O'REILLY®

Architecting Data and Machine Learning Platforms

Enable Analytics and AI-Driven Innovation in the Cloud

Early Release

RAW & UNEDITED

Marco Tranquillin,
Valliappa Lakshmanan
& Firat Tekiner

Architecting Data and Machine Learning Platforms

Enable Analytics and AI-Driven Innovation in the Cloud

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Marco Tranquillin, Valliappa Lakshmanan, and Firat Tekiner

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

Architecting Data and Machine Learning Platforms

by Marco Tranquillin, Valliappa Lakshmanan, and Firat Tekiner

Copyright © 2024 Marco Tranquillin, Vallliappa Lakshmanan, and Firat Tekiner. 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*.

Acquisitions Editor: Megan Laddusaw

Development Editor: Virginia Wilson

Production Editor: Gregory Hyman

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

December 2023: First Edition

Revision History for the Early Release

- 2023-04-24: First Release
- 2023-06-01: Second Release
- 2023-06-29: Third Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098151614* for release details.

The O’Reilly logo is a registered trademark of O’Reilly Media, Inc. *Architecting Data and Machine Learning Platforms*, the cover image, and related trade dress are trademarks of O’Reilly Media, Inc.

The views expressed in this work are those of the authors and do not represent the publisher’s views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-15161-4

Preface

What is a data platform? Why do you need it? What does building a data and ML platform involve? Why should you build your data platform on the cloud? This book starts by answering these common questions that arise when dealing with data and ML projects. We then lay out the strategic journey that we recommend you take to build data and ML capabilities in your business, and wrap up all the concepts in a model data modernization case.

Why do you need a Cloud data platform?

Imagine that the CTO of your company wants to build a new mobile-friendly e-commerce website. “We are losing business,” he claims, “because our website is not optimized for mobile phones, especially in Asian languages.”

The CEO trusts the CTO when he says that the current website’s mobile user experience isn’t great, but she wonders whether customers who access the platform through mobile phones form a profitable segment of the population. She calls into the head of operations in Asia and asks “*What is the revenue and profit margin on customers who reach our e-commerce site on mobile phones? How will our overall revenue change over the next year if we increase the number of people making purchases on mobile*?”

How would the regional leader in Asia go about answering this question? It requires the ability to relate customer visits (*to determine the origin of HTTP requests*), customer purchases *(to know what they purchased*), and procurement information (*to determine the cost of those items*). It also requires being able to predict the growth in different segments of the market. Would the regional leader have to reach out to the IT department and ask them to pull together the necessary information from all these different sources and write a program to compute these statistics? Does the IT department have the bandwidth to answer this question, and the skills to do predictive analysis?

How much better would it be if the organization has a *data platform*? In this case, all the data will have already been collected, cleaned up, and available for

analysis and synthesis across the organization. A data analyst team could simply run an interactive, ad hoc query. They could also easily create or retrieve forecasts of revenue and traffic patterns by taking advantage of built-in AI capabilities and allow a data-driven decision to be made on the CTO's request to invest in a new mobile-friendly website.

Modern organizations increasingly want to make decisions based on data. Our example focused on a one-time decision. However, in many cases, organizations want to make decisions repeatedly, in an automated manner for every transaction. For example, the organization might want to determine whether a shopping cart is in danger of being abandoned and immediately show the customer options of low-cost items that can be added to the shopping cart to meet the minimum for free shipping. These items should appeal to the individual shopper and therefore require a solid analytics and ML capability.

To make decisions based on data, organizations need a data and ML platform that simplifies:

- Getting access to data
- Running an interactive, ad hoc query
- Creating a report
- Making automated decisions based on data
- Personalization of the business' services

As you will see in this book, cloud-based data platforms reduce the technical barrier for all these capabilities: it is possible to access data from anywhere, carry out fast, large-scale queries even on edge devices, and take advantage of services that provide many analytics and AI capabilities. However, being able to put in place all the building blocks needed to achieve that can sometimes be a complex journey. The goal of this book is to help readers in having a better understanding of the main concepts, architectural patterns and tools available to build modern cloud data platforms to gain better visibility and control of their corporate data to make more meaningful and automated business decisions.

We, the authors of this book, are engineers who, for the past seven years, have been helping enterprises in a wide variety of industries and geographies build

data and ML platforms. These enterprises want to derive insights from their data but often face many challenges with getting all the data they need in a form where it can be quickly analyzed. Therefore, they find themselves having to build a modern data and machine learning (ML) platform.

Who is this book for?

This book is for architects who wish to support data-driven decision making in their business by creating a data and ML platform using public cloud technologies. It is also relevant for a data engineer, data analyst, data scientist, or ML engineer, who will find several useful concepts to gain a high-level design view of the systems that they might be implementing on top of.

Digitally native companies have been doing this already for several years.

As early as 2016, Twitter explained[1] that their data platform team maintains "*systems to support and manage the production and consumption of data for a variety of business purposes, including publicly reported metrics, recommendations, A/B testing, ads targeting, etc.*" In 2016, this involved maintaining one of the largest Hadoop[2] clusters in the world. By 2019, this was changing to include supporting the use of a cloud-native data warehousing solution[3].

Etsy, to take another example, says[4] that their machine learning platform "*supports machine learning experiments by developing and maintaining the technical infrastructure that Etsy's ML practitioners rely on to prototype, train, and deploy ML models at scale.*"

Both Twitter and Etsy have built modern data and ML platforms. The platforms at the two companies are different, to support the different types of data, personnel, and business use cases that the platforms need to support but the underlying approach is pretty similar. In this book, we will show you how to architect a modern data and ML platform that enables engineers in your business to:

- Collect data from a variety of sources ranging from operational databases, customer clickstream, IoT devices, SaaS applications, etc.
- Break down silos between different parts of the organization
- Process data while ingesting it, or after loading it
- Analyze the data routinely or ad-hoc

- Enrich the data with prebuilt AI models
- Build ML models to carry out predictive analytics
- Act on the data routinely or in response to triggering events or thresholds
- Disseminate insights and embed analytics

This book is a good introduction to architectural considerations if you work with data and ML models in enterprises, because you will be required to do your work on the platform built by your data or ML platform team. Thus, if you are a data engineer, data analyst, data scientist, or ML engineer, you will find this book helpful for gaining a high-level systems design view.

Even though our primary experience is with Google Cloud, we endeavor to maintain a cloud-agnostic vision of the services that underlie the architectures by bringing in examples from all the three major cloud providers (i.e. AWS, Azure and Google Cloud).

Organization of this book

The book has been organized in 12 chapters that map to the *strategic steps to innovate with the data* that will be explained in detail in Chapter 2. The book concludes with a model use case scenario to showcase how an organization might approach its modernization journey.

The visual representation of the book flow is reported in Figure P-1.

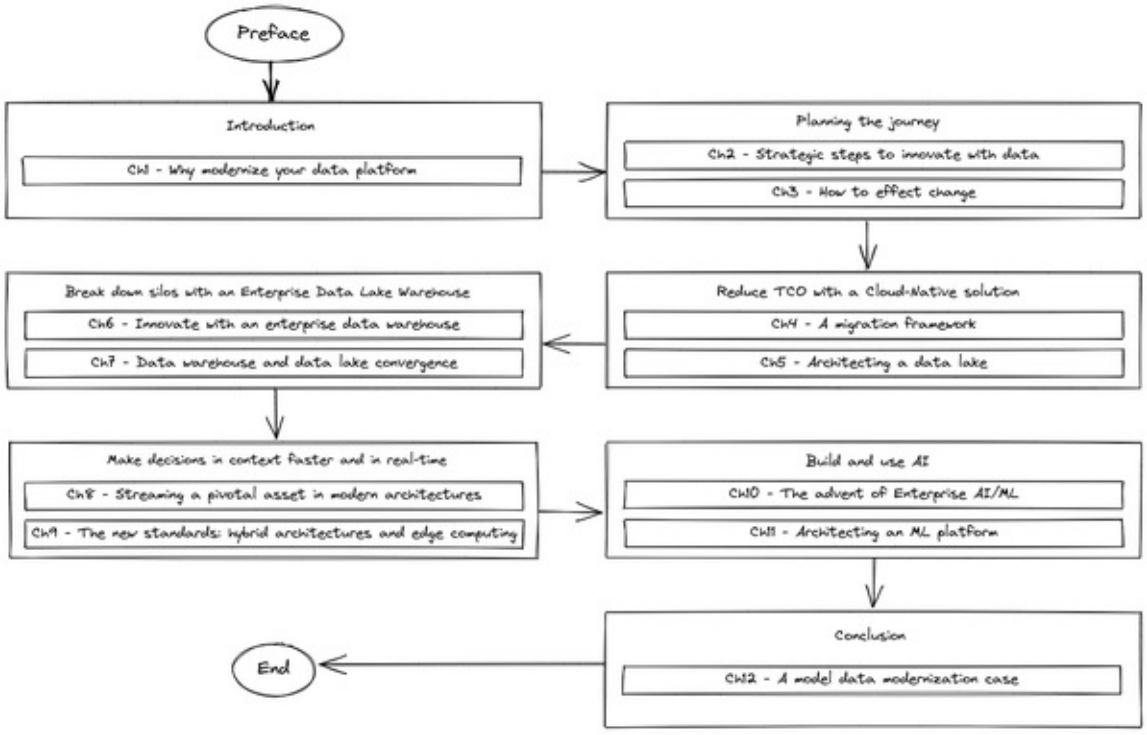


Figure P-1. Figure P-1. Book flow diagram

The first chapter is an introduction to understand why organizations should modernize the data platform. It covers approaches, technology trends and core principles in data platforms.

In Chapters 2 and 3 we dive more into how to plan the journey identifying the strategic steps to innovate and how to effect change. Here we will discuss concepts like reduction of the total cost of ownership (TCO), the removal of data silos, and how to leverage AI to unlock innovation. We also analyze the building blocks of a data lifecycle, discuss how to build a data culture, and plan how to introduce it into a company.

In Chapters 4 and 5 we investigate how to effectively reduce the TCO by adopting a migration framework; here we review approaches in data modernization, ways to estimate costs, and handle data at scale. We then describe how to architect a modern data lake.

In Chapters 6 and 7 the focus is on how to break down silos with enterprise data lake warehouses. We review how to introduce innovation with modern data warehouses and then evolve to an architecture where there is a convergence between data warehouse and data lake.

In Chapters 8 and 9 we examine how to make decisions in contests faster and in real time via the introduction of streaming patterns and the definition of hybrid architecture with an expansion/ integration with the edge.

Chapters 10 and 11 cover how to build and use AI/ML in enterprise environments and how to design architectures to design, build, serve and orchestrate innovative models.

Finally in Chapter 12, we will have a look at a model data modernization case journey with a focus on how to migrate from a legacy architecture to the new one explaining what is the process that brings an organization to select one specific solution.

If you are a cloud architect tasked with building a data and ML platform for your business, read all the chapters of the book in order.

If you are a data analyst whose task is to create reports, dashboards and embedded analytics, read Chapters 1, 5-7 and 12.

If you are a data engineer who builds data pipelines, read Chapters 1-9. Skim the remaining chapters and use them as a reference when you encounter the need for a particular type of application.

If you are a data scientist charged with building ML models, read Chapters 1-3 and 9-12.

If you are an ML engineer interested in operationalizing ML models, skim through Chapters 1-9 and study Chapters 10-12 carefully.

---

1 *https://blog.twitter.com/engineering/en_us/topics/insights/2016/discovery-and-consumption-of-analytics-data-at-twitter*

2 A set of open source tools that streamlines the resolution of problem leveraging networks of interconnected computers that perform data intensive computations

3 *https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/democratizing-data-analysis-with-google-bigquery*

4 *https://codeascraft.com/2021/12/21/redesigning-etsys-machine-learning-platform/*

Chapter 1. Modernizing Your Data Platform: An Introductory Overview

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the first chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

Almost every company these days understands the importance of having full control of their data to make meaningful business decisions. In order to become a data driven company, these organizations need to build an ecosystem for data analytics, processing, and insights. This is particularly true because there are multiple different types of applications (web sites, dashboards, mobile applications, machine learning models, distributed devices, etc.) that create data and consume data. There are also multiple divisions within the company (finance, sales, marketing, operations, logistics, etc.) that require data-driven insights. Data use cases may differ in whether they are transactional, the kind of throughput they need to support, who is allowed to do different things with the data, and the number of transactions or queries that the task needs to support. Throughout this book you will learn how to design modern data platforms that enable the development of such an ecosystem utilizing the foundational building blocks available on the major public clouds. At the end of the book you will have a better understanding of the core concepts, architectural patterns and tools

available to build modern data platforms that today's organization should adopt to gain better visibility and control of their corporate data and to make more meaningful and automated business decisions.

In this introductory chapter we will focus on what a data platform is, what a data platform requires, and why traditional data products don't cut it. We will then discuss technology trends in data analytics and AI, and how to build data platforms for the future using the public cloud.

This chapter will give you a bird's eye view of the core topics we will cover in detail in the rest of the book (e.g. data silos removal, convergence of data lake and data warehouse, hybrid architecture, ML in the enterprise, etc.). You should consider this chapter as a sort of mini-book that you can read by itself to get a glimpse of general concepts, and then rely on the following chapters for a deeper study.

Why do organizations need a data platform?

Traditionally, organizations' data ecosystems consist of independent solutions that are used to provide different data services. Unfortunately, such task-specific data stores have led to the creation of silos within the organization. The resulting siloed systems operate as independent solutions that are not working together in an efficient manner. Siloed data is silenced data – it's data from which insights are difficult to derive. To broaden and unify enterprise intelligence, securely sharing data across lines of business leads is critical.

If the majority of the solutions are custom built, it becomes difficult to plan for business continuity (BC) and disaster recovery (DR). If each part of the organization chooses a different environment to build their solution in, and each environment comes with its own way to guarantee high availability and methodology for DR (i.e. infrastructure vs software approach), the complexity becomes overwhelming. In such a scenario, how could you cope with requirements such as ensuring privacy or being able to audit changes to data?

One solution is to develop a data platform and, more precisely, a *cloud* data platform. Its purpose is to allow *analytics and machine learning to be carried out over all of an organization's data in a consistent, scalable and reliable way*. The data platform should leverage, to the maximum extent possible, managed

services so that the organization can focus on business needs instead of operating infrastructure. Infrastructure operations and maintenance should be delegated totally to the underlying cloud platform. In this book, we will cover the core decisions that you need to make when developing a unified platform to consolidate data across business units in a scalable and reliable environment.

Data silos means data movement tools

Because organizations tend to have a multitude of solutions to manage their data, gaining a unified view of the information that they possess is challenging. Today, organizations usually solve this problem by leveraging data movement tools. Extract Transform Load (ETL) applications allow data transformations and data transfer between different systems in order to build a unique source of truth.

For example, an ETL tool might be built to periodically retrieve the latest transactions from a transactional database and archive them into an analytics store. Over time, ETL tools are built for every database table that is required for analytics, so that analytics can be carried out without having to go to the source system each time as illustrated in Figure 1-1.

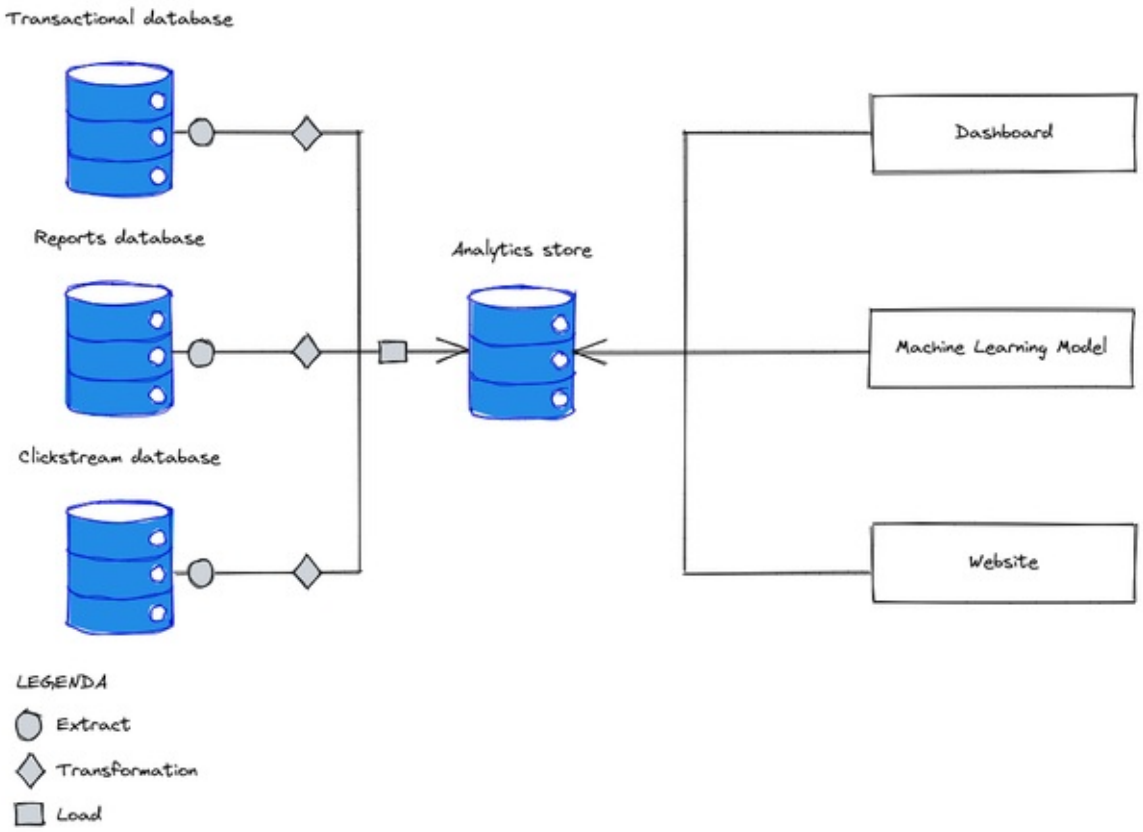


Figure 1-1. Extract Transform Load (ETL) Tools can help break down data silos

The central analytics store that captures all the data across the organization is referred to as either a *data warehouse* or as a *data lake* depending on the technology being used. A high level distinction between the two approaches is based on the way the data is stored within the system: if the analytics store supports Standard Query Language (SQL) and contains governed, quality controlled data, it is referred to as a *data warehouse*. If instead it supports tools from the Apache ecosystem (such as Apache Spark) and contains raw data, it is referred to as a *data lake*. Terminology for referring to in-between analytics stores (such as governed raw data or ungoverned quality-controlled data) varies from organization to organization – some organizations call them data lakes and others call them data warehouses. As you will see later in the book (in Chapters 5, 6 and 7), this confusing vocabulary is not a problem because data lake and data warehouse approaches are converging.

There are a few drawbacks to relying on data movement tools to try building a consistent view of the data:

- **Latency**. ETL tools introduce latency. For example, if the ETL tool to

extract recent transactions runs once an hour, and takes 15 minutes to run, the data in the analytics store could be stale by up to 75 minutes. This problem can be addressed by streaming ETL where events are processed as they happen.

- **Bottleneck**. ETL tools typically involve programming skills. Therefore, organizations set up bespoke data engineering teams to write the code for ETL. As the diversity of data within an organization increases, an ever increasing number of ETL tools need to be written. The data engineering team becomes a bottleneck on the ability of the organization to take advantage of data.
- **Maintenance**. ETL tools need to run routinely. This requires system administrators to be familiar with the ETL tools and to be able to troubleshoot errors. At the same time it is important to consider that the underlying infrastructure system needs to be continually updated as well:
    - to be able to cope with the increase of compute and storage capacity needed to perform all the task carried out by the system (e.g. more ETLs, more compute and storage needed)
    - to guarantee the reliability of the overall system (i.e. high availability, business continuity, and disaster recovery).
- **Change management.** Changes in the schema of the input table require the Extract code of the ETL tool to be changed. This either makes changes hard to do, or results in the ETL tool being broken by upstream changes.
- **Data gaps**. it is very likely that many errors have to be escalated to the owners of the data, the creators of the ETL tool, or the users of the data. This adds to maintenance overhead, and very often to tool downtime. There are quite frequently large gaps in the data record because of this.
- **Governance**. As ETL processes proliferate, it becomes increasingly likely that the same processing is carried out by different processes, leading to multiple sources of the same information. It’s common for the processes to diverge over time to meet different needs, leading to inconsistent data being used for different decisions.

Often data needs to be pre-processed before it can be “trusted” to be made available in production. Generally data coming from the upstream systems can be considered pretty *raw* and the majority of the time it can contain noise or even bad information if not correctly cleaned and transformed. Let’s imagine for example web-logs of an e-commerce system: there could be some information written in a specific format that needs to be transformed before being used (e.g. the ID of the purchased item maybe written within a URL string) or there could be some false transactions made by bots that need to be filtered out. As with the extraction code in ETL tools, the data processing tools (i.e. the Transformation part) have to be built in a task-specific manner. There may be no global data quality solution or common framework to handle quality issues, and every solution might involve making the data fit for purpose.

While this situation is reasonable when considering one data source at a time, the total collection (see Figure 1-2) leads to chaos.

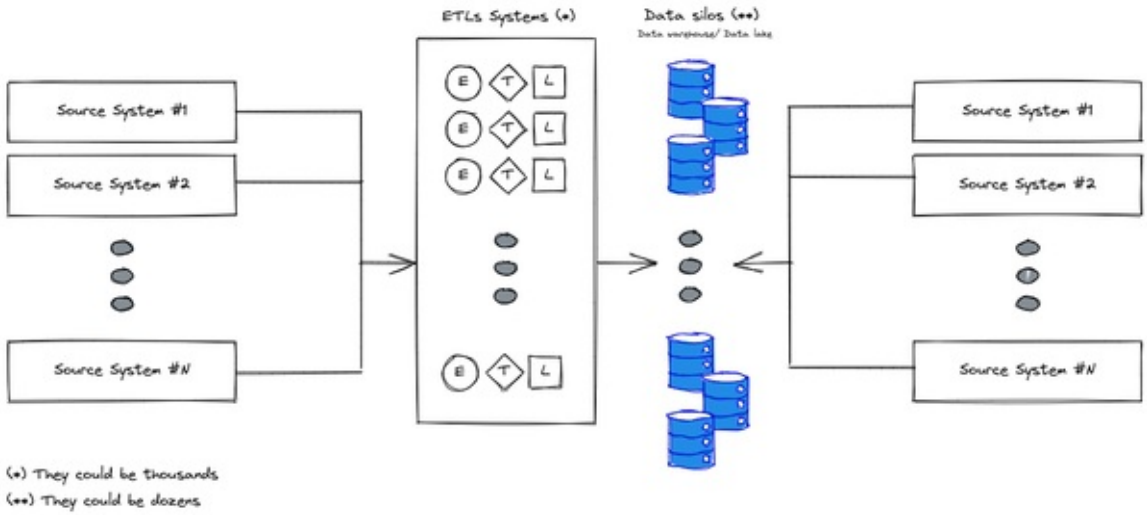


Figure 1-2. Data ecosystem and challenges

The proliferations of storage systems, together with tailored-made data management solutions developed to satisfy the desires of different downstream applications, bring about a situation where analytics leaders and CIOs face the following challenges:

- Their data warehouse/ data lake are unable to keep-up with the ever growing business needs
- Increasingly, digital initiatives (and competition with digital natives) have

transformed the business to be one where massive data volumes are flooding the system

- The need to create separate data lakes, data warehouses, and special storage for different data science tasks ends up creating multiple data silos
- Data access needs to be limited or restricted due to performance, security, and governance challenges
- Renewing licenses and paying for expensive support resources become challenging

It seems pretty clear that this kind of approach cannot scale in accordance with the new needs of the business not only because of the technology complexity but even because of all the burden of the security and governance requirements that this model drags in.

Centralization of control

To try to address the problem of having siloed, spread and distributed data managed via task-specific data processing solutions, some organizations have tried to centralize everything in a single, monolithic platform under the control of the Information Technology (IT) department. As shown in Figure 1-3, the underlying technology solution doesn’t change — instead, the problems are made more tractable by assigning them to a single organization to solve.

