O'REILLY®

coalesce®

REPORT

# Automating Data Transformations

## Enable Your Organization to Solve the Largest Bottleneck in Analytics

**Satish Jayanthi & Armon Petrossian**

# Automating Data Transformations

## *Enable Your Organization to Solve the Largest Bottleneck in Analytics*

*Satish Jayanthi and Armon Petrossian*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

**Automating Data Transformations**
by Satish Jayanthi and Armon Petrossian



Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.



**Acquisitions Editor:** Aaron Black
**Development Editor:** Virginia Wilson
**Production Editor:** Elizabeth Faerm
**Copyeditor:** nSight, Inc.
**Proofreader:** Elizabeth Faerm
**Interior Designer:** David Futato
**Cover Designer:** Karen Montgomery
**Illustrator:** Kate Dullea

April 2023: First Edition

**Revision History for the First Edition**
2023-04-25: First Release



This work is part of a collaboration between O'Reilly and Coalesce. See our statement of editorial independence.

978-1-098-14756-3

[LSI]

# Table of Contents

# Preface

Many have heard the term *data velocity*, popularized by Oscar Herencia as a part of his five V's of data: volume, velocity, variety, veracity, and value. For the past two decades, the first four V's have grown exponentially, but what about the most important—value? Data transformation exists to deliver value and drive tangible improvements in business outcomes. While many organizations have built data warehouses and lakes, growing the volume of their information, how many have seen their data's *value* grow proportionally?

Today, integrating data into business operations is table stakes. Winning organizations will have the most performant and scalable architectures, prioritize the value of their outputs, and place the greatest emphasis on results. We've seen this pattern in other operational groups: engineering, sales, and marketing, to name a few. From workplace wikis to a slew of Slack apps, automation and tooling drive process improvement, which multiplies the value-add of passionate, curious employees—this is the premise of the modern data stack (MDS).

The MDS is the practitioner's toolbelt. Under its umbrella are products for data ingestion, storage, transformation, analytics, and governance. These solutions enable data teams to be lean and efficient: only a few years ago most were built from scratch, limiting robust data analysis to organizations with a fleet of data engineers and architects. As a result, those who leverage the MDS are capable of delivering more value at a more rapid pace.

Though we've seen much innovation in the MDS, one category has lagged behind: data transformation. In our report, we'll walk through the value-add of the MDS and dive deep into

transformation: its origins, the current state, and how we see it evolving. When we say transformation, we're referring to downstream data processing—the kind taking place in data warehouses. We'll discuss common approaches and demonstrate how present solutions create an analytics bottleneck. Finally, we'll present our solution to automating the transformation problem—a hybrid framework combined with data architecture as a service (DAaaS) for consistent, distributed development of data warehouses at scale. While data processing has come a long way, there's still some distance to scalable, accessible transformation tooling for data teams of any size.

## What You Will Learn

From this report, you'll gain:

- A conceptual understanding of the MDS
- Context on where current tooling shines and where it falls short
- A vision for the future of data transformation: how we see transformation tooling evolving
- A framework for automating the transformation layer
- A path to a scalable, robust data system that creates business value

## Who This Report Is For

Our intention is to deliver value to data stakeholders: directors of analytics, data leaders, and CDOs/CTOs. That being said, we feel that a diverse group of data and product practitioners will benefit from this report. A successful tech stack takes a village—not just the data team. The DevOps, infrastructure, and product teams are all working to pull data initiatives forward, ideally from the same end of the rope. Here's a brief overview of what various groups might gain from our discussion:

- Analytics and data leaders will learn more about the MDS and current industry trends. This will improve decision making and provide relevant context on services, frameworks, and build/buy decisions in the data space.

- Data/analytics engineers will gain a better understanding of why a particular technology makes sense in the context of their organization. Pondering high-level decisions can be a good exercise for individual contributors and a window into strategic planning. We challenge our readers in this group to think critically about their own infrastructure and how it fits into the MDS.
- DevOps/infrastructure engineers are often some of the closest partners with the data team. Because of the tight connection between cloud services and the MDS, these engineers are frequently deploying infrastructure at the behest of data stakeholders. Understanding the MDS can lead to empathy for, and insight into, the processes of data teams.
- Data architects are experts in the patterns and architecture of our data. While they know best how to structure data systems, their design choices rely heavily on downstream users (data and product teams). From this report, data architects will be enabled to make informed decisions and consider the perspective of their partners on the data team.

If you come from another background, whether that be in engineering, product management, or the C-suite, we feel you carry a tremendous amount of power in enabling a data-driven culture. Data is produced and consumed primarily for product and business: *you* are the solution to the difficult problems of democratizing data and extracting its value.

## Why We Wrote This

Our goal is to provide background on the evolution of the MDS and give context to the current renaissance in data transformation. We hope to present you with today's popular solutions and our critiques of their functionality, providing a window into why current data transformation methods must evolve. Above all, we hope to convey an understanding of how to implement a robust, bespoke solution that *delivers actionable data* to your team.

To support our hypotheses, we synthesized information from a number of presentations, academic papers, blog posts, and websites. Nonetheless, many of the ideas expressed in our report are relatively

new. As such, the source material is rapidly evolving and subject to change.

Lastly, while we write as the CEO and CTO of Coalesce, we strive to maintain neutrality and provide you with an objective lens into the evolution of the MDS. We sincerely hope to see our ideas challenged and look forward to the resulting discussion.

## O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

## How to Contact Us

Please address comments and questions concerning this report to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

Email *bookquestions@oreilly.com* to comment or ask technical questions about this report. For news and information about our books and courses, visit *https://oreilly.com*.

# CHAPTER 1
# Today's Modern Data Stack

Quite often, data teams develop a myopic focus on the pursuit of the "perfect" process. While optimization is important, it's easy to overlook *why* companies have sunk millions into data operations. Drawn to the attractiveness of real-time data or the hype of machine learning (ML), AI, and cutting-edge techniques, many seek overly complex solutions when value can be derived from much simpler processes.

The goal of the modern data stack (MDS) is to simplify and democratize access to insight that can enable any organization to improve decision making, delivering value to the business. Working backward to achieve our desired outcome, the *tools* that support a product-led data team must:

- Be simple to implement and easy to understand (democratize access to data)
- Scale with the growth of the company, both in head count and data maturity
- Limit technical debt and vendor lock-in

In this chapter, we'll provide a brief overview of the MDS and walk through basic concepts before diving into the creation of a successful data framework. Afterward, we'll discuss the importance of automation and what data transformation currently lacks.

# What Is the MDS?

The MDS is a term pioneered by Fivetran to describe the solutions that comprise an organization's system for capturing, enriching, and sharing data. We've seen tremendous advancement in data processes—only a few years ago, every element of data ingestion and transformation had to be built from scratch or adapted from an open source library.

In the early 2010s, we saw an explosion of services that provided off-the-shelf functionality for data integration (Fivetran, Stitch), cloud data warehousing (BigQuery, Amazon Redshift), transformation (dbt Labs, Matillion), and analytics (Tableau, Looker). Innovation has continued with the maturation of the data warehouse: the separation of storage/compute and rise of serverless architecture. We now have access to mature, cloud native technologies for data warehouses and operational analytics, with a multitude of tools available at every step of the way. While this has been a boon for productivity in storage, compute, and ingestion, transformation and metadata have lagged behind.

