O'REILLY®
Technical Guide

Understanding ETL

Data Pipelines for Modern Data Architectures

Early Release
RAW & UNEDITED

Compliments of
databricks

Matt Palmer

Databricks

databricks

Delta Live Tables

Reliable data pipelines made easy

Delta Live Tables (DLT) is the first ETL framework that uses a simple declarative approach to building reliable data pipelines. DLT automatically manages your infrastructure at scale so data analysts and engineers can spend less time on tooling and focus on getting value from data.

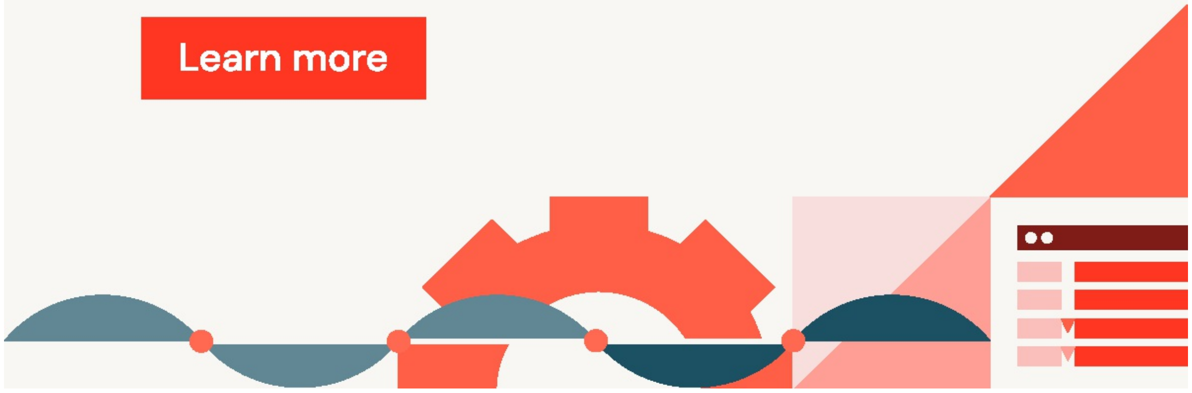
Accelerate ETL Development

Automatically manage your infrastructure

Have confidence in your data

Simplify batch and streaming

Learn more



Understanding ETL

Data Pipelines for Modern Data Architectures

Matt Palmer

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

Understanding ETL

by Matt Palmer

Copyright © 2024 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

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

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Gary O'Brien and Aaron Black

Production Editor: Kristen Brown

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

April 2024: First Edition

Revision History for the Early Release

- 2023-10-23: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781098159252 for release details.

978-1-098-15925-2

[LSI]

Preface

The Bread and Butter of Data Engineering

If you are a Data Engineer or any sort of practitioner in the data space (looking at you, Data Scientists and Data Analysts!), you might be familiar with the term *ETL* (or at least heard it being tossed around). ETL stands for "Extract Transform Load," and the term speaks to the most foundational process Data Engineers are tasked with implementing–bringing data into a system and making it ready and available to be used in any variety of ways. Need to build a dashboard that helps business leaders understand marketing data from the past year? You need ETL. Want to use a large data set to train the next iteration of a machine learning model? You need ETL. Need to organize data to be securely stored and archived to meet compliance requirements? You guessed it, you need ETL. Really, anytime you want to do *anything* with data, you most likely need a reliable process or pipeline (leaning into the common metaphor of data engineers as the plumbers of the data world). This fundamental truth holds true to the classic workloads of business intelligence (BI) and analytics and the most cutting-edge new advances in the space.

The Brave New World of AI

The data world has seen many trends come and go; some have transformed the space, and some have turned out to be short-lived fads. The most recent trend is, without a doubt, generative AI. It seems like someone is talking about AI, large language models (LLMs), or chatbots everywhere you look. This recent fascination with AI, largely brought by the release of OpenAI's ChatGPT, extends beyond the media's interest and among researchers - it is now seen by many as a strategic investment companies must make or get left behind. In many cases, Large Language Models' (LLMs) great potential is found in a company's proprietary data or expertise. If used to train LLMs, this data can unlock business value in exciting ways. When looking at what that data looks like, where it is coming from, how it is being generated, and where it is - it very quickly becomes clear that to even begin utilizing it for AI, it needs to be moved, made ready, and consolidated in a single location or, in other words *extracted*, *transformed*, *and loaded*. Yes, you guessed it, even the most exciting thing happening in the tech world today relies, first and foremost, on ETL.

A Changing Data Landscape

Beyond the most recent trend of generative AI, other trends have changed the data landscape in the past decade. One example is the growing dominance of streaming data. Today, companies are generating vast amounts of data in real-time from sensors, websites, mobile applications, and more. This reality demands that data be ingested and processed in real time so it can be used for real-time decision-making. Data engineers must now go beyond batch processing and build and manage continuous running pipelines that can handle vast amounts of streaming data.

Another example is the emergence of data lakehouse architectures. The data lakehouse is a new paradigm (first introduced in this paper) that aims to unify data warehouses and data lakes. Using new storage technologies like Delta Lake that add reliability and performance to data lakes, the lakehouse brings the best of both worlds–the cheap and scalable data storage found in data lakes with the high-performant transactions of a data warehouse. This unification allows running both AI workloads (usually done by data scientists on data lakes) alongside analytics workloads (usually done by data analysts on data warehouses) and removing the complexity associated with redundant dual architectures, inconsistent data governance, and data duplication.

Even though ETL is a term that has been around for a while and is still relevant as ever, you should revisit it and see how you can implement it in the modern data landscape. How can ETL tackle both batch and streaming data ingestion and processing? How can it be implemented in a data lakehouse architecture? This guide will shed light on these questions and help you, the reader, understand ETL in the context of recent trends.

What About ELT (and Other Flavors)?

Beyond ETL, if you have also encountered a term like ELT–fear not! This is not a typo. ELT is a permutation of the Extract Transform Load process, which differs from ETL in the order of Load and Transform.

Since many of the considerations and principles discussed in this guide are common to both ETL and ELT (as well as to other permutations such as reverse ETL) we will be using the term ETL moving forward as a general term to describe all of these similar processes. Even if the ETL does not accurately describe the process you are working to implement, understanding data ingestion, transformation, and orchestration is still important, as are best practices around observability, troubleshooting, scaling, and optimizations. Therefore, we encourage you to read on and hope you find this guide useful.