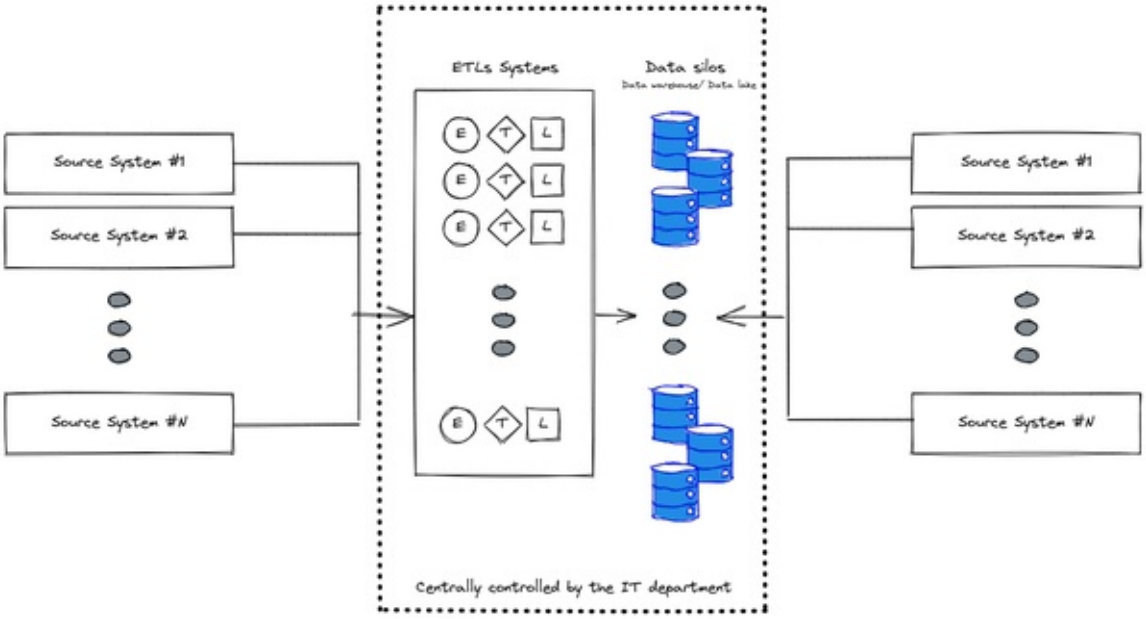


Figure 1-3. Data ecosystem and challenges with IT centrally controlling data systems

Such centralized control by a unique department comes with its own challenges and trade offs. All business units (BU) – IT itself, data analytics, and business users – struggle when IT controls all data systems:

- **IT**: The challenge that IT departments face is the diverse set of technologies involved in these data silos. IT departments rarely have all the skills necessary to manage all of these systems. The data sits across multiple storage systems on premises and across clouds, making it costly to manage data warehouses, data lakes and data marts. It is also not always clear how to define security, governance, auditing etc. across different sources. Moreover it introduces a scalability problem in getting access to the data: the amount of work that IT needs to carry out linearly increases with the number of source systems and target systems that will be part of the picture because this will surely increase the number of data access requested by all the related stakeholders/ business users. This can quickly become a bottleneck slowing down the innovation process within the organizations.
- **Analytics**: One of the main problems hindering effective analytics processes is not having access to the right data. When multiple systems exist, moving data to/from one monolithic data system becomes costly resulting in unnecessary ETL tasks, etc. In addition, the pre-prepared and readily available data might not have the most recent sources or there might

be other versions of the data that provide more depth and broader information such as having more columns or having more granular records. It is impossible to give your analytics team free rein whereby everyone can access all data due to data governance and operational issues. Organizations often end up limiting data access at the expense of analytic agility.

- **Business**: Getting access to data and analytics that your business can trust is difficult. There are issues around limiting the data you give the business so you can ensure the highest quality. The alternative approach is to open up access to all the data the business users need even if that means sacrificing quality. The challenge then becomes a balancing act on the quality of the data and amount of trusted data given.

Despite so many challenges, several organizations adopted this approach throughout the years creating, in some cases, frustrations and tensions for business users who were delayed in getting access to the data they needed to fulfill their tasks. As an aside, this proliferated the anti-pattern of *shadow IT* where entire departments develop and deploy proper solutions to work around such limitations but in the process make the problem of siloed data worse.

The technologies underlying data warehouses and data lakes

The challenges around a siloed centrally managed system created huge tension and overhead for IT. They had to manually build and manage the glue across all these systems you have seen in the previous high level architectures while trying to minimize the impact of the trade offs.

In the first instance, data was stored in on-premises relational and analytical databases called *data warehouses* often serving different workloads across industries and for which we will dig into more details in Chapters 3 and 6. Data warehouses are a powerful solution that allows business users to design and deploy their own business data into structured data models (e.g. in retail, healthcare, banking, insurance, etc.) allowing them to easily get information about the current and the historical business (e.g. the amount of revenue of the last quarter, the number of users who played your last published game in the last week, the correlation between the time spent on the help center of your website

and the number of tickets received in the last 6 months, etc.). One reason data warehouses became so popular is that everything was SQL based, a technology that many business users are comfortable with. Over many decades (and continuing today), we have seen organizations developing data warehouse solutions leveraging different technologies (e.g. Oracle, Teradata, Vertica) and implementing several different applications on top of it. However, these technologies on premises are severely constrained by capacity. IT teams and data stakeholders struggle with the challenge of (*vertical*) scaling infrastructure, finding critical talent, improving costs, and ultimately managing the growing expectation to deliver valuable insights.

This scalability issue led to the advent of *Big Data* solutions based on the Apache Hadoop ecosystem. Hadoop introduced distributed data processing (*horizontal scaling*), realized via the *MapReduce* programming model[1], and the Hadoop Distributed File Service (HDFS) (inspired by Google File system paper[2]) on cheap commodity servers. As you will see in Chapters 3 and 5, this new approach allowed organizations to cope with the explosion of data not only in terms of volume but even in terms of analysis velocity and data model variety ("the three Vs"). This approach opened the door to a brand new era of data analysis enabling use cases that were previously only possible with high end (and very costly) specialized hardware. Every application running on top of Hadoop was designed to tolerate node failures, thus making it a cost effective alternative for some traditional data warehouse workloads. This gave rise to a new concept called *data lake*, which soon became a core pillar of data management alongside the data warehouse.

The idea was that while core operational technology divisions carried on with their routine tasks, all data was exported for analytics into a centralized data lake. The intent was for the data lake to serve as the central repository for analytics workloads and for business users. Data lakes began expanding their capabilities beyond storage of raw data to include advanced analytics and data science on large volumes of data. This enabled self-service analytics across the organization, but it required an extensive working knowledge of advanced Hadoop and engineering processes in order to access the data. The Hadoop Open Source Software (Hadoop OSS) ecosystem grew in terms of data systems and processing frameworks (Hbase, Hive, Spark, Pig, Presto, SparkML, and more) in parallel to the exponential growth in organizations' data, but this led to

additional complexity and cost of maintenance.

The cloud as a solution for main challenges

Data warehouse and data lakes technologies enabled IT to build the first iteration of a data platform to break down data silos and to enable the organization to derive insights from all their data assets. The data platform enabled data analysts, data engineers, data scientists, business users, architects, and security engineers to derive better real time insights and predict how their business will evolve over time.

However, running the above mentioned technologies in an on-premises environment presents some inherent challenges (as you will see in Chapter 2) with scaling and operations costs. This has caused organizations to review their approach and to start considering the cloud (especially the public version of it) as the de facto environment for such a platform. Why? Because it allows them to:

- Reduce cost by taking advantage of new pricing models (*pay per use model*)
- Speed up innovation by taking advantage of best-of-breed technologies
- Scale on-premises resources by bursting to the cloud
- Plan for business continuity by having data in multiple regions
- Manage disaster recovery in an automated fashion leveraging fully managed services

When users are no longer constrained by the capacity of their infrastructure, organizations are able to democratize data across their organization and unlock insights. The cloud supports organizations in their modernization efforts as it minimizes the toil and friction by offloading the administrative, low-value tasks. A cloud data platform promises an environment where you no longer have to compromise and can build a comprehensive data ecosystem that covers the end-to-end data management and data processing stages from data collection to serving. And you can use your cloud data platform to store vast amounts of data in varying formats without compromising on latency.

Cloud data platforms promise:

- Centralized governance and access management
- Increased productivity and reduced operational costs
- Greater data sharing across the organization
- Extended access by different personas
- Reduced latency of accessing data

To what extent do they deliver? What are the considerations to keep in mind when using cloud technologies to solve data science workflows? We will dive into these concepts throughout the book and will provide you with the tools needed to understand how to address the mentioned challenges.

Now that you have a clearer understanding of why organizations need a cloud data platform, let's have a look at how *convergence* is playing a key role in designing and building modern architectures.

Creating a Unified Analytics Platform

In the previous section, you saw that data warehouses and data lakes are the core of data platforms even if they come with different approaches: data warehouses support structured data and SQL, whereas data lakes support raw data and programming frameworks in the Apache ecosystem. In the public cloud environment, the lines between the two technologies are blurring because cloud infrastructure (specifically, the separation of compute and storage) enables a convergence that was impossible in the on-premises environment. Today it is possible to apply SQL to data held in a data lake, and it's possible to run what is traditionally a Hadoop technology (e.g. Spark) against data stored in a data warehouse. In this section we will give you an introduction to how this convergence works and how it can be the basis for brand new approaches that can revolutionize the way organizations are looking at the data; you'll get more details in Chapter 7.

Drawbacks of Data Warehouses and Data Lakes

One thing that every IT department understood in the last 40 years is that data warehouses are often difficult to manage and become incredibly costly. The legacy systems that worked well in the past (such as Teradata, Netezza, etc.) have proven to be difficult to scale and very expensive, and pose a lot of challenges around data freshness and cost (even setting aside license fees). Furthermore, they cannot easily provide modern capabilities like access to AI/ML or real-time features without bolting on that functionality after the fact.

Data warehouse users tend to be analysts, often embedded within a specific business unit. They may have ideas about additional datasets that would be useful to enrich their understanding of the business. They may have ideas for improvements in the analysis, data processing, and requirements for business intelligence functionality. However, in a traditional organization, they often don't have direct access to the data owners, nor can they easily influence the technical decision makers who decide datasets and tooling. In addition, because they don't have access to raw data, they are unable to test hypotheses or drive a deeper understanding of the underlying data.

Data lakes have their own challenges. In theory, they are low cost and easy to scale, but we repeatedly notice that organizations have seen a different reality in their on-premise implementations. Planning for and provisioning sufficient storage can be expensive and difficult, especially for organizations that produce highly variable amounts of data. It may be expensive to provision the computational capacity of data lakes for peak periods. So, different business units often end up fighting over scarce computational resources.

On-premise data lakes can be brittle, and maintenance of existing systems takes time. Almost always, the engineers who would otherwise be developing new features are relegated to the maintenance of data clusters and scheduling jobs among different business units. Overall, the total cost of ownership is higher than expected for many companies. Said more bluntly, data lakes do not create value and many businesses find that the return on investment is negative.

With data lakes, governance is not easily solved, especially when different parts of the organization use different security models. Then, the data lakes become siloed and segmented, making it difficult to share data and models across teams.

Data lake users typically are closer to the raw data sources and need programming skills in order to use data lake tools and capabilities, even if it is

just to explore the data. In traditional organizations, these users tend to focus on the data itself and are frequently held at arm's length from the rest of the business. On the other hand, business users do not have the programming skills to derive insights from data in a data lake. This disconnection means that business units miss out on the opportunity to gain insights that would drive their business objectives forward to higher revenues, lower costs, lower risk, and new opportunities.

Convergence of Data Warehouses and Data Lakes

Given these tradeoffs, many companies end up with a mixed approach, where a data lake is set up to graduate some data into a data warehouse or a data warehouse has a side data lake for additional testing and analysis. However, with multiple teams fabricating their own data architectures to suit their individual needs, data sharing and fidelity gets even more complicated for a central IT team.

Instead of having separate teams with separate goals — where one explores the business, and another knows the business — you can unite these functions and their data systems to create a virtuous cycle where a deeper understanding of the business drives exploration, and that exploration drives a greater understanding of the business.

Starting from this principle, the data industry has started shifting towards a new approach: *Lakehouse* and *Data mesh* which work best together because they help solve different challenges within an organization

- Lakehouse enables data access bringing the data warehouse and data lake together allowing different types and higher volumes of data
- Data mesh enables data teams designing strategy for enterprise data platform architecture.

As an added benefit, this architecture combination also brings in more rigorous data governance, something that data lakes typically lack. Data Mesh empowers people to avoid being bottlenecked by one team, and thus enables the entire data stack. It breaks silos into smaller organizational units in an architecture that provides access to data in a federated way.

Lakehouse

Historically, organizations have implemented siloed and separate architectures. Data warehouses store structured, aggregate data (primarily used for BI and reporting), whereas data lakes store large volumes of unstructured and semi-structured data (primarily used for ML workloads). This approach often results in complex ETL pipelines because of extensive data movement, processing, and duplication. Operationalizing and governing this architecture is challenging, costly, and reduces agility. As organizations are transitioning out from their on-premise data centers and leveraging cloud environments, they want to overcome these challenges.

The data lakehouse architecture, which combines the key benefits of data lakes and data warehouses, can help overcome the aforementioned challenges. This architecture offers a low-cost storage format typical of the data lake approach that is accessible by various processing engines like the SQL ones of the data warehouses while also providing powerful management and optimization features.

Databricks pushes the lakehouse architecture because of its origins with Spark, and the need to support business users who are not programmers. Hence, data in Databricks is stored as in a data lake, but business users can use SQL to access the data. However, the lakehouse architecture is not limited to Databricks.

Data warehouse running in cloud solutions like Google Cloud BigQuery or Azure Synapse allow you to create a lakehouse architecture: it allows you to treat the data warehouse like a data lake by applying SQL over Parquet files in cloud storage. This could reduce the cost of storage[3] and allow Spark jobs running on parallel Hadoop environments to leverage the data stored on the underlying storage system rather than requiring a separate storage layer.

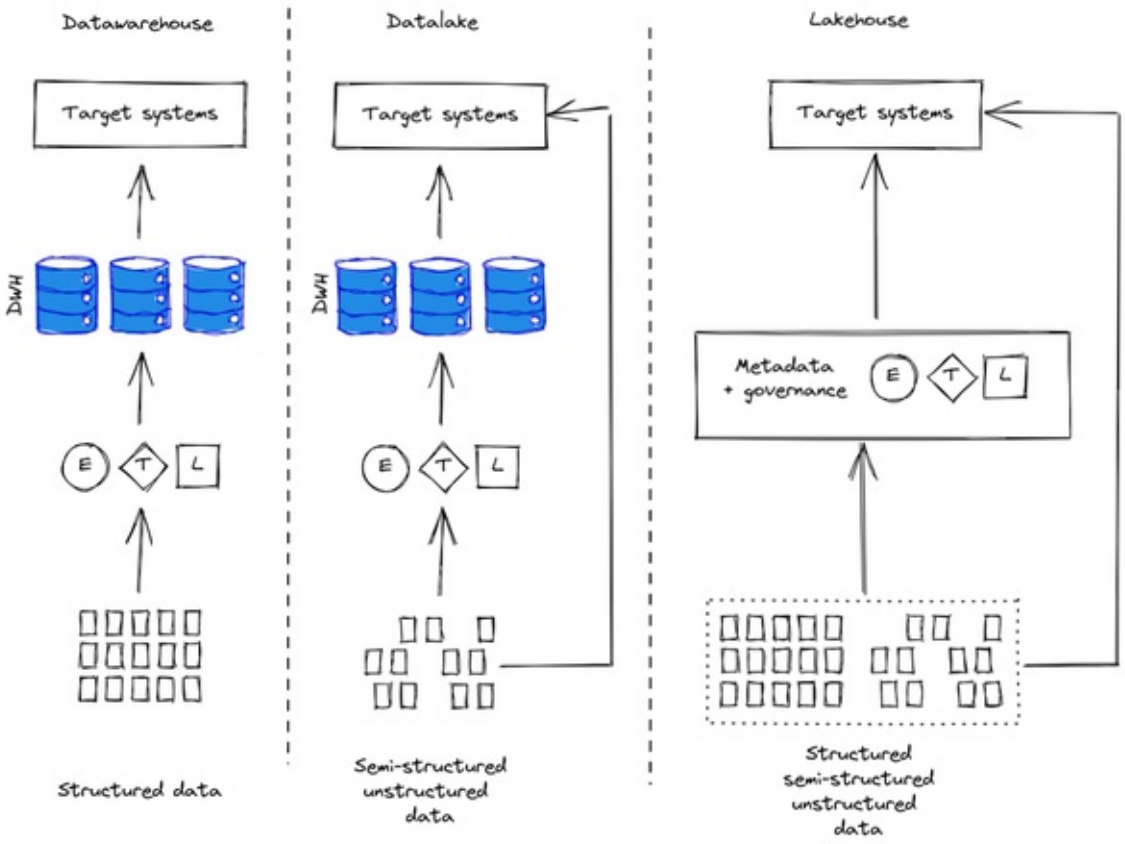


Figure 1-4. Data warehouse/ Data lake and lakehouse patterns

The lakehouse pattern offers several advantages over the traditional approaches:

- Decoupling of storage and compute that enable
  - Inexpensive, virtually unlimited, and seamlessly scalable storage.
  - Stateless, resilient compute.
  - ACID-compliant storage operations.
  - A logical database storage model, rather than physical.
- Data governance via the support of data enforcement and evolution.
- Support for data analysis via the native integration with business intelligence tools
- Native support of the typical multi-version approach of a data lake approach (i.e. bronze, silver and gold)

- Data storage and management via open source format like Apache Parquet for example
- Support for different data-types in the structured or unstructured format
- Streaming capabilities with the ability to handle real time analysis of the data
- Enablement of a diverse set of applications varying from business intelligence to machine learning

A lakehouse, however, is inevitably a technological compromise. The use of standard formats limits the storage optimizations and query concurrency that data warehouses have spent years perfecting. Therefore, the SQL supported by lakehouse technologies is not as efficient as that of a native data warehouse (i.e., it will take more resources, and cost more). Also, the SQL support tends to be limited, with features such as geospatial queries, machine learning and data manipulation, not available or incredibly inefficient.

The lakehouse approach enables organizations to implement the core pillars of an incredibly varied data platform that can support any kind of workload. But what about the organizations on top of it? How can users leverage the best of the platform to execute their tasks? In this scenario there is a new operating model that is taking shape and it is Data Mesh.

Data Mesh

Data Mesh[4] is a decentralized operating model of tech, people and process to solve the most common challenge in Analytics – the desire for centralization of control in an environment where ownership of data is necessarily distributed, as shown in Figure 1-5. Another way of looking at Data Mesh is that it introduces a way of seeing data as a self-contained product rather than a product of ETL pipelines.

Distributed teams in this approach own the data production and serve internal/external consumers through well defined data schema. As a whole, Data Mesh is built on a long history of innovation from across data warehouses and data lakes, combined with the scalability, pay-for-consumption models, self-service APIs, and close integration associated with Data Warehouse technologies

in the public cloud.

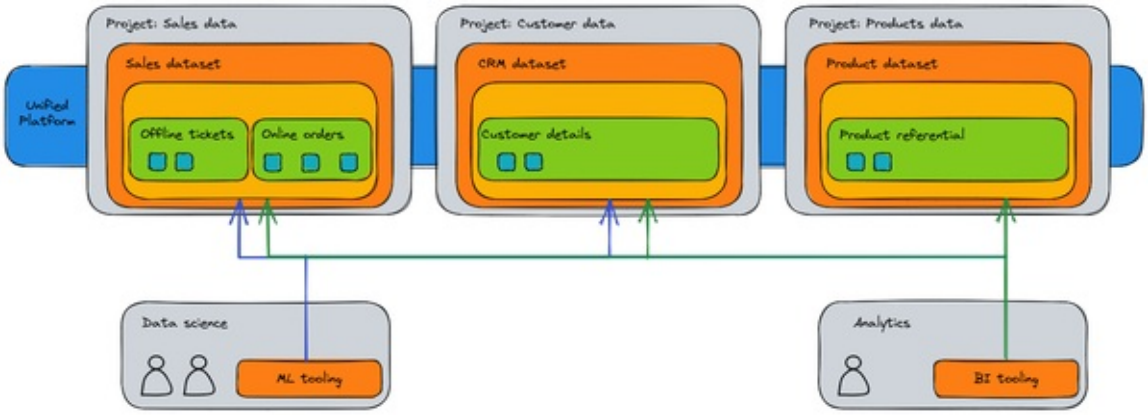


Figure 1-5. A Data Mesh centralizes control in a unified platform on the cloud, while retaining ownership of the data in distributed domains.

With this approach, you can effectively create an on-demand data solution. A Data Mesh decentralizes data ownership among domain data owners, each of whom are held accountable for providing their data as a product in a standard way (see Figure 1-6). A Data Mesh also enables communication between various parts of the organization to distribute datasets across different locations.

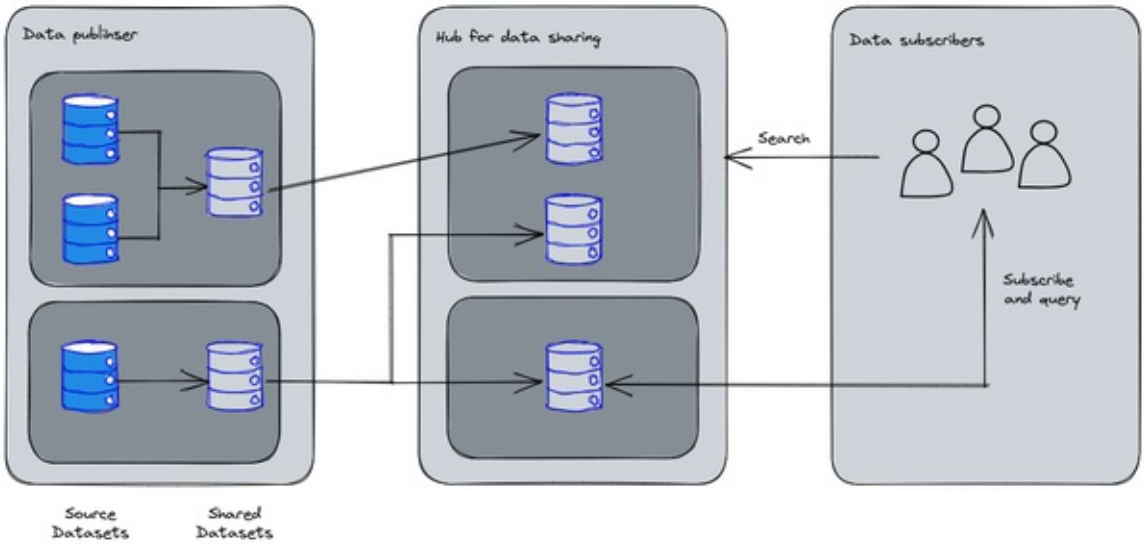


Figure 1-6. Data as a Product

In a Data Mesh, the responsibility for generating value from data is federated to the people who understand it best; in other words, the people who created the data or brought it into the organization must also be responsible for creating consumable data assets as products from the data they create. In many

organizations, establishing a “single source of truth” or “authoritative data source” is tricky due to the repeated extraction and transformation of data across the organization without clear ownership responsibilities over the newly-created data. In the Data Mesh, the authoritative data source is the Data Product published by the source domain, with a clearly assigned Data Owner and Steward who is responsible for that data.

Having access to this unified view from a technology perspective (Lakehouse) and from an organizational perspective (Data mesh) means that people and systems get data delivered to them in a way that makes the most sense for their needs. In some cases this kind of architecture has to span across multiple environments generating, in some cases, very complex architecture: let’s see how companies can manage this challenge.

> **TIP**
>
> For more information about data mesh, we recommend you read Zhamak Dehghani’s book *Data Mesh: Delivering Data-Driven Value at Scale* (O’Reilly 2022).

Hybrid Cloud

When designing a cloud data platform, it might be that one single environment isn’t enough to manage a workload end to end. This could be because of regulatory constraints (i.e. you cannot move your data into an environment outside the organization boundaries), or because of the cost (e.g. the organization made some investments on the infrastructure that did not reach the end of life), or because you need a specific technology that is not available in the cloud. In this case a possible approach is adopting a hybrid pattern. A hybrid pattern is one in which applications are running in a combination of various environments. The most common example of hybrid pattern is combining a private computing environment, like an on-premises data center, and a public cloud computing environment. In this section we will explain how this approach can work in an enterprise.

Reasons why hybrid is necessary

Hybrid cloud approaches are widespread because almost no one today relies entirely on the public cloud. Many organizations have invested millions of dollars and thousands of hours into on-premises infrastructure over the past few decades. Almost all organizations are running a few traditional architectures and business-critical applications that they may not be able to move over to public cloud. They may also have sensitive data they can't store in a public cloud due to regulatory or organizational constraints.

Allowing workloads to transition between public and private cloud environments provides a higher level of flexibility and additional options for data deployment. There are several reasons that drive hybrid (i.e. architecture spanning across on-premise, public cloud and edge) and multi cloud (i.e. architecture spanning across multiple public cloud vendors like Amazon AWS, Microsoft Azure and Google Cloud Platform for example) adoption.

Here are some key business reasons for choosing hybrid and/or multi cloud:

- **Data Residency regulations**. Some may never fully migrate to the public cloud, perhaps because they are in finance or healthcare and need to follow strict industry regulations on where data is stored. This is also the case with workloads in countries without a public cloud presence and a data residency requirement.
- **Legacy investments.** Some customers want to protect their legacy workloads like SAP, Oracle, Informatica on prem but want to take advantage of public cloud innovations like Databricks and Snowflake.
- **Transition**. Large enterprises often require a multi year journey to modernize into cloud-native applications and architectures. They will have to embrace hybrid architectures as an intermediate state for years.
- **Burst to cloud**. There are customers who are primarily on-premises and have no desire to migrate to the public cloud. However, they have challenges of meeting business SLAs due to ad-hoc large batch jobs, spiky traffic during busy periods, or large-scale ML training jobs. They want to take advantage of scalable capacity or custom hardware in public clouds, and avoid the cost to scale up onprem infrastructure.
- **Best of breed.** Many organizations choose different public cloud providers

for different tasks in an intentional strategy to choose the technologies that best serve their needs. For example, Uber uses AWS to serve their web applications[5], but uses Cloud Spanner on Google Cloud[6] for its fulfillment platform. Twitter runs its news feed on AWS[7], but runs its data platform on Google Cloud[8].

Now that you understand the reasons why you might choose a hybrid solution, let's have a look at the main challenges you will face when using this pattern — these challenges are why hybrid ought to be treated as an exception, and the goal should be to be cloud-native.

Challenges of Hybrid Cloud

There are several challenges that Enterprises face when implementing hybrid or multi cloud architectures:

- **Governance**: It is challenging to apply consistent governance policies across multiple environments. For example, compliance security policies between on premises and public cloud are usually dealt with differently. Often, parts of the data are duplicated across on-premise and cloud. Imagine your organization is running a financial report – how would you guarantee that the data used is the most recent updated copy if there are multiple copies that exist across platforms?
- **Access Control**: User access controls and policies differ between on-premise and public cloud environments. Cloud providers have their own user access controls (called Identity and Access Management or IAM) for the services provided whereas on-premises uses technologies such as Local Directory Access Protocol (LDAP) or Kerberos. How do you keep them synchronized or have a single control plane across distinct environments?
- **Workload interoperability**: When going across multiple systems it is inevitable to have inconsistent run time environments that need to be managed.
- **Data Movement:** If both on-premises and cloud applications require access to some data, the two datasets must be in sync. It is costly to move data between multiple systems – there is a human cost to create and manage the

pipeline, there may be licensing costs due to software used, and last but not least it consumes system resources such as computation, network and storage. How can your organization deal with the costs from multiple environments? How do you join heterogeneous data that is siloed across various environments? Where do you end up copying the data as a result of the join process?

- **Skill sets**: Having the two clouds (or on-premises and cloud) means teams have to know and build expertise in two environments. Since the public cloud is a fast moving environment, there is a significant overhead associated with upskilling and maintaining the skills of employees in one cloud, let alone two. Skill sets can also be a challenge for hiring systems integrators (SIs) – even though most large SIs have practices for each of the major clouds, very few have teams that know two or more clouds. As time goes on, we anticipate that it will become increasingly difficult to hire people willing to learn bespoke on-premises technologies.
- **Economics**: the fact that the data is split between two environments can bring unforeseen costs: maybe you have data in one cloud and you want to make it available to another one incurring in egress costs.

Despite these challenges, a hybrid setup can work. We'll look at how in the next subsection.

Why Hybrid can Work

Cloud providers are aware of these needs and these challenges. Therefore, they provide some support for hybrid environments. These fall into three areas:

- **Choice**: Cloud providers often make large contributions to open-source technologies. For example, although Kubernetes and TensorFlow were developed at Google, they are open-sourced so that managed execution environments for these exist in all the major clouds and they can be leveraged even in the on-premises environments.
- **Flexibility**: Frameworks such as Databricks and Snowflake allow you to run the same software on any of the major public cloud platforms. Thus, teams can learn one set of skills that will work everywhere. Some public

cloud products, such as BigQuery Omni, support running on all the clouds even though the native cloud (Google Cloud, in the case of BigQuery) tends to be more efficient. Note that the flexibility offered by tools that work on multiple clouds is different from total choice – you are now locked into the specific data processing or data warehouse technology, rather than into the specific cloud platform. You will have to choose between lockin at the framework level and flexibility at the cloud level (offered by technologies such as Databricks or Snowflake), and lockin at the cloud level and flexibility at the framework level (offered by the cloud-native tools). On Google Cloud, for example, you can choose between Google BigQuery and Dataproc, while on Amazon you can choose between Amazon Redshift and EMR whereas the interoperability between Snowflake and Databricks will be quite limited.

- **Openness**. Even when the tool is proprietary, code for it is written in a portable manner because of the embrace of open standards and import/export mechanisms. Thus, for example, even though Redshift runs nowhere but on AWS, the queries are written in Standard SQL and there are multiple import and export mechanisms. Together, these capabilities make Redshift (and BigQuery and Synapse) open platforms. This openness allows for use cases like Teads where data is collected using Kafka on AWS, aggregated using Dataflow and BigQuery on Google Cloud, and written back to AWS Redshift (see Figure 1-7).

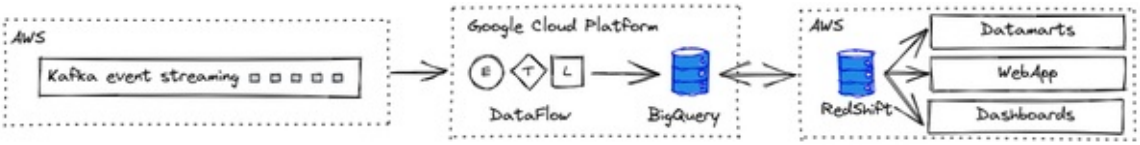


Figure 1-7. Hybrid analytics pipeline at Teads. Figure based on an article by Alban Perillat-Merceroz and published in Teads Engineering.

Cloud providers are making a commitment to choice, flexibility, and openness by making heavy investments in open source projects that help customers use multiple clouds. Therefore, multi-cloud data warehouses or hybrid data processing frameworks are becoming reality. So you can build out hybrid and multi-cloud deployments with better cloud software production, release, and management—the way you want, not how a vendor dictates.

Edge Computing

Another incarnation of the hybrid pattern is when you may want to have computational power spanning outside the usual data platform perimeter, maybe to interact directly with some connected devices. In this case we are talking about *edge computing*. Edge computing brings computation and data storage closer to the system where data is generated and needs to be processed. The aim in edge computing is to improve response times and save bandwidth. Edge computing can unlock many use cases and accelerate digital transformation. It has many application areas such as Security, Robotics, Predictive Maintenance, Smart Vehicles, etc.

As edge computing is adopted and goes mainstream, there are many potential advantages for a wide range of industries:

- **Faster response time**: In edge computing, the power of data storage and computation is distributed and made available at the point where the decision needs to be made. Not requiring a roundtrip to the cloud reduces latency and empowers faster responses. In preventive maintenance, it will help stop critical machine operations from breaking down or hazardous incidents from taking place. In active games, edge computing can provide the millisecond response times that are required. In fraud prevention and security scenarios, it can protect against privacy breaches and denial-of-service attacks.
- **Intermittent connectivity**: Unreliable internet connectivity at remote assets such as oil wells, farm pumps, solar farms or windmills can make monitoring those assets difficult. Edge devices' ability to locally store and process data ensures no data loss or operational failure in the event of limited internet connectivity.
- **Security and compliance**: Edge computing can eliminate a lot of data transfer between devices and the cloud. It's possible to filter sensitive information locally and only transmit critical data model building information to the cloud. For example, smart devices, watch-word processing such as listening for "OK Google" or "Alexa" can happen on-device itself. Potentially private data does not need to be collected or sent to the cloud. This allows users to build an appropriate security and compliance

framework that is essential for enterprise security and audits.

- **Cost-effective solutions:** One of the practical concerns around Internet of Things (IoT) adoption is the upfront cost due to network bandwidth, data storage, and computational power. Edge computing can locally perform a lot of data computations, which allows businesses to decide which services to run locally and which ones to send to the cloud, which reduces the final costs of an overall IoT solution.
- **Interoperability**: Edge devices can act as a communication liaison between legacy and modern machines. This allows legacy industrial machines to connect to modern machines or IoT solutions and provides immediate benefits of capturing insights from legacy or modern machines.

All these concepts allow architects to be incredibly flexible in the definition of their data platform. In chapter 9 we will deep dive more into these concepts and we will see how this pattern is becoming a standard.

Applying AI

Many organizations are thrust into designing a cloud data platform because they need to adopt Artificial Intelligence (AI) technologies — when designing a data platform, it is important to ensure that it will be future-proof in being capable of supporting AI use cases. In this section, we'll do a quick survey of industry uses of AI and then will delve into how the public cloud is used by organizations adopting AI.

Artificial Intelligence (AI) defines the class of problems you can solve when computers think/act like humans (see Figure 1-8). You can make computers think like humans by explicitly coding in the decision process that experts take ("expert systems"). For example, this is how the early chess-playing and medical diagnosis computer programs were built.

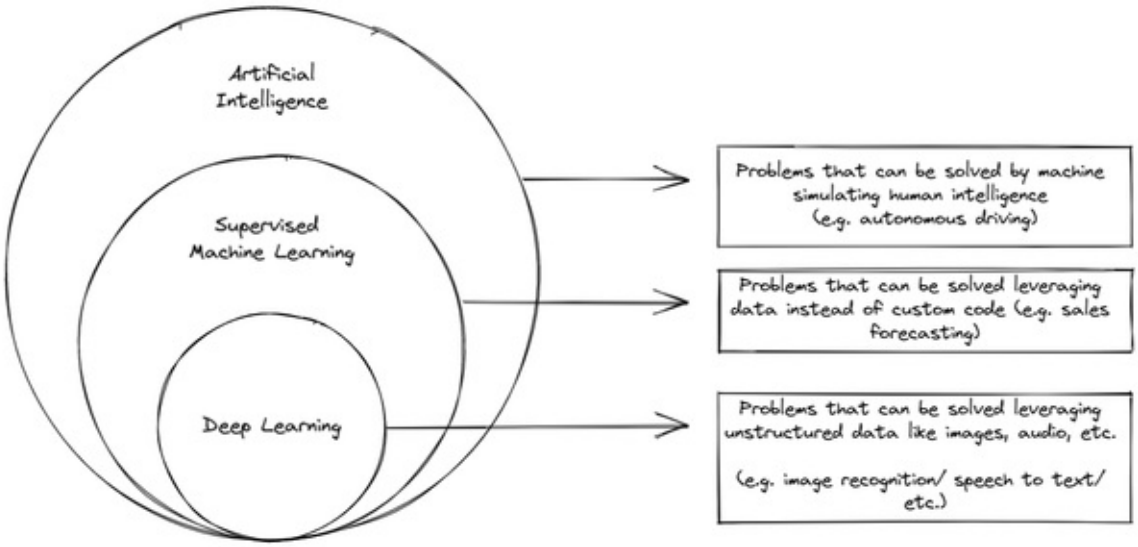


Figure 1-8. Machine Learning is a way of solving AI problems using data rather than custom code.

Considering the great impact AI is having on society and its diffusion within the enterprise environments, let's have a quick deep dive on what Machine Learning is and how it can be implemented in an enterprise environment. You will find a more deep dissertation in Chapters 10 and 11.

Machine Learning

These days a branch of AI called supervised machine learning has become tremendously successful to the point where the term AI is more often used as an umbrella term for machine learning. As shown in Figure 1-9, ML works by showing the computer program lots of examples where the correct answer (called labels) is known. The ML model is a standard algorithm (i.e. the exact same code) that has tunable parameters that "learn" how to go from the provided input to the label. Such a learned model is then deployed to make decisions on inputs for which the correct answer is not known.

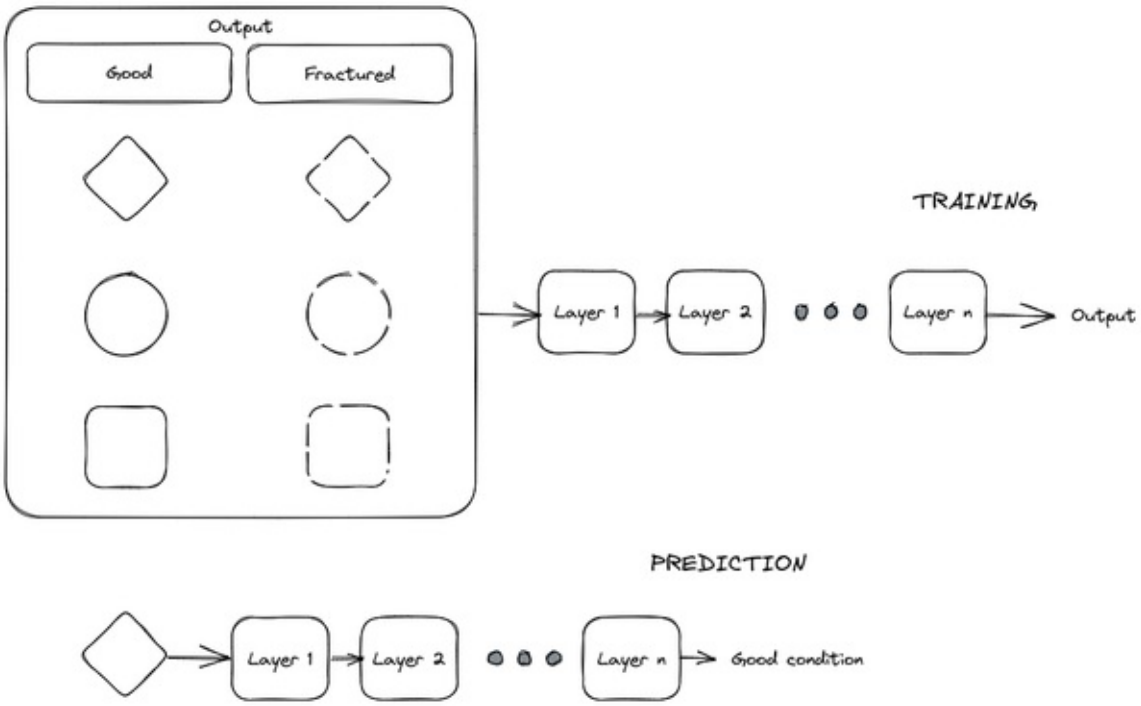


Figure 1-9. Training an ML model (top) and using it for predicting (bottom).

Unlike expert systems, there is no need to explicitly program the AI model with the rules to make decisions. Because many real-world domains involve human judgment where experts struggle to articulate their logic, having the experts simply label input examples is much more feasible that capturing their logic.

Modern day chess-playing algorithms and medical diagnostic tools use ML. The chess-playing algorithms learn from records of games that humans have played in the past[9] whereas medical diagnostic systems learn from having expert physicians label diagnostic data, as shown in Figure 1-10.

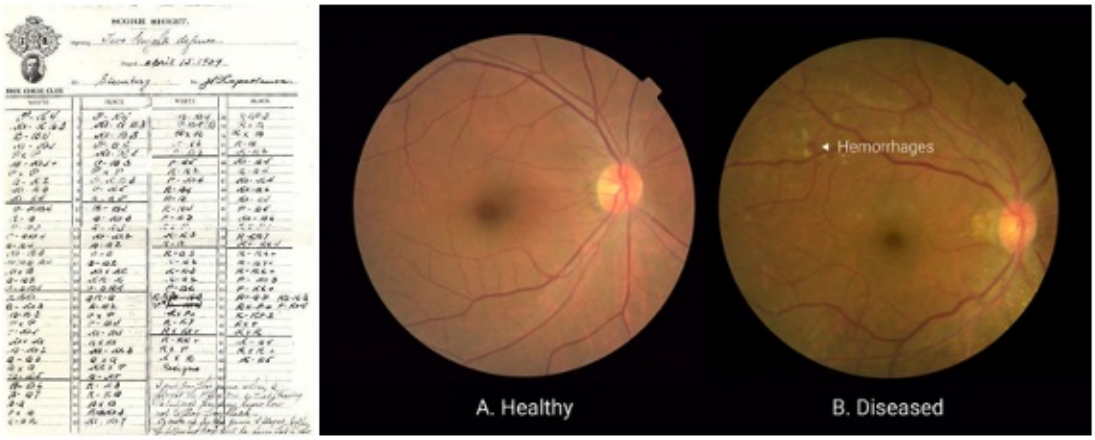


Figure 1-10. Machine learning systems are trained based on data. Left: chess games. Right: Retinal scans.

Images from Wikipedia and Google blog.

In order for these ML methods to operate, they require tremendous amounts of training data and readily available custom hardware. Because of this, organizations adopting AI start out by building a cloud data and AI platform.

Uses of ML

Given the rather technical nature of the explanation above, it may be hard to understand the reason that machine learning has taken off so dramatically. There are a few key reasons:

- **Data is easier.** It is easier to collect labeled data than to capture logic. Every piece of human reasoning has exceptions that will be coded up over time. It is easier to get a team of ophthalmologists to label a thousand images than it is to get them to describe how they identify that a blood vessel is hemorrhaging.
- **Retraining is easier**. When ML is used for systems such as recommending items to users or running marketing campaigns, user behavior changes quickly in order to adapt. It is important to continually train models. This is possible in ML, but much harder with code.
- **Better user interface**. A class of machine learning called deep learning (see Figure 1-8) has proven capable of being trained even on unstructured data such as images, video, and natural language text. These types of inputs are notoriously difficult to program against. This enables you to use real-world data as inputs – consider how much better the user interface of depositing checks becomes when you can simply take a photograph of a check instead of having to type all the information into a web form.
- **Automation**. The ability of ML models to understand unstructured data makes it possible to automate many business processes. Forms can be easily digitized, instrument dials can be more easily read, and factor floors can be more easily monitored because of the ability to automatically process natural language text, images, or video.

Given these advantages, it is not surprising that a Harvard Business Review article found that AI generally supports three main business requirements:

- automating business processes: typically automating back-office administrative and financial tasks.
- gaining insight through data analysis, and
- engaging with customers and employees.

Machine Learning increases the scalability to solve those problems using data examples and without needing to write a custom code for everything. Then ML solutions such as deep learning allows solving these problems even when that data consists of unstructured information like images, speech, video, natural language text, etc.

Why Cloud for AI?

A key impetus behind designing a cloud data platform might be that the organization is rapidly adopting Artificial Intelligence (AI) technologies such as deep learning. In order for these methods to operate, they require tremendous amounts of training data. Therefore, an organization that plans to build ML models will need to build a data platform to organize and make the data available to their data science teams. The ML models themselves are very complex, and training the models requires copious amounts of specialized hardware called Graphics Processing Units (GPUs). Further, AI technologies such as speech transcription, machine translation, and video intelligence tend to be available as SaaS software on the cloud. In addition, cloud platforms provide key capabilities such as democratization, easier operationalization, and the ability to keep up with the state-of-the-art.

Cloud infrastructure

The bottom line is that high-quality AI requires a lot of data – a famous paper titled “Deep Learning scaling is predictable, empirically” found that for a 5% improvement in a natural language model, it was necessary to train twice as much data used to get the first result. The best ML models are not the most advanced ones – they are the ones that are trained on more data. The reason is that increasingly sophisticated models require more data, whereas even simple models will improve in performance if trained on a sufficiently large dataset.

Large quantities of data are required to train modern ML models. To give you an idea, image classification models are routinely trained on one million images and leading language models are trained on multiple terabytes of data.

As shown in Figure 1-11, this sort of data quantity requires a lot of efficient, bespoke computation – provided by accelerators such as GPUs and custom Application-Specific Integrated Circuit (ASIC)[10]s called Tensor Processing Units or TPUs – to harness this data and make sense of it.

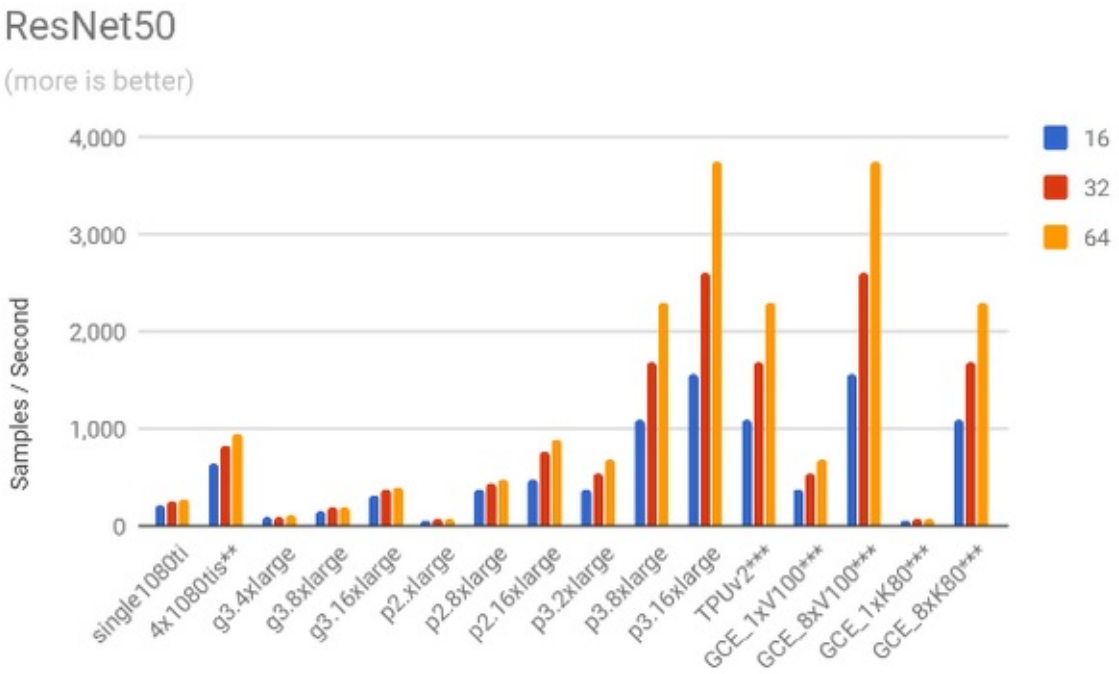


Figure 1-11. Machine learning performance increases dramatically with greater memory, more processors, and/or the use of TPUs and GPUs. Graph from AVP Project.

Many recent AI advances can be attributed to increases in data size and compute power. The synergy between the large datasets in the cloud and the numerous computers that power it has enabled tremendous breakthroughs in machine learning. Breakthroughs include reducing word error rates in speech recognition by 30% over traditional approaches, the biggest gain in 20 years.

Democratization

Architecting ML models, especially in complex domains such as time-series processing or natural language processing, requires knowledge of ML theory. Writing code for training ML models using frameworks such as PyTorch, Keras,

or TensorFlow requires knowledge of Python programming, and linear algebra. In addition, data preparation for ML often requires data engineering expertise, and evaluating ML models requires knowledge of advanced statistics. Deploying ML models and monitoring them requires knowledge of DevOps and software engineering (often terms MLOps). Needless to say, it is rare that all these skills are present in every organization. Given this, leveraging ML for business problems can be difficult for a traditional enterprise.

Cloud technologies offer several options to democratize the use of ML:

- **ML APIs.** Cloud providers offer pre-built ML models that can be invoked via APIs. At that point, a developer can consume the ML model like any other web service. All they require is the ability to program against REST web services. Examples of such ML APIs include Google Translate, Azure Text Analytics, and Amazon Lex – these APIs can be used without any knowledge of natural language processing.
- **Customizable ML models**. Some public clouds offer "Auto ML" which are end-to-end ML pipelines that can be trained and deployed with the click of a mouse. The Auto ML models carry out "neural architecture search", essentially automating the architecting of ML models through a search mechanism. While the training takes longer than if a human expert chooses an effective model for the problem, the Auto ML system can suffice for lines of businesses that don't have the capability to architect their own models. Note that not all Auto ML is the same – sometimes what's called Auto ML is just parameter tuning – make sure you are getting a custom-built architecture rather than simply a choice among pre-built models double checking there are various steps that can be automated (*e.g. feature engineering, feature extraction, feature selection, model selection, parameter tuning, problem checking, etc.*)
- **Simpler ML**. Some data warehouses (BigQuery and Redshift at the time of writing) provide the ability to train ML models on structured data using just SQL. Redshift and BigQuery support complex models by delegating to Vertex AI and Sagemaker respectively. Tools like DataRobot and Dataiku offer point-and-click interfaces to train ML models.
- **ML Solutions**. Some applications are so common that end-to-end ML

solutions are available to purchase and deploy. Product Discovery on Google Cloud offers an end-to-end search and ranking experience for retailers. Amazon Connect offers a ready-to-deploy contact center powered by machine learning. Azure Knowledge Mining provides a way to mine a variety of content types. In addition, companies such as Quantum Metric and C3 AI offer cloud-based solutions for problems common in several industries.

- **ML Building Blocks**. Even if no solution exists for the entire ML workflow, parts of it could take advantage of building blocks. For example, recommender systems require the ability to match items and products. A general purpose matching algorithm called Two Tower Encoders is available from Google Cloud. While there is no end-to-end back office automation ML model, you could take advantage of Form Parsers to help implement that workflow quicker.

These capabilities allow enterprises to adopt AI even if they don't have deep expertise in it, thereby making AI more widely available.

Even if the enterprise does have expertise in AI, these capabilities prove very useful because you still have to decide whether to buy or build an ML system. There are usually more ML opportunities than there are people to solve them. Given this, there is an advantage to allowing non-core functionality to be carried out using pre-built tools and solutions. These out of the box solutions can deliver a lot of value immediately without needing to write custom applications. For example, data from a natural language text can be passed to a pre-built model via an API call to translate text from one language to another. This not only reduces the effort to build applications but also enables non ML experts to use AI. On the other end of the spectrum, the problem may require a custom solution. For example, retailers often build machine learning models to forecast demand so they know how much product to stock. These models learn buying patterns from a company's historical sales data, combined with in-house, expert intuition.

Another common pattern is to use prebuilt, out-of-the-box models for quick experimentation and once the ML solution has proven its value, a data science team can build it in a bespoke way to get greater accuracy and hopefully more differentiation against the competition.

Real-time

It is necessary for the ML infrastructure to be integrated with a modern data platform because real-time, personalized machine learning is where the value is. As a result, speed of analytics becomes really important as the data platform must be able to ingest, process, and serve data in real-time, or opportunities are lost. This is then complemented by the speed of action. Machine-learning drives personalized services, based on the customer's context but has to provide inference before the customer context switches – there's a closing window for most commercial transactions within which the ML model needs to provide the customer with an option to act. To achieve this, you need the results of ML models to arrive at the point of action in real-time.

Being able to supply ML models with data in real-time and get the ML prediction in real-time is the difference between preventing fraud and discovering fraud. In order to prevent fraud, it is necessary to ingest all payment and customer information in real-time, run the ML prediction, and provide the result of the ML model back to the payment site in real-time so that the payment can be rejected if fraud is suspected.

Other situations where real-time processing saves money are customer service and cart abandonment. Catching customer frustration in a call center and immediately escalating the situation is important to render effective customer service – it will cost a lot more money to reacquire a customer once lost than it is to render them good service in the moment. Similarly, if a cart is at risk of being discarded, offering an enticement such as 5% off or free shipping may cost less than the much larger promotions required to get the customer back on the website.

In other situations, batch processing is simply not an effective option. Real-time traffic data and real-time navigation models are required in order for Google Maps to allow drivers to avoid traffic.

As you will see in the chapter on streaming, the resilience and autoscaling capability of cloud services is hard to achieve on premises. Thus, real-time ML is best done on the cloud.

ML Ops

Another reason that ML is better in the public cloud is that operationalizing ML is hard. Effective and successful ML projects require operationalizing both data and code. Observing, orchestrating, and acting on the ML lifecycle is termed *ML Ops*.

Building, deploying and running machine learning applications in production entail several stages, as shown in Figure 1-12. All these steps need to be orchestrated and monitored; if, for example, data drift is detected, the models may need to be automatically retrained. Models have to be retrained on a constant basis and deployed, after making sure they are safe to be deployed. For the incoming data you have to perform data preprocessing and validation to make sure there are no data quality issues, followed by feature engineering, followed by model training and ending with hyperparameter tuning.

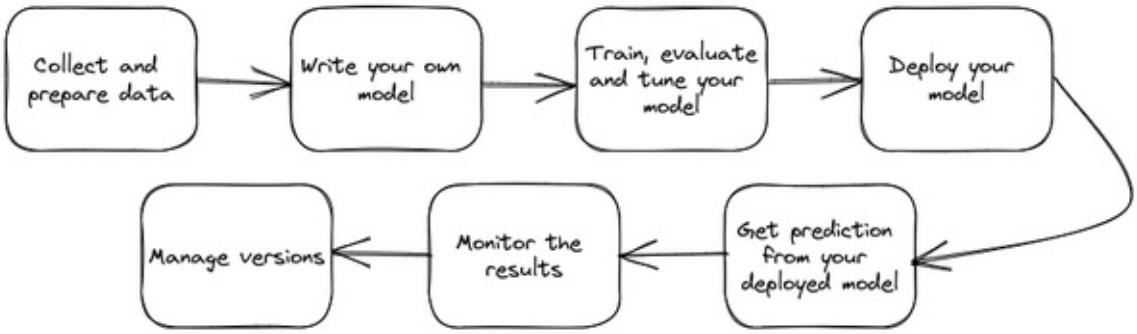


Figure 1-12. Stages of an ML workflow. Image from Google Cloud documentation.

In addition to the data-specific aspects of monitoring discussed above, you also have the monitoring and operationalization that is necessary for any running service. A production application is often running continuously 24/7x365, with new data coming in regularly. Thus you need tooling that makes it easy to orchestrate and manage these multiphase machine learning workflows, and run them reliably and repeatedly.

Cloud AI platforms such as Google's Vertex AI, Microsoft's Azure ML, and Amazon's Sagemaker provide managed services for the entire ML workflow (see Figures 1-13, 1-14, and 1-15). Doing this on-premises requires you to cobble together the underlying technologies and manage the integrations yourself.

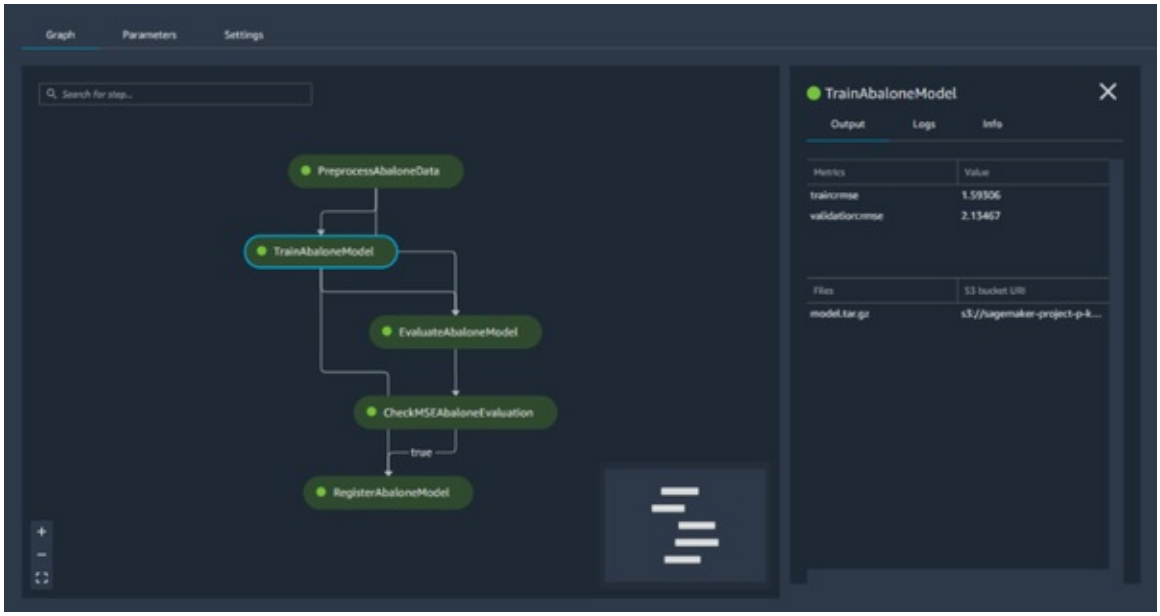


Figure 1-13. Managed pipeline in AWS Sagemaker

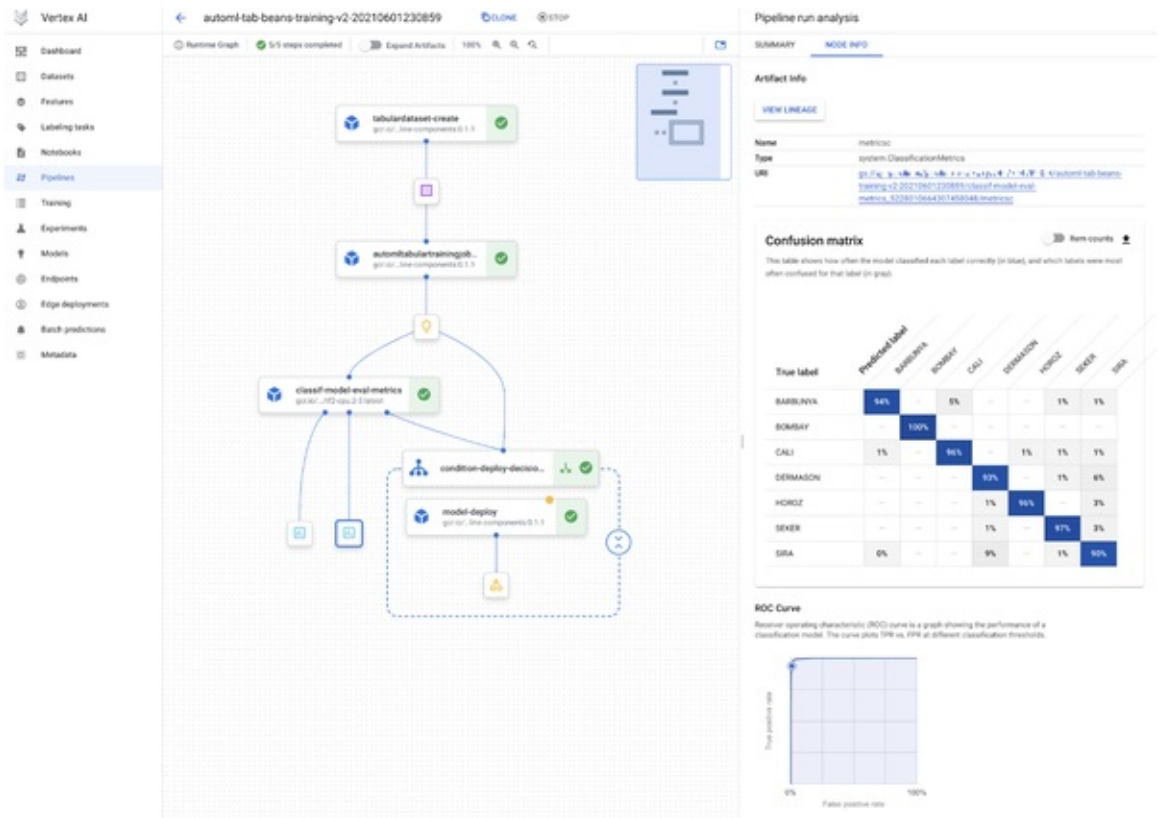


Figure 1-14. Managed pipeline in Google Cloud Vertex AI

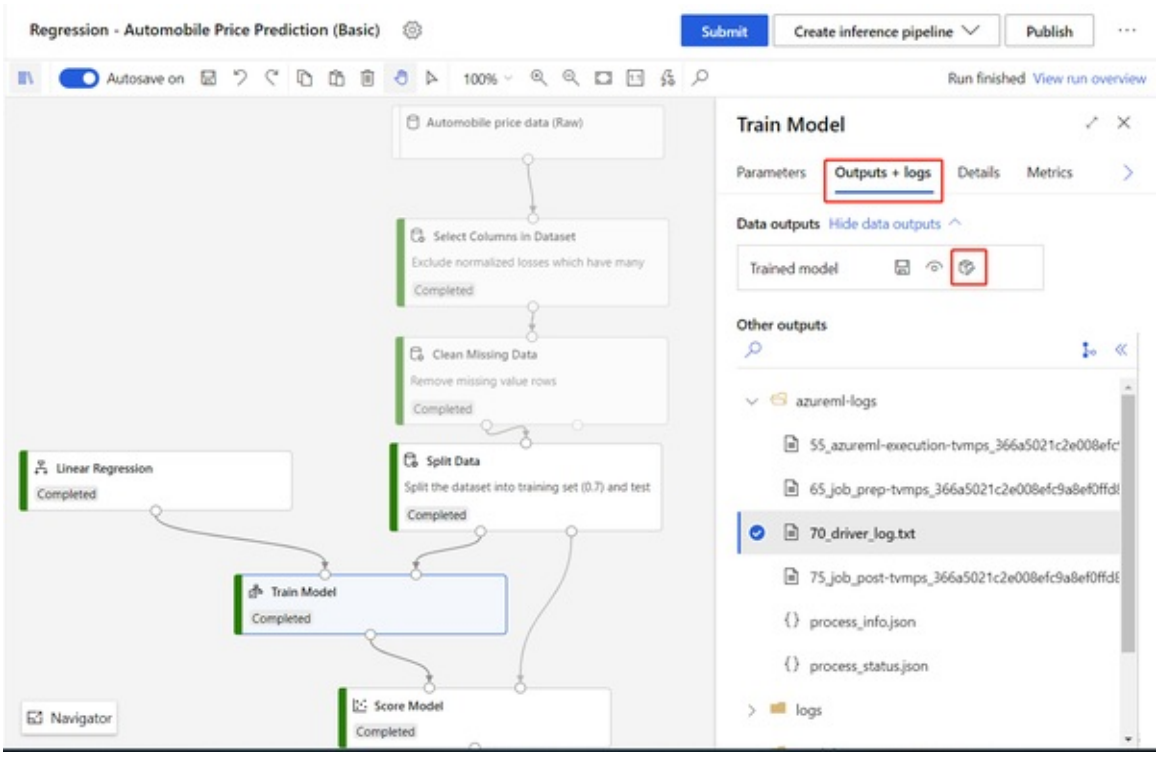


Figure 1-15. Managed pipelines in Azure ML.

At the time of writing this book, ML Ops capabilities are being added at a breakneck pace to the various cloud platforms. This brings up an ancillary point, that with the rapid pace of change in ML, you are better off delegating the task of building and maintaining ML infrastructure and tooling to a third party and focusing on data and insights that are relevant to your core business.

In summary, a cloud-based data and AI platform can help resolve traditional challenges with data silos, governance, and capacity while enabling the organization to prepare for a future where AI capabilities become more important.

Core Principles

When designing a data platform, it can help to set down key design principles to adhere to and the weight that you wish to assign to each of these principles. It is likely that you will need to make tradeoffs between these principles, and having a predetermined scorecard that all stakeholders have agreed to can help you make decisions without having to go back to first principles or getting swayed by the squeakiest wheel.

Here are the five key design principles for data analytics stack that we suggest, although the relative weighting will vary from organization to organization:

- **Deliver serverless analytics, not infrastructure**. Design analytics solutions for fully managed environments and avoid a lift-and-shift approach as much as possible. Focus on a modern serverless architecture to allow your data scientists (we use this term broadly, to refer to data engineers, data analysts, and ML engineers) to keep their focus purely on analytics and move away from infrastructure considerations. For example, use automated data transfer to extract data from your systems and provide an environment for shared data with federated querying across any service. This eliminates the need to maintain custom frameworks and data pipelines.
- **Embed end-to-end ML**. Allow your organization to operationalize machine learning end-to-end. It is impossible to build every ML model that your organization needs, so make sure you are building a platform within which it is possible to embed democratized ML options such as pre-built ML models, ML building blocks, and easier-to-use frameworks. Ensure that when custom training is needed, there is access to powerful accelerators and customizable models. Ensure that ML Ops is supported so that deployed ML models don't drift and become no longer fit for purpose. Make ML lifecycle simpler on the entire stack so that organizations can derive value from their ML initiatives faster.
- **Empower analytics across the entire data lifecycle.** The data analytics platform should be offering a comprehensive set of core data analytics workloads. Ensure that your data platform offers data storage, data warehousing, streaming data analytics, data preparation, Big Data processing, data sharing and monetization, Business Intelligence (BI), and ML. Avoid buying one-off solutions that you will have to integrate and manage. Looking at the analytics stack much more holistically will, in return, allow you to break down data silos, power applications with real-time data, add read-only data sets, and make query results accessible to anyone.
- **Enable Open Source Software (OSS) technologies.** Wherever possible, ensure that open source is at the core of your platform. You want to ensure

that any code that you write uses OSS standards such as Standard SQL, Apache Spark, TensorFlow, etc. By enabling the best open source technologies, you will be able to provide flexibility and choice in data analytics projects.

- **Build for growth.** Ensure that the data platform that you build will be able to scale to the data size, throughput, and number of concurrent users that your organization is expected to face. Sometimes, this will involve picking different technologies (e.g. SQL for some use cases and NoSQL for other use cases). If you do so, ensure that the two technologies that you pick interoperate with each other. Leverage solutions and frameworks that have been proven and used by the world's most innovative companies running their mission critical analytics apps.

Overall, the factors above are listed in the order that we typically recommend them. Since the two primary motivations of enterprises in choosing to do a cloud migration are cost and innovation, we recommend that you prioritize serverless (for cost savings and freeing employees from routine work) and end-to-end ML (for the wide variety of innovation that it enables).

In some situations, you might want to prioritize some factors over others. For startups, we typically recommend that the most important factors are serverless, growth, and end-to-end ML. Comprehensiveness and openness can be sacrificed for speed. Highly regulated enterprises might favor comprehensiveness, openness, and growth over serverless and ML (i.e. on-premises might be necessitated by regulators). For digital natives, we recommend in order end-to-end ML, serverless, growth, openness, and comprehensiveness.

Summary

This was a high-level introduction to data platform modernization. We looked at the evolution of data processing, starting with the ETL, Data warehousing and data lakes. We covered the cloud data platform, how AI works in the Cloud, and why building a unified analytics platform enables you to do AI with the masses. Finally, we looked at data mesh and lakehouse together with the hybrid cloud and edge computing. The key takeaways from this chapter are:

- Traditionally, organizations' data ecosystems consist of independent solutions that lead to the creation of silos within the organization.
- Data movement tools can break data silos, but impose a few drawbacks: latency, data engineering resource bottlenecks, maintenance overhead, change management, and data gaps.
- Centralizing control of data within IT leads to organizational challenges. IT departments don't have necessary skills, analytics teams get poor data and business teams do not trust the results.
- All data can be exported from operational systems to a centralized data lake for analytics. The data lake serves as the central repository for analytics workloads and for business users. The drawback, however, is that business users do not have the skills to program against a data lake.
- Data warehouses are centralized analytics stores that support SQL, something that business users are familiar with.
- Organizations are building a cloud data platform to obtain best of breed architectures, handle consolidation across business units, scale on-prem resources, or plan for business continuity.
- The ethos of the data lakehouse is about believing that all users can and should be data users, regardless of technical capabilities. By providing an underlying and centralized effort to make the data accessible, different tools can sit on top of the lakehouse that meets each user's capabilities.
- Data Mesh introduces a way of seeing data as a self-contained product. Distributed teams in this approach own the data production and serve internal/external consumers through well defined data schema.
- Cloud and hybrid cloud environments are required to meet the realities of the enterprise world. They provide a pragmatic approach presenting real world architectures adopted by enterprises.
- Cloud data platform leverages modern approaches and aims to enable data-led innovation through replatforming data, breaking down silos, democratizing data, enforcing data governance, enabling decision-making in real-time and using location information, and moving seamlessly from

descriptive analytics to predictive and prescriptive analytics.

- The ability of the public cloud to provide ways to manage large datasets and provision GPUs on demand makes it indispensable for machine learning. In addition, cloud platforms provide key capabilities such as democratization, easier operationalization, and the ability to keep up with the state-of-the-art.
- The five core principles of a cloud data platform are to prioritize serverless analytics, end-to-end ML, comprehensiveness, openness, and growth. The relative weights will vary from organization to organization.

Now that you have this holistic view and understand why you might want to design and implement a modern data platform, you're ready to dive into more detail about all these core concepts. Up next, you will learn the six strategic steps you can take to foster data innovation within your organization.

---

1 *https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf*

2 *https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf*

3 For example, at the time of this writing, native BigQuery storage is more expensive than Google Cloud Storage.

4 You can find more information on this suggested book from O'Reilly: *https://learning.oreilly.com/library/view/data-mesh/9781492092384/*

5 *https://aws.amazon.com/blogs/startups/how-uber-survives-its-busiest-nights-of-the-year/*

6 *https://eng.uber.com/building-ubers-fulfillment-platform/*

7 *https://techcrunch.com/2020/12/15/twitter-taps-aws-for-its-latest-foray-into-the-public-cloud/*

8 *https://venturebeat.com/2021/02/25/inside-twitters-growing-interest-in-google-cloud/*

9 Recent machine learning systems such as Alpha Go learn by looking at games played between machines themselves: this is an advanced type of machine learning called reinforcement learning, but most industrial uses of ML are of the simpler supervised kind.

10 An application-specific integrated circuit (ASIC) is an integrated circuit (IC) chip customized for a particular use, rather than intended for general-purpose use.

Chapter 2. Strategic steps to innovate with data

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the second chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

The reason your leadership is providing the funds for you to build a data platform is very likely because they want the organization to innovate. They want the organization to discover new areas to operate in, to create better ways of operating the business, or to serve better quality products to more customers. Innovation of this form typically happens through better understanding of customers, products, or the market. Whether your organization wants to reduce user churn or acquire new users or predict the repair cycle of a product or identify whether a lower cost alternative will be popular, the task starts with data collection and analysis. Data is needed in order to analyze the current state of the business, identify shortcomings or opportunities, implement ways to improve upon the status quo, and measure the impact of those changes. Often, business unit-specific data has to be analyzed in conjunction with other data (both from across the organization, and with data from suppliers and marketplaces). Therefore, when building a data platform, it is important to be intentional and keep the ultimate "why" (of fostering innovation) firmly in mind.

In this chapter, you will learn the seven strategic steps to take when you are building a platform to foster innovation, why these steps are essential, and how

to achieve them using present-day cloud technologies. We can think of these steps as forming a pyramid (as depicted in Figure 2-1), where each step serves as the foundation for the steps that follow. For example, Step 1 which involves defining the strategies for building a data platform (data lake and/or data warehouse) on the cloud is foundational to Steps 2-6, whereas Step 7 which involves treating the data as a real product is the pinnacle. Being able to master the data to get the most from it and bring the organization to take meaningful decisions based on facts is not an easy journey: it requires a significant investment of time and a great commitment of the entire company over all the involved steps but the competitive advantage you can get is much more rewarding.

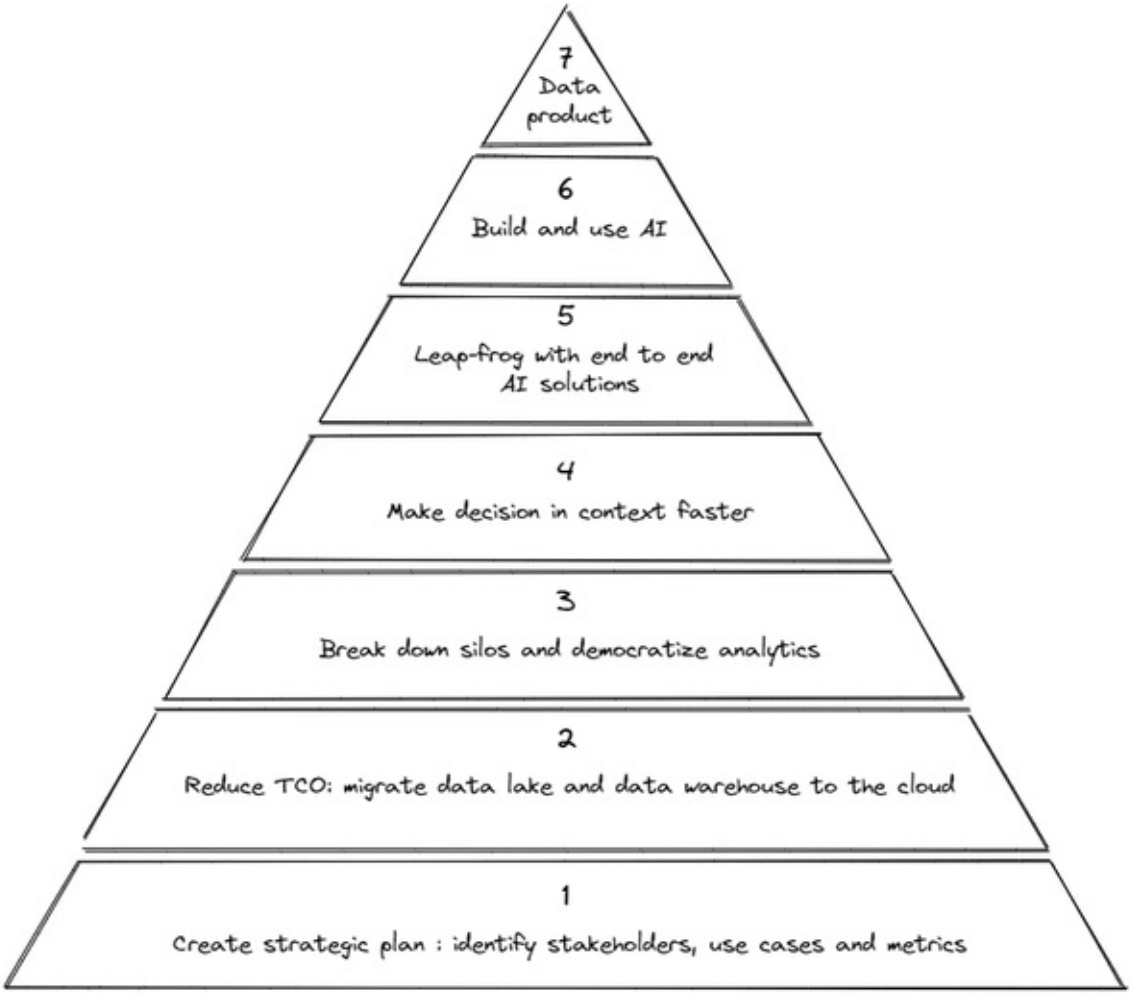


Figure 2-1. The 7-steps journey we suggest to build a cloud data and AI platform

In this chapter, we will crystallize the concepts that underlie all the steps.

Detailed implementation of the steps we will discuss in later chapters. For example, while we will describe the concept of breaking down silos in this chapter, we'll describe the architecture of the analytics hub or data mesh approaches to do so in Chapters 5, 6 and 7.

Step 1: Strategy and Planning

For the final six steps to be successful, you need to first formulate a strategic plan wherein you identify three main components:

- *Goals*: What are the ambitions that the organization has when it comes to making the best use of data? It is important to dig deeper and identify goals that go beyond cost savings. Specifically, it is important to identify the use cases for data and the metrics by which you can know the transformation has been successful.
- *Stakeholders*: Who are the people in the organization who have the mandate to sponsor and drive deeper transformations? It is important to ensure that you bring together all these stakeholders — in our experience, IT projects tend to be under-resourced and always in imminent risk of failure whereas business driven projects have longer-term executive support and funding. Business projects also have a higher return on investment.
- *Change management process*: how to effectively cascade and communicate the approach to the entire organization to get sponsorship of the final users

This strategic planning needs to be periodically revisited. Are the goals still the same? Are there new stakeholders who need to be briefed? Is there discontent brewing in the ranks?

Let's look at these three components one-by-one.

Strategic goals

You should be clear-headed about your goals when building a data and AI platform. Very often, stakeholders will frame goals in terms of the limitations of the current platform. For example, the goal might be stated as "*we want to be able to create monthly statements for our 3 million customers within 3 hours*" if

your current most painful problem is that reporting workloads cause cascading outages. However, you don't want your goal in building a platform to be narrowly defined by a single use case. Instead, you want to start from a clear-headed view of the strategic goals you want to achieve.

Design your system based on the strategic direction of the business. What is the desired turnaround time for shipping information to be provided for new purchases? What is the anticipated growth in customer numbers? Will the mix of customers who arrive via mobile vs. via brokers change over time? What is the expected headcount in the information technology team? Does the business wish to enable field personnel to make more decisions? Do you want to send out monthly statements, or do you want to dynamically generate reports of historical activity on demand? The inability of the current platform to support these needs will naturally fall out from these strategic concerns, but it allows you to holistically frame the requirements of the system instead of being tied down by a single use case and old ways of thinking. This also helps you communicate the need for the system to non-technical executives.

If possible, get numeric estimates for these strategic business goals over the next 3-5 years to help you make cost-effective decisions. For example, it can be tempting to simply say "*queries should run as fast as possible*," but that is a recipe for building an over-engineered and costly system. Instead, if you know that you will have, at peak, 1500 concurrent queries each processing 100 GB of data that need to take less than 10 seconds to run, you can choose technology that achieves your goal without breaking the bank. This also works in reverse. Knowing that the business is acquiring customers at a 60% year-over-year basis, it might be clear that the clickstream dataset is poised to be on the order of 1 TB/day. This will prevent you from making shortsighted decisions that need to be unwound.

This is often limited by the time horizon over which you can foresee business growth. It is also subject to real-world events. For example, the COVID pandemic of 2020 upended many businesses' plans and accelerated a move towards digital and omnichannel experiences. Sometimes, you build a system that has to be scrapped and rebuilt, but you can minimize how much this happens by thinking expansively about contingencies.

We find that most organizations end up prioritizing the following goals because

they strike the right balance between pragmatism and ambition:

1. Reduce the cost of operating the data and AI platform. In particular, linearly growing information technology costs with growth in data set sizes can be unsustainable.
2. Increase the speed of innovation by accelerating the time to get value from new projects. Experimenting on new ideas should not face months-long delays procuring machines, installing software, getting access to datasets, etc.
3. Democratize insights from data. Enable domain experts and people in the field to interact with the data directly and gain insights.
4. Incorporate predictive analytics, not just descriptive analytics, into decision making. For example, rather than simply measuring the amount of material used last week, predict the amount of material needed over the following week based on the amounts used in the recent past.

The relative prioritization varies between businesses (as it should). Many startups and digital natives emphasize speed of innovation and flexibility to grow whereas many mature enterprises emphasize cost over flexibility. For example, a startup might use a petabyte-scale data warehouse even though its data is small because it expects to see 10x annual growth. A more mature business might choose to use batch processing because it is less expensive than stream processing. These different starting points impact the type of innovation and growth possible – the petabyte scale data warehouse might allow the startup to target recommendations based on every payment transaction as they happen, whereas the more mature business might only send recommendation emails daily and only to customers who made large orders.

Identify stakeholders

A solid definition of the strategy starts with the correct requirements gathering. To do that successfully it is incredibly important to identify the right people within the organization who are able to understand the needs and effectively collaborate across all the different business units (BUs) to reduce the risk of choosing the wrong approach and solution. But who are the right people?

Are we talking about people coming from the business (e.g. CEO/ CFO) or is it better to rely on the IT team (e.g. CIO)? Well, it really depends on the organization. In our experience we have seen so many different approaches, but the one common theme is that this kind of transformation journey usually has the highest rate of success when supported directly by the business. Why? Many times, the Information Technology (IT) organization may be mandated only with keeping things running and reducing costs year over year. If your stakeholders are only in IT, their incentives are very different than if your group of stakeholders includes people from business units who need to develop new products, reach more customers, or fundamentally change the business.

By making the definition of a new data platform more than just a pure IT activity, you can raise the visibility of the transformation and ensure that the new platform allows the organizations to solve so many business problems that were not addressable before (e.g. real time analysis).

Even within the area of the company that supports the initiative, it is crucial to have full commitment from all the people involved. But these may be very busy people with insufficient time and expertise to spend on this transformation project. Therefore, another key question to ask is: does the company have enough internal people to support the initiative? Or do you need to get someone outside the company to steer the project in the right direction? It is not just a matter of technical knowledge but also a matter of leadership and management to ensure that the design and implementation of the data platform are successful and that the business outcomes are positive.

The importance of change management

Once you have identified the goals and the people who should steer toward the objectives, next you must define a strategy for the change management. Organizations can have the most ambitious goals supported by the most influential people, but they will have tremendous difficulty implementing the project if there is no clear mission to effectively cascade the message down the chain.

When embracing a project like data-driven transformation, we have seen so many companies forgetting to put the focus on the *cultural aspect* of the transformation and treating it as a merely technological project. Business users,

and employees who will leverage the data platform in general, should be ready to embrace the change and this can be achieved only via adequate processes and right skills.

As shown in Figure 2-2, change management is an intersection among People, Technology and Process:

- *People and technology*: training, training, training! It is incredibly important to work on an extensive training program to enable people to leverage the new assets made available within the organizations. It can be company driven (*internally delivered*) or it can be partner driven (*externally delivered*). The greater the focus that organizations have in upskilling their workforce, more effective will be the realization of the overall business journey.
- *People and process:* It is always a matter of leadership. Leaders within the company have to foster the overall message: when people are supported by a mandate from the leadership (and this is another link to the fact that stakeholders are super important!) the level of adoption increases. We have seen so many projects failing because of the lack of proper support from the same people who launched the initiative. It is important that leaders work on several internal campaigns to properly spread the message across the company helping people to embrace the change. Some common questions to ask are: How are the teams structured? Have they got executive sponsorship? How are cloud projects budgeted, governed, assessed?
- *Process and technology:* This is related to the ability of the organization to take advantage of cloud-native services adoption for scaling. Some common questions are: what is the extent to which the organization abstracts away the infrastructure with managed and serverless cloud services? What is the level of implementation of the automation processes and the programmable infrastructure code that runs through it? Automation is a critical asset for success because from one side it reduces the shuman effort and in parallel it helps in making low risk and frequent changes that is the key ingredient for innovation.

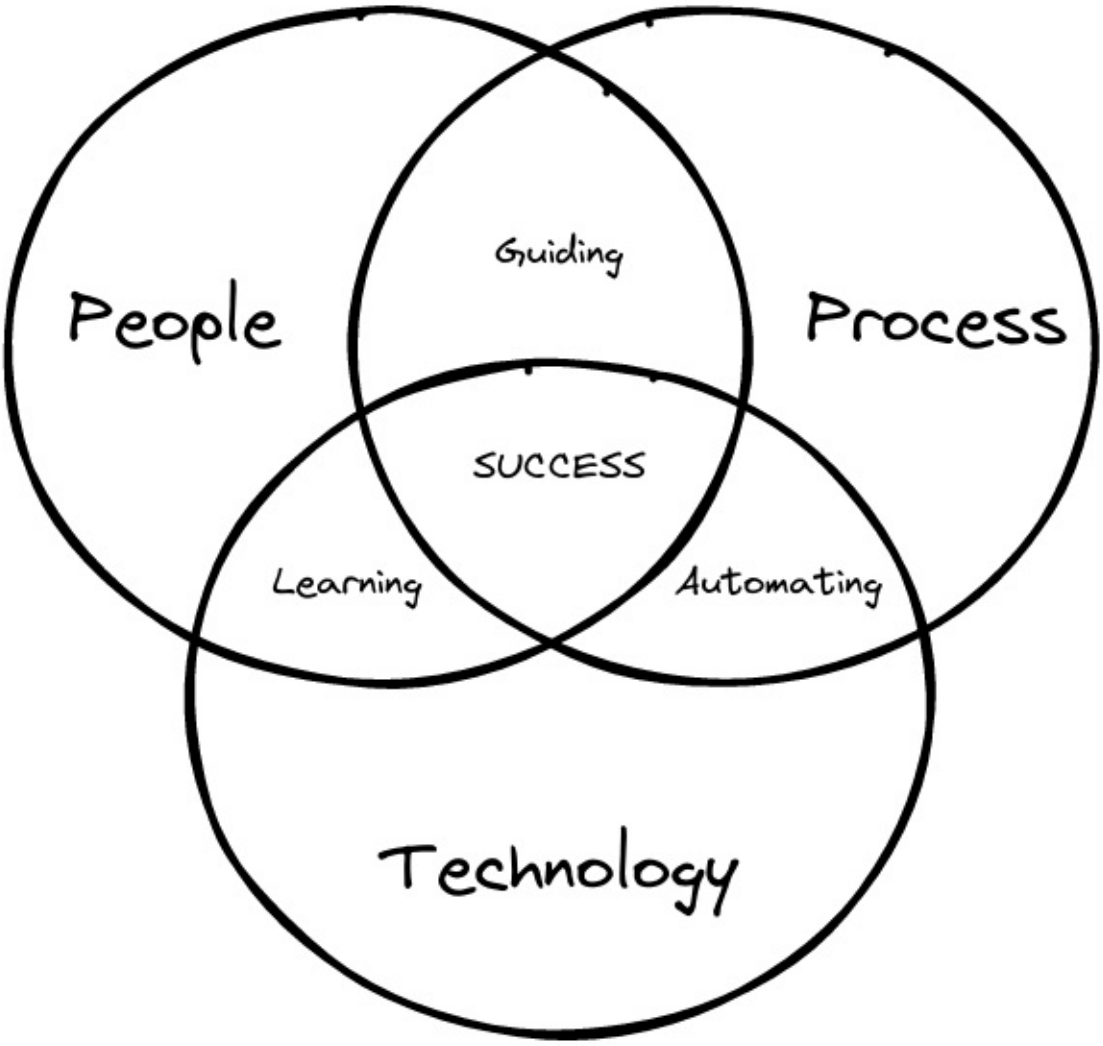


Figure 2-2. People, Process and Technology working together toward success.

Success requires all three elements to work together cohesively. Many organizations have achieved this by setting up a dedicated group of people called "*Cloud Center of Excellence (CCoE)*" whose goal is to set the direction and drive the company towards people-process-technology harmony. We will revisit this concept with a concrete example in Chapter 12.

Craft the digital transformation journey

Be clear about the goals and objectives of the company (the mission) and consider how to better fulfill the mission by employing a more modern and capable platform. This is a case where we recommend taking it one step at a time.

Based on our experience helping many organizations go through a digital transformation journey, there are five steps in the journey:

1. Reduce total cost of ownership and operations by migrating data
2. Break down silos, democratize analytics, and build a data culture
3. Make decisions in context, faster
4. Incorporate end-to-end packaged AI solutions
5. Empower data and ML teams with scaled AI platforms

As shown in Figure 2-1, step 2 forms the foundation for step 3, both steps 2 and 3 are needed to be successful in step 4. However, you don't need to move all your data to the cloud before you move on to Step 3. Instead, you can do this one workload or set of workloads at a time. Doing it by workload adds the complication that you may need to maintain data into two places until migration is complete.

The rest of the chapter delves into each of the slices of the pyramid, starting with the one marked as Step 2.

Step 2: Reduce Total Cost of Ownership adopting a cloud approach

The first step for most enterprises is to define (*and find*) a budget. Moving your enterprise data warehouse and data lakes to the cloud can save you anywhere from 50% to 75% of the cost of a legacy implementation. Let's look at why this is, and how you can set yourself up for maximum savings.

Why Cloud costs less

The reason that migrating data to the cloud can save you money is because of several factors:

1. **Reduction in operating costs.** On premises, your company bears the entire costs to operate the system, and much of the maintenance is done manually. Cloud providers (who we will call hyperscalers), on the other hand, have

built incredibly cost-efficient ways to manage large clusters of machines. Amazon, for example, brings their expertise running one of the world's largest and most reliable websites in a very low-margin business to provide cloud infrastructure at a very low cost. Similarly, Google runs nine services that have to be run very efficiently because over a billion users of each of these services (like Search, GMail, YouTube) use them for free. Users of the public cloud benefit from the low cost due to the high degree of automation built into the operation of cloud services. The majority of cloud services do not require maintenance because the majority of the activities (e.g. hardware maintenance, security checks, packages updates, etc.) are managed automatically under the hood.

2. **Right-sizing of compute and storage**. Instead of purchasing equipment that matches anticipated peak usage, cloud providers allow you to scale computational resources according to demand and usage. For example, you could start your systems small and increase the number of machines as the number of reports to be created grows over time. This benefit applies both to services from the cloud providers (such as AWS EMR, Google Cloud Dataproc, and Azure HDInsight) and to third-party tools such as Databricks or Teradata Vantage that run on top of cloud infrastructure.

3. **Autoscaling of workloads.** Many cloud services (e.g. Azure Cosmos DB, AWS Aurora, Google Cloud Composer, Snowflake, Actian Avalanche) allow you to assign more machines during peak hours and fewer machines during off-hours. Note that we said fewer machines, not zero. Although it can be tempting to completely shut down services during off-hours, consider whether you really want to retain that brick-and-mortar model. Your company's website, hopefully, is not shut down at night. Your backend systems should not be, either. Retaining the ability to service the occasional, urgent request at night tends to pay off in dramatic fashion.

4. **Serverless workloads**. A few modern cloud services (e.g. BigQuery on GCP, Athena on AWS, Azure Functions) are serverless – you get to submit code and the system takes care of executing it for you. Think of a serverless system as an autoscaling cluster that is shared among all the hyperscaler's customers. Thus, serverless systems bring the cost benefits of operating cloud infrastructure all the way up the stack. Because labor tends to be the

most expensive line item on an IT budget, serverless cloud services lead to the most cost-effective solutions. Note that many vendors position their autoscaling services as “serverless”, so you should verify that the service in question is truly serverless – you get the cost benefits of serverless only if it is multitenant. If the cluster belongs to you, you will need to manage the cluster (for example, find out who runs what jobs on the cluster and when), and so, you are not getting the labor cost advantage that accrues to serverless solutions.

How much are the savings?

In Figure 2-3, we have shown the results of a Proof of Concept (POC) we carried out on a real-world data lake. We first moved the workload as-is to the cloud, then put it on autoscaling infrastructure and finally modernized it. We measured what the cloud costs would be. Because this was a POC, the systems were not run long enough in these configurations to measure personnel costs to operate these systems.

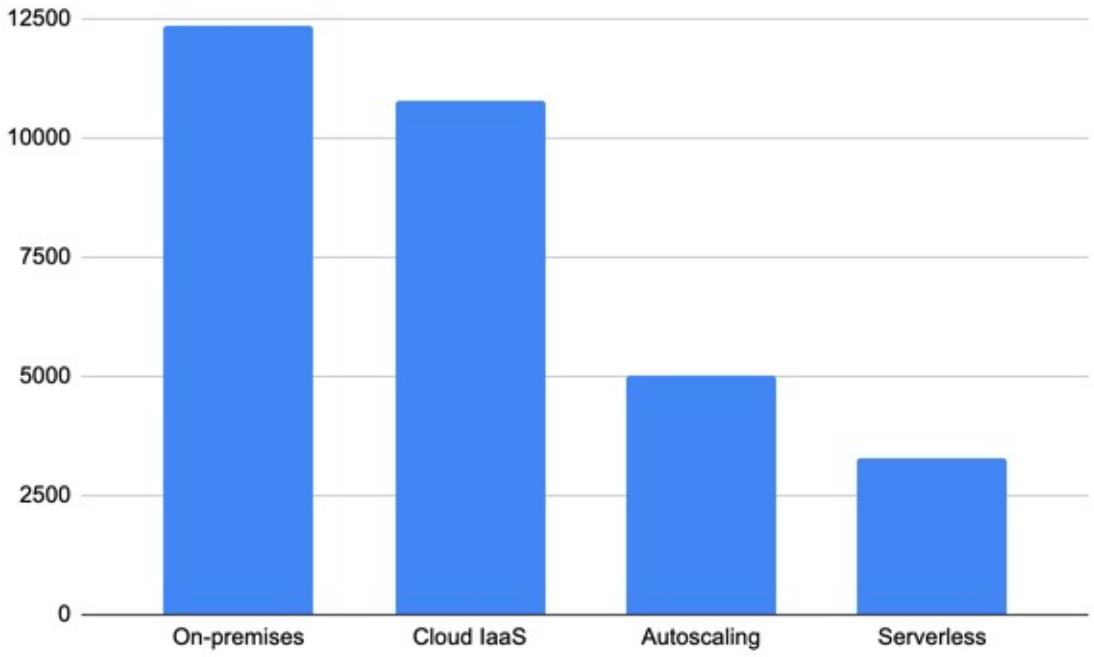


Figure 2-3. Monthly cost (in USD) of operating a 100-node data lake (measurements from a proof of concept conducted by the authors on a real workload). Personnel costs to operate the system are not included in the analysis.

Actual savings will vary, of course, depending on the specific details of your platform and workload. As a ballpark estimate (see Figure 2-3), simply moving a workload as-is to the cloud tends to provide a savings of about 10%. Adding right-sizing tends to add an additional 5%. Autoscaling tends to add a 40% savings, and serverless tends to tack on an additional 30%. If you take advantage of all these savings, for example by changing a workload that uses Hive on-premises to using BigQuery on the cloud, the cost savings can be as high as 95%.

When does Cloud help?

Adding fuel to all of this is basic cloud economics. Ignoring the impact of pricing discounts, it costs the same to run a job on 10 machines for 10 hours as it does to run the job on 100 machines for 1 hour or to run the job on 1000 machines for six minutes. The ability to give you access to 1000 "warm" instances for six minutes out of a multitenant cluster is what makes serverless so cost-effective. Of course, it is not just cost – what is the business benefit of taking a job that takes 10 hours today and having the results be available in six minutes? Very often, what happens is that an operation that used to be done once a month gets done every day because the more timely decision carries a business value that far exceeds the increase in computational cost.

What kind of workloads don't benefit from a move to the cloud? Any workload that is consistent (i.e. does not need to grow and does not have spikes) and large-scaled. An example of a workload that doesn't benefit from moving to the cloud is that of running a global-scale numerical weather forecasting model. Today, this requires specialized hardware (shared memory, high-speed interconnects) which obviates the hardware cost advantage of cloud, specialized operations personnel who understand weather models, and experiences almost the same load day-in and day-out.

Ephemeral and spiky workloads tend to benefit the most from a cloud move mostly by reducing the need to spend valuable time doing resource provisioning. Ephemeral and spiky workloads will also benefit from autoscaling and the cloud economics of pay-for-what-you-use. So, when prioritizing a cloud move based on cost, start with the ephemeral and spiky workloads.

Step 3: Break down silos

Once you have migrated all your data to the cloud, you can start to look at how to get more value from it. One of the best ways to start getting value from data is to break down data silos. In other words, avoid having multiple, disjointed and sometimes invisible datasets. We are now in the second layer of the Figure 2-1 pyramid.

Breaking down data silos involves striking the right balance between decentralization and value. Decentralization is good because data quality reduces the farther away from the domain experts the data gets. So, you have to give domain experts control over the data. Don't centralize data in IT. At the same time, remember that you get the greatest AL/ML value by combining data that you have across your organization and even data shared by partners. Breaking silos between different parts of the organization is key.

How do you square this circle? How do you allow different parts of the organization to maintain control of their data, but provide access to the data to anyone who's allowed to do so? We explore how below.

Centralizing data

What will not work is to have each team put its data on a cluster and then manage access to that cluster. Instead, centralize data on the cloud. Note that a centralized storage location does not mean a centralized ownership structure. For example, the data could be stored on Azure Blob Storage, but every department could put "their" data in "their" bucket.

Access to data should be managed through the cloud provider's Identity and Access Management (IAM). Avoid the temptation of carrying over on-premises authentication mechanisms like LDAP or Kerberos to the cloud. If you need to maintain a hybrid infrastructure, map on-premises Kerberos roles to Cloud IAM. If you are using software that needs its own authentication mechanism (eg. MySQL), use an authentication proxy to avoid proliferation of login mechanisms. Avoid using software that provides neither IAM nor proxies to IAM. Having data and insights locked in will cause you a lot of heartache in the long term whatever that software's near-term benefits.

If you are multi-cloud, ensure that you standardize on a SaaS single sign-on

(SSO) authentication mechanism such as Okta and then map the Okta authentication to each of the clouds' IAMs. An alternative approach, if you have a "main" cloud is to federate that cloud's IAM with others: for example, you might federate Google Cloud Identity to use Azure Active Directory if Azure is your main cloud, but you wish to have some data workloads on Google Cloud.

Make sure that access to data is auditable based on the actual user making the request, not through service accounts that break the link back to an individual user. Because privacy and government regulations on access to data continue to become stricter, avoid getting locked into any software that operates in their own cloud project or reads data in an opaque way.

What this means is that each department would manage their data, and classify their data. For example, they could tag some of the columns in their data warehouse as containing financial data. The data governance policy might be that the only people allowed to view financial data are people in accounting, and vice presidents and above. This policy is enforced by the IT department (not the data owner) using cloud identity access management.

Don't fall into the temptation of centralizing the control of data to break down silos. Data quality reduces the further away from the domain experts you get. You want to make sure that domain experts create datasets and own buckets. This allows for local control but access to these datasets will be controlled through Cloud IAM roles and permissions. The use of encryption, access transparency, and masking/dynamic views can help ensure orgwide security even if the responsibility of data accuracy lies with the domain teams.

Choosing storage

Where should you store the data?

Store structured and semi-structured data in a location that is optimized for SQL analytics. Google BigQuery, AWS Redshift, Azure Synapse, Actian Avalanche, Snowflake, etc. are good choices. These tools allow you to centralize the data and still have different datasets managed by different teams but as part of the same larger data warehouse.

Another option is to store the structured or semi-structured data in an open format such as Parquet using a distributed table format such as Apache Iceberg

or Databricks Delta Lake on top of a cloud blob store such as AWS S3. While you may take a bit of a performance hit in SQL analytics when you store data in these open formats (as opposed to native storage mechanisms like Capacitor in BigQuery), the lower cost of storage and the flexibility to support non-SQL analytics (such as machine learning) might make this a worthwhile tradeoff.

Unstructured data should be stored in a format and location that is optimized for reading from a variety of computational engines – Spark, Beam, TensorFlow, PyTorch, etc. Aim to use standard cloud-friendly formats such as Parquet, Avro, TensorFlow Records, and Zarr and store the files on Google Cloud Storage, Azure Blob Storage, or AWS S3. CSV and JSON are human readable and relatively easy to process, and so have their place as well.

If the data is held by a fully managed service, make sure that you have direct, live access to the data without having to go through its query interface. When using Databricks, for example, you have the option to store data as Apache Parquet files on any cloud storage. BigQuery, for example, offers a Storage API to directly read the columnar data instead of going through its query interface or exporting data.

Our recommendation to choose the storage layer based on type of data might seem surprising. Shouldn't you store "raw" data in a data lake, and "clean" data in a data warehouse? As mentioned in Chapter 1, data lakes and data warehouses are converging and it doesn't make sense to treat them separately any more. Instead, you want to think about the characteristics of the data and the type of processing that you will want to do on the data. Some of your "raw" data, if it is structured, will be in Redshift/BigQuery and some of your final, will reside in a blob storage service.

Typically, each analytics dataset or bucket will be in a single cloud region (or multi-region such as EU or US). We term such a storage layer a "*distributed data layer*" to avoid getting sidetracked by the lake vs. warehouse debate.

Encourage teams to provide wide access to their datasets ("default open"). Data owners control access but are responsible for classifying the data subject to org-wide data governance policies. Specialized teams may also have the ability to tag datasets (for privacy, etc.). Permissions to their datasets are managed by the data owners. Upskill your workforce so that they are discovering and tagging datasets and building integration pipelines to continually increase the breadth

and coverage of your distributed data layer.

Semantic layer

One problem you will run into when you build a democratized data culture is that you will start to see analytics silos. The same variable may be called by different column names in different parts of the organization. Each time a Key Performance Indicator (KPI) is calculated is one more opportunity for it to be calculated in a wrong or inconsistent way. So, encourage data analytics teams to build a semantic layer[1] (so that vocabularies can be standardized and KPIs can be computed once and reused everywhere else) and apply governance through that semantic layer – see Figure 2-4.

Tools like Looker, Informatica, and Collibra can help in defining and standardizing the semantic layer. Using such tools has the advantage of being multi-cloud and spanning between on-premises and cloud. Thus, you can standardize your data governance across all your environments. On the cloud, the actual queries are carried out by the underlying data warehouse, so there is no data duplication when using these tools to create dashboards.

Regardless of where you store the data, you should bring computational resources to that data. For example, you might store your data on Azure Blob Storage as Parquet files and use Databricks or HD Insight to process the data using Spark. On Google Cloud, the compute and storage are separate and you can mix and match. For example, your structured data can be in BigQuery, but you can choose to do your processing using SQL in BigQuery, Java/Python Apache Beam in Cloud Dataflow, or Spark on Cloud Dataproc.

Do not make copies of data. Extracts and copies increase security risk, make data dependencies hard to track, and decrease the timeliness of analysis. Establish a lightweight semantic layer and bring compute to the single source of data.

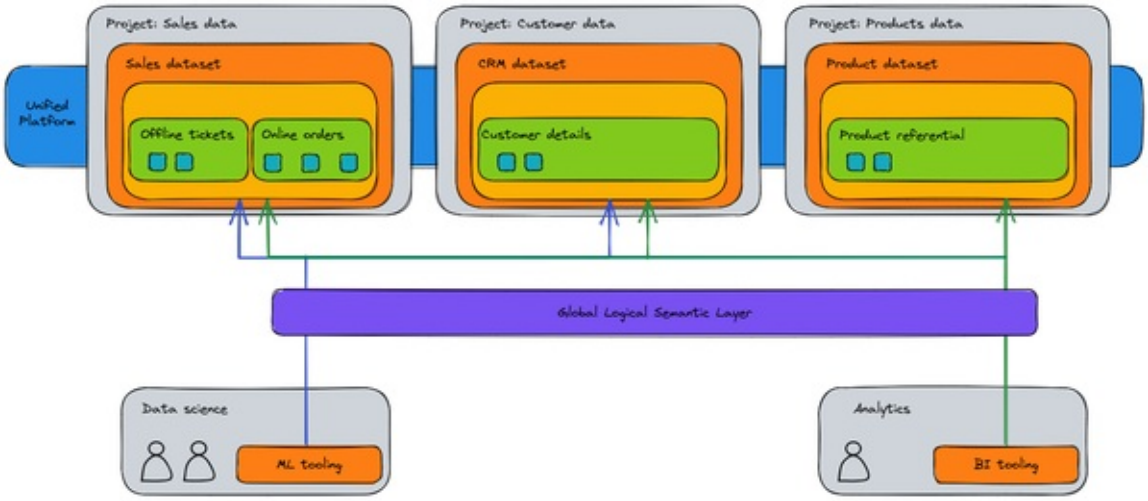


Figure 2-4. A well-defined, governed, secured data mesh architecture allows teams to maintain ownership of their data while providing access to teams that need to join datasets across multiple domains. Semantic consistency can be provided to unify KPIs and definitions across the different domains while eliminating the need for data extracts.

There is also a trend towards providing consistent control panes across different environments. Google's BigQuery Omni, for example, allows you to process data in AWS S3 buckets, Azure Blog Storage, and BigQuery, MySQL, and/or Spanner on Google Cloud from the Google Cloud BigQuery interface. Tools like Informatica, Collibra, Looker etc. provide a consistent interface to data in different clouds and on-prem environments.

As you have seen, removing silos is a key step in unlocking the power of the data because it enables better visibility and better collaboration among teams. Let's see now how you can move into the next steps to leverage this amount of data at your disposal in an even faster way.

Step 4: Make decisions in context faster

The value of a business decision decreases with latency and distance. For example, suppose you are able to approve a loan in 1 minute or in 1 day. The 1-minute approval is much, much more valuable than the 1-day turnaround. Similarly, if you are able to make a decision that takes into account spatial context (whether it is based on where the user currently lives, or where they are currently visiting), that decision is much more valuable than one devoid of spatial context. Therefore, an important modernization goal of your platform should be that you can do GIS, streaming, and machine learning on data without

making copies of the data. The principle of the previous section, of bringing compute to the data, should apply to GIS, streaming, and ML as well.

Batch to Stream

In many of the organizations that we work with, the size of data has been increasing between 30 and 100% year on year. Due to the power of compounding, this translates to planning for a 4x to 32x data growth over the next 5 years.

One of the counterintuitive aspects of a dramatic increase in data volumes is that it starts to make sense to process the data *more* frequently the larger the data volume gets. For example, suppose a business was creating a daily report based on its website traffic and this report took 2 hours to create. If the website traffic grows by 4x, the report will take 8 hours to create unless the business puts 4 times the number of machines on the job. Rather than do this, an approach that makes the reports more timely is to compute statistics on 6 hours of data 4 times a day and aggregate these 6 hourly reports to create daily reports that are updated four times a day, as shown in Figure 2-5. The computational cost of both these approaches is nearly the same, yet, the second approach can bring considerable business benefits. Extrapolate this approach, and it makes sense to have a constantly updating dashboard – you can see 24h aggregates that are up-to-the-minute. As data volumes increase, many businesses have this conversation and change from batch data processing to stream processing.

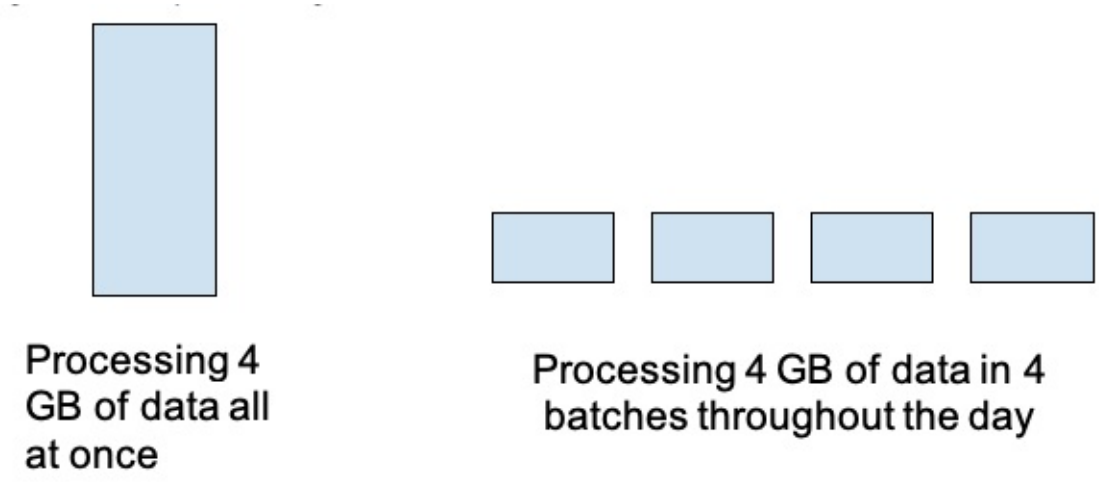


Figure 2-5. Spreading out processing can lead to lower latencies, fewer spikes, and less computational overhead.

Contextual information

Another key element of speeding up decision making is automating them. As data quality increases, or as the business changes its focus to its long tail of customers, there is an increasing need to cut down friction in the user experience. A frequent, expert user of your products will put up with a lot more than an occasional, less sophisticated user. Being able to catch frustrated users and provide them contextual help becomes important.

Real-time, location-based visualizations are increasingly how decisions get made. In many cases, these visualizations are built into the application that the user is using. For example, Shopify provides vendors with graphs and charts that depict how their store is performing. In order to do this at scale, the graphics are actually embedded into the website, rather than being a standalone dashboard product. It is a best practice to ensure that location information is part of your data schema, whether it is the location of a store or the location of a delivery truck. So, if your schema includes an address, make sure the schema requires that the address be geocoded and made to be in canonical form. It is very difficult to retroactively add clean geographic information to datasets.

Cost management

While few technology executives would quibble with the above, streaming has the reputation of being expensive to implement, monitor, and maintain over time. How can you enable real-time decision making without breaking your budget?

First, do not build two systems, one for batch and the other for streaming. Instead, treat batch processing as a special case of streaming. Software tools like Apache Flink and Apache Beam make this possible. Second, do not custom build monitoring, observability, late arrival, scaling, and so on. Flink and Beam are open-source technologies, but to execute them, leverage fully managed services such as Kinesis Data Analysis on AWS or Cloud Dataflow on GCP – this is because the skills to manage and troubleshoot streaming infrastructure is quite rare.

The alternate approach is to treat stream processing as a special case of batch. People who do this try to do microbatching, by processing tiny bits of data as fast as possible. This kind of approach can work when very fresh data, but not

necessarily real-time, is needed. It means that it is not acceptable to wait an hour or a day for a batch processing to run, but at the same time it is not important to know what happened in the last few seconds.

Second, land the streaming data into a data warehouse or storage tier that provides the latest information to readers. In other words, as long as you can land the data in real-time, all analytics on the data will reflect the latest data without any further effort on your part.

Now that you have seen how we can leverage streaming to work on up to date information, let's have a look at how to infuse AI/ML to have a better understanding of the data.

Step 5: Leapfrog with packaged AI solutions

One of the most exciting developments in technology in the 2010s was the rise of deep learning, a branch of Artificial Intelligence (AI). AI encompasses the class of problems where a computer can be used as a decision-making tool (see Figure 2-6). Commonly, AI systems were built by programming computers to think or act like humans. In order to do so, the thought process of experts had to be carefully encoded into rules that the computers could follow.

Because humans often can not precisely explain their judgment, such expert systems rarely performed very well. Machine learning (ML) is a class of AI techniques where, instead of capturing human logic, the ML "model" is shown a large number of correct decisions and the model is expected to infer how to make a correct decision in the future. Because collecting data can be easier than capturing logic, ML was able to solve a wide range of problems. However, the data usually had to be structured data, of the sort held in relational databases. In the mid-2010s, a set of techniques called deep learning achieved prominence – these techniques, which employed "deep neural networks", were capable of understanding unstructured data like images, speech, video, natural language text, etc. That in turn, has led to the development of technology like Google Photos (i.e. ability to search pictures using natural language queries) or Alexa (i.e. interaction via natural language processing). In the enterprise, too, deep learning has enabled organizations to extract information from unstructured data like product catalogs and user reviews.

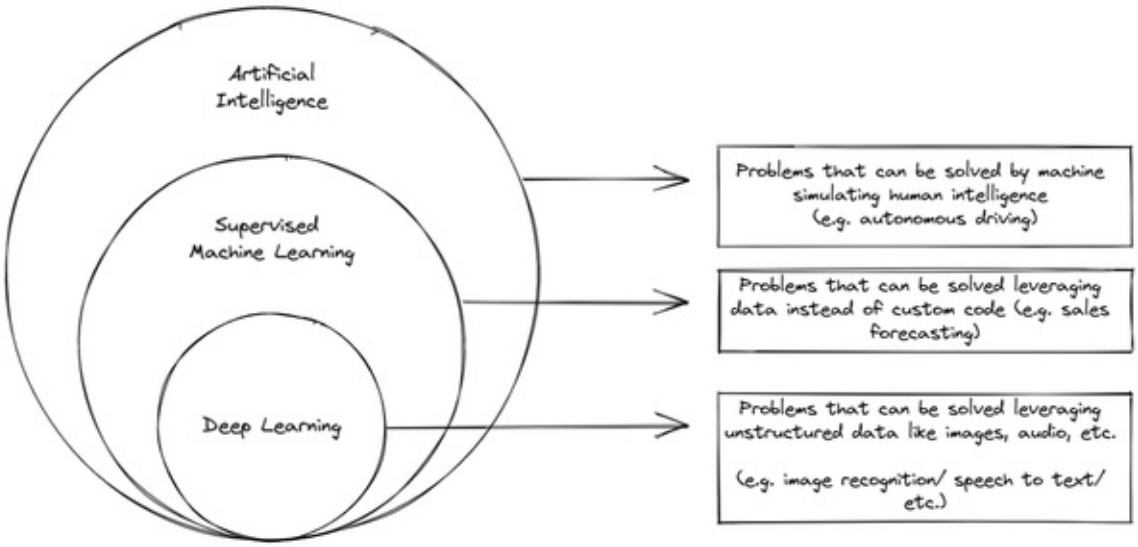


Figure 2-6. Deep learning is a type of machine learning which is itself a type of AI. Deep learning has enabled computers to think/act like humans when presented data examples consisting of unstructured data like images and text.

Because AI is getting more mature, it is no longer necessary to invest considerable engineering time into building AI capabilities in your organization. Instead, you can leverage the benefits of AI through the many AI solutions that you can buy or customize. These fall into a few categories: predictive analytics, unstructured data, personalization, and packaged solutions.

Predictive analytics

Machine learning models are trained from examples of correct decisions. An enterprise data warehouse is often a great source of such training examples. For example, suppose you are in the business of buying used cars, repairing them, and selling them. You would like to create a system to estimate the cost of repairing a vehicle bought at auction. It is clear that the historical data you have of your business is a good source of what the repair costs actually were for vehicles that you purchased and fixed up. In other words, the correct answers for historical data are present in the data in your data warehouse (see Figure 2-7) and can be used to train an ML model. The trained machine learning model can then be used to predict the cost of repairing vehicles that subsequently come up for auction.

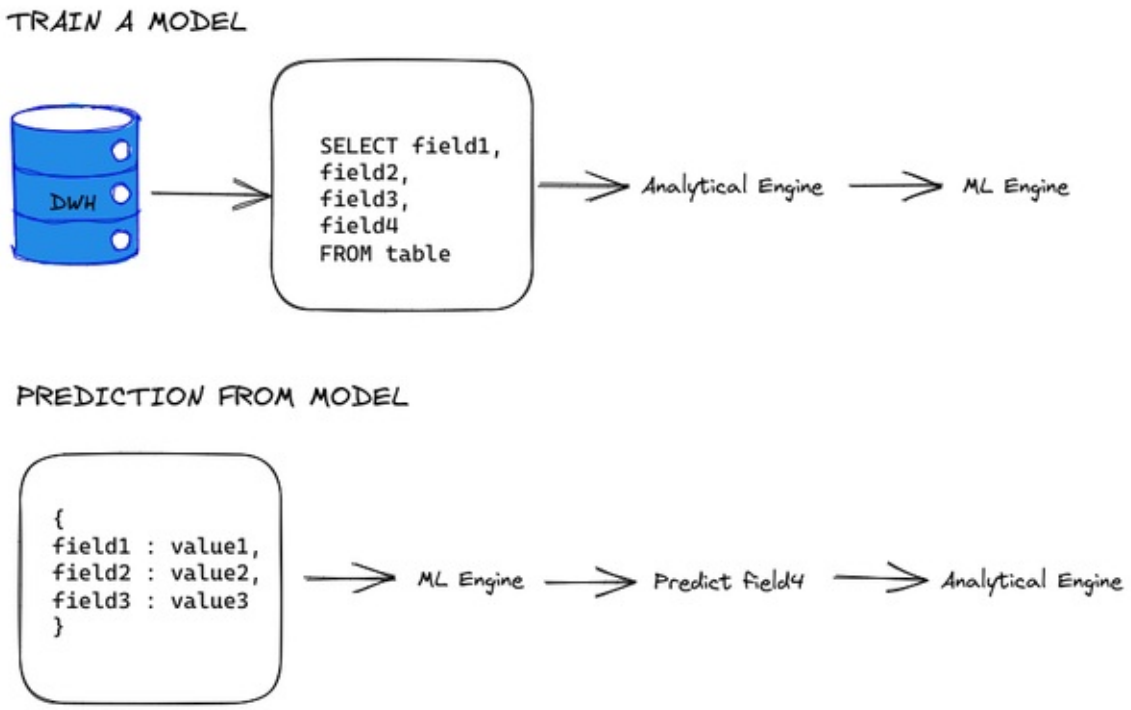


Figure 2-7. The enterprise data warehouse is a source of training examples for ML models.

Problems like detecting a fraudulent transaction, estimating when a machine is going to fail, whether an ad will be clicked on, how many items will be sold, whether a customer will make a purchase, and so on are examples of predictive analytics problems – these problems can be trained by teaching the model to predict one value in the historical record based on the other factors that have also been captured.

Understanding what factors affect something like the repair cost, and bringing all the data in the organization that bears upon this estimate into the data warehouse are prerequisites to successfully do predictive analytics. Once you have built an enterprise data warehouse, there are a large number of prebuilt forecasting solutions available to create the necessary model. Indeed, data warehouses such as AWS Redshift and Google BigQuery provide the ability to train custom ML models without moving the data out of the data warehouse by connecting to AWS Sagemaker and Google Cloud Vertex AI respectively.

Unstructured data

Problems like identifying eye disease from retinal images, detecting an illegal left turn from a traffic camera, transcribing text from videos, and identifying

abusive language in reviews are examples of using machine learning models to interpret unstructured data: images, videos, or natural language.

Deep learning has revolutionized the understanding of unstructured data, with each successive generation of models lowering the error rate to the point that products like question-answering in Google Home, Smart Reply in Gmail, and photograph retrieval in Google Photos are highly accurate.

Unlike with predictive analytics, it is rarely necessary to create and train models to understand unstructured data. Instead, prebuilt models like Azure Vision API, Google Video Intelligence API, and AWS Comprehend can be used as-is. Use these APIs to enrich your data. For example, even with no machine learning knowledge, a developer can use Google's NLP API to extract the sentiment of reviews and add the sentiment as an extra column in your data warehouse.

What if the label provided by these prebuilt models is insufficient for your use case? Perhaps the image recognition API returns a response indicating that the image contains a screw, but you want the API to return a value that it is Item# BD-342-AC in your catalog. Even in that case, it is not necessary to train models from scratch. Auto ML models (i.e. a standard ML model for a problem space such as image recognition that can be trained with custom data) such as those from Google Cloud, Azure, H2O.ai, Data Robot, etc. are customizable by simply fine-tuning them on your own data, often with as few as a dozen examples of each type. Auto ML models exist for images, video, and natural language. Use them to get high-quality customized APIs that work for your problem.

Personalization

Rather than provide the exact same product to all users, machine learning offers the opportunity to provide a more tailored experience. Problems such as customer segmentation, targeting, and product recommendations fall into the category of personalization.

Personalization is driven by the historical behavior of your customers. For example, if you are a retailer, you might recommend products to users based on other people's behavior ("People who bought this item also bought …"), based on your own purchase history, or based on item similarity. All this information is present in your enterprise data warehouse (or at least, it can be). Therefore, you

can power recommendation engines off your storage engine.

And indeed, Google BigQuery ML has a recommendation module and Google's Recommendations AI operates off BigQuery data. Similarly, recommendation modules in Azure ML operate off Azure Synapse.

In all three categories of AI considered in this section, it is possible to stand up a quick prototype in a matter of hours to days. The models all exist; the data requirements are not onerous, and you can train ML models with a single click or single SQL query.

Packaged solutions

In addition to the individual ML models and techniques discussed in the previous sections, always be on the lookout for packaged solutions that solve domain-specific problems in a turnkey way.

For example, Conversational AI can accurately handle common, routine questions like finding out store hours of operation or a request for a restaurant reservation. Conversational AI is now capable of populating the screen of a call center support agent with suggested answers to the customer's question. Such solutions are readily integratable into an enterprise's telephone system.

Solutions ilke Order to Cash, Inventory Management, Shelf-Out identification, etc. are ready to integrate into your business. Doubtless, by the time you are reading this, many more packaged and turn-key solutions will be available. We encourage you to leapfrog the limitations of what you can realistically achieve with your current technology stack by adopting these packaged solutions.

Step 6: Operationalize AI-driven Workflows

The last stage of your data platform strategy is to go beyond simple predictions and decisions to automating end-to-end workflows. You want to be able to solve business problems in a fully automated and autonomous way. For example, you don't want to merely predict when a machine will fail next, you want to schedule maintenance for the machine before it fails. In fact, you want to do this optimally, so that your operating cost is lower, machines last longer, and your repair facility is not overwhelmed. To achieve all this you must first of all

understand what is the level of automation that you want to have within your organization (e.g. fully automated or just assistance?) and then you need to focus on building a data culture to build a path toward your desired goal. Last but not least you need to reinforce your data scientists team that is the fuel for the realization of your AI desires.

Identifying the right balance of automation and assistance

In our experience, most organizations are making many decisions in a one-off way. If they used a data-driven approach and invested in making those data-driven decisions more systematically, they would reap hefty dividends.

Such automation is the reason to break down silos, gain a consolidated view of data, make prescriptive recommendations – as depicted in Figure 2-8, this is the logical end-station of the data transformation journey. As the data maturity of an enterprise deepens, leadership needs to encourage the movement of the enterprise from one stage to the next.

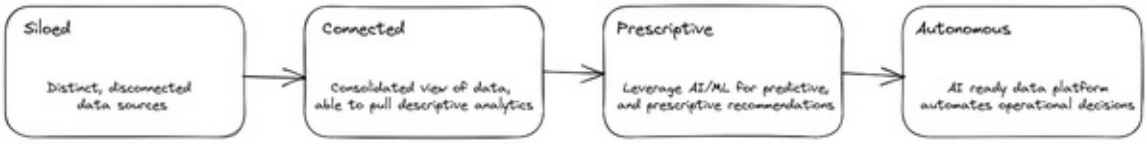


Figure 2-8. Increasing data maturity of an enterprise.

To achieve this increasing maturity, it is necessary for both executives and staff to be clear-headed about what the final goal is. Is it full automation of decisions? Is it AI-guided decision making? In other words, are humans out of the loop? Or are they "over" the loop?

Consider, for example, the difference in the way navigation instructions are generated by Google Maps and by your nation's Air Traffic Control system. The former is an example of humans-out-of-the-loop. It is fully automated. In the air traffic control case, the system provides guidance, but it is a human who provides the final navigation instructions to landing aircraft. Of course, in both cases, humans receive and act on the navigation instructions.

The right balance of automation vs. assistance is something that organizations usually settle on after a fair bit of experimentation. It varies from industry to industry and from organization to organization. The relative mix is often one of the ways that different businesses in the same industry compete with each other.

As such, this is usually something that is led by product management, not engineering.

Building a data culture

It will be necessary for the organization to build applications and systems to achieve the end-goal, whether the goal is full automation or to provide assistance to experts. Building such a system will require the organization to inculcate a data culture.

It is not enough to simply put in place an analytics data platform. It is also necessary to change the culture of the organization to one where data-driven experiments are the norm, and successful experiments get operationalized and scaled.

Building a data culture[2] is key to unlocking innovation (*we cover this in more detail in Chapter 3*) because you will enhance data literacy within the organization spreading the knowledge needed to read and work with the data. Just because you have built a platform that enables data silos to be broken doesn't mean that they will be. Just because decisions can be made in a data-driven manner does not mean that old heuristics will be easily discarded.

Successful organizations undertake a transformation program that involves providing training on data tools (such as business intelligence dashboards and embedded analytics) to their entire creative workforce. They seek to change the way employees are rewarded, to encourage risk taking and entrepreneurship. They seek to put in place ways to measure everything that is important to the business. Finally, they are looking to equip their workforce with data tools so that they can effect change.

Populating Your Data science team

There are several roles that your data platform will need to enable for your organization to realize value from its data and AI investments: data engineers, data scientists, ML engineers, developers, and domain experts. Unless all these groups are actively collaborating on your data platform, you can not say that you have built a future-ready data platform.

Data engineers ingest, prepare, and transform data. They use Extract Transform Load (ETL) tools to land the data in the data warehouse. They monitor the data pipelines to ensure that the pipelines are running correctly and not corrupting the data feeds.

Data scientists carry out data analytics to help decision makers gain visibility into how the business is performing, answer hypotheticals, model outcomes, and create innovative models. A key decision maker here is the product management team. Data science teams do analysis that help inform product managers on ROI of different items on the product roadmap. For example, a data scientist in an agricultural technology company might provide answers to questions about the yield per acre of different seeds and how it depends on the soil type. They might answer hypotheticals such as the anticipated profits if the product mix at a store were to be changed to have more of one seed than another. They might also model answers to questions such as the ROI of improving availability in smaller stores. Finally, data scientists may break down a business strategy, such as creating personalized harvest planning, into component models and build them.

Once a model is created, it needs to be run routinely. ML engineers encapsulate the entire workflow in a ML pipeline and then ensure that it is executed in such a way that it can be monitored. Models are monitored both for resilience (is the model running? Is it handling all the users requesting predictions?) and for accuracy (are the results correct?). Models tend to drift over time. Sometimes this is because the environment itself changes (a new type of pest starts to affect a certain type of seed, so its yield estimates are no longer valid) and sometimes it is because users adapt to the model (farmers start to plant more soybeans than corn in response to price changes of the corresponding seed varieties, and this changes the demand for certain types of fertilizers). ML engineers look for such drift, and retrain the model with new data. This is called ML Ops.

Deployed models are available as APIs. Developers invoke the models and depict their results in the applications that end-users use.

Domain experts, also known as business analysts, employ predictive analytics to enable data-driven decision making. Data scientists observe the decisions that experts make, and look at ways to speed up the decision making by breaking down data silos that prevent this data from being accessed routinely. They use packaged AI solutions to enrich the data, and thus continue the cycle of data-

powered innovation.

Step 7: Product Management for Data

To maximize the leverage you get from data, apply product management principles. A few years ago, what many executives meant by “treating data as a product” was that they wanted to monetize their data directly such as by selling it on a data marketplace. However, today, such marketplaces tend to mostly contain data created by companies that specialize in aggregating data across many sources (e.g. retail footfall, credit card receipts, product reviews). Few companies have found success monetizing their 1st party data.

So, what does it mean today when a typical business aspires to treat data as a product? There are several competing, but complementary, definitions. The Tableau definition — any application or tool that uses data to help businesses improve their decisions and processes is a data product — emphasizes the usefulness of data. The McKinsey definition — a high-quality, ready-to-use set of data that people across an organization can easily access and apply to different business challenges — emphasizes standardization. The Montecarlo definition — that data is available within the company in a form that’s usable (even if the final mile involves self-service transformations) — emphasizes data governance.

Applying Product Management Principles to Data

Our preferred way to think about this is to combine the desired outcome and the process to get there. The desired outcome is that your organization will maximize the leverage it gets from its data by treating it as a product, and here the characteristics highlighted by the definitions above (usefulness, standardization, governance) are important. Like Tableau, we take an expansive view of what a data product is — datasets qualify, but so do data pipelines, dashboards, data-reliant applications, and ML models.

Desired outcomes are valuable only when accompanied by a path to get there. To treat data as a product, apply product management principles when conceiving and building data products. What product management principles? (1) Have a product strategy, (2) be customer-centric, (3) do lightweight product discovery, and (4) focus on finding market fit. We recommend adopting 10 data practices

(See Figure 2-9) aligned to these principles.

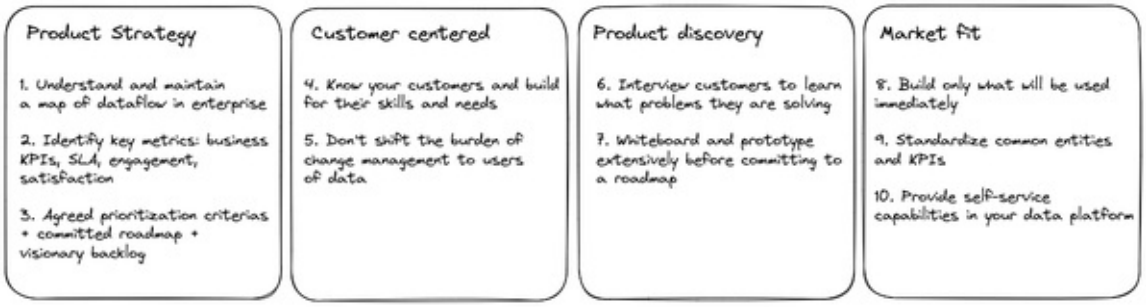


Figure 2-9. Product Managing Data

1. Understand and maintain a map of data flows in the enterprise

One key job of a product manager is simplification. Too often, when someone asks “what data do you have”, the answer is a spreadsheet of hundreds of datasets collected by surveying the varying business units across the company. This is not very useful. Treating data as a product means that you (the data product team) maintain a high-level model of data flows in the business that can be easily communicated for discoverability. You need to maintain this map at multiple levels of granularity.

Imagine that you have an e-commerce site. At the highest level, for the e-commerce site, it might be:

- Web Traffic
- Product Catalog
- Web Content
- Orders
- Inventory
- Customer Survey

At the next level of granularity, web traffic might be broken down into session data, page data, etc. Capture how each dataset is collected, how they are processed, what roles can access and how, whether PII or other attributes are present, what quality assurances are made, etc. Also, capture the production use

cases for each dataset. As you go from higher levels of granularity to lower-levels, the mapping starts to include details of your data platform implementation. It starts to become a data catalog.

2. Identify key metrics

A data catalog is simply a record what currently exists. It does not capture why the data is important or whether the data is fit-for-purpose. It doesn't tell you what needs to be improved. An important part of your data product strategy is to get alignment across the enterprise on your key metrics — what you will measure, how you will measure it, and what the target number for the metric is (goals will change over time). The universe of metrics that you track should include:

1. Business KPIs: what business outcomes need to be enabled by data?
2. SLA: What is the data availability? data quality? refresh rate?
3. Engagement: How widely and how often is the data used across the company?
4. Satisfaction: How satisfied are customers (could be internal) are with what data is available and how easy it is to use?

For the hypothetical e-commerce site introduced in the previous step, the business outcomes might involve increasing customer lifetime value, increasing free-tier conversions, etc. The SLA for the inventory displayed to internal purchasers (for restocking) might be that it's available 99.99% of the time, at an hourly refresh, and is maintained to be above the next week's predicted sales. You might want the inventory predictions to be used, not only by internal purchases, but also by logistics teams and incorporated into dashboards. And you might have a measure of how often the predicted inventory amounts are overridden.

3. Agreed criteria, committed roadmap, and visionary backlog

The data catalog is a record of what currently exists. The metrics capture what

your goals are. Neither of these explains where you are going next.

It is important to adapt the product vision over time based on customer feedback, stakeholder input, and market conditions. During all this, your stakeholders will ask you for features and timelines and expect you to keep your commitments. To handle change and user feedback, you need three things:

1. Prioritization criteria are what stakeholders agree on beforehand — this enables transparency and buy-in across the org on the product roadmap.
2. The product roadmap itself is informed by a process of product discovery so that the team can avoid agreeing to timelines in the absence of information and prototyping.
3. Things that you think are important, but are yet to be roadmapped will be captured in a product backlog. Typically, the product backlog consists of customer problems that need to be solved (not features that have to be built). In many ways, the backlog (not the roadmap) forms your longer-term product vision. Organize the backlog to tell a clear story.

The roadmap needs to be high commitment — you should be able to commit to the timelines and features on the roadmap. A great way to do this is to get agreement on prioritization criteria, do product discovery, and maintain a product backlog.

Recalling that one of our hypothetical data products (see previous step) is inventory predictions a week ahead, we need to agree on how we measure how good the predictions are— is it that we rarely run out? That we minimize the costs of procuring and storing the items? Is the running out at the warehouse level? Or at the company level? These form the prioritization criteria. If someone asks you to customize the inventory model for perishable goods, is it worth doing? You will initially add it to a product backlog. Then, you'll do product discovery to determine the ROI of doing such a project — this will include the cost of increasing/decreasing refrigeration at the warehouses, for example. Only when you know the value will you add this to your product roadmap.

4. Build for the customers you have

Too often, data teams get caught up in technology slogans: they only provide

APIs, or insist that everyone publishes data into their enterprise data warehouse, or expect conformance to a single dictionary.

Take a leaf out of product management, and develop a deep knowledge of who your customers are. What are they building? A mobile app or a monthly report? What do they know? SQL or Java? What tools do they use? Dashboards or Tensorflow? Do they need alerts whenever the data changes? Do they need moving averages of the data in real-time? Do they care about test coverage?

Then, serve data in ways that your target customers can use them. For example, you might serve the data in a data warehouse (to data analysts), make it accessible via APIs (to developers), publish it in feature stores (to data scientists), or provide a semantic layer usable in dashboards (to business users).

If the hypothetical inventory prediction data product (that we are using as our example) will be leveraged by internal purchasers who are business users, the predictions will have to be served in the application that is used for ordering replenishments. So, the predictions will likely have to be accessible via an API for the application developers to use.

5. Don’t shift the burden of change management

Change and conflict are inevitable. The suppliers of data will change formats; the consumers of data will have new needs; the data velocity will change; the same data might be provided in multiple channels; your customers will move to an alternate supplier due to cost. These are not solely the problem of the team that makes the changes or the team that uses the data.

A big part of treating data as a product is to ensure that users of data are not stuck with change management responsibilities. As much as possible, make sure to evolve schema and services so that changes are transparent to downstream users.

When backwards-incompatible change inevitably happens, version the changes and work with stakeholders to move them from older versions of the data to newer versions. This might involve creating a migration team whose job is to move the enterprise from one version to the next.

What’s true of change management is also true of security. Make sure to build

safeguards for PII and compliance instead of shifting the burden to users of your data products.

Suppose our hypothetical inventory prediction data product is customized to include predictions of perishable goods. If this involves requesting additional information on the items being sold, you will have to take on the responsibility of ensuring that your item catalog is enhanced for all existing items. This data engineering work is part of the scoping of the project, and feeds into the ROI of whether that work is worth doing.

6. Interview customers to discover their data needs

How do you evolve the product backlog, prioritize needs, and add to the roadmap? An important discipline is to ensure that you are constantly talking to customers and discovering what data they need to solve the problems that they are encountering. What shortcomings of the current data products are they having to work around? These problems feed into your product backlog, for you to prioritize and solve.

It is important that before any new data product idea enters the product roadmap that the need for the product has been validated by potential (internal or external) customers. Building on spec (“build it and they will come”) is extremely risky. Much safer is to build implementations of ideas that have already been validated with customers.

How do you do that?

7. Whiteboard and prototype extensively

Whiteboard the design of the data product with customers who want it. This ensures that what you land on in the data platform will meet their needs in terms of quality, completeness, latency, etc. Walk through potential uses of data with them before you build any data pipelines or transformations.

One of the best tools here is a prototype. Many use cases of data can be validated by building a minimum viable prototype. What do we mean? If the sales team believes that building a customer data platform will help them cross-sell products, validate this by picking up a set of records from the individual products’ sales pipelines, doing the match manually, and trying to cross-sell the

resulting customers.

We recommend using a prototype, plus interviews with potential users of the final product, to scope the problem in terms of:

1. what needs to be built: identify everything, from data pipelines to user interfaces that are needed for the project to succeed
2. the ROI that you can expect in terms of business KPIs

Do this before you write any code. It's only when you have a clear idea of what needs to be built and the expected ROI that you should add the project to your roadmap. Until then, keep the problem in your backlog.

In the case of our hypothetical inventory predictions data product, you would have validated the input schema and how the predictions will be used with the key product users, checked how much more refrigeration warehouses can accommodate, etc. You'll do this before you write any code, perhaps by doing the predictions in a spreadsheet and game-playing the whole set of scenarios for a wide variety of products.

8. Build only what will be used immediately

Prioritize going to production quickly over having all the necessary features built. This means that you should be using agile, iterative processes to build only the datasets, data pipelines, analytics, etc. that are immediately required.

Use the product backlog to capture future needs. Build those capabilities only after you have identified customers who will use those features and can give you feedback in whiteboarding/prototyping sessions.

9. Standardize common entities and KPIs

Provide canonical, enriched datasets for common entities and KPIs that will be standard across the business. Usually, these enriched entities power a large number of high-ROI use cases (e.g. customer data platform, content management platform) or are required for regulatory/compliance purposes (e.g. the way to calculate taxes).

Typically, you'll have only a handful of these standardized datasets and metrics,

because such enrichment requires significant collaboration across business units and reduces their release velocity.

10. Provide self-service capabilities in your data platform

You have to balance flexibility and standardization in a way that fits your organization. Do not go overboard with #9. Do not build centralized datasets that have everything anyone could ever want. Instead, enable teams to be self-sufficient. This is the microservices principle as applied to data.

One way to achieve this balance is to provide small, self-contained datasets that customers can customize by joining with other datasets in domain-specific ways. Often, this is implemented as a data mesh, with each business unit responsible for the quality of the datasets that it publishes into a shared analytics hub.

Summary

In this chapter you have understood more about the process that organizations should put in place to innovate with data. Everything starts with planning with the ultimate goal to transform your data in a real product with a well defined (and evolving) end to end strategy. The key takeaways from this chapter are:

- Create a strategic plan by identifying key business goals, gathering the right stakeholders, and setting in place change management across the enterprise. Ensure that your set of stakeholders includes the business units that will most benefit from the data platform if they were to adopt it.
- Reduce total cost of ownership by moving to the Cloud. When doing so, do not try to replicate the technology choices you made on-premises. Instead, select technologies that autoscale, separate compute and storage, embrace no-ops, and allow you to power all sorts of applications without data movement.
- Break down data silos so that data from across the organization can be joined together. It is important to ensure that each individual department controls its data and is responsible for its quality. However, all the data is available on a uniform platform to enable cross-organization access.

- make decisions in context faster by streaming data into your data warehouse in real-time and ensuring that data access always reflects the latest data. The data tables should, wherever relevant, include location information.
- Take advantage of customizable and adaptable AI models so you don't have to build bespoke AI models. These help regardless of whether your need is for predictive analysis, understanding unstructured data, or personalization. Predictive analytics and personalization can be driven from your data platform. Unstructured data can be added to your data platform by processing it using prebuilt AI models.
- Expand beyond one-off ML models and into automating entire workflows. This step is often led by product management and analysis here often informs the product roadmap.
- build a culture of innovation by hiring and staffing teams that consist of data engineers, data scientists, ML engineers, developers, and business analysts.
- Use product management principles to formulate your data product strategy, be customer-centric, discover products through whiteboarding and prototyping, and find the right balance between standardization and flexibility.

Now that you are equipped with the foundational knowledge on how to infuse innovation with data within an organization, you will focus in the next chapter on the building blocks of modern data platforms reviewing different approaches and understanding why they are an accelerator for the business.

---

[1] A layer of abstraction that provides a consistent way to understand data. It translates complex information into familiar business terms such as product, customer, or revenue to offer a unified, consolidated view of data across the organization.

[2] HBR Article - 10 steps to creating a data-driven culture (*https://hbr.org/2020/02/10-steps-to-creating-a-data-driven-culture*)

Chapter 3. Creating a Modern Data Analytics Capability

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the third chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

An important element to keep in mind when building a data platform, is that the overall activity is more than just an IT project. Data is one of the most valuable assets that a company has because it allows organizations to make meaningful and data-driven decisions that add value to their business. Using data allows you to go beyond gut feel and hunches and instead make decisions based on measurable goals or KPIs, via the analysis of actual facts. You can leverage patterns identified from the data to envision new business strategies and implement activities that benefit the business in a number of areas. Given the centrality of data to strategic transformation, we will expand the lens in this chapter beyond the data platform and consider the overall modernization of how data is used.

To take one example, Google in 2013 improved the favorability score of its management from 83% to 88%[1] via a data-driven approach based on the collection of more than 10k performance reviews. Google's human resources team used employee retention rates to identify common behaviors of high-performing managers and created training programs to develop these competencies. In turn, this ability to retain employees led to a whole host of

better outcomes across the business. The project of improving retention rates was strategic in nature. Another example of a strategic data project was carried out at Amazon. The e-commerce giant implemented a recommendation system based on customer behaviors that drove 35% of purchases in 2017[2]. The Warriors, a San Francisco basketball team, is yet another example; they enacted an analytics program to help them change tactics and turn around a game where they were behind to eventually win 119-104.[3] All these — employee retention, product recommendations, improving win rates — are examples of business goals that were achieved by modern data analytics.

In the rest of this chapter, you will learn how to set in place such a modern data analytics capability, by first analyzing the building blocks of a data platform (i.e. data lifecycle) then the related security and governance implications, and reviewing the limitations and challenges of an enterprise's current data platform. Finally you will see why it is better to think about *workflows* rather than *platforms*.

The Data Life Cycle

In this section you will have a look at the different steps we apply to data in order to move from raw data to insightful information.

The journey to wisdom

Data helps companies to develop smarter products, reach more customers and increase ROI. Data can also be leveraged to measure customer satisfaction, profitability and cost. But the data by itself is not enough. Data is raw material that needs to pass by a series of steps before being useful and generating insights and knowledge. This sequence of stages is what we call a *data lifecycle*. There are many definitions available in the literature but from a general point of view we can identify 5 main stages in modern data platform architecture:

1. **Collect**: data has to be acquired and injected into the target systems (e.g. manual data entry, batch loading, streaming ingestion, etc.)
2. **Store**: data needs to be persisted in a durable fashion with the ability to easily access it in the future (e.g. file storage system, database)

3. **Process / transform**: data has to be manipulated to make it useful for subsequent steps (e.g. cleansing, wrangling, transforming)
4. **Analyze/ visualize**: data need to be studied in order to derive business insights via manual elaboration (e.g. queries, slice and dice) or automatic processing (e.g. enrichment using ML APIs)
5. **Activate**: surfacing the data insights in a form and place where decisions can be made (e.g. notifications that act as a trigger for specific manual actions, automatic job executions when specific conditions are met, ML models serving that send back feedback to devices)

Each of the stages above feeds into the next, similar to the flow of water through a set of pipes.

The water pipes structure analogy

To have a clearer understanding of what a data life-cycle is, you can think of these stages as a part of a simplified water pipes structure that could take various shapes and have multiple paths to transfer and transform the water from an aqueduct to serve a group of houses (see Figure 3-1).

The water that will flow into these tubes need to be first **collected**, for example from an underground aquifer, and then **stored** for additional usage into a collection well. At this stage water has to be **transformed**, for instance it can be filtered to remove impurities and **analyzed** to check if it can be drunk. Once terminated, water has to be piped into a complex network of tubes, designed by plumbing engineers, to be distributed to final users that can use it ("**activate** it") in several different ways.

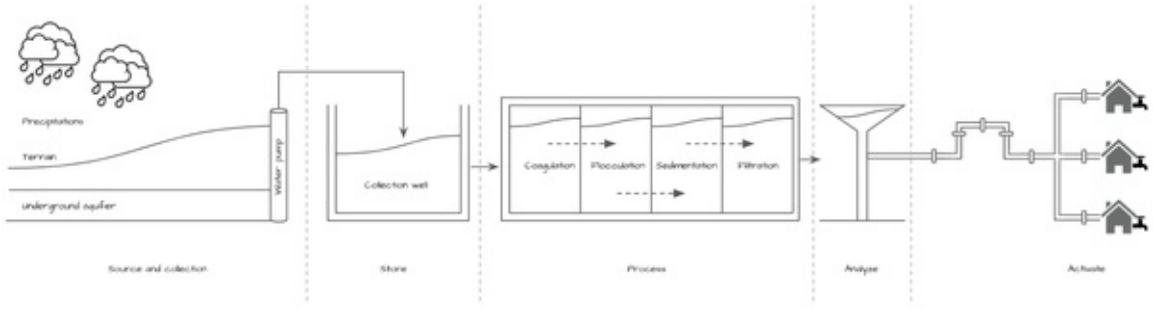


Figure 3-1. Water lifecycle, providing an analogy for the 5 steps in the data lifecycle

If you translate this example into the data world you can immediately identify several analogies not only in terms of stages but even in the actors involved in the process: plumbing engineers can be equated to data engineers who are designing and developing all the transformation steps and pipelines needed to make the data usable while people who are performing micro analysis on the water can be compared to data analysts/ data scientists who are performing analysis on the data to extract insights. This is of course a simplification because there are several other roles in a company that are making use of data (in fact every single employee of an organization is leveraging data) like executives, developers, business users, security administrators etc. but it should provide you with an easy way to remember the main concepts.

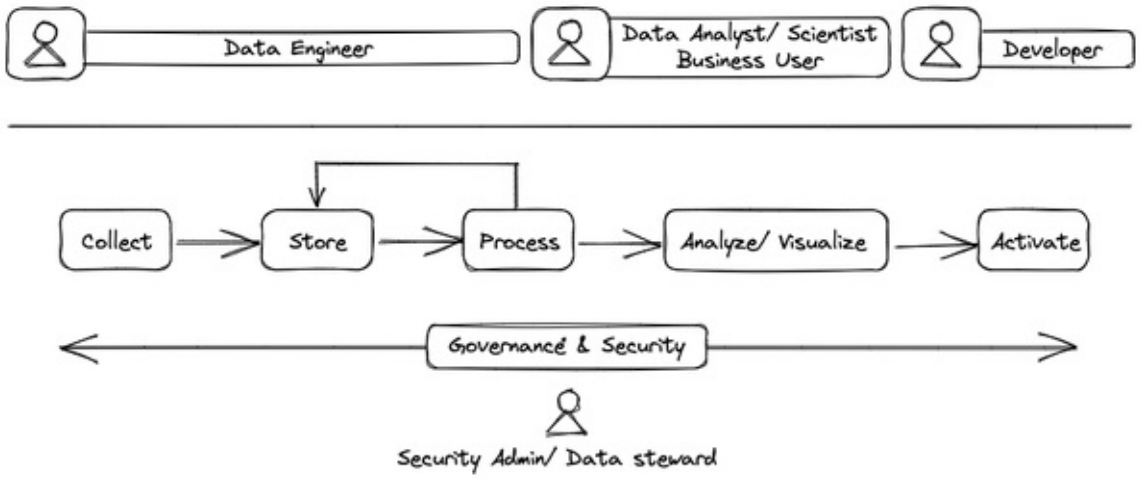


Figure 3-2. Simplified data lifecycle

In the canonical data lifecycle, shown in Figure 3-2, data engineers collect and store the data in an operational database and/or analytics store. The stored raw data is processed, typically in multiple stages, using a variety of tools. If the tools involve programming (e.g. Spark), the processing is typically done by data engineers. If the tools are declarative (e.g. SQL), the processing is typically done by data analysts. The stored data is made fit for use in the process stage. This more usable data is then analyzed by data analysts, business users, and data scientists. Business users tend to activate the insights using domain-specific tools – for example, marketing users might launch a marketing campaign and logistics personnel may issue refunds for packages that were not delivered. Data scientists may train ML models on the data and then activate the models as APIs to be consumed by developers.

What you have seen above is an idealized description of how a modern data platform architecture and roles should work. Of course, the real world is a little bit different – most of the time you have solutions where stages tend to be shrinked into a single one (e.g. storage and process) or the order is different (e.g. process before store as in ETL rather than store before process as in ELT). However, such changes typically result in important limitations in terms of scalability and reliability. For example, combining storage and processing into a single stage means that there is no decoupling of storage and compute. Therefore, it has the unfortunate side-effect of resource wastage (e.g. if data sizes grow, the only way to get more storage is to simultaneously scale the compute power also) and scalability issues (e.g. limitations of infrastructure make it hard to extra storage/ extra compute).

Now that we have defined what a data life cycle is and summarized the various stages that take part in the data journey from raw data collection to activation, let's delve into each of the five stages of the data lifecycle in turn.

Collect

The first step that you have to design for is the ingestion step. In the ingestion step, you transfer data from a source that could be anywhere (i.e. on-premise, on devices, in another cloud …) into a target system where it can be stored for further analysis. This is the very first moment where you have to start reflecting about the 3Vs[4]:

- *Volume*: what is the size of the data? Usually when dealing with BigData this means TB or PB of data.
- *Velocity*: what is the speed of the data coming in? Generally this is MB/s or TB/day. This is often termed the throughput.
- *Variety*: what is the format of the data? Tables, flat files, images, sound, text, etc.

To answer these questions, you need to identify the key attributes of the data being ingested. This involves very basic questions like "*what is the data that I am going to collect?" , "is it structured (e.g. a table), semi-structured (e.g. JSON file), unstructured (e.g. image) or could the format change over time or by*

source?", "is it something that will be generated continuously or at specific intervals?" Based on the responses to these questions, you could adopt one of several approaches:

- *Batch ingestion*: where data collected over a period of time will be transferred at regular scheduled intervals
- *Streaming ingestion*: where events or messages will be transferred in real time as they happen
- *Hybrid ingestion*: that is a mixture of batch and real time approach that usually take part in what is called lambda architecture

Since different parts of the organization could be interested in different data sources you need to design this stage as flexible as possible. There are several commercial and open source solutions that could be leveraged, each specialized to a specific data type/ approach mentioned earlier. Your data platform will have to be comprehensive and support the full range of volume, velocity, and variety required among all the data that needs to be ingested into the platform. You could have simple tools that have to transfer files between FTP servers on regular intervals or you could have complex systems even geographically distributed that have to collect data from IoT devices in real time. One possibility to simplify data architecture designs is to assume that data will not be modified/ transformed by this stage.

Store

This is the stage where the raw data collected in the previous step will be physically put into the target system. Note the order here – you ingest the data and immediately store it without performing any transformation of data values. Not allowing data modification, and ingesting the raw data as-is allows you to potentially perform additional data reprocessing in the future – you need to preserve the original raw format to guarantee the correct results.

You have seen that data can come in many different shapes, sizes and formats. Based on the different technical (*and sometimes commercial*) decisions made during the definition of the data platform architecture, you could have different options to store the data: object storage systems, Relational Database

Management System (RDBMS), data warehouses solutions, and data lakes are only a few options available. Your choice will be driven to some extent by whether the underlying hardware, software and artifacts are able to cope with the analytics requirements of your desired use cases as detailed below:

Scalability

The ability to grow and manage increased demands in a capable manner.

There are mainly two ways to achieve that:

- *Vertical scalability* where the capability of the storage system is increased simply adding extra expansion units to the same node;
- *Horizontal scalability* where instead of adding new expansion units to a single node, one (or more) additional nodes will be added. This sort of distributed storage is more complex to manage but it could achieve improved performance and efficiency.

It is extremely important that the underlying system is able to cope with the volume and velocity required by modern solutions that have to work in an environment where the data is exploding and its nature is transitioning from batch to real-time: we are living in a world where the majority of the people are continuously generating and requiring access to the information leveraging their smart devices; organizations need to be able to provide their users (both internal and external) with solutions that are able to provide real time responses to the various requests.

Performance vs. Cost

Depending on the kinds of data that you need to manage, you could end up with a hierarchy that could be, for example, based on the business importance of the information you need to store. You can accordingly have a tiered storage structure that allows you to store data at the top of the identified hierarchy (i.e. *the hot data*) into a high performance storage system and that relies on less performant and less expensive storage systems for the data at the bottom of the hierarchy (i.e. *the cold data*). There are then situations where you want to achieve even higher performance (e.g. when dealing with interactive data exploration) and you may want to leverage caching techniques that load a

meaningful portion of your hot data into a volatile storage tier that comes with a higher price but much better performance especially in getting read-only access to the data.

High availability

High availability means having the ability to be operational and deliver access to the data when requested. This is usually achieved via hardware redundancy to cope with possible physical failures/ outages. This is achieved in the cloud by storing the data in at least three *availability zones*. Zones are not physically separated (i.e. they may be on the same "campus"), but will tend to have different power sources, etc. Hardware redundancy is usually referred to as system *uptime* and modern systems usually come with 4 9s or more.

Durability

Durability is the ability to store data for a long-term period without suffering data degradation, corruption, or outright loss. This is usually achieved through storing multiple copies of the data in physically separate locations. Such data redundancy is implemented in the cloud by storing the data in at least two *regions* (e.g. in both London and Frankfurt). This is extremely important when dealing with data restore operation in the face of natural disasters: if the underlying storage system has a high durability (modern systems usually come with 11 9s), then the entire data can be restored with no issues unless a cataclysmic event takes down even the physically separated data centers

Openness

As far as possible, use formats that are not proprietary and that do not generate lock-in. Ideally, it should be possible to query data with a choice of processing engines without generating copies of the data or having to move it from one system to another. That said, it is acceptable to use systems that use a proprietary or native storage format as long as they provide an easy export capability.

As with most technology decisions, openness is a tradeoff and the ROI of a proprietary technology may be high enough that you are willing to pay the price of lock-in. After all, one of the reasons to go to the cloud is to reduce operational costs — these cost advantages tend to be higher in fully managed/serverless systems than on managed open-source systems. For example, if your Big Data

use case requires transactions, Databricks (which uses a quasi-open storage format based on Parquet called Delta Lake) might involve lower operating costs than Amazon EMR or Google Dataproc (which will store data in standard Parquet on S3 or GCS respectively) — the ACID transactions that Databricks provides in Delta Lake will be expensive to implement and maintain on EMR or Dataproc. If you ever need to migrate away from Databricks, export the data into standard Parquet. Openneness, per se, is not a reason to reject technology that is a better fit.

Process / Transform

Here the magic starts happening: the data in raw format will be transformed to make it useful for further analysis. It is the stage where data engineers implement data pipelines to make data available to a larger audience of non-technical users in a meaningful way. There are several frameworks that can be leveraged and each of them comes with capabilities that depend on the storage method you selected on the previous step.

From a general point of view activities related to this stage can be bucketed into the following ones:

- *Data cleansing*: the process of preparing the raw data removing duplicated or corrupted elements from the dataset.
- *Data wrangling, munging, transformation*: the process of transforming raw data that can be disorganized or incomplete into a standard format that facilitates further consolidation and analysis (e.g. data mapping) with the aim to make the processed data reusable in different scenarios; there could be some use cases where ad-hoc transformations are required.
- *Data integration*: the ability to combine data coming from multiple sources into a unified view. Typical solutions belonging to this group are Extract Transform and Load systems (ETLs) that take data from source systems and convert/ transform/ enrich it in order to make it digestible by the target systems.

In general, engines that allow you to query and transform your data using pure SQL commands (e.g. AWS Athena, Google BigQuery, Azure Data Warehouse,

and Snowflake) are the most efficient, cost effective, and easy to use. However, the capabilities they offer are limited in comparison to engines based on modern programming languages, usually Java, Scala or Python (e.g., Apache Spark, Apache Flink, or Apache Beam running on Amazon EMR, Google Cloud Dataproc/Dataflow, Azure HDInsight, and Databricks). Code-based data processing engines allow you not only to implement more complex transformations in batch and in real time but also leverage other important features such as proper unit and integration tests.

Another consideration is that SQL skills are typically much more prevalent in an organization than programming skills. The more of a data culture you want to build within your organization, the more you should lean towards SQL for data processing. This is particularly important if the processing steps (such as data cleansing or transformation) requires domain knowledge.

This stage also includes *data virtualization* solutions (that will not be discussed any further in this book[5]) that abstract multiple data sources, and related logic to manage them, to make information directly available to the final users for analysis.

Analyze/ Visualize

Once you arrive at this stage, the data starts finally to have value in and of itself – you can consider it *information*. Users can leverage a multitude of tools to dive into the content of the data to extract useful insights, identify current trends, and predict new outcomes. At this stage, visualization tools and techniques that allow users to represent information and data in a graphical way (e.g. charts, graphs, maps, heat maps, etc.) play an important role because they provide an easy way to discover and evaluate trends, outliers, patterns and behavior.

Visualization and analysis of data can be performed by several types of users. On one hand are people who are interested in understanding business data and want to leverage graphical tools to perform common analysis like slice & dice roll-ups, and what-if analysis. On the other hand there could be more advanced users (“power users”) who want to leverage the power of a query language like SQL to execute more fine grained and tailored analysis.

Parallel to that there might be data scientists who can leverage ML techniques to

implement new ways to extract meaningful insights from the data, discover patterns and correlations, improve customer understanding and targeting, and ultimately increase a business' revenue, growth and market position.

Activate

This is the step where end-users are able to make decisions based on data analysis and ML predictions, thus enabling a data decision making process. From the insights extracted or predicted from the available information set, it is the time to take some actions.

To make it clearer, consider the scenario shown in Figure 3-3. This depicts a connected machine in a factory that has to manufacture plastic bottles. The machine has to complete several stages in order to transform a piece of plastic into a bottle and for each stage there are some configurable actuators that have to complete some activities (e.g. compress, cut, blow the plastic). The machine is continuously collecting data describing how actuators are working and the information is sent to the company data platform. Here the data is collected, stored and transformed to make it usable by a ML algorithm that has to understand if actuators are working properly or if they need to change the way they are operating. If the system recognizes that there are some elements to be tuned or fixed, it activates the generation of some messages that have to be sent back to the machine. The machine will apply new information to the selected actuators to improve the way the machine is operating.

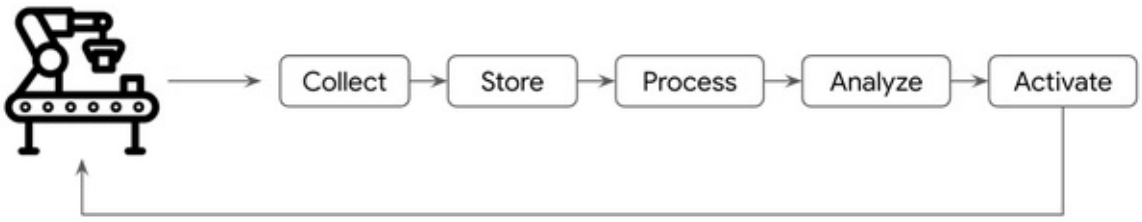


Figure 3-3. Machinery activation example

The actions that can be carried out fall into three categories:

- **Automatic actions**. Automatic systems can leverage the outcomes of a recommendations system to provide tailored suggestions to customers. This can help the top-line of the business by driving more sales.

- **SaaS integrations**. Actions can be carried through integration with third party services. For example, a company might implement a marketing campaign to try to reduce its customer churn — it could have analyzed the data and have implemented a propensity model to find which customers will positively react to a new commercial offer. The list of customer email addresses can then be sent to a marketing tool to activate the campaign.
- **Alerting**. It is possible to implement applications that monitor data in real time and that send out personalized communication when some specific conditions are met. For example, the pricing team might receive proactive communications when the traffic to an item listing webpage exceeds a certain threshold so that they can check that the item in question is not mispriced.

The technology stack for these three scenarios is different.

For automatic actions, the “training” of the ML model is carried out periodically usually by scheduling an end-to-end ML pipeline (this will be covered in Chapter 11). The predictions themselves are achieved by invoking the ML model deployed as a web service using such tools as AWS Sagemaker, Google Cloud Vertex AI, or Azure ML.

In this section we have seen the main building blocks of a modern data platform. Now let’s have an overview of the current approaches in implementing analytics and AI platforms adopted by the majority of the companies in the market to have a better understanding of how technology evolved and why the cloud approach can make a big difference.

Foundational elements

Historically, the main tool to manage data within a company has always been the spreadsheet that allowed anyone, especially people who did not come with a technological background, to perform analysis. It has quickly become the de facto standard as a decision making tool. As the volume of data started to grow and the level of analysis started to become increasingly complex, the limitations of a spreadsheet approach became abundantly clear.

From spreadsheet to the data warehouse

For large datasets and complex analysis, the tool of choice for managing and analyzing data is the relational database management system (RDBMS). RDBMS enabled people to take a less ad-hoc and more systematic approach:

- **Schema**. RDBMSs introduced a well defined way to describe the data via schemas, the blueprint of how the data is organized and connected;
- **SQL**. RDBMS introduced a standard query language (SQL) to query the data. This declarative language (you say what you want “SELECT x FROM Y” but not how the database should find x in Y) has become the standard way of extracting and analyzing data in relational databases.
- **Centralization**. RDBMSs enabled centralization of data repositories (i.e. the databases) with the ability to manage a bigger volume of data compared with legacy approaches.

Beyond analytics, RDBMSs have become the canonical backend for enterprise applications. Because applications have to save the state of objects over time, databases introduced transactions to serialize the writes from multiple concurrent users. In order to support transactions, databases store frequently changing data separately from “dimension data” to which they are connected through “foreign keys”. This method that RDBMSs store and process data is called database normalization, and it has become very common..

Unfortunately, RDBMSs are not efficient for analytics. They can not cope with the advanced analytics capabilities required to process the greater volume of data generated by enterprise because of three reasons:

- **Data layout**. It is very difficult to have a holistic overview of the data since the information is spread across multiple databases.
- **Data size**. The hardware on which RDBMs run are not able to scale and cope with analysis requirements as data sizes increase.
- **Use cases**. Data analysis needs are becoming more frequent and needed in more use cases by more business units across the enterprise.

The only way to cope with these challenges was to augment the CPU power and

memory of RDBMs — this was costly and after a certain threshold even inefficient.

Because of the limitation of RDBMSs, it became necessary to:

- Centralize even more the data to be analyzed
- Remove the tight interconnection with enterprise applications
- Standardize the management of who has access to the data
- Get rid of the intrinsic limitations of the normalization like the need to perform several table joins to have an holistic view of the data making indexing somewhat inefficient.

This led to the adoption of *Data Warehouse (DWH)*. An DWH is a repository for enterprise data (complete with its historical values) coming from various sources (mainly operational systems). There are two schools of thought when it comes to whether you build a centralized data warehouse for the entire enterprise (an Enterprise Data Warehouse or EDW) or separate data warehouses for different business units (called data marts).

The goal of an DWH is to develop an analytic platform to facilitate the understanding of business data and to help stakeholders in decision making based on real facts. A general architecture, aligned with the data lifecycle we described earlier, is made of several interconnected components as you can see in Figure 3-4.

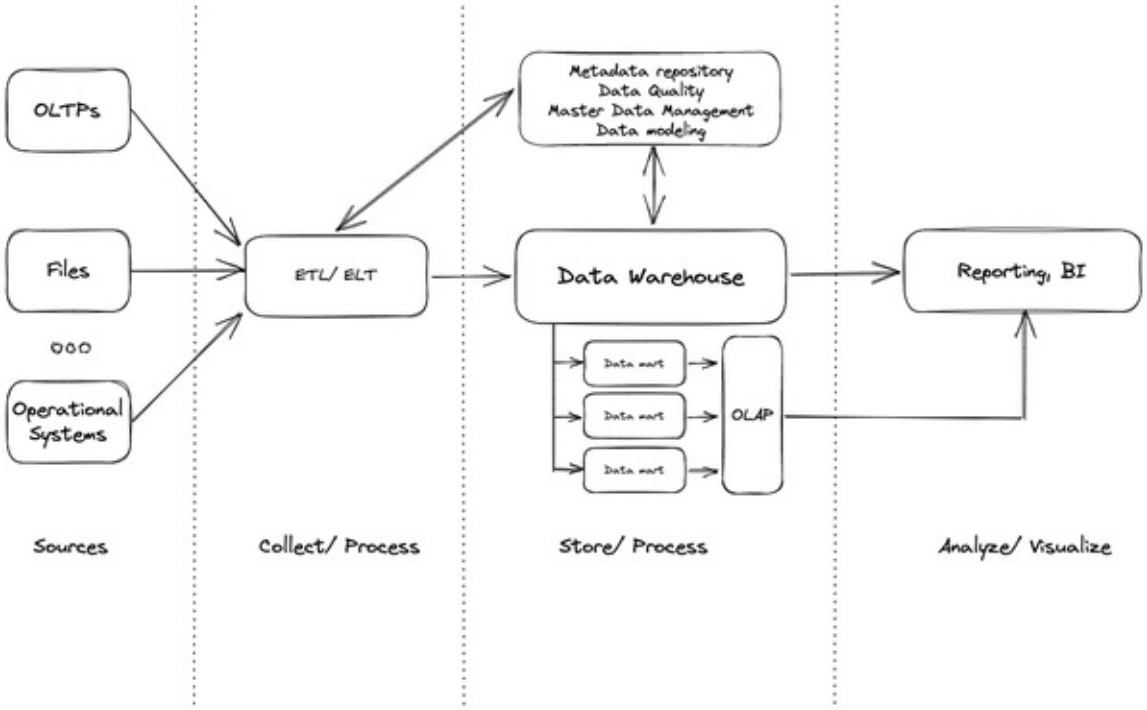


Figure 3-4. Data life cycle in a standard Enterprise Data Warehouse architecture

The core of this solution is still a database engine but instead of being driven by the transactional nature of the data (i.e. Online Transaction Processing or OLTP), it is focused on the analytical side of it (i.e. Online Analytics Processing or OLAP) generating the needs of new approaches in collecting and organizing information into new structures called *data marts*. The design of these new data structures generally follows two kinds of approaches called *star schema* and *snowflake*. While both approaches are similar at heart having a central *fact table* surrounded by *dimensions* table, they differ in the way they are structuring the dimensions themselves. In star schema, each logical dimension is denormalized in one table (that is for each dimension there is a table that contains all the details about the dimension itself). In snowflake schema, dimensions are highly normalized (that means that for each dimension you can have multiple tables joined).

Both approaches are pretty popular and have pros and cons. Which one you adopt should be mainly driven by the use case: if it is more metric analysis the star schema is the easiest path whereas if the goal is more dimension analysis the snowflake approach is the better choice.

The data coming from various data sources has to be prepared and transformed

before being stored in the DWH. There are several techniques and tools you can leverage but the most important are

- *ETL*: it has to convert numerous representations of the data into a well defined format called *conforming dimension* to be aligned with what is expected by the data mart where data will be uploaded. Several activities can be done at this stage like calculation of derived fields, concatenation, filtering, selection etc. These processing steps are carried out using an engine such as Apache Spark that is separate from the EDW and is not based on SQL. Once prepared, data can be loaded into the final system.
- *ELT*: same as before but with the major advantage that the data is loaded as-is into the DWH to be transformed by the DWH engine itself (using stored procedures, user-defined functions, views, etc.).

On modern cloud systems it is more common to see the adoption of ELT approach mainly because of its flexibility and the fact that the storage and the analytical resources are becoming more powerful and with a lower cost. It is then common to see organizations loading data in a raw format into the data warehouse and then leverage its analytical engine to perform all the needed transformations to generate new tables containing final and elaborated information to be leveraged for the further analysis.

Once the data is available in the DWH it used to be a common approach to generate *OLAP cubes* to facilitate the final analysis and reporting activities. These are generally pre-computed multidimensional datasets that allow final users to analyze the data from different dimensions (e.g. you are a sales person and you want to have a better understanding of your sales; you may want to know what are the most sold items or what is the most important brand or what is the geographical location where you are generating more revenue). The most common operations on top of OLAP cubes are:

- *Roll-up,* also known as drill-up operation, performs a dimension reduction removing the selected dimension from the given cube showcasing the data aggregated by remaining dimensions.
- *Drill-down* is the reverse of roll-up operations that introduces one or more additional dimensions in the analysis.

- *Slice & dice* performs a selection on one (*slice*) or more (*dice*) dimensions of a given cube generating a new sub-cube.
- *Pivot*, also known as *rotate*, rotates the data axes in view to provide an alternative data presentation.

The world of DWH is pretty big and complex and it is not limited only to the building blocks we have just described but there is a series of outline tools/ solutions that support its definition and related evolution. The most relevant examples are the following:

- *Metadata repositories:* solution to persist metadata. Metadata is data describing data and in this context it defines the various data warehouse objects. It could define for example the data lineage of a query (i.e. history of the data and the sequence of transformations applied to it) but even the structure of a data mart (e.g. schema, views, dimensions, etc.). It plays a pivotal role because it allows tools and users to better understand how to interpret the content of the DWH; for example it can be used to locate contents of the DWH or as a guide to perform some transformation operation.
- *Data quality:* these are tools that allow users to define rules (*at scalar, field, record or data set level*) that have to be applied to the data and detect violations in case of issues with the goal to fix them via an automatic or a manual operation.
- *Master data management:* these solutions are used to generate master lists for specific entities (e.g. customers, products, suppliers, etc.). They are helping in standardizing the data (e.g. leveraging the same time zone), removing duplicated data and reconciling data that could come from different data sources (e.g. an item could be identified by its product ID in one system and by its EAN code in another table). The goal is then to generate what is called a *golden record.*
- *Data modeling:* these are solutions to facilitate the development of the relational schema but even helping in understandability and extensibility.

Data warehouses have gained a lot of popularity within companies and over time, and we have seen several evolutions in the intrinsic patterns to try to

achieve better performance and scalability:

- *Massively parallel processing system (MPP)* that spreads the load across multiple servers running into a distinct entity called cluster enabling unmatched scalability compared with the traditional single server solutions.
- *DWH Appliances* that are high performance databases running on proprietary hardware and software facilitating the deployment and management.
- *Columnar Store databases* where data is accessed at column level instead of row level to speed up analytics elaborations where the access to a subset of fields is required instead of the entire record.
- *In-Memory databases* with the goal to store as much as data as possible into memory to speed up read operations.

Recalling the data lifecycle concept you have seen before and applying it to the Data Warehouse, as reported in Figure 3-4, you can see that it flows as the following: data can be collected from various data sources and then prepared for ingestion into the DWH engine system via ETL solutions. Data modelers can be leveraged to design a data model where to fit and process the data in data marts. OLAP cubes can be eventually generated to facilitate further data explorations and analysis with enhanced performance. As we said earlier, there are then several solutions that enrich the architecture, like for example data governance and quality tools, to provide final users with a complete solution for end-to-end data analysis.

AI when present was termed *advanced analytics* where statistical models and predictive models have been developed and used to address several business problems like risk measurement or predictive maintenance. In this area, several companies have made a lot of investments to develop specialized tools to allow advanced users to develop sophisticated solutions that have been incorporated into business life and that opened the door to the future of ML.

The advent of Hadoop and the birth of Data Lake

With the ubiquity of Internet services and the rise of mobile devices that were increasing their popularity, in the first decade of 2000 it turned out that Data

Warehouses were not enough to cope with the volume, velocity and variety of the data and it was necessary to develop a brand new technology. Google in 2004 published a famous paper called "*MapReduce, Simplified Data Processing on Large Cluster*" that presented a brand new algorithm they developed internally to be able to work with the data of their Internet index. The concept was very simple: specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key leveraging a distributed system made by a large cluster of commodity machines. One important aspect of this algorithm that helped in increasing its popularity is related to the fact that it is fully automated: data partition, inter-machine communication, failures management are fully managed by the underlying framework. This characteristic enabled the massive adoption even by people who lack experience in managing parallel systems. Google never released an implementation of this Map/Reduce algorithm but it opened the door to the open source ecosystem that, leveraging the knowledge of this paper, implemented Hadoop which is one of the most used solutions in the BigData space.

The foundation of Hadoop is its file system called *Hadoop File System* or *HDFS*. HDFS is massively parallel, highly fault-tolerant and self-healing. From an architectural point of view it is made by a master/ follower configuration. There is a single *Name node* that manages the file system namespace and regulates access to files by clients and one or more *Data nodes* (usually one per server in a cluster) which manage the underlying storage. HDFS exposes a file system namespace and allows user data to be stored in files. Data in Data nodes is stored in blocks (usually 128MB) replicated (generally 3 copies) across multiple servers so in case of failures of some nodes the data is still available for processing without affecting its availability. The fact that you have multiple copies of the same data facilitates even the loading spread across multiple servers so if the system recognizes that there is a load peak it can leverage other nodes for processing.

The good thing about HDFS is that it is essentially a file system and it means that it can handle any kind of data and not only files with a predefined schema as in a DWH system. Having said that, it is of course needed to define a schema or a structure when processing the data and this is where the concept of *schema on read* comes in: data is applied to a plan or schema as it is pulled out of a stored

location. This kind of approach enables the system to be able to manage any kind of data with minimal effort and open the door to the processing even of semi-structured and unstructured data that was impossible before with DWH technology without a pre-elaboration.

On top of HDFS sits Apache YARN (Yet Another Resource Negotiator) which is Hadoop's cluster resource management system. YARN was introduced in Hadoop 2 to improve the MapReduce implementation, but it is general enough to support other distributed computing paradigms as well. The fundamental goal of YARN is to separate the processing layer from the resource management layer, helping in organizing the usage of underlying resources to improve efficiency and performance. YARN allows the data in HDFS to be processed by various processing engines (i.e. not only MapReduce) to enable not only batch processing but even streaming and graph processing.

Thanks to these solid foundations, the Hadoop project has gained notable interests and it quickly became a rich ecosystem of services spanning across Ingestion tools, Relational databases, NoSQL databases, security and governance (see Figure 3-5). Several distributions are now available both in an open-source and commercial fashion (e.g. Cloudera or MapR). One of the most important developments is surely *Spark* that was initially developed in 2009 and has become part of Apache ecosystem with its 1.0 version in 2013. It can be seen as an evolution of the MapReduce concept that offers improved performance and flexibility especially in handling interactive or iterative computing jobs. It is a unified engine for large-scale distributed data processing that provides in-memory storage for intermediate computations incorporating libraries for composable APIs for ML (MLlib), SQL for interactive queries (Spark SQL), stream processing (Structured Streaming) for interacting with real-time data and graph processing (GraphX). Even if it is written in Scala, it can be used leveraging several programming languages like Java, Python and R and even SQL via SparkSQL extension.