In this chapter, we'll dive into the individual components of the MDS and walk through a framework for building *the right* data stack. First, it's necessary to consider how we think about data.

# Managing Data as a Product (DaaP)

Traditionally, data has been separate from product: analytics leaders report directly to the CTO or another executive. Under this structure, the data team is isolated. While analysts might be embedded in other product groups, development of infrastructure and initiatives lacks a shared understanding of the business. Frequently, this can lead to silos, specifically between the operational units of a business and the engineering teams.

As one might expect, this structure can result in data that lacks context, even from the best-intentioned engineers. Centralized data ownership can result in obfuscated requirements from product teams. For example, often a single individual will be responsible for data directives, from the top down. It's quite easy for these teams to focus on *outputs* rather than *outcomes*. This diverges from our mandate of value creation: how can engineers contribute when they are isolated from the business?

The solution? Manage data as you would a *product*. While it might not be feasible to hire a data product manager, there are a number of best practices that can bring a team closer to the product mindset and more focused on the data "consumer:" business users. First proposed by Zhamak Dehghani in her book, *Data Mesh* (O'Reilly), the DaaP approach revolves around a decentralized framework that includes the following attributes:

*Data services as code*
: Data and query modeling immediately jump to mind, though data teams may also construct data discovery, observability, and product interfaces as APIs to programmatically share information. Codifying data practices can occur at every stage in the MDS, from engineering to analytics/science.

*Data microservices*
: There has been a movement toward microservices in software development, and for good reason. While data teams may necessitate a monolithic element more than other software teams, domain ownership of data is crucial. Those closest are best qualified to understand how and when to leverage analytical data. Microservices bring distributed architecture and decentralized governance to the data space.

*Outcome-oriented teams*
: While activity-oriented teams are effective, they do not effectively scale what matters most: delivering insight built on quality and trustworthy data. Outcome-oriented teams, by contrast, are constructed to deliver business value. The tools data teams leverage must fit this outcome-oriented framework.

*Extensive support for metadata*
: Metadata is the foundation of trust in, as well as understanding and democratization of, data—its importance cannot be overstated. Without accurate documentation, lineage, and governance, the amount of energy spent on triaging issues and answering questions will slowly eclipse the amount contributed to finding insight.

The ultimate goal of a product-first mindset is to deliver value. In the context of data, that means organizing resources, services, and talent in a way that improves the quality of insight derived from data sources.

Of course, a shift from activity- to outcome-oriented teams requires new processes. Data tools and techniques must support cross-disciplinary users, from technical experts to novices, and should support a *self-service analytics* framework. That is, they should enable business users to access data and *democratize* information across an organization. The following are some characteristics for a new generation of product-led data initiatives:

*Flexible*
: Data tooling must be malleable. Without flexibility, many will resort to hacky solutions that skirt best practices or opt for building their own solutions.

*Shallow learning curve*
: As solutions mature, the barrier to entry must fall. While some technical proficiency will always be necessary, building outcome-oriented teams requires *every* member to be able to triage problems and implement solutions. Complex, code-first products limit collaboration to only the most technically advanced users and impose barriers to development and progress.

*User-friendly*
: While the days of the command line are mostly over, some hang on to code-only solutions. Self-service tooling that democratizes access to information requires an element of user-friendliness. While this doesn't *exclude* a coding element, it does imply that intuitive graphical user interfaces (GUIs) should be present.

*Metadata-first*
: Like other areas of product, data must be well-documented. Unlike those areas, however, documentation, governance, and lineage is an incredibly expensive task. For data to be valuable, it must be discoverable, understandable, and engaging across *all* teams in a business. The next generation of data platforms must put metadata first to unlock value.

*Version-controlled*
: For data to be managed as a product, it must be built with software best practices. Tooling, regardless of code volume, should be version-controlled, and data teams should iterate to construct robust, reliable pipelines, business intelligence (BI), and models. Version control enables collaboration at scale.

As the market is flooded with new customers (data teams), solutions must enable a DaaP mindset without the need to build and scale full data engineering, analytics, and science teams. These products will replace in-house solutions and enable drastic productivity gains, further lowering the barrier to entry in analytics, science, and ML, and raising the bar for the average company to compete.

# Basic Terms and Concepts in the MDS

The core areas of the MDS are ingestion, storage, transformation, and analytics. We'll break down each to understand more about how the individual components of the MDS have shaped its trajectory.

## Ingestion

Data ingestion refers to the process of extracting and loading data, the first two steps in an ELT (extract, load, transform) architecture. This means that ingestion is the first step to building a data stack. Like the foundation of a house, a rock-solid ingestion framework is essential to the stability of subsequent bricks in the construction of a data warehouse.

Data ingestion solutions are tailored to the problems they solve: where does data originate, and where is it being written? Today, a number of ingestion tools offer pre-built connectors and abstract away the many headaches of custom pipelines. Fivetran is the current market leader, while upstarts like Airbyte and Meltano offer an open-source alternative.

What does this mean for the modern data practitioner? Data ingestion is a developed space with a number of great options. It currently allows a data team to go from 0 to 1 in hours/days instead of weeks/months. Still, there is a need for technical prowess with hard-to-find datasets.

## Storage

Data storage is perhaps one of the most mature aspects of the MDS. Products like Snowflake, Redshift, and BigQuery have innovated and helped to spur competition in the online analytical processing (OLAP) space. Modern data warehouses are optimized for analytic processes and can scale to handle loads that older platforms

(PostgreSQL, MySQL) simply can't match, thanks to parallel computing and column-oriented architecture.

When these products first emerged, storage and compute were bundled. Thus, many users found their usage limited by the constraints of the service: they had either plenty of storage but hit analytical limits, or analytical bandwidth but found themselves out of storage. Increasing the availability of *both*, however, required a tremendous price increase.

Today, storage and compute can be completely separate. The change (and competition) adds much-needed flexibility in data warehouse cost structure. Even the concept of server maintenance has been abstracted away—virtually no infrastructure management is necessary with products like Azure, Redshift Serverless, BigQuery, or Snowflake.

With these changes, many are opting for a data lake architecture, where data is first staged in Amazon Simple Storage Service (Amazon S3) or Google Cloud Storage (GCS), then loaded and transformed in a warehouse mentioned earlier. This approach allows for greater breadth of storage with nice implications for disaster recovery.

What does this mean for the modern data practitioner? Choosing the right platform will always be a difficult decision, but great options like BigQuery, Redshift, and Snowflake are ubiquitous. These are poster children for the MDS: they remove complexity and DevOps from the equation and allow data teams to focus on what matters.

## Transformation

Data transformation is the manipulation and enrichment of data to improve access, storage, and analysis. This most frequently occurs on data that has already been structured and stored in a data warehouse. Hence, transformation is accessible to many via SQL and Python. For this reason, we won't consider upstream frameworks, like PySpark, Dask, and Scala, which are limited to implementation by data and software engineers. While these are important tools, their highly technical nature makes them difficult to democratize. As data volume grows, so does the importance of a solid transformation layer—refining and revealing the most pertinent information to analysts, stakeholders, and data scientists.

Still, transformation is one of the most nascent aspects of the MDS. While dbt was the first in transformation to appropriate software engineering best practices to analytics, many are still operating on some combination of Apache Airflow for orchestration plus dbt Core for execution. While this combination *works*, it is very technically demanding and *does not* scale well. Alternatives, like Matillion, are more focused on a GUI, which, while adding simplicity, removes necessary customization and locks users into predefined routines.

Transformation is due for an overhaul. We feel that transformation lacks a metadata-driven element that operates on the column level or, as we say at Coalesce, is *column-aware*. Furthermore, a more cohesive combination of GUI and code is necessary to include *every* member of a data team, while allowing the more technically-minded to fine-tune models to their needs. Once this transformation pattern is established, we feel data teams will be prepared to leverage a DAaaS, introduced in Chapter 3, to automate the transformation layer and provide value at scale.

What does this mean for the modern data practitioner? There currently exists a number of established products that allow data teams to transform integrated data. What they lack, however, is the *automation* of transformation processes: a metadata-driven, column-aware architecture that combines GUI/code and maximizes code reusability. We'll touch on this more throughout the report.

## Analytics

When we say "analytics," we're broadly referring to the process of using SQL and GUI platforms to generate visualizations, reports, and insight. Data analytics and BI are separate in many product organizations, but we'll just call them "analytics" for simplicity's sake.

Data analysis has a highly human component and is one of the most challenging aspects of deriving value from data—from an *entire* data lake, an analyst, scientist, or engineer has to think critically to choose the correct data for the task. The upside: improving efficiency in business and product with massive payoff. The downside: potential misinformation and the creation of data silos—*negative* returns on investment.

BI is another part of the MDS that has been highly refined. Services like Tableau and Looker have been under development for *decades*

(Tableau being released in 2003). While newcomers like Metabase, Apache Superset, Sigma, and ThoughtSpot are challenging the status quo and introducing intuitive new ways of presenting data (at a dramatically lower cost), there are well-established leaders who will continue to serve the majority of the market.

However, the *data exploration* part of analytics is undergoing a transformation of its own. There are entire websites dedicated to finding the best hosted notebook tool—while a few leaders stand out (Hex, Deepnote), the field is wide open. Hosted notebooks increase efficiency of data science/analytics teams by integrating SQL, Python, and R into the already popular Jupyter Notebook format.

What does this mean for the modern data practitioner? There are a variety of choices in the data analytics space, but some combination of BI plus an exploratory data science/analytics (EDA) platform will be a good fit for most organizations. Many BI products are mature and expensive, but newcomers are looking to disrupt that trend. Newer EDA tooling might lead to a shift in how analytics/science teams operate.

## Governance

Data governance is inherently broad, but the umbrella term refers to all under data security, privacy, accuracy, availability, and usability. Data governance carries many benefits but is often overlooked or delayed due to the (seemingly) high cost of implementation. In reality, sound data governance principles will pay dividends far beyond the cost of investment. Three of the most important aspects of data governance are cataloging, observability, and lineage:

*Cataloging*
: Describes the data itself: column names, enum values, and context. It's the documentation that allows an outside viewer to *understand* your data without the years of experience working in it. Many transformation platforms have catalog functionality—Coalesce, dbt, Google Dataform, etc. Still, dedicated frameworks for metadata tracking, like Amundsen, are also popular.

*Observability*
: Makes sure that data is reliable and *as expected*. With more data than ever, how do you know there aren't silent errors in your pipelines? Data observability solutions fit this niche. Databand, Datafold, and Monte Carlo all provide observability solutions.

*Lineage*
: Describes the path data took, from ingestion to visualization. Many popular tools contain table-level lineage in the form of a directed acyclic graph (DAG) as built-in functionality (see Figure 1-1).

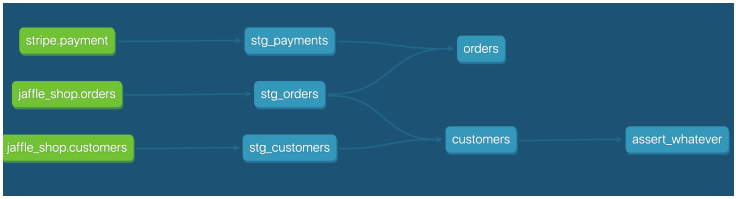


*Figure 1-1. An example of table-level data lineage in transformation*

What does this mean for the modern data practitioner? Data governance is a far-from-developed part of the MDS. It is a natural fit for part of the transformation step—it's efficient to document new tables/data at the source when they're created. Unfortunately, not all data tooling provides column-level lineage, replete metadata tracking, and automation of data governance. We see these as emergent trends in the transformation layer.

## Considerations When Creating an MDS

While we've provided a brief overview of each phase of the MDS, we've yet to dig in to how you can apply this knowledge to build your own performant data system. As we've suggested, software choices can be tricky. Aside from the difficulty in obtaining accurate information, biases, dogma, and bureaucracy can weigh on decision makers, company-wide.

Rather than thinking from first principles to address the problems that exist and potential solutions, many teams will opt for solutions that are the cheapest, most popular, or most familiar. These are common pitfalls that will lead to suboptimal solutions. Instead, we feel teams should prioritize:

*Completing the objective (solving the problem)*
: Does the solution actually solve the objective? Every team is different—just because a solution is popular or ubiquitous doesn't mean your team *needs* it.

*Considering the total* cost of labor *in decisions*
: Rather than balking at a steep asking price for a product (though some skepticism is necessary), consider if the product will result in less labor. If possible, attempt to quantify the time saved. Labor is often the greatest expense for a technology team. This is a multidimensional calculation: we need to consider the technical proficiency necessary for a product, the amount of time it will save, *and* the number of employees required to scale the solution. Tools that win on all three fronts have the potential to drastically reduce cost and represent huge wins for the organization.

*Addressing technical debt/the scalability of various solutions*
: Does implementing the free solution create a mountain of tech debt that will need to be addressed? Will a greater up-front cost result in long-term savings? Only *you* can answer these questions, but a long-term approach can have exponential payoff.

## Automation in the MDS

From our discussion, it should be clear that some aspects of the MDS are more developed than others. There has been a trend in data processing toward automation: the abstraction of administrative tasks and reduction of operational overhead, which allow data teams to focus on precisely what matters.

For the MDS to deliver what it promises, automation is key. Thus, to address what's *missing* in the MDS today you need only compare progress in automation across categories. For example, take data ingestion. While a more constrained problem (taking data from a source and moving it to a target is narrow in scope), the data ingestion problem has been nearly commoditized: you can now find a number of extremely polished products that "just work."

By contrast, transformation has not yet reached this stage. While the process of transforming data has already undergone a renaissance, it's due for another. We've seen a plateau in the progress of transformation tooling as many companies work to adopt existing solutions.

While current solutions are great advancements, each lacks one or more of the components we highlighted as key to truly automating data transformation. A new wave of technologies will complete automation in the MDS by eliminating this current bottleneck.

## Summary

Using the guiding principle that data should serve to create value and generate tangible business outcomes, the DaaP framework is a way to think about *how* your tools of choice should function.

By providing an overview of the MDS as it exists today, we've highlighted the strengths and weaknesses of current solutions. Using history as a guide, it is evident that a second renaissance in data processing is here. In this iteration pain points and bottlenecks in data transformation are being overhauled, and data governance is making its way downstream—originating at the column level in the transformation stage.

In Chapter 2, we'll dive into the specifics of data transformation—examining the pros and cons of transformation frameworks, comparing code and no-code approaches, and ultimately presenting what we like to call the "golden middle" of transformation: a solution that provides *just* the right balance of flexibility and consistency.

CHAPTER 2

# A Renaissance in Data Transformation

In this chapter, we'll expand on the transformation layer. We will provide a brief overview of the importance of the transformation, discuss the importance of ETL/ELT, and jump into existing solutions. We'll then present the benefits and challenges these solutions pose, framing each as code- or GUI-first. Taking the best of both worlds, we'll present a solution that finds the "golden middle" for a flexible, yet user-friendly, experience. This golden middle of data transformation represents the second revolution in data processing and the first true automation of the transformation layer. Finally, we'll provide direct examples of how you can use this framework to further analytics and engineering efforts on your team.

## Why Data Transformations Matter

With the growing volume and variety of data, it becomes the task of a robust transformation framework to concisely filter, aggregate, and present findings in a manner that's easily understandable. Data transformation is essential to extract (pun intended) value from all this information.

For this reason, it's essential to implement a framework that provides consistent outputs with as little overhead as possible. *Every* member of the data team should be able to contribute, not just those with technical backgrounds. Furthermore, this solution *must efficiently scale* to handle both tremendous quantities of data *and* an

ever-expanding domain (schemas, tables, views) within any number of data warehouses. Additionally, the scope will be downstream from big data processing—at the warehouse layer, where most work is done in SQL.

In Chapter 3, we'll discuss existing solutions in the transformation space, including where they shine and why they fall short for many. From these shortcomings, we'll present a hybrid option that suits the needs of *most* data teams, formalizing our vision for a transformation engine of the future.

## ETL Versus ELT

Two concepts central to data transformation are ETL (extract, transform, load) and ELT (extract, load, transform). The first serves as the origin of data transformation, dating back to the first days of databases themselves, while the second is a more recent rendition of data processing:

*ETL*

In the early years, data warehouses looked very different. For a company to have a database, it needed to have a server. As long as that server was running, the company could host a database like PostgreSQL or MySQL. Hence, all but the largest teams were constrained by the costliness of server maintenance and the difficulty of physically upgrading components.

It's easy to see how analytics teams might be limited by these processes in the early days. This is how ETL originated: data was first extracted, then transformed to keep what was truly important, and finally loaded into the target (see Figure 2-1).

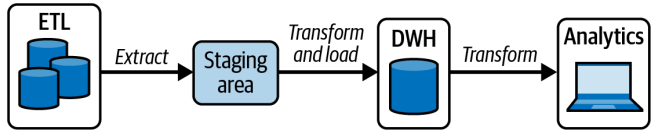


*Figure 2-1. The traditional ETL process*

Being resource-constrained by the cost of managing and upgrading on-prem systems, data had to be carefully curated *before* taking up space on-site. Hence, ETL is a legacy transformation option that has been around for the better part of 30 years.

*ELT*

It wasn't until the mid-2000s and early 2010s that cloud technology began to gain traction. With the introduction of autoscaling compute and storage, it became easier to manage data resources.

In the mid- to late 2010s, we saw tremendous competition in the cloud computing space, contributing to a sharp decline in cost. Business teams could tailor data warehouses and cloud storage precisely to their requirements and cost tolerance at the click of a button.

In recent years, we've seen the proliferation of the data lake (enabling Extract and Load processes to precede Transformation). Data lakes are built around storing almost *everything* in a low-cost, semistructured format (e.g., JSON or Parquet files), using new technologies to analyze directly from the lake *or* transform for a data warehouse (see Figure 2-2).

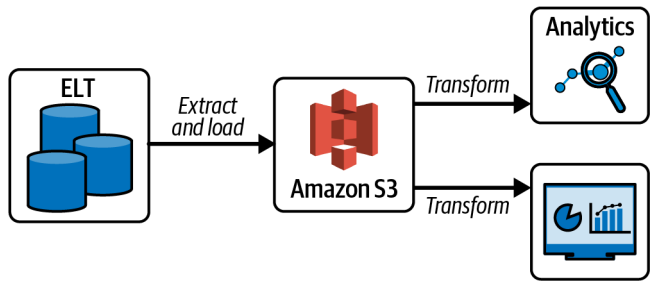


*Figure 2-2. The ELT process*

This unlocks incredible power in exploratory data analysis. Now, many are facing the problem of *too much data*, sparking a heightened demand for data analytics, data science, and ML engineering. The same is true for data engineering: never before has the need to filter and transform data been so great.

# Data Transformation: Existing Solutions

Now we'll discuss data transformation approaches in the modern data era. Note that many of these are still "new" in the sense that they're patterns that have been established in the last one to three years. We do not consider legacy products, like Informatica or Talend. Furthermore, we will narrow our focus to transformations

taking place *within the data warehouse*. While useful functionality can live upstream—batch and stream processing with PySpark and Dask, or other distributed transformation frameworks like Scala and Java, etc.—it is necessarily limited to data/software engineers. To revolutionize data transformation, we'll narrow our focus to the data warehouse, where SQL and Python make transformation accessible to a much broader audience.

## SQL Plus Orchestration Tooling

Much of data transformation is a bit of Python and a large amount of SQL, orchestrated according to some order/hierarchy. Thus, the most bare-bones and homegrown transformation method is to: 1) write SQL; and 2) orchestrate it using Airflow or another platform.

Given that this is the foundation of many code-first modern data frameworks, it seems like a simple enough approach: find an orchestrator and establish a pattern for referencing sources and targets within SQL; use that orchestrator to generate a DAG and execute code in order; and host that code in a version-controlled manner.

From personal experience, this gets out of hand quite quickly. We've seen firsthand the havoc caused by such a system: changes to infrastructure can be incredibly hard to test and maintain, resulting in frequent job failures that are notoriously hard to triage. The absence of formal logs and tests result in vague errors that often take hours or days to resolve.

## Code-First

Many code-first open source frameworks for data transformations have been created in the past few years. At their core these products are transformation engines, providing the backbone for automating a series of SQL manipulations while providing the infrastructure to define macros, references, and metadata. Popular open source tools, especially among small and midsize businesses and in the startup ecosystem, include dbt and Airflow.

Code-first solutions enable reusability, easy versioning, and the structure necessary for most data teams to stand up a robust transformation pipeline within a data warehouse. The catch? They are extremely technical, with a steep learning curve. Most work will be done entirely on the command line and in code editors. Many

readers will know—there's quite a bit of manual work and often arduous setup involved.

While one can often find a script or add-on package to automate away some of the headache, the functionality of code-first solutions is quite limited. A team of engineers is necessary to manage orchestration, continuous integration and continuous deployment (CI/CD), and other essential functionality. Most code-first solutions offer cloud-hosted alternatives that handle deployment and orchestration, at a price, but these are often equally difficult to configure and maintain.

While we have different beliefs about what a truly scalable, enterprise data transformation solution should look like, we would be remiss if we didn't recognize the impact of creating open source solutions in addressing the challenges of legacy data transformation products. Code-first frameworks paved the way for future advancement.

## GUI-First

Matillion and similar products differ from dbt and align more closely with the legacy products, like Talend or Informatica. Built around a GUI, they're tailored toward teams without a dedicated engineering component or with limited technical resources.

The low-code approach is accessible to a broader range of organizations, and the inclusion of scripting functionality means that there *are* opportunities to tailor the experience to niche demands.

Often, the proprietary configuration can be synced via Git, and collaborators can edit projects in real time, similar to many code-first solutions. As these products have matured, they've introduced functionality similar to Fivetran for data ingestion along with monitoring functionality. Some have even stepped up to reduce complexity for the custom API calls.

Most aim to be an all-in-one solution that provides a quick way to enable several parts of the modern data stack. Unfortunately, products that try to do too much often end up doing little well.

Perhaps the largest disadvantage of a GUI-first framework is the lack of flexibility and customization. Many GUI-first products are older and started as data integration platforms, not transformation platforms. As a result, the UI is dated and functionality can

seem bizarre. The inflexibility introduced by a lack of code means that replicating components and automating data transformation is actually quite difficult, despite the implied promise of simplicity.

While they solve some of the headaches of a code-first implementation and decrease the complexity of the solution, they often end up requiring just as much time and bring less scalability to the table (see Table 2-1).

*Table 2-1. GUI versus code*

| Item | GUI-first | Code-first |
|---|---|---|
| **Technical proficiency** | Low to medium | High |
| **Metadata tracking** | Absent in most, if not all, cases. | Feasible, but often repetitive. For example, tracking source and references in dbt Core is an incredibly time-consuming task and requires repeating large amounts of code/text. |
| **Flexibility** | Low—tends to concede flexibility for user-friendliness. | High—can be customized to the heart's content, at a price: a high degree of technical skill. |
| **Vendor lock-in** | High—often notoriously difficult to export/transfer. | Medium—while code-easier to export, proprietary formats may exist and other barriers may arise. |
| **Cost to build/ maintain** | Low—with a gentle learning curve, GUI-first products can be adopted and managed by a wider audience, mitigating labor cost. | High—finding and retaining the skilled human resources required to build and maintain a code-first solution can be burdensome. |
| **Community support** | Low to medium—few GUI-first solutions are open source, and subsequently there are few communities around these products. | Medium—open sourced solutions tend to have a wide body of community support. Choosing popular products and researching community adoption is always wise if your organization is committed to an open source solution. |
| **Customer support** | High—given the lower complexity, GUI-first products can be more easily supported and triaged. Vendors may even provide suggestions on how to use the product in the most efficient manner possible. | Low/none—it will be extremely difficult to troubleshoot errors arising from custom, code-first solutions. In the case of open source products, community support will be the only option. |
| **Ease of configuration** | High | Low |

| Item | GUI-first | Code-first |
|---|---|---|
| **Scalability** | Low—GUI-based tools *usually* have trouble handling transformations at scale. There often isn't a way to programmatically generate transformations, resulting in a point-and-click nightmare. | Low—maintaining transformations, metadata, linting, and CI/CD for a pure code solution will soon become a daunting task. The *discipline* required to enforce uniformity and standards can consume a large amount of valuable engineering resources. |
| **Column-aware architecture** | None (so far) | None (so far) |

# Data Transformations: Finding the Golden Middle

So far, we've explored the foundations of data transformation, including homegrown SQL plus orchestration efforts, open source (typically code-first) solutions, and GUI-first alternatives. Hopefully, it's evident that each of these alone is insufficient to address the characteristics of tools we identified in Chapter 1: flexible, user-friendly, metadata-first, and version-controlled with a shallow learning curve. Materializing these characteristics in the transformation layer, specifically, the ideal solution will have the following:

*Transformations as code*
: The ability to define custom transformations, either through Python or SQL, that can be templated and distributed across an implementation.

*GUI support*
: Products with a GUI are often more approachable than the command line and help to democratize data transformations. They allow technical and nontechnical members alike to contribute to data warehouse development.

*Column-level metadata awareness*
: For data to be valuable, it must be discoverable, understandable, and engaging across *all* teams in a business. Starting with the finest grain possible—the column—enables maximum data efficacy.

*End-to-end automation*
: A truly scalable tool will be one that automates *all* the repetitive processes in transformation—from code generation to metadata definition.

These are the features that are essential for automated data transformation to work toward the goal of delivering the greatest amount of value possible. Naturally, these elements combine the best of code- and GUI-first, so a true solution must be either a hybrid approach or one that finds the golden middle of data transformation.

## Hybrid Approaches

A true hybrid transformation approach that balances the flexibility of code with the gentle learning curve of GUI has yet to emerge as mainstream in the data space. While hybrid products are not for every team, they are a solution tailored to the majority of data teams. While some users will require the raw flexibility of code (and have the resources to maintain it), others will need the stark simplicity of a GUI.

When evaluating a hybrid data transformation platform, here is some key functionality to look for:

- GUI elements that allow technical and nontechnical users alike to build data pipelines quickly and simply.
- Code-first elements that enable programmatic generation of pipelines and in-kind widgets/modules without the need for hundreds of clicks or individual components.
- Flexible in the sense that almost everything is customizable.
- Rigid enough to enforce standardization in tracking and metadata management.
- Column-level lineage—we must track data movement at the column level to automate analytics documentation and simplify the triaging of data-related issues.
- Automation of administrative tasks typically required by analytics/data engineers. In the case of code-first tools, this might be declaring sources, pulling through metadata, and performing any other repetitive task that "always has to happen."

These features will result in accelerated development and scalability—for both code and metadata management. By automating away tedious processes and ensuring that metadata is sourced at the column level, data warehouse configurations will be well defined—from sources to curated views—enabling rapid development of data pipelines.

## Summary

The falling cost and barriers to cloud computing have given rise to a data landscape of the future. This landscape is much more friendly to large-scale data processing and incentivizes a "store first, analyze later" approach. As a result, data lake architectures have become the norm for most product teams.

Many current data transformation solutions in the modern data stack are insufficient for this advance. The most bare-bones approach is unscalable and unstable. Code-first solutions isolate nontechnical users and are equally difficult to scale. While GUI-first is more user-friendly, rigidity makes it prone to time-consuming manual processes, a lack of community support, and a number of headaches in implementation and upkeep.

Thus, we believe the golden middle will revolutionize data transformation by providing a fully automated and scalable approach to data transformation that takes the best of code- and GUI-first products, combines them, and optimizes for modern data teams. Hybrid transformation frameworks will address problems critical to a majority of data teams.

Unfortunately, creating value with the MDS isn't as simple as buying a product. In Chapter 3, we'll provide a framework for creating business value from data and discuss how it's the *automation* of the transformation layer that opens the door to self-service analytics and data democratization.

CHAPTER 3

# Delivering Value with Data Transformations Through Automation

Though there are numerous ways for data systems to create value, each follows a core set of principles. A successful operation needs to be built on simplicity, flexibility, user-friendliness, and a metadata-first approach. A foundation in metadata is essential. Specifically, metadata originating at the column level allows for the precise dissemination of business context. While wikis and ad hoc questions might work for startups, this quickly becomes untenable at large enterprises.

More importantly, a metadata-based implementation enables a data architecture as a service (DAaaS) approach. As we'll present in the following section, DAaaS leverages "data patterns" that can be used to break down data silos, eliminate analytics bottlenecks, and automate the many pain points that exist in the transformation layer today.

## Principles of Data Value

The concept of providing value with data is not new, though it has grown in popularity and depth in the last few years. We feel data value is best approached through the data mesh framework. When viewed through this lens, it becomes apparent that a decentralized approach will be transformative in the data space. Decentralization

of data skill, combined with a column-aware architecture and automation in the transformation layer, will serve to deliver value in the most efficient way possible.

## Product-First

In Chapter 1 we introduced the concept of data as a product (DaaP), in which valuable data is easily discovered, understood, trusted, and explored. A product-first mindset means developing data resources with the following characteristics in mind:

*Discoverability*
: How do stakeholders know about that great summary table an analyst built? How can you avoid duplicate queries that return *slightly* different answers to the same question? Data discoverability reduces the friction to finding the data that already exists and the work that has already been done. While metadata collection is a prerequisite, dissemination of your team's data universe is key. Without the ability to educate users on *what exists* in near real time (as assets are created), many efficiency improvements will be overlooked! Be sure to evaluate the discoverability of your data as you invest in data governance and transformation tooling.

*Understanding*
: Establishing *understandable* data begins with the transformation layer. Solutions that allow for column-level lineage are key as they provide the greatest amount of context. Enforcing rigor in data assets is essential, as it reduces the *discipline* required to construct consistent outputs: in many data platforms, the responsibility falls on one or a handful of individuals to police code, ensuring that conventions are followed, *or* to construct complicated CI/CD processes to lint and verify work.

*Trust*
: Building trust in data is an arduous process. It requires a high degree of consistency and a dedication to robust and accurate pipelines. Failures, whether of data jobs or the ability to deliver accurate reporting, will not soon be forgotten. This will undermine even the *best* attempts to integrate a data-driven culture.

  Using the modern data stack, trust is acquired through production systems built for *uptime* and *consistency*. A rigorous

adherence to data as code and documented processes will help to ensure services that inspire trust in the end user.

*Exploration*
: Explorable data empowers stakeholders to quickly find answers to their questions. Becoming comfortable with data systems will enable them to generate *new questions and ideas*, unlocking the true power of distributed data knowledge and kicking off the virtuous cycle of the scientific method.

Data practitioners should seek to implement a framework that enables many members of their team to create data systems that adhere to the preceding four characteristics.

This starts with a column-first mindset.

## Column-First

In recent years, data observability has become mainstream. At one time it was incredibly complex, but now firms like Atlan, Monte Carlo, and Datafold have helped data teams to better understand lineage, comply with regulation, and even detect silent errors/unintended outputs without the need for DIY tracking systems or anomaly detection services.

What drives data observability? Understanding is not possible without scrutiny at the finest grain possible—the column level. By recognizing how each column is transformed and *where* these columns go, a wealth of functionality abounds. Unfortunately, observability and transformation have evolved separately in the MDS, though the two are inexorably linked. While the aforementioned tools work well, they're separate from the source of data changes: the transformation layer.

Additionally, current code- and GUI-first transformation solutions are built at the table level. To understand columnar data, one has to purchase, implement, and maintain a separate observability service. This is inherently inefficient, as many of these operate via reverse engineering that attempts to circumvent table-level shortcomings in the transformation layer.

The current, circuitous approach is disjointed: every column in a data warehouse is sourced from a raw table that's created through ingestion. From ingestion, these columns are *transformed* to the tables and views used for decision making (see Figure 3-1).

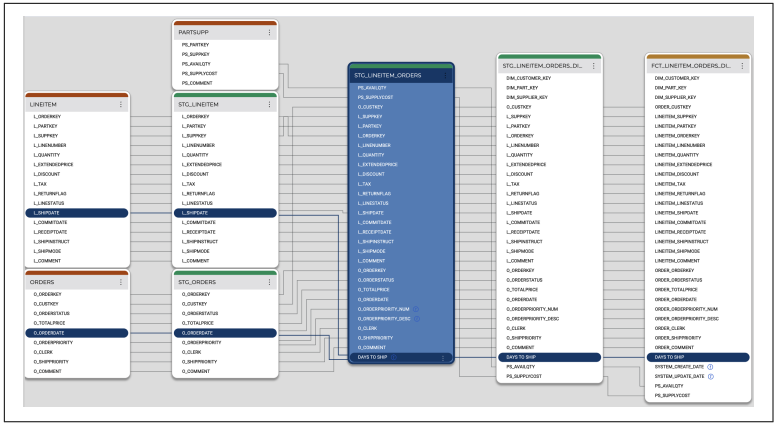


*Figure 3-1. Column-level lineage is the most visual benefit of a column-aware architecture*

Thus, if you track metadata at the column level, starting from your entry point and interwoven with your transformation tool, it should be possible to tie each column directly back to the source. It follows that this should take place in the transformation layer, not a separate solution that tries to back in to observability. The benefits of a column-based architecture go far beyond metadata management: the ability to craft and distribute the building blocks of data transformation at scale is made possible only through column awareness.

It's our belief that the transformation platform that delivers the most value will be one founded on a column-first approach, not only with hybrid code and GUI implementation but also by being built on the foundations of data transparency and observability. Value in data transformation starts at the column level. At the finest grain possible, you can leverage the true power of automated transformation, enhancing the discoverability, understanding, trust, and exploration of our datasets and delivering value with the MDS.

# Optimizing the Transformation Layer

The concepts of data mesh and DAaaS/DaaP revolve around a decentralized structure for data curation. The ultimate goal is to enable every team to build relevant data infrastructure. It might sound trivial for a small company, but this approach becomes critical as the company grows.

In an excellent talk at the 2022 Amplify product conference, Geoff Coyer described how he supported *over 700 monthly active users* with a data team of six. Perhaps more surprising is that his team still finds time to progress on data science and engineering initiatives. How is this possible? *Enabling self-serve analytics*. Though much time and energy has been spent on developing "self-serve" tooling, we feel current methods miss the mark. In this section, we'll discuss our approach to an accessible data framework.

## Enabling Analytics at Scale

While self-service analytics and "data democratization" have been some of the hottest data buzzwords of late, we feel this to be misguided. While it is necessary to have a BI layer that enables *nontechnical* stakeholders to explore data, this assumes a solid data foundation. How can a CMO craft a compelling story when a marketing analyst is waiting on support to implement an attribution framework? What about a financial analyst who is held up by a transaction pipeline? While self-service is indeed a powerful concept, the assumption is that every arm of the business has access to adequate data resources, which often isn't the case.

To deliver value with transformation at scale, we must enable all teams to build and maintain their own data infrastructure while still working collaboratively with each other. Traditionally, this approach has been incredibly messy and accelerated data siloing. This can be avoided with a change in methodology—the adoption of DAaaS.

## DAaaS

Consider two auto manufacturers. Company A operates under a "custom-built" or tailor-made model: each component is handcrafted and unique, making the final product one of a kind. Company B takes the opposite approach: manufacturing parts according to exact specifications and with rigorous precision, then assembling downstream to create the final products, all of which are nearly identical.

Neither company is right or wrong, but their biggest differentiators are scale and cost. While Company A might be able to produce 200–300 cars each year, increasing output will lead to lower quality and inconsistency. Additionally, those 300 cars will come at quite the price, since only the most skilled workers will be able to make them.

This model does not scale. This may be sustainable for a company like Ferrari, but it will not work for one like Tesla.

At Company B, however, production can scale up or down to output cars in the millions, as long as economics are maintained. Thanks to the production process, Company B has two categories in production: architects/engineers who design parts and those in assembly who implement them. The only prerequisite to assembly is domain knowledge, which lowers the barrier to entry and lets Company B allocate more talent to that task.

Data transformation is not an assembly line, and relational data is neither Ferrari nor Tesla, but the analogy of a bespoke data solution versus a scaled, consistent one is apt. The former will be tenable in small software-as-service (SaaS) companies with cutting-edge products, a passionate team of engineers, and about 25–50 employees. Past that, inconsistency and cost will balloon. What we need, then, is an approach that is:

- Consistent and rigorous in its outputs (i.e., data architecture, naming conventions, and implementations) *across* business organizations.
- Scales indefinitely from a 50-person organization to a 50,000-person organization.
- Cost-effective in the headcount required to implement and maintain. Specifically, maintenance and migrations are often overlooked.

DAaaS is the solution. DAaaS is an approach that involves technical team members—typically, architects and engineers—creating development templates, or "patterns," that can be implemented downstream by almost anyone in an organization. DAaaS has a number of benefits across all organizational sizes:

*Removes bottlenecks (centralized data team)*
: This is accomplished thanks to a parallelized/asynchronous data implementation.

  *Parallelized*
  : Every team can work simultaneously to build its own, slightly different infrastructure.

*Asynchronous*
: Data transformation often acts as a bottleneck to analysis. Under DAaaS, the most skilled practitioners only need to review implementations, not create them from scratch. This enables asynchronous development of shared resources, eliminating bottlenecks.

*Adds context*
: Puts those who are closest to the business context in charge of their data.

*Breaks down silos*
: Multiple business arms are integrated to a larger data monolith: a data microservice approach.

It's easy to see how DAaaS fits into the data mesh approach to development: a centralized data team creates data patterns and builds the foundations of a data warehouse, with column-level metadata on foundational tables. This enables the following:

- Decentralized business users and subject matter experts (SMEs) to understand relational data and build derivative resources without advanced technical skill
- Keeping ownership of data transformation in the hands of the SME (the relevant product/business group) and allowing those with the most context to create data resources
- Removing the "gatekeeper" status from the centralized data team, freeing up time and resources to pursue larger projects

A solution that enables a DAaaS architecture will parallelize the implementation of data systems, automate the hard parts of transformation, and enable a data-driven business at scale. In our opinion, a unified platform that allows architects and implementers to operate in the same domain will be most efficient. A single platform is important for cohesion and reduced friction, as well as process improvement. Often, it's those closest to a process who invent improvements and optimizations.

Thus, our transformation solution should work to both enable advanced technical members to show their true potential *and* democratize pattern implementation to business users in a friendly way. Learning data systems is a practice of trial and error—asking managers about the context of a column, checking a query

with peers, and verifying the correctness of an analysis. This platform should streamline this process by intertwining column-level metadata and transformation to draw clear lineage between data resources.

While one may separately purchase and maintain data tooling (observability, metadata tracking, and transformation platforms), this is both expensive and time-consuming. The complexities of working with three separate vendors for enterprise-level software should not be understated. Additionally, product dependencies and integrations may be complex. We feel an all-in-one solution will be the more straightforward and cost-effective approach.

## Data Patterns

When we say "data patterns," we're referring to common patterns in data modeling that we've seen throughout the years. Frequently, these patterns are implemented in the transformation layer as SQL or Python to produce a table that provides additional context or insight. Take, for example, an SCD type 2 table. SCD stands for *slowly changing dimension*. Type 2 indicates that a new row is created for each change to an existing record in the source table. This is a common pattern for transactional tables that provide a "change log" of the table contents.

Creating an SCD type 2 table is an exact science. An analytics or data engineer will write SQL to transform source transactional data into a summary table. *However*, it is a learned skill, with many possibilities for error. The engineer must have a solid grasp on window functions, cursors, and primary keys to ensure they accurately replicate the source data. Complexity compounds when data is transformed incrementally. This means that implementing SCD type 2 tables is often reserved for a centralized data team, which can quickly bottleneck work *and* result in data teams working on problems outside their area of expertise. Metadata tracking and naming conventions are additional areas of potential error.

*Data patterns* are the solution and a byproduct of a DAaaS approach. If data architects and engineers construct an SCD type 2 pattern, downstream users can implement corresponding tables without having to worry about complexity or convention. Breaking down common tables into patterns and building reusable "templates" can dramatically accelerate data warehouse development and reduce error—it's the automation of the transformation layer.

So if you sit on finance as a junior analyst and know you need a change log of subscription status, you can implement an SCD2 pattern by merely picking a piece of pre-built code off the shelf and selecting appropriate cursor columns. Then you can be confident you're getting the correct results and proceed with your analysis.

Other examples of common patterns in data warehousing and analytics include hubs, satellites, and links in a data vault, as well as the dimensions and facts in a star schema. Data-loading patterns such as incremental loading using change data capture (CDC) are also common, as are data transformation patterns such as deduplication, aggregation, and standardization.

Optimally, patterns are integrated in the same platform as transformation, with visibility in their implementation. Then the junior analyst can observe and learn about the patterns they're implementing, should they choose.

Patterns leverage advanced technical talent to abstract away tricky processing, incrementality, and materialization logic—democratizing access to transformation and eliminating bottlenecks created by centralized data teams. This is especially important for enterprise-scale teams, where talent and domain knowledge are wildly varied.

# Culture Shift

We've discussed data patterns as a method for parallelizing the development of the transformation layer, eliminating the bottlenecks of a centralized data team. Distributing tasks according to technical prowess will allow your team to deliver more value by enabling the *entire organization* to contribute to developing data infrastructure. Column-level metadata begets this process by providing context *at the finest grain possible*. This reduces the likelihood of data silos and swamps. Our assumption is that your organization will *adopt and thrive* in this environment. For that to be true, however, a culture shift is necessary.

## Democratizing Data Transformation

The shift toward data democratization is one from activity-oriented to outcome-oriented teams. Popularized by Martin Fowler, the idea is that teams need to focus on *outcomes*, not *activities*. Our framework for automating data transformation is a new and radical

approach that starts from first principles of process design and eliminates inefficiency.

The shift here is *away* from a monostructural manufacturing-line type of assembly, where analysts and engineers alike are ultraspecialized. Instead, we embrace the liberalization of the data process, a microservices approach where a junior analyst might realize they need a piece of data and have the ability to implement it with only a brief code review. Rather than struggling with a non-optimal GUI, waiting for a ticket to be filled by the data team, or meeting with a manager to walk through complex SQL, an automated transformation layer will enable teams to *create value* almost immediately through a top-down DAaaS approach. In theory, this sounds perfect—in practice, it requires a shift in mindset.

Every member of the organization will need to think *a bit more critically* to truly understand their data—both in context and how to manipulate patterns to achieve a desired result. Only a few will need to build patterns, but *most* will use them to access information. All will need to think toward *outcomes*, not *activities*. For some, this might be easy, but for many this will necessitate a shift in approach. With recent advancements in artificial intelligence (AI) and generative content, many—especially creative types—have become skeptical of processes that automate away large swaths of work. We expect similar pushback on the automation of the transformation layer, especially from those who've made careers of the work being automated. A common fear is that more automation means less work to do, which can sound scary.

The truth about automation, repeated throughout history, is that it does not eliminate work. It merely shifts the type of work we do, often to something more challenging and enjoyable. If it now takes three analysts to build infrastructure instead of ten, the other seven will still have plenty to do—focusing on higher-level tasks like implementing new patterns, optimizing design, and tackling new *creative* problems that were previously consumed by manual process and day-to-day operations.

In the same way that rote data engineering processes can be automated and integrated into the transformation layer, this does not result in fewer data engineers but ultimately *more value creation* from the existing team, which drives business success and creates *more jobs* in the long term. For example, take Fivetran's automation

of data ingestion: once a core responsibility of data engineers, the ingestion problem has been largely solved, yet demand for data and data teams has only increased exponentially. The same will be true for the transformation layer.

Automation, then, is not about creating a smaller team, but rather one that achieves at a higher standard. Getting more done begets success and a *larger* team capable of achieving tremendous results.

## Implementation

As we think about building discoverable, understandable, trustworthy, and explorable data systems, there are some common pitfalls that can create undue work. We've discussed our ideal framework—a DAaaS solution built around a hybrid transformation approach—but every team's data journey will differ. Following are some guiding principles for your team to consider as you implement and scale your data systems:

*Simplicity in architecture design*
: The ideal data infrastructure is simple but robust. Good systems should *reduce* complexity—they take time to implement and maintain, but this trade-off is worth it for the gains in productivity.

*Reliable, scalable, maintainable*
: As espoused in the book *Designing Data-Intensive Applications* by Martin Kleppmann (O'Reilly), functional systems are built on reliability, scalability, and maintainability. In addition to being simple, a system should:

  - Work correctly, even under challenging conditions (reliable)
  - Be able to handle growth in volume or complexity (scalable)
  - Be easy to maintain and adaptable to new functionality (maintainable)

  This ties to our discussion and promotion of cloud native, end-to-end solutions. While many of these carry a high sticker price, the management cost is much lower, the reliability of cloud infrastructure is nearly unmatched, and a lot of maintenance is abstracted away:

*Total cost of ownership versus price*
: We briefly discussed how time and resource savings can quickly compound, but it's worth reiterating. When implementing a solution, spend a good deal of resources considering the cost not only to implement but also maintain that solution. This should be broken down by time, labor, and the actual cost of the product being implemented. The cost of *changing* a solution must also be considered, as the only constant in software (and especially in data) is change.

*Alignment with stakeholders*
: While data is ideally treated as a product, data teams do not operate independently from the rest of an organization. Furthermore, conflicts of interest and bureaucracy can muddy initiatives and result in shifting priorities. Before embarking on any project, ensuring stakeholder alignment is critical. Far too often, we have seen well-intentioned work miss the mark due to a lack of communication. While this is costly for an individual contributor, it can be more consequential at the managerial level. Be sure that projects are aligned to stakeholder goals and the initiatives undertaken are in the interest of delivering value.

# Summary

We've stepped beyond the modern data stack to examine what it means to deliver value with data: from the importance of column awareness to an entirely new method of infrastructure development (DAaaS). The building blocks of these concepts are not new: breaking down technical barriers and standardizing the design and distribution of infrastructure, asynchronous distributed work, and the automation of rote processes have led to many advancements across data and software—we feel it's time these were brought to data transformation.

Optimizing the modern data stack for value is not simply intelligent software engineering but also a practice of social engineering. It involves building a cohesive system of tools and components and requires leadership and effective management according to proven frameworks. To deliver value, data teams need to implement performant solutions *and* champion their company-wide adoption. The modern data stack is built to solve the first, more technical, problem, but only *you* can solve the second. We believe the future

of data transformation will start with a column-aware DAaaS foundation, atop which confident data practitioners and leaders, like yourselves, can champion systems of tremendous value and help organizations win.

# CHAPTER 4
# Summary and Further Reading

In our report, we sought to tie transformation to the act of creating value from data. Underlying this daunting task is a set of prerequisite organizational characteristics: thinking about "data as a product" (a core tenet of data mesh); building a data-driven culture; enabling DAaaS; and building data from a column-aware, metadata-first framework.

While a number of guiding principles and prevailing philosophies help the data practitioner in their journey, every situation is unique, demanding a bespoke solution. It's our hope that this report can help your organization achieve efficiency and automation in transformation, unlocking the full value of the modern data stack and providing a lens into the past and a vision for the future of data transformation.

To further your understanding, we highly recommend the following as further reading:

*Data Mesh* by Zhamak Dehghani (O'Reilly, 2022)
: This book was revolutionary in its introduction to the DaaP concept, and Zhamak Dehghani's approach to a decentralized data team of the future is applicable to most data teams. For those looking to build out their organization *or* restructure, *Data Mesh* is a must-read.

*Data Pipelines Pocket Reference* by James Densmore (O'Reilly, 2021)

A down-to-earth manual on how to solve common data problems in the pipeline step, this text walks through pipelines—from definition to implementation—with considerations for maintenance, testing, and alerting. This text is highly recommended for readers as they build out their ETL systems.

*The Data Warehouse Toolkit* by Ralph Kimball and Margy Ross (Third Edition, Wiley, 2013)

Penned by data warehousing legend Ralph Kimball, this book is an exhaustive guide to dimensional modeling. Though the book was first published in 1996, there have been a number of revisions, and it remains one of the foremost texts on data warehousing convention. We recommend interleaving Kimball's ideas with your own to account for how the data warehouse has evolved.

*Designing Data-Intensive Applications* by Martin Kleppmann (O'Reilly, 2017)

This is a practical and comprehensive guide to solving problems in the data space. Martin Kleppmann provides various technologies and the pros and cons of each, allowing the reader to make logical, informed decisions about the technologies they choose to implement.

*Fundamentals of Data Engineering* by Joe Reis and Matt Housley (O'Reilly, 2022)

Covering the data engineering lifecycle from start to finish, Joe Reis and Matt Housley map out best practices, technologies, and undercurrents in the space. Their approach introduces principles that have stood the test of time and encompass all relevant technologies. Readers will walk away with an understanding of how to apply data engineering patterns to real-world problems.

## About the Authors

**Satish Jayanthi** is CTO and cofounder of Coalesce. Prior to that, he was senior solutions architect at WhereScape, where he met his cofounder, Armon.

**Armon Petrossian** is CEO and cofounder of Coalesce. Previously, he was part of the founding team at WhereScape in North America, where he served as national sales manager for almost a decade.