. Security can be maintained by role- or attribute-based security design approaches but must be consistent even when a federated data domain owner thinks they are special and wants to be treated as such.

This is one key reason the core team must govern and enforce the architecture that trumps optimal data engineering designs. Additionally, data rights (to do *something* with the consumed data), fair use (the amount of data that can be accessed in a time period), entitlement (ability to see that data exists versus the data content versus its metadata), and ownership/lineage (who is the domain owner and what value-added enrichment was performed) in the federated governance model add facets to the security architecture of the data mesh. Think about security as a graph problem, and what will come out will be a solution that can be flexible and extensible over time and be a source of consistency in the solutions' architecture.

To provide a certification that a data mesh dataset is correct, of high quality, and of timely value so that it can be considered ready for consumption is a contract. The contract needs to be set and enforced. Part of that contract is the clear line of sight to the dataset's raw origin and the rights that were set when the raw data was acquired.

Often data is purchased with contracted strings attached that define these rights. Even in an enterprise such rights exist and need to be enforced. And what you can enforce you can also have audited. This assures the domain owner and end user that rights are being preserved. You can go so far as to put all this into a blockchain for 100% auditability, but tools to track and validate lineage with metadata are still maturing. The goal is to certify data as if it were a precious jewel that requires expert ratification of genuine value. Once this thought syncs into the reader's mind, you can see that the data that the data factory curates is to be treated with a level of correctness and auditable genuineness so that data trust is always preserved.

It is recommended that the architect and data engineer look at the tools to be integrated, the conflicting security models of the cloud and third-party vendors and the organization's priorities, and create a unified security architecture for your data mesh.

Note

There is no silver bullet solution today that can be purchased and plugged in – it is a custom solution with trusted data at the center. Do not be oversold into thinking a difficult integration can be handled by the CISO – it is an architecture issue and implication required for the data mesh, and it is fundamentally a shared solution.

Test first design (TFD)

We develop software in an agile manner, but we do not yet release data in an agile manner. This is a common observation. When did you last check a dataset into GitHub and not have it bounced right back out? Even when zipping it up, you just can't do this! You need to think of data as being versioned in the data mesh, yet there are few tools to add that kind of versioning control to curated datasets. It is essential that there be a change set mentality and a new form of release processes for data in the mesh.

When you have implemented metadata quality checking, made datasets self-describing, and applied formal data quality checks throughout the various stages of the developed data pipeline, you are thinking about engineering a robust system with contract validation in mind. You do this for code released into the system via CI/CD pipelines, but what about the data tests so that the data may be released with quality?

At various levels of data quality testing, we need to leave enough of a trace to forensically fix anomalies when they occur. We will need to retain that trace with data lineage metadata. We need to do this for the data and not just for the software code affecting the state of the data.

You will also need to shift the data testing efforts as far left as possible in the development process, with the CI/CD quality checking steps aligned with the process. The effectiveness of the data pipeline execution needs to be verified to remain effective. A key goal is to not allow bad data to propagate. Knowing what is bad is a subjective measure transformed into an objective measure with a chain of quality checks with auditable results.

Data profiling

Garbage data in leads to garbage data out.

That has never changed. You need to assess the data's profile before it is used and considered valuable.

What does it mean to *profile* data? The answers lie in the need for data to be semantically correct (has data arrived and is it of the right class?). We will want to know if it is complete (has a partial receipt been received due to an error?) in that enough data arrived to process it as a cohesive dataset. We will also want to know if data is overstated (has a file arrived twice because of a transmission glitch?). Does the arriving data break the expected trend (it could be that arriving data does have increased volume but is not 100x the norm; maybe somebody had a really sticky keyboard)? Do the data's metrics and measures (what we've always referred to as **second-order data**) break trends or pass the test of being credible?

You will want to gauge data anomalies carefully by creating analytic probes, made operational as data profiling steps running within the data pipeline. Data quality is not always myopically observable after a single data pipeline processing step is complete, but only after being rolled up into a metric or having a calculated measure applied will anomalies be discoverable. Additionally, machine learning techniques for pattern identification, matching, and common feature clustering (followed by classification) allow anomaly thresholds to be exceeded and issues identified that are not obvious.

For example, in a prior engaged retail analytic setting, these machine learning techniques identified the occurrence of gray market retail items in retail store sales volumes that should not have been reported as true sales. This was detected by seeing a pattern of sudden rise in sales in a store within a particular region. The ability to detect anomalies and provide enough signals to enable you to forensically identify the root cause of anomalies is essential to preserving the credibility of any derived consumer-driven insight.