Chapter 1. Data Ingestion

Data ingestion, in essence, involves transferring data from a source to a designated target. Its primary aim is to usher data into an environment primed for staging, processing, analysis, & AI/ML. While massive orgs may focus on moving data internally (among teams), for most of us, data ingestion emphasizes pulling from external sources and directing it to in-house targets.

In an era where data holds central importance in both business and product development, the significance of accurate and timely data cannot be overstated. This heightened reliance on data has given rise to a multitude of “sources” from which teams extract information to refine decision-making processes, craft outstanding products, and a multitude of other actions. For instance, a marketing team would need to retrieve data from several advertising and analytics platforms, such as Meta, Google (including Ads and Analytics), Snapchat, LinkedIn, Mailchimp, and more.

However, as time marches on, APIs and data sources undergo modifications. Columns might be introduced or removed, fields could get renamed, and new versions might replace outdated ones. Handling changes from a single source might be feasible, but what about juggling alterations from multiple sources

—five, ten, or even a hundred? The pressing challenge is: "How can a budding data team efficiently handle these diverse sources in a consistent and expandable way?" As data engineers, how do we ensure our reputation for providing reliable and straightforward data access, especially when every department's demands are continuously escalating?

Data Ingestion : Now vs. Then

While much has changed in the past decade about data ingestion, the two largest shifts affecting ingestion result from the volume and velocity of our sources.

We've had multiple industry changes to accommodate this— movement to the cloud, the warehouse to the data lake to the lakehouse, and the simplification of streaming technologies, to name a few. This has been manifested as a shift from extract-transform-load workflows to extract-load-transform, the key difference being that *all* data is now loaded into a target system.

We refrain from being too pedantic about the terms ETL and ELT, however we'd like to emphasize that almost every modern data engineering workflow will involve staging almost all data in the cloud. The notable exception is cases where hundreds of trillions of rows of highly-granular data are processed daily (i.e., IoT or sensor data) where it makes sense to aggregate or discard data before staging.

Despite constant changes, the fundamental truth of extraction is that data is pulled from a source and written to a target. Hence, the discussion around extraction must be centered on precisely that.

Sources and Targets

While most associate ingestion with *Extraction*, it's also tightly coupled with *Loading*, after all, every source requires a destination. In this guide, we're working on the premise that you have an established warehouse or data lake — storage will not be a primary topic in this chapter. So, while storage takes a backseat, we'll still shine a light on best practices for staging and highlight the hallmarks of ideal storage implementations.

It's our mission to arm you with a toolkit for architecture design, keeping in mind that a "perfect" solution might not exist. We'll navigate a framework for appraising sources and untangling the unique knots of data ingestion. Our high-level approach is designed to give you a birds-eye view of the landscape where you can make informed, appropriate decisions.

The Source

When ingesting data, our primary consideration should be the characteristics of the data source— as we've alluded, you will *likely* have more than one data source (if not, congrats, your life just got much easier), so each must be weighted separately.

With the sheer volume of data sources and the nature of business requirements (I've seldom been asked to *remove* sources, but adding sources is just another Thursday), it's *highly* likely that you'll encounter one or many sources that do not fit into a single solution.

As in the ebb and flow of life, data engineering dances between the need for adaptability, embracing shifting requirements and unconventional sources, and precision. While building trust takes weeks, months, and years, it can be lost in a day. So, what's important in choosing a source?

Examining sources

As a practical guide, we take the approach of presenting time-tested questions that will guide you towards *understanding* the source data, both its characteristics and how the business will get value.

We recommend taking a highly critical stance: it is always possible that *source data is not needed* or *a different source* will better suit the business. You are your organization's data expert, and it's your job to check and double check assumptions. It's normal to bias action and complexity, but imperative we consider *essentialism* and *simplicity*.

When examining sources, keep in mind that you'll likely be working with software engineers on upstream data, but downstream considerations are just as important. Neglecting these can be highly costly, since your error may not manifest itself until weeks of work have taken place.

Here are some questions to ask:

Who will we work with?

> In an age of artificial intelligence, we prioritize real intelligence. The most important part of any data pipeline is *the people* it will serve. Who are the stakeholders involved? What are *their* primary motives— OKRs (objectives and key results) or organizational mandates can be useful for aligning incentives and moving projects along quickly.

How will the data be used?

> Closely tied to "who," *how* the data will be used should largely guide subsequent decisions. This is a way for us to check our stakeholder requirements and learn the "problem behind the problem" that our stakeholders are trying to solve. We highly recommend a list of technical yes/no requirements to avoid ambiguity.

What's the frequency?

> As we'll discuss in further detail later, most practitioners immediately jump to batch vs. streaming. Any data can be processed as a batch or stream, but we are commonly referring to the characteristics of data that we would like to stream. We advocate first considering if the data is *bounded* or *unbounded*, i.e. does it *end*, for example the 2020 Census ACS dataset, or is it continuous, i.e. log data from a fiber cabinet.
>
> After bounds are considered, the minimum frequency available sets a hard limit for how often we can pull from the source. If an API only updates daily, there's a hard limit on the

frequency of your reporting. Bounds, velocity, and business requirements will inform the frequency at which we *choose* to extract data.

What is the expected data volume?

Data volume is no longer a limiter for the ability to store data — after all, “storage is cheap,'' and while compute can be costly, it’s less expensive than ever (by a factor of millions). However, volume closely informs how we choose to write and process our data and the scalability of our desired solution.

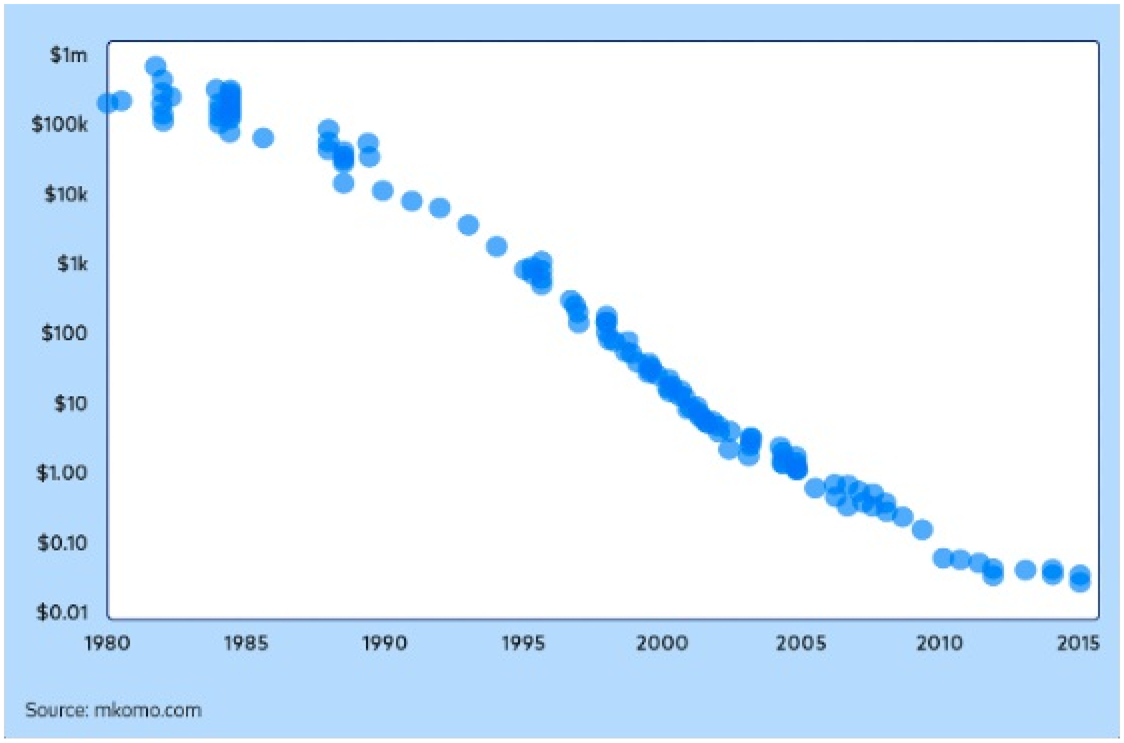


Figure 1-1. Hard drive costs per GB, 1980 to 2015

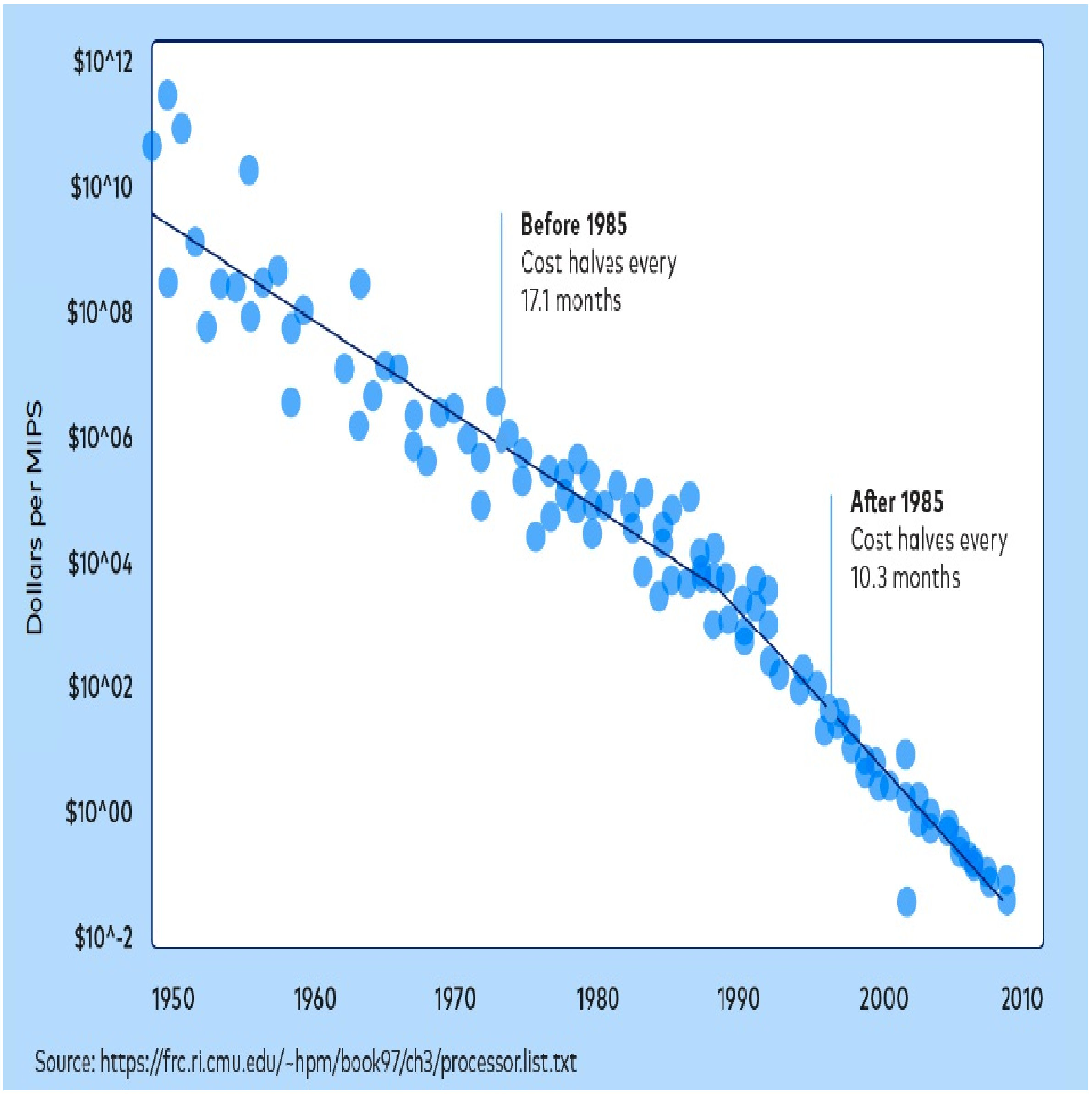


Figure 1-2. Cost of compute, millions of instructions per second (MIPS)

What's the format?

While we will eventually choose a format for storage, the input format is an important consideration. How is the data being delivered? Is it via a JSON payload over an API? Perhaps a FTP server? If you're lucky, it already lives in a relational database somewhere. What does the schema look like? Is there a schema? The endless number of data formats keeps our livelihoods interesting, but also presents a challenge.

What’s the quality?

> The quality of the dataset will largely determine if any transformation is necessary. As data engineers, it’s our job to ensure consistent datasets for our users. Data might need to be heavily processed or even enriched from external sources to supplement missing characteristics.

We’ll use these characteristics to answer our final question:

How will the data be stored?

> As we mentioned, this report assumes some fixed destination for your data. Even then, there are a few key considerations in data storage: to stage or not to stage (is it really a question?), business requirements, and stakeholder fit are the most important.

Source Checklist

For every source you encounter, consider these guiding questions. Though it might seem daunting as the number of sources piles up, remember: this isn’t a writing task. It’s a framework to unpack the challenges of each source, helping you sketch out apt solutions that hit your targets.

While it might feel repetitive, this groundwork is a long-term time and resource saver.

Question	Note
Who will we collaborate with?	Engineering (Payments)
How will the data be used?	Financial reporting & quarterly strategizing
Are there multiple	Yes

sources?	
What’s the format?	Semi-structured APIs (Stripe & Internal)
What’s the frequency?	Hourly
What’s the volume?	Approximately 1k new rows/day, with an existing pool of ~100k
What processing is required?	Data tidying, such as column renaming, and enrichment from supplementary sources
How will the data be stored?	Storing staged data in Delta Tables via Databricks

The Destination

While end-to-end systems require considerations that are just that, we assume most readers will be tasked with building a pipeline into an *existing* system where the storage technology is already chosen.

Choosing data storage technology is beyond the scope of what we are focusing on in this guide but we will give a brief consideration of destinations, as they are *highly* important for the total value created by a data system. Thus, when analyzing (or considering) a destination, we recommend making use of a similar checklist to that of a source. Usually, there are far fewer destinations than sources, so this should be a much simpler exercise.

Examining Destinations

A key differentiator in destinations is that the *stakeholder* be prioritized. “Destinations have a unique trait: they pivot around stakeholders. These destinations either directly fuel BI, analytics, and AI/ML applications, or indirectly power them when dealing with staged data, not to mention user-oriented apps. Though we recommend the same checklist, frame it slightly towards the stakeholder to be sure it meets their requirements, while working towards engineering goals.

We fully recognize *this is not always possible*. As an engineer, your role is to craft the most fitting solution, even if it means settling on a middle ground or admitting there’s no clear-cut answer. Despite technology’s incredible strides, certain logical dilemmas do not have straightforward solutions.

Staging Ingested Data

We advocate for a data lake approach to data ingestion. This entails ingesting most data into Cloud Storage systems, such as S3, GCP, or Azure, before loading it into a data warehouse for analysis. One step further is a lakehouse — leveraging metadata management systems like Unity Catalog and Delta to bring structure to an otherwise turbulent sea of data and, with it, the ability to perform most analysis without the need for a warehouse

A prevailing and effective practice for staging data is utilizing metadata-centric parquet-based file formats, including Delta, Iceberg, or Hudi. Grounded in Parquet—a compressed, columnar format designed for large datasets—these formats incorporate a metadata layer, offering features such as time travel, ACID compliance, and more.

Integrating these formats with the medallion architecture, which processes staged data in three distinct quality layers, ensures the preservation of the entire data history. This facilitates adding new columns, retrieving lost data, and backfilling of historical data.

The nuances of the medallion architecture will be elaborated upon in our chapter dedicated to data transformation. For the current discussion, it’s pertinent to consider the viability of directing all data to a ’staging layer’ within your chosen Cloud Storage provider.

Change Data Capture (CDC)

Change Data Capture (CDC) is a design pattern in data engineering that captures and tracks changes in source databases to update downstream systems. Rather than batch-loading entire databases, CDC transfers only the changed data, optimizing both speed and resource usage.

This technique is crucial for real-time analytics and data warehousing, as it ensures that data in the target systems is current and in sync with the source. By enabling incremental updates, CDC enhances data availability and consistency across the data pipeline. Put simply by Joe Reis and Matt Housely in *Fundamentals of Data Engineering*, “CDC… is the process of ingesting changes from a source database system.”

CDC becomes important in analytics & engineering patterns— like creating SCD Type 1 & 2 tables, a process that can be unnecessarily complex and time consuming. Choosing platforms or solutions that natively support CDC can expedite common tasks, letting you focus on what matters most. One example is Delta Live Tables (DLT) on Databricks, which provides native support for SCD Type 1 & 2, in both batch & streaming pipelines.

Destination Checklist

Here’s a short checklist for questions to consider when selecting a *destination*:

Question	**Note**
Who will we collaborate with?	Marketing - Analytics
How will the data be used?	Financial reporting and quarterly planning
Are there multiple destinations?	Data is first staged in GCP then written to BQ.
What’s the format?	Parquet in GCP. Structured/semi-structured in BQ.
What’s the frequency?	Batch
What’s the volume?	~1k rows/day new data, ~100k existing

What processing is required?	Downstream processing using dataform/dbt
How will the data be stored?	Source tables in Google BigQuery

Ingestion Considerations

In this section, we outline pivotal data characteristics. While this list isn't exhaustive and is influenced by specific contexts, aspects like *frequency, volume*, *format, and processing* emerge as primary concerns.

Frequency

We already mentioned that the first consideration in frequency should be bounds, i.e. is the data set *bounded* or *unbounded*. Bounds and business needs will dictate a frequency for data ingestion— either in batch or streaming formats.

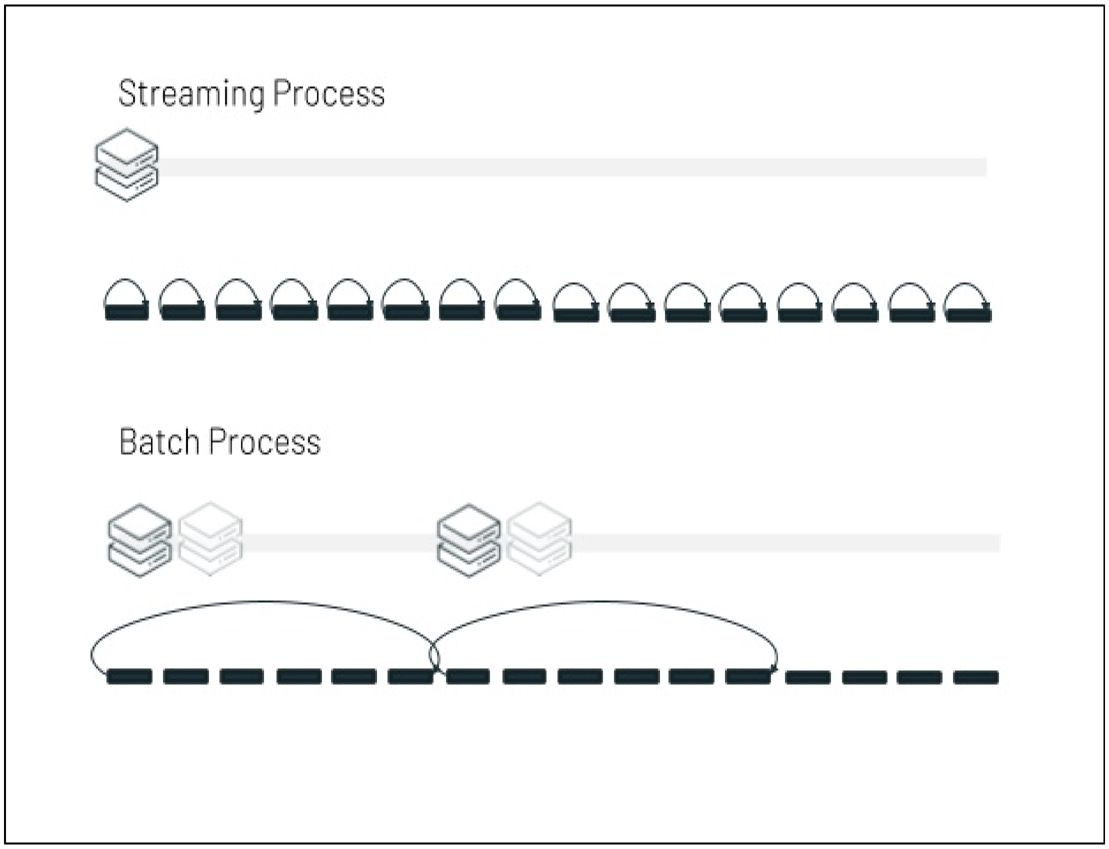


Figure 1-3. Latency is the property which defines “batch” or “streaming” processes. Past some arbitrary latency threshold, we consider data “streamed.”

We’ll present batch, micro-batch, and streaming along with our thoughts to help you select the most appropriate frequency and a compatible ingestion solution.

Batch

Batch processing is the act of processing data in *batches* rather than all at once. Like a *for loop* that iterates over a source, batch simply involves either chunking a bounded dataset and processing each component *or* processing unbounded datasets as data arrives.

Micro Batch

Micro-batch is simply “turning the dial down” on batch processing. If a typical batch ingestion operates daily, a micro-batch might function hourly or even by the minute. Of course you might say “Matt, at what point is this just pedantics? A micro-batch pipeline with 100ms latency seems a lot like streaming to me.”

We agree!

For the rest of this chapter (and report), we’ll refer to low-latency micro-batch solutions as streaming solutions— the most obvious being Spark Structured Streaming (SSS). While “technically” micro-batch, latency in the hundreds of milliseconds makes SSS effectively a real-time solution.

Streaming

Streaming refers to the continuous reading of datasets, either bounded or unbounded, as they are generated.

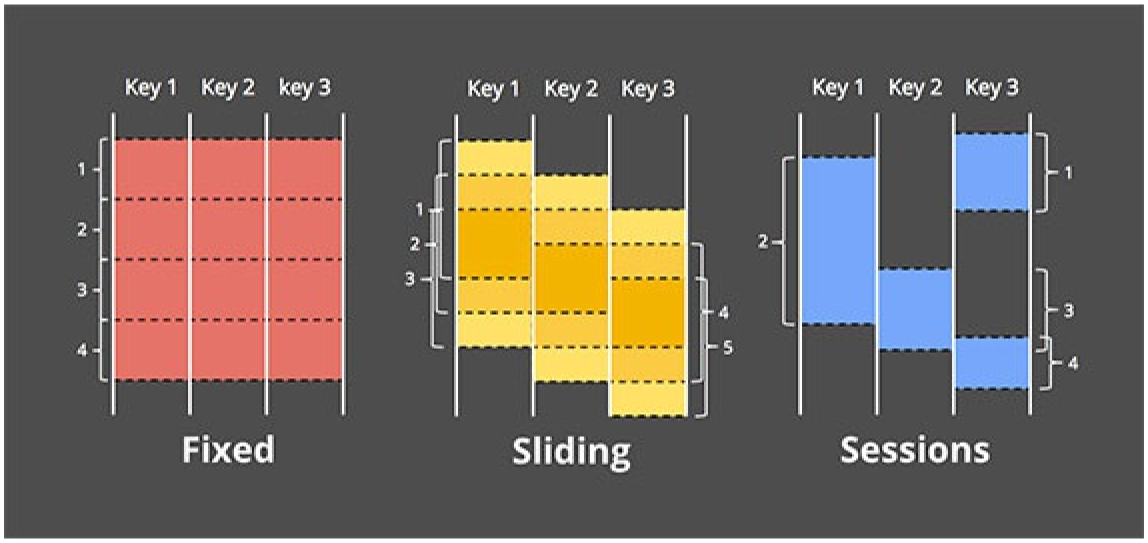


Figure 1-4. [Caption to come]

While we will not discuss streaming in great detail, it does warrant further research. For a comprehensive understanding of streaming data sources, we suggest exploring resources like *Streaming 101*, *Streaming Databases,* and, of course, *The Fundamentals of Data Engineering*.

Methods

Common methods of streaming unbounded data include:

Windowing

Segmenting a data source into finite chunks based on temporal boundaries.

Fixed windows

Data is essentially “micro-batched” and read in small fixed windows to a target.

Sliding windows

Similar to fixed windows, but with overlapping boundaries.

Sessions

Dynamic windows in which sequences of events are separated by gaps of inactivity— in sessions, the “window” is defined by the data itself.

Time-agnostic

Suitable for data where time isn’t crucial, often utilizing batch workloads.

It’s crucial to differentiate between the *actual* event time and the processing time, since discrepancies often arise.

Message Services

When we say “message services,” we refer to “transportation layers” or systems for communicating and transporting streaming data.

Kafka

Originated at LinkedIn in 2011, Apache Kafka started as a message queue system but quickly evolved into a distributed streaming platform. While Kafka’s design allows for high throughput and scalability, its inherent complexity remains a hurdle for many.

Redpanda

> Developed as an alternative to Kafka, Redpanda boasts similar performance with a simplified configuration and setup. Redpanda is based on C++ rather than Java and compatible with Kafka APIs.

Stream Processing Engines

Stream processing is about analyzing and acting on real-time data (streams). Given Kafka's longevity, the three most popular and well known stream processing tools are:

Flink

> An open-source engine that continuously processes both unbounded and bounded data sets with minimal downtime. Apache Flink ensures low latency through in-memory computations, offers high availability by eliminating single points of failure, and scales horizontally.

Spark Structured Streaming

> An arm of the Apache Spark ecosystem, designed to handle real-time data processing. It brings the familiarity and power of Spark's DataFrame and DataSet APIs to streaming data. Spark might be an attractive option given the popularity of Apache Spark in data processing and ubiquity of the engine in tools like Databricks.

Kafka Streams

> A library built on Kafka that provides stateful processing capabilities, but ties to Java can be limiting.

Simplifying Stream Processing

Several relatively new solutions simplify stream processing by offering straightforward clients, with a focus on performance and simple development cycles.

Managed Platforms

> Taking Databricks as an example: leveraging tools like Delta Live Tables or simply running Spark Streaming jobs on the Databricks runtime can be a powerful abstraction of complexity and drastically simplify the process to building streaming systems.

Confluent-Kafka

> An attempt to bring Kafka capabilities to Python, although it remains rudimentary compared to its Java counterpart. Confluent-Kafka is simply a client-library in the same way psycopg2 is a Postgres client library.

Bytewax

> A library that aims to bridge the gap by offering a more intuitive, Pythonic way of dealing with stream processing, making it more accessible to a wider range of developers. Built on Rust, Bytewax is highly performant, simpler than Flink, and boasts shorter feedback-loops and easy deployment/scalability.

Still newer tools that seek to unify stream processing— like Apache Beam or Estuary Flow—or combine stream processing directly with databases (streaming databases) are growing in popularity. We recommend *Streaming Systems* and *Streaming Databases* for an in-depth look.

The streaming landscape, while complex, has seen strides in simplification and user-friendliness, especially when considering managed platforms, like Databricks, and low-latency micro-batch solutions, like Spark Structured Streaming.

While many think of real time data as the ultimate goal, we emphasize a “right-time” approach. As with any solution, latency can be optimized infinitely, but the cost of the solution (and complexity) will increase proportionally. Most will find going from daily or semi-daily data to hourly/sub-hourly data a perfectly acceptable solution.

Payload

Volume

Volume is a pivotal factor in data ingestion decisions, influencing the scalability of both processing and storage solutions. When assessing data volume, be sure to consider factors like:

Cost

> Higher volumes often lead to increased costs, both in terms of storage and compute resources. Make sure to align the cost factor with your budget and project needs— that includes storage/staging costs associated with warehouses, lakes, or lakehouses, depending on your solution.

Latency

> Depending on whether you need real-time, near-real-time, or batch data ingestion, latency can be a critical factor. Real-time processing requires more resources and, thus, may cost more.

Throughput/Scalability

> It’s essential to know the ingestion tool’s ability to handle the sheer volume of incoming data. If the data source generates large amounts of data, the ingestion tool should be capable of ingesting that data without causing bottlenecks

Retention

> With high volumes, data retention policies become more important. You’ll need a strategy to age out old data or move it

to cheaper, long-term storage solutions.

For handling sizable datasets, using compressed formats like Avro or Parquet is crucial. Each offers distinct advantages and constraints, especially regarding schema evolution.

Structure and Shape

Data varies widely in form and structure, ranging from neat, relational “structured” data to more free-form “unstructured” data. Importantly, structure doesn’t equate to quality; it merely signifies the presence of a schema.

In today’s AI-driven landscape, unstructured data’s value is soaring as advancements in large-language and machine learning models enable us to mine rich insights from such data. Despite this, humans have a penchant for structured data when it comes to in-depth analysis, a fact underscored by the enduring popularity of SQL—*Structured* Query Language—a staple in data analytics for nearly half a century.

Unstructured

As we’ve alluded, unstructured data is data without any predefined schema or structure. Most often, that’s represented as text, but other forms of media represent unstructured data, too. Video, audio, and imagery all have elements that may be analyzed numerically. Unstructured data might be text from a Pierce Brown novel:

> *“A man thinks he can fly, but he is afraid to jump. A poor friend pushes him from behind.” He looks up at me. “A good friend jumps with.”*

Such data often feeds into machine learning or AI applications, underlining the need to understand stakeholder requirements comprehensively. Given the complexity of machine learning, it’s vital to grasp how this unstructured data will be utilized before ingesting it. Metrics like text length or uncompressed size may serve as measures of shape.

Semi-structured

Semi-structured data lies somewhere between structured and unstructured data— XML and JSON are two popular formats. As data platforms continue to mature, so too has the ability to process and analyze semi-structured data *directly*.

The following snippet shows how to parse semi-structured JSON in Google BigQuery to pivot a list into rows of data:

```
WITH j_data AS (
    SELECT
        (
        JSON '{"friends": ["steph", "julie", "thomas", "tommy",
"michelle", "tori", “larry”]}'
        ) AS my_friends_json
        ), l_data AS (
    SELECT
        JSON_EXTRACT_ARRAY(
            JSON_QUERY(j_data.my_friends_json, "$.friends"), '$'
        ) as my_friends_list
    FROM j_data
)
    SELECT
        my_friends
    FROM l_data, UNNEST(l_data.my_friends_list) as my_friends
    ORDER BY RAND()
```

In some situations, moving data processing downstream to the analytics layer is worthwhile— it allows analysts and analytics engineers greater flexibility in how they query and store data. Semi-structured data in a warehouse (or accessed via external tables) allows for the flexibility of changing schemas or missing data while getting all the benefits of tabular data manipulation and SQL.

Still, we must be careful to ensure this data is properly validated to be sure missing data isn’t causing errors. We’ve seen many invalid queries due to improper consideration of NULL data. We’ll discuss this more in Chapter 5 on scalability.

Describing the shape of JSON frequently involves discussing keys, values, and the number of nested elements. We highly recommend tools like https://jsonlint.com and https://jsoncrack.com/ for this purpose, VSCode extensions also exist to validate and format JSON/XML data.

Structured

The golden "ideal" data, structured sources are neatly organized with *fixed* schemas and unchanging keys. For over 50 years, SQL has been the language of choice for querying structured data. When storing structured data, we frequently concern ourselves with the number of columns and length of the table (in rows). These characteristics inform our use of materialization, incremental builds, and, in aggregate, an OLAP vs. OLTP database (column-/row-oriented).

Though much data today lacks structure, we still find SQL to be the dominant tool for analysis. Will this change? Possibly, but as we showed, it's more likely that SQL will simply adapt to accommodate semi-structured formats. Though language-based querying tools have started appearing with AI advancements, SQL is often an intermediary. If SQL disappears from data analysis, it will likely live on as an API.

OLAP VS. OLTP DATABASES

The biggest choice in data warehousing is whether to use a cloud-native database or a cloud-hosted traditional database—the main difference being the distributed nature and column-oriented architecture of cloud-native solutions. These are more commonly referred to as OLAP (Online Analytical Processing) and OLTP (Online Transactional Processing).

- OLAP systems are designed to process large amounts of data quickly. This is commonly accomplished via distributed processing and a column-oriented architecture. Newer, cloud-native databases are OLAP systems: the big three OLAP solutions are Amazon Redshift, Google BigQuery, and Snowflake.
- OLTP systems are engineered to handle large amounts of transactional data originating from multiple users. This usually takes the form of a row-oriented database. Many traditional database systems are OLTP: Postgres, MySQL, etc.

OLAP systems are most commonly used by analytics and data science teams for their speed, stability, and low maintenance cost. Here are some considerations for data warehouse selection:

Computing platform of choice

The integration of these technologies largely depends on the rest of your tech stack. For example, if *every* tool in your org lives in the Amazon ecosystem, Redshift might be more cost effective and simple to implement than Google BigQuery.

Functionality

Snowflake made a name for themselves by being the *first* to support the separation of storage and compute. BigQuery has great support for semi-structured data and ML operations. Redshift Spectrum has proven to be a useful tool for creating external tables from data in S3. Every warehouse has strengths and weaknesses, these need to be evaluated according to your team's use-case.

Cost

Pricing structures vary wildly across platforms. Unfortunately, pricing can also be opaque— in most cases, you won't *truly* know how a database will be used until you're on the platform. The goal with pricing should be to understand *how to use a database to minimize cost* and what the cost might be if that route is taken. For example, in cases where cost is directly tied to the amount of data scanned (BigQuery), intelligent partitioning and query filtering can go a long way. There will be no direct answer on cost, but research is worthwhile, as warehouses can be expensive, especially at scale.

Format

In what format is the source data? While JSON and CSV are common choices, an infinite number of format considerations can arise. For instance, some older SFTP/FTP transfers might arrive compressed, necessitating an extra extraction step.

The data format often dictates the processing requirements and the available solutions. While a tool like Airbyte might seamlessly integrate with a CSV source, it could stumble with a custom compression method or a quirky Windows encoding (believe us, it happens).

If at all possible, we advise opting for familiar, popular data formats. Like repairing a vehicle, the more popular the format, the easier it will be to find resources, libraries, and instructions. Still, in our experience it’s a rite of passage to grapple with a perplexing format, but that’s part of what makes our jobs fun!

Variety

It’s highly likely you’ll be dealing with multiple sources and thus varying payloads. Data variety plays a large role in choosing your ingestion solution — it must not only be capable of handling disparate data types but also be flexible enough to adapt to schema changes and varying formats. Variety makes governance and observability particularly challenging, something we’ll discuss in Chapter 5.

Failing to account for data variety can result in bottlenecks, increased latency, and a haphazard pipeline, compromising the integrity and usefulness of the ingested data.

Choosing a Solution

The best tools for your team will be the ones that support your sources and targets. Given the unique data requirements of each organization, your choices will be context-specific. Still, we can’t stress enough the importance of tapping into the knowledge of mentors, peers, and industry experts. While conferences can be pricey, their knowledge yield can be priceless. For those on a tight budget, data communities can be invaluable. Look for them on platforms like Slack, LinkedIn, and through industry newsletters

When considering an ingestion solution, we think in terms of *general* and *solution-specific* considerations— the former applying to all tools we'll consider, while the latter is specific to the class of tooling.

- *General considerations:* extensibility, the cost to build, the cost to maintain, and switching costs. For a deep-dive on general solutions, we highly recommend Meltano's 5-day data integration guide.

Solution-specific considerations are dependent on the class of tooling, which commonly takes one of two forms:

- *Declarative solutions* dictate outcomes, i.e. I leverage Databricks to natively ingest data from my cloud storage provider or I create a new Airbyte connection via a UI.
- *Imperative solutions* dictate actions. For example, I build a Lambda that calls the Stripe API, encodes/decodes data, and incrementally loads it to Snowflake.

Each of these solutions has its pros and cons, we'll briefly discuss each and present our recommended method for approaching data integration.

Declarative Solutions

We classify declarative solutions as either "legacy" or "modern."

Legacy

> Think Talend, Wherescape, and Pentaho. These tools have robust connectors and benefit from a rich community and extensive support. However, as the data landscape evolves, many of these tools lag behind, not aligning with the demands of the Modern Data Stack (MDS). Unless there's a compelling reason, we'd recommend looking beyond legacy enterprise

tools.

Modern

Here’s where Fivetran, Stitch, and Airbyte come into play. Designed around “connectors,” these tools can seamlessly link various sources and targets, powered by state-of-the-art tech and leveraging the best of the MDS.

Native

In the first two solutions, we’re working from the assumption that data must be moved from one source to another— but what if you had a managed platform that supported ingestion, out-of-the-box? Databricks, for example, can natively ingest from message buses and cloud storage.

```
CREATE STREAMING TABLE raw_data

AS select *

FROM cloud_files(“/raw_data”, “json”);

CREATE STREAMING TABLE clean_data

AS SELECT SUM(profit)...

FROM raw_data;
```

What is the main allure of declarative solutions in data ingestion? The reduced “cost to build/maintain.”

Cost to build/maintain

Here’s where you get a bang for your buck. Dedicated engineers handle the development and upkeep of connectors. This means you’re delegating the heavy lifting to specialists. Most paid solutions come with support teams or dedicated client managers, offering insights and guidance tailored to your needs. These experts likely have a bird’s eye view of the data landscape and can help you navigate ambiguity around specific data problems *or* connect you with other practitioners.

However, what separates one service from another, and a factor you'll need to weigh, is extensibility.

Extensibility

This revolves around how easy it is to build upon existing structures. How likely is that new connector to be added to Airbyte or Fivetran? Can you do it yourself, building on the same framework? Or do you have to wait weeks/months/years for a product team? Will using Stitch suffice or will you need a complementary solution? Remember, juggling multiple solutions can inflate costs and complicate workflows.

A significant drawback, however, can be the cost to switch tools.

Cost to switch

This is where the limitations of declarative tools come to light. Proprietary frameworks and specific Change Data Capture (CDC) methods make migrating to another tool expensive. While sticking with a vendor might be a necessary compromise, it's essential to factor this in when evaluating potential solutions.

Imperative Solutions

Imperative data ingestion approaches can be in-house Singer taps, Lambda functions, Apache Beam templates, or jobs orchestrated through systems like Airflow.

Typically, larger organizations with substantial resources find the most value in adopting an imperative methodology. Maintaining and scaling a custom, in-house ingestion framework generally requires the expertise of multiple data engineers or even a dedicated team.

The biggest benefit of declarative solutions is their extensibility.

Extensibility

By nature, to be declarative is to be custom— that means each tap and target is *tailored to the needs of the business*. When exploring data integration options, it quickly becomes apparent that no single tool meets every criterion. Standard solutions inevitably involve compromises. However, with an imperative approach, there's the freedom to design it precisely according to the desired specifications.

Unfortunately, this extensibility is incredibly expensive to build and maintain.

Cost to build/maintain

While imperative solutions can solve complex and difficult ingestion problems, they require engineers to understand sources *and* targets. One look at the Stripe ERD should be enough to convince you that this can be incredibly time consuming. Additionally, the evolving nature of data—like changes in schema or the deprecation of an API—can amplify the complexity. Managing a single data source is one thing, but what about when you scale to multiple sources?

A genuinely resilient imperative system should incorporate best practices in software design, emphasizing modularity, testability, and clarity. Neglecting these principles might compromise system recovery times and hinder scalability, ultimately affecting business operations. Hence, we suggest that only enterprises with a robust data engineering infrastructure consider going fully imperative.

Cost to switch

Transitioning from one imperative solution to another might not always be straightforward, given the potential incompatibility between different providers' formats. However, on a brighter note, platforms based on common frameworks, like Singer, might exhibit more compatibility, potentially offering a smoother switch compared to purely declarative tools such as Fivetran or Airbyte.

Hybrid Solutions

Striking the right balance in integration often means adopting a hybrid approach. This might involve leveraging tools like Fivetran for most integration tasks, while crafting in-house solutions for unique sources, or opting for platforms like Airbyte/Meltano and creating custom components for unsupported data sources. Alternatively, you might leverage the power of *Native* solutions— like Databricks cloud ingestion or AWS Glue for moving data from S3 to Redshift.

Contributing to open-source can also be rewarding in a hybrid environment. Though not without faults, hybrid connectors, like those in Airbyte or Singer taps, benefit from expansive community support. Notably, Airbyte's contributions to the sector have positively influenced market dynamics, compelling competitors like Fivetran to introduce free tiers. We also encourage a proactive approach to exploring emerging libraries and tools. For instance, 'dlt' is an open-source library showing significant promise.

Consider data integration akin to choosing an automobile. Not every task demands the prowess of a Formula One car. More often, what's required is a dependable, versatile vehicle.

However, while a Toyota will meet 99% of uses, you won't find a Sienna in an F1 race. The optimal strategy? Rely on the trusty everyday vehicle, but ensure the capability to obtain high-performance when necessary.

Table 1-1. Declarative/Imperative Matrix

	Declarative	**Imperative**	**Hybrid**
Extensibility	Low/moderate	High	High
Cost to build/maintain	Low	High	Moderate
Lock-in	High	Low	Low

Data Ingestion Checklist

Stage	Topic	Note
Upstream	Who will we collaborate with?	Engineering (Payments)
	How will the data be used?	Financial reporting & quarterly strategizing
	Are there multiple sources?	Yes
	What’s the format?	Semi-structured APIs (Stripe & Internal)
	What’s the frequency?	Micro-batch
	What’s the volume?	Approximately 1k new rows/day, with an existing pool of ~100k
	What processing is required?	Data tidying, such as column renaming, and enrichment from supplementary sources
	How will the data be stored?	Storing staged data in Delta Tables via Databricks
Downstream	Who will we collaborate with?	Marketing (Analytics)
	How will the	Financial reporting & quarterly

	data be used?	strategizing
	Are there multiple destinations?	Data is first staged in GCP, then written to BigQuery.
	What’s the format?	Parquet in GCP. Structured/semi-structured in BigQuery.
	What’s the frequency?	Batch
	What’s the volume?	Approximately 1k new rows/day, with an existing pool of ~100k
	What processing is required?	Downstream processing using Dataform/dbt
	How will the data be stored?	Source tables in Google BigQuery
Solutions		
Upstream	1. FiveTran for connections 2. Orchestrated batch API for others	Medallion stage
Downstream	GCP Databricks to BigQuery	External tables

About the Author

Matt Palmer is a Data Engineer at Underline Infrastructure. He graduated from Swarthmore College, where he majored in physics and economics. Matt enjoys using analytics to understand products and make data-driven decisions. Engineering is a passion for him because he loves making things more efficient by building new tools.