Does the **third-order data** (the *insight*) generated hold true during backtesting? You will want to perform backtesting of any insight using past data as part of data quality testing. This allows the truth of the insight to have been discovered in the past if such a test existed when prior data was received. Backtesting an insight's hypothesis adds credibility to the insight and solidifies it as a truth!

Data profiles need to be created and all data testing automated to define the trustworthiness of a dataset. Software development frameworks do not satisfy the need alone. Data profiling frameworks do! So, a data profiling framework is needed in the solution to support answers to the questions posed in this section. What frameworks exist for data profiling? You should look at Alteryx's Trifacta {https://packt-debp.link/UJPS9u}, Datameer {https://packt-debp.link/cMTXQS}, DataFlux {https://packt-debp.link/jUXuaI} (but you will most probably be working with SAS since it is a subset of SAS now), Hightouch {https://packt-debp.link/tA0Zm5}, and AWS Glue {https://packt-debp.link/mRLiTl}.

Data profiling is a goal. The tooling choices needed to support that goal may be part of the solution. Even if homegrown or chosen from over-the-counter solutions and then integrated, data profiling is essential to gauge the shape of the data to have it semantically fit the need and be of high enough quality for consumption.

Metadata design

Metadata patterns and solutions can be endlessly discussed and are often implemented badly. Refer to prior sections where various vendors and offerings are mentioned. See how cloud provider solutions are beginning to focus on the need for structural metadata to explain the chain of Cloud PaaS dependencies of the customer-implemented data services their pipelines require (for example, Microsoft Azure Purview {**https://packt-debp.link/OET4Vj**}).

Today, in many data processing systems, you can't easily trace the data journey from end to end. You can't afford **master data management** (**MDM**) systems of the past, and they don't fit the cloud model anyway. The risk of a full-on data quality meltdown exists at many levels. This is prevented with the clarity that a solid metadata blueprint provides. Structural metadata (cloud data transform operations with the data organization definition), metadata for data at rest (semantics, shapes, and contexts including time), and data lineage metadata (enrichment and transformations) are all required in the data mesh metadata solution.

Audit

We've mentored this thought to many teams in the past and often say the following:

It is not good enough to get the job done; it's essential that any individual following you do IT as well if not better than you!

Going from *hero to zero* is not just a cliché, but rather a very distinct possibility if you do not build that mountain of support!

Build a mountain of support for your success, and do not climb the flagpole with your banner to obtain your career heights!

We often begin prep talks as follows:

It is not good enough to just get it working; IT must work, IT always has to work, and IT can never fail! If you just get IT working, you've done one-third of your job!

So, what does this have to do with the topic of auditing and the data mesh pattern? The answer is, please leave breadcrumbs! It is essential that there be an audit trail containing the essence of *what your software does* and the effect on the data pipeline's released output after each stage of the data pipeline journey.

What the pipeline code does to the dataset's state has to make sense when looking backward at the audit trail. This data journey audit trail leads to its hookup into the data mesh (or fabric if in the cloud), which then needs to be discoverable. What your IT solution does to the state of your data is what the customer will see, use, and enrich. Change is a stream, but quality is discrete and assigned at the stage boundaries. Even if the stages are small, changes from one release state to the next must be explainable by those functioning as quality assurance team members. This will be needed when the analytics user wants to use that data, which leads to impactful insight.

Imagine presenting to a company's board a revenue forecast that is based on erroneous summary retail data! The analyst will often be the critical downstream user to be serviced;

any change better be explainable, to everyone! With data being self-service, this is even more an issue for obvious reasons.

On another note, debug levels in your software quality analysis are analogous to the data trace levels of your data quality audit reports. Think about **data forensic** use cases when building data quality and assessment tools to clarify the features needed for the designed audit trail. In this manner, you only collect good trace and not piles of useless clutter. Sifting through it for insights can be fruitless.

Tools such as InsightFinder {https://packt-debp.link/AvNA70} exist (for assessing IT operations patterns), but ML algorithm-driven tools can miss important and necessary details. Changing data pipeline code after the fact to collect more audit trace and waiting for the off chance that a previously observed anomaly reoccurs, and doing this literally through spiral iterations, leads to many lost cycles, frustration, and bitterness directed toward IT developers. It is proof that an architect did not get involved early enough to stop the erroneous thinking in advance.

The cost of building a safety net for trace assessment may be high, but without this safety assurance environment, the ability to add new quality checks becomes difficult. If insights gleaned from trace assessment are missing, the feedback loop to improving quality checking is vastly slowed down. Here, we leave you with one of our favorite thoughts; we call it the first principle of paranoid programming:

What can go wrong will go wrong, and it's going to happen to me! So, I better just be ready to handle IT!

Turning a difficult software or data quality issue into a series of opportunities makes it possible for you to build a ratchet toward success rather than a house of cards doomed to many failures.

ML-based anomaly detection

The human capital cost of curating a complex or vast dataset can be expensive. Leveraging ML minimizes this manual effort.

Using ML-based anomaly detection reduces quality costs and leads to a future-proof solution!

Any quality application of ML or DL approaches to address data quality anomaly detection needs to be of high efficacy. Obtaining trusted repeatable results is the key. When data science meets data engineering, you want a repeatable quality assessment framework for ML results, with real-time observability approaches to be integrated to know the following:

- When any retraining is required
- When drift is occurring
- When precision/recall within a category (a logical segment of data) goes awry

There are many advanced ML quality metrics {https://packt-debp.link/F1a3Id}, and many are applicable to a tuned-up data engineering and data science team. Listed next are a few of these metrics:

- Area Under Curve

- Classification Accuracy
- Concordant Ratio and Discordant Ratio
- Confusion Matrix
- Cross Validation
- F1 Score
- Gain Chart and Lift Chart
- Gini Coefficient
- Kolomogorov Smirnov Chart
- Logarithmic Loss
- Mean Absolute Error
- Mean Squared Error
- R-Squared/Adjusted R-Squared
- **Root Mean Squared Error** (**RMSE**)
- Root Mean Squared Logarithmic Error

In the past, when building data engineering processes for data scientists, it was like trying to tell an artist that their art is not logical. It is where art meets science and where there is much contention. Often, this is due to mismatches between the language of the following:

- Statistics, computational linguistics, and mathematics
- Computer, software, and data engineering

Math is the common language of science and engineering, so bring it all down to the numbers, the proofs, and the algorithms.

You will want to build a workbench of processes so that what works in an inherently fuzzy processing area works well over time and can be understood by all mindsets. Any stochastic approach can go off track if assumptions are not codified, built as contracts, and tested to exist before applying a model. Once the contract is verified, it needs to be monitored for operational compliance to make sure drift does not occur, and retraining and then re-contracting take place within the ML workbench.

With hundreds of parameterized ML models, this can become an area for systemic meltdown without a framework. With the addition of any new dataset or new instance data aligned with an existing approved dataset, this meltdown can happen. No AI solution can remain effective without an engineering discipline to minimize risk to the data domain owner.

What are the key foundational takeaways?

In this overview section, a basic level set was provided leading to the generation of principles to be itemized in the next section. Each area has key takeaways that need to be kept in mind as the principles are elaborated upon so that they can be made part of your organization's data engineering approach:

- **Data lake, mesh, and fabric**: Know the architecture differences between data lake, data mesh, and data fabric and think of data as a curated factory product subject to all the life cycle needs of a product.
- **Data immutability**: Establish data zones such as Raw, Bronze, Silver, Gold, and Consumable to separate classes of data as being fit for purpose within each zone. Understand that the justification for this approach is a pragmatic one, given current technology limitations and vendor offerings that often are at odds with your integration goals.
- **Third-party tool, cloud PaaS, and framework integrations**: Focus on DQM in your data factory architecture and data engineering designs. Data as a product needs to trump niche third-party vendor offerings that may be difficult to integrate or force the necessary architecture principles to be violated.
- **Data mesh principles**: Grok {https://packt-debp.link/Y84Gs8} the data mesh pattern fully:
    - Domain-driven ownership of data
    - Data as a product
    - Self-serve data platform
    - Federated computational governance

Build out the architecture knowing the conceptual, logical, and physical architecture implications for the data engineering effort.

- **Data mesh metadata**: Metadata is a key enabler of the data mesh pattern. It is formed in three classes:
    - Structural data
    - Data at rest (semantics)
    - Data in transit (lineage)

Metadata needs to be discoverable via a data catalog and operationalized to be effective. It also must overcome and bridge cloud and third-party vendor gaps/inconsistencies in this key capability.

- **Data semantics in the data mesh**: Metadata explains data at rest semantically but the *how* is subject to debate. The need for a semantic web was raised by Tim Berners-Lee long ago, but that vision was not realized. The data mesh needs data semantics to enable quality and federated domain curation of data through the mesh. The data engineer's platform must address the need for comprehensive metadata capabilities.

You need to weigh the effort to create a formal domain model creation (in OWL 2/RDF, for instance) as proposed by Stardog, Ontotext, and Cambridge Semantics in their approach versus the use of **labeled property graphs** (**LPGs**) proposed by Neo4j, JanusGraph, TigerGraph, and Microsoft Graph. This will be required as part of the logical and physical architecture definition of any modern data platform.

- **Data mesh, security, and tech stack considerations**: Security is the first capability of the dataset contract for service that is needed to preserve the quality and integrity of any curated dataset. It must be well architected for proper and effective fit and service of the data platform. It must not become the Rube Goldberg {**https://packt-debp.link/pWgl9k**} implementation task of the data engineer:
    - **Test First Design** drives the CI/CD pipelines need to be applied to data quality and not just software quality.
    - **Data profiling** is needed to transform data after it is validated, normalized, and proven to be of high enough quality to pass pipeline gating checks before propagating to the next stage of the factory.
    - **Metadata design** is key to the data mesh's implementation of the federate and domain ownership principles.
    - **Audit capabilities** are to be built for compliance, and forensic and governance needs to preserve the contract and validate the level of trust you can give to a curated data set.
    - **ML-based anomaly detection** reduces the cost of maintaining the data quality contract, which leads to a future-proof solution.

In this overview section, you were given a walk-through of our thinking regarding what is needed before generating your principles. The key takeaways will be in line with your vision as the principles are elaborated upon in the next section.

Architecture principles in depth

The following principles will help guide your decisions when architecting and engineering solutions are built into a modern data platform.

Principle #1 – Data lake as a centerpiece? No, implement the data journey!

This may sound shocking and really an anti-follow-the-herd mentality, but it is true! Thinking that the data lake was envisioned as a source for all data that can be miraculously understood and repurposed over time leading to great insights is naïve. It can become a data swamp and a costly liability without semantics, context, time series structures, and a clear metadata pattern with governance principles aligned with the data mesh and operational data fabric capabilities.

Data needs to be curated in the factory from raw form to consumable form, and it needs structure and life cycle along its assembled journey through various zones (such as a number of logical data lakes) until ready for consumption. Data needs to be released like a product. You cannot just wave a magic wand over the data to get it to this state of readiness for the consumer. That data journey is worth noting, logging, preserving, contracting, securing, controlling with entitlements, and assigning value-added rights for long term sustainability.

Principle #2 – A data lake's immutable data is to remain explorable

A logical dataset should first undergo its contracted life cycle's curation process steps (as defined as assembly instructions) along with quality metrics within a zone. Logical data is then ready for propagation to the next stage in the data journey or made ready for placement into the downstream zone. Datasets will remain explorable and cataloged at the zone boundaries since that is where contracts are enforced. Datasets will need to have lineage recorded along the journey and then have the lineage condensed (or rolled up) once the dataset is readied for release. It is important that data catalogs have the capability to separate temporary lineage trails from final trails that must be retained with the released dataset. Many tools today do not have these parallel metadata trails available nor a way to roll up the trails to the release set. This is even more complex in a cloud environment where many platform as a service (PaaS) offerings are utilized along the data journey. Cloud metadata services provide tracking services (such as Azure Purview), but these need to be augmented to correctly implement this principle. *Once metadata for the data journey is available, the data should be made explorable*. Without the data and its metadata being linked together and made available for search systems, you cannot effectively identify a dataset's worth, its context, or fitness for use when being leveraged by a data engineer or consumer. Datasets and their metadata and search indices will need to be locked down and readiness state be a primary search facet when looking for data across the entire data factory platform-enabled system. It is far too easy to lose track of a widget on a physical factory floor, and this is true of a dataset element in whatever state it may be, across the entirety of the data factory.

Principle #3 – A data lake's immutable data remains available for analytics

After all, code is code, and it has bugs that need to be addressed in time.

Errors will exist in data, metadata, second-order and third-order data/metadata, and various quality processing steps. Data states will be affected every time a code snippet is adjusted, and with that, the history of the change and its effect on prior processed data is subject to auditing and downstream explanation. Analytics snippets are applied throughout the data factory's data pipelines. These are part of the shift-left testing and TFD methods that must be core to the software design approaches applied throughout the **software development life cycle** (**SDLC**). The analytics steps will need to be self-contained and cataloged. Also, the analytics output needs to be clearly cataloged as metadata for primary, second-order, and third-order data of the pipeline.

Data quality analytics, trend assessments, and consumer analytics will indicate that there may be errors or deviations to the contracted dataset's state. What do you do with corrective algorithms? In a prior system, corrective algorithms piled up change on change for a curated dataset so high that the truth of the original dataset was lost. How data corrective steps are applied must not change the immutable source but be applied as data lenses on the source data so that if they were removed – the original could still be leveraged. If necessary, all value added lenses should be rebuilt when gross errors are encountered.

Some data lenses are temporary in that they will apply only for a limited release or duration of time to correct a gross gap, understatement, or overstatement of a dataset's raw availability. These also need to be cataloged and auto-removed when no longer pertinent (such as for the next data release).

Data analytics is not an art but a science and when engineered into a data factory will result in the need to explain the state or quality of curated datasets. This will require change when the explanation is: *That's an error!* When errors arise, new data corrective steps are added that need cataloging and life cycle management as data lenses rather than changes to the immutable raw or curated data.

Principle #4 – A data lake's sources are discoverable

A dataset's source will need to be clear when leveraging the data catalog's search capabilities with the data lineage trail, and all steps, stages, and corrections evident. This gets very complex and can look like a tree's roots seen from a bird's eye view. There are lots of branches, merges, algorithm processing nodes, and ultimately, way down deep, you will find the row and column of the source. When tracking data lineage metadata, it is important that the journey be backward discoverable. This is not easy since a processing step may have implemented a nested update such as *Change a column on a table using a transform after a join that first merged a lookup table with a conditional*. What caused a particular row/column change was just lost. Many metadata tracking systems just report the query and leave it to the forensic analysts to figure out if the transform was effective. Worse yet, when the update join operation was performed, it may be that the lookup table itself was being updated in another parallel processing step at just the time the join was being made.

This really happened in a prior experience, and the audit trace looked like a perfectly executed operation when it was really producing a gross error since the joined lookup table was being updated at that exact moment. You could have fixed this with a global system table lock but that would have slowed the entire factory down. The update was being performed in an optimized service that was out of the development team's control. A snapshot of the table followed by a quality check was implemented to fix the problem, and then the snapshot persisted for backward traceability once the pipeline was completed.

It is important that you do not forget the need to assess a data source's original source when looking backward from the end state. Being able to get past a data pipeline processing *black-box stage* by making sure they have a *gray box* level of inspection trace will enable the state of the original data source to be discoverable.

Principle #5 – A data lake's tooling should be consistent with the architecture

Many third-party tools and cloud services are over-marketed to data engineering management. These tools and services are often only 80% production ready, even though they have been launched as generally available. They come with the need for excessive training since they exhibit peculiar error-handling characteristics that will appear only at the edge of performance, scale, or functional use! A **proof of concept** (**POC**) may never expose these characteristics. Only a data engineer with a bulletproof quality standard with clear methodology will be able to smoke out these bugs (and they are *not* features) yet a developer must code around the handling of these anomalies. Sometimes the workarounds do not warrant the use of the tool, and this can come to light too late in the project and force re-solutioning of the pipeline. This is not refactoring. It's a disaster of architecture and design caused by believing in a vendor's marketware rather than proven software.

The need to assess tools and services correctly with an objective due diligence process, with a clear demonstrable winner with the optimum fit for the architecture, must precede the POC to vet vendor fit. Once deemed fit, a POC should go forward to vet all integration needs, and then finally the iterative agile development of the solution.

All too often, the industry herd moves to the next shiny thing that is offered by a start-up vendor. The result is a fragile mess of a solution that exposes the business to risk. The architecture is implemented based on principles and required capabilities. It stands above the niche vendor's capabilities. It should be used to guide and align any selected vendor capabilities and reject any misaligned capabilities and marketware.

Principle #6 – A data mesh defines data to be governed by domain-driven ownership

We are suggesting that datasets in the data mesh be curated by data pipeline processes with contracts set by domain owners. This will mean that datasets stand alone and do not cross domain boundaries. This is not always the case since data can be adjusted with downstream value-added additions. Ownership becomes shared between the primary domain owner and the value-added owner. The contracts for subsequent downstream consumption are then a mixture between the primary owner and a chain of value-added owners.

With ownership comes rights for entitlements, fair use, and commoditization with or without redistribution rights for the consumable dataset. You can even go so far as to put all contracts and lineage into a blockchain, but that capability has yet to mature into easily integrateable third-party offerings. However, it is still required to maintain contracts and make them discoverable in a data catalog along with the metadata for any data at rest with lineage as well as all value-added adjustments.

The domain owner's data change life cycle and potential for contract default exists, and as such, trust can be eroded over time. Who has not used data thinking it would be refreshed yearly only to find out that it was only partially refreshed due to the business failure of the domain owner? Departments in a large organization come and go, ownership changes, funding stops, or management is shaken up. The data consumer may never be notified of contract breaches before they impact production. It is required that contracts be subject to compliance tests and predictive alerting enabled before failure arises. This way, the consumer has time to react. If data is domain owned and a product, it must be treated as such.

The total cost for the domain owner must consider the intangible revenue and costs to the contracted consumers. The proper data-engineered solution must provide that dashboard to the financial accountants, who often do not know that the domain owner's business data obligations impact consumers. The changes in a domain owner's funding or viability have ripple effects on the business. Centralizing domain ownership into a core IT group is often the solution for orphaned domain owners' data products. Over time, the data contract changes since a steward has replaced the domain ownership. Consumers need to see this in the changed contract when the architecture refreshes yearly and tech debt is re-assessed. Additional data debt remediation costs are incurred implicitly, which need to be added to the strategic plan's rolling 5- (or 10-) year total cost estimate as the future state horizon moves out in time.

There are many practical data engineering effects on domain ownership of data in the data mesh. These need to be discussed and planned for since what is a great idea on day one must meet the practicalities of the long-term run/managed solution. That solution is developed in the context of a fast paced dynamic business environment.

Principle #7 – A data mesh defines the data and derives insights as a product

A dataset (with cataloged metadata), its derived data, its quality analytics process steps, its transforms, its corrections/adjustments, and various versions as it appears in the data pipeline's zones are considered products in the data mesh. As a product, a dataset comes with all the implications of it being a product as defined by Zhamak Dehghani in her work on the data mesh concept while at Starburst {**https://packt-debp.link/77sMny**}. What you also have to know is that any directly observed insights as well as implicit insights/inferred insights are *also* products. Imagine acting on a dataset's insight today only to have it change tomorrow or next month because of the dataset's restatement, a software bug correction fixed in reprocessing, or a trend break that was standard for years. It could spell financial disaster for a consumer. Heads will roll if the impact is not assessed and the change provisions of the data contract are not communicated.

We can't begin to tell you how much time was spent in the past explaining data changes and why a data change was made going forward or, worse yet, in the past. Products can be recalled and so can data. You must be ready for reality. Data and insights do not just get profitably produced but can be recalled and restated. You must handle the ugly parts of treating data as a product and not just build anticipation for the revenue that the thought evokes. There are real costs to maintaining data contracts, and some of these are practical corrective costs for restatements, parallel version maintenance, differential sets (for **change data capture** (**CDC**) log replays), communications of data inventories in stock, quality metrics, and change logs (dataset inventories that could even be snapshots of the data catalog at the time of distribution/consumption).

We caution the reader to look at all aspects of data being a product and not skimp on any capability that puts the data contract at risk or the dataset's quality into an unknown state.

Principle #8 – A data mesh defines data, information, and insights to be self-service

We have observed that data being self-service is a great concept but rarely do analysts want to spend an enormous time curating it from its raw state. Consider the bricklayer working with clay and sand and cement rather than bricks and mortar when building a wall. It would be insane not to build with some degree of prebuilt material. Items must fit together seamlessly rather than having to be manufactured on demand at the time of assembly to become an insight. Data, therefore, must have edges or facets that align with other data. You can't have a year be defined as a calendar year when the data is aligned with hundreds of different company corporate years. Retail calendars have to align for weekly, quarterly, and yearly data to be analyzed comparably. Datasets subject to be used as self-service need to be faceted, subject to contracts, and fully discoverable in the data catalog with lineage explaining contracts. Only trusted clear data can be put out as self-service. Any difficult analysis that requires internal data facets to be exposed must be wrapped as second- and third-order data that then becomes subject to self-service.

Once any complex data is too hard to explain, it can't be made available self-service! So, to solve this, you can provide open services and analytics to put into code and scripts to interpret complex data. These analytic wrappers and pre-canned services help solve the complexity problem while leading to data maintenance issues. Analytic code changes in Microsoft Power BI or a big data notebook can be made incorrectly, leading to unchecked data abuse. Financial information providers are acutely aware of data abuse issues in analytics. The financial numbers should tell a story but even a small gap in a time series can lead to analytics errors.

For self-service data mesh goals to be truly effective, self-service data must be correct, complete, timely, high quality, and at all times aligned with the data contract. But what if the data has gaps or is missing key fields? Can it be augmented with synthetic data, and if so, how can that be good regarding maintaining the data contract?

It all depends on how the data contract is written. If data must be 100% complete and the raw data is not, does this practical and real lack of some data points hold up the assembly of the final product? The answer is, “No!” Can the contract enable the factory pipeline to produce data points in a dataset that are fiction just to maintain a trend or fill a gap or subject points that are redundant? The answer is, “Yes!” The data’s use case is the important goal. The purity of the dataset as a whole has to meet the terms of the data contract to apply to the entire dataset, not just individual data points in the set.

Principle #9 – A data mesh implements a federated governance processing system

Federated governance has an implication. There is a central governance function, and some governance can be delegated to others. It is a fine balance that must be maintained between the two major divisions of data governance: central versus distributed. The effect is an ability to change and adapt as data and contracts are brought online in the data mesh. The enterprise’s organization must be made ready to handle this model of governance. Also, the organization must promote the federated model with standards from the highest level (also known as the core architecture) and incentivize compliance.

Martin Fowler {https://packt-debp.link/o4dhRZ} points out that centralized golden datasets are no longer pertinent. You have to comply with the data mesh federated principles and established architecture while maintaining data contracts within the data mesh.

Principles drive how the following subject areas impact your designs:

- Data quality standards
- Data contracts
- Security and entitlement
- Audit and regulations
- Modeling and cataloging features
- Self-service and metadata governed
- Metrics and measures captured
- Code steps and transforms
- Error detection and correction

These subject areas then drive the need for a federated governance model in order to keep your solution relevant in the future.

Principle #10 – Metadata is associated with datasets and is relevant to the business

Metadata defines the data and how it became the end-state dataset used by downstream consumers, whether that be internal users, automated processing steps, big data repositories, or **business intelligence** (**BI**)-tooled end-user analysts. Metadata facets are exposed in data catalogs and subject to time series organization to provide filtering capabilities of the data in the data mesh, or data fabric if hosted in the cloud. Metadata is important since it represents the data and can be used to certify data in the mesh as meeting the data-contracted requirements of the business. It establishes the trust required to enable the data itself to be self-serviced.

Capturing metadata and retaining auditable versions of it in the past enables past data to be verifiable and understood in a prior context. It also provides change traceability that adds to current data credibility.

Metadata provides for the creation of facets that may be discovered through dictionary search capabilities. Metadata lineage capabilities aid in the forensic analysis of data quality issues. Domain ownership of key datasets and any value-added business enrichment may be discovered from metadata lineage. Shortening the time to assess errors, finding owners, and determining entitlement issues is an accelerator for the business. Exposing data facets created from metadata also enables you to implement facet intersection and, as such, obtain new insights. If you were to be given the goal to *do more with less*, you would want to leverage metadata.

Principle #11 – Dataset lineage and at-rest metadata is subject to life cycle governance

Metadata, whether it is semantically definitive of the domain owner's truth or reflective of the data journey as lineage, will need to be tracked as if it were a core dataset. This form of second-order data being put into change control and treated as a product makes sense since it feeds the searchable data catalog of the architecture. **Life cycle governance** is needed for all datasets, including the metadata defining pipeline curated datasets. A key addition is the linkage between the curated dataset and its metadata instance as part of the named branch noted as third-order metadata.

You can envision multiple instances of datasets with linked metadata within a pipeline, with the final instance being the change set that is allowed to be released and, as such, propagated to the downstream zones. What determines the acceptability of the releasable instance of data with its metadata is the quality metric. Governance rules check for those contract conditions to be met, and if not, direct action is taken to make them acceptable. These include reprocessing, gap filling, synthetic data creation, data removal, trend smoothing or fitting, factorization, and other statistical enrichment processes.

Principle #12 – Datasets and metadata require cataloging and discovery services

The data catalog is not a static service. It is a searchable up-to-date reflection of the state of data within the data mesh. It maintains the data facets that make the data self-serviceable and the contacts that make it trustworthy.

Discovery and visualization will be key to the data mesh since over time, data loses pertinence. A consumer will want to dial in data at various levels. Often, data must be timely, or it needs to be up to date and correct for some other fixed point in time desired by the analyst. The ability to dial in the zoom when looking for insights must be a key self-service capability of the discovery tool of the data engineering solution.

Principle #13 – Semantic metadata guarantees correct business understanding at all stages in the data journey

It's all about the semantics when bringing data together for analysis. Then, it's all about the quality of the combined dataset before the assessment of an insight's hypothesis for truth. You know that you can't combine apples with oranges and get a valid resulting fruit. They don't cross-pollinate, although you can do this for plumcot, which is a plum and apricot hybrid. The details of what can and can't be combined are in the dataset's semantic at-rest metadata. Preserving the domain owner's understanding of the curated dataset enables the fair use, entitlements, and rights of the consumer.

Clear semantics may be preserved in the creation of a model in an OWL 2/RDF model knowledge graph that supports forward and backward inferencing. As an alternative, you may create an **labeled property graph** (**LPG**) knowledge base, but this is inherently unsuitable for reasoning purposes. For advanced knowledge graph capabilities in an LPG, you need a lot of code to glue the semantics together, but direct inferencing capabilities may have to be given up. A knowledge graph with instance data forms a knowledge base. With a formal knowledge base, you have a very strong and enforceable representation of the domain owner's data in the mesh.

Principle #14 – Data big rock architecture choices (time series, correction processing, security, privacy, and so on) are to be handled in the design early

With this principle, you are being asked to handle architecture choices early and set up a framework for data engineering design success:

- Time series data
- Data corrections
- Data entitlement, data rights, and data privacy

As stated earlier, data is time series sensitive. This was noted as a key issue requiring a solution in a data mesh. Often, data loses value based on age, and this affects how it is to be represented over time. In the simple case, it can be partitioned in a relational system and rolled off after the audit period expires, but that removes it from use unless rolled up as summary info when aged out. How this affects its metadata lineage is also important. Data with its associated metadata together have to be rolled off (or partitioned off) when the primary data ages out. Modern knowledge graphs often lack partitioning capability making this possible and, as such, must be cloned or custom copied over into special archival semantic models created to support a compressed form of historical data.

Likewise, current data corrections, historical restatements, and the effect on downstream data pipeline consumers must be clear in the design of the data mesh. This can't be left to an afterthought without the data mesh, losing the trust it was built to preserve. We've built data systems that preserve data if the assessed change is less than 10% but create a new set if the change exceeds 10% deviation because of correction before becoming a restatement. That percentage should be a dialed-in number and not fixed as per the domain owner's contract, but once set, it should not change for a given contract. A correction is a small change, and a restatement requires full downstream reprocessing.

Entitlement to view the existence of a dataset, the contents of a dataset, or reflect on (inspect) the dataset's metadata must be separate types of entitlement. Additionally, you will want to add enhanced entitlements for fair use (how often data can be accessed), redistribution (ability to send data to another), value add (to enrich data), and resale (to redistribute with profit). Basically, if you can protect the datasets with a constraint, it can be an entitlement. Once you protect data with entitlements, you want to define the type of entitlement structure in your design. This is either via roles (RBAC) or attributes (ABAC), but you must deal with legacy group/ID-based IAM security of third-party products and convoluted cloud security services. These tools and services implement the latest security system *du jour* and do not integrate well with RBAC/ABAC– and you still have an OKR to implement a zero-trust secure system. Our suggestion is to generate the security architecture and keep to it even when faced with many integration obstacles.

Principle #15 – Implement foundational capabilities in the architecture framework first

For this principle the foundational principles componnents include: zero trust data security, test first design, data profiling, metadata design, auditing, and machine learning anomaly detection

The architecture of a future-proof data engineered modern solution will need to address a number of key areas identified in the principles discussed in the various sections of this chapter. These are the following:

- Security
- Test first design (TFD)
- Data profiling
- Metadata design
- Audit
- Machine learning (ML) based anomaly detection

Do not dive into the details of the data mesh implementation without scoping out necessary logical components first because you will not be able to fit them into the solution later if you do not handle them here.

Summary

One needs to grasp the concept of the data mesh and its core principles but apply practical data fabric DataOps features to make the data mesh implementation work as expected.

Data being a product makes sense, but it needs to be fabricated, produced, and curated to the point that data-derived information is ready for self-service, which can be leveraged for knowledge pop-out insights, leading to wise decisions. This progression in the past was forced through big systems software development, but today and going forward, it will involve automated intelligence brought about by the data engineer.

Data needs to stand alone and be smart as well as enabled for insight harvesting. Data needs to be smarter than it is today! The wording of that implies a personification of data. This is a data engineer's goal. Engineered data is no longer hindered by today's big data approaches. Data that is of the highest quality, fully understood, linkable, transparently curated, and contract enforced (to maintain trust and truth) is data enabled for consumption. The principles described need to sink into your data engineering approach. The result will be the creation of systems that lead to the discovery of insights that become data, information, and newer insights in a rapid progression. Intelligence is driven by facts that are supported by information organized by knowledge. So, when we engineer systems to support artificial intelligence, we must not forget that the future-proofed data organization must support similar structures and processes we use to bring about human intelligence. In the next chapter, we'll elaborate on a principle-based conceptual architecture that you will develop based on what you have learned in this chapter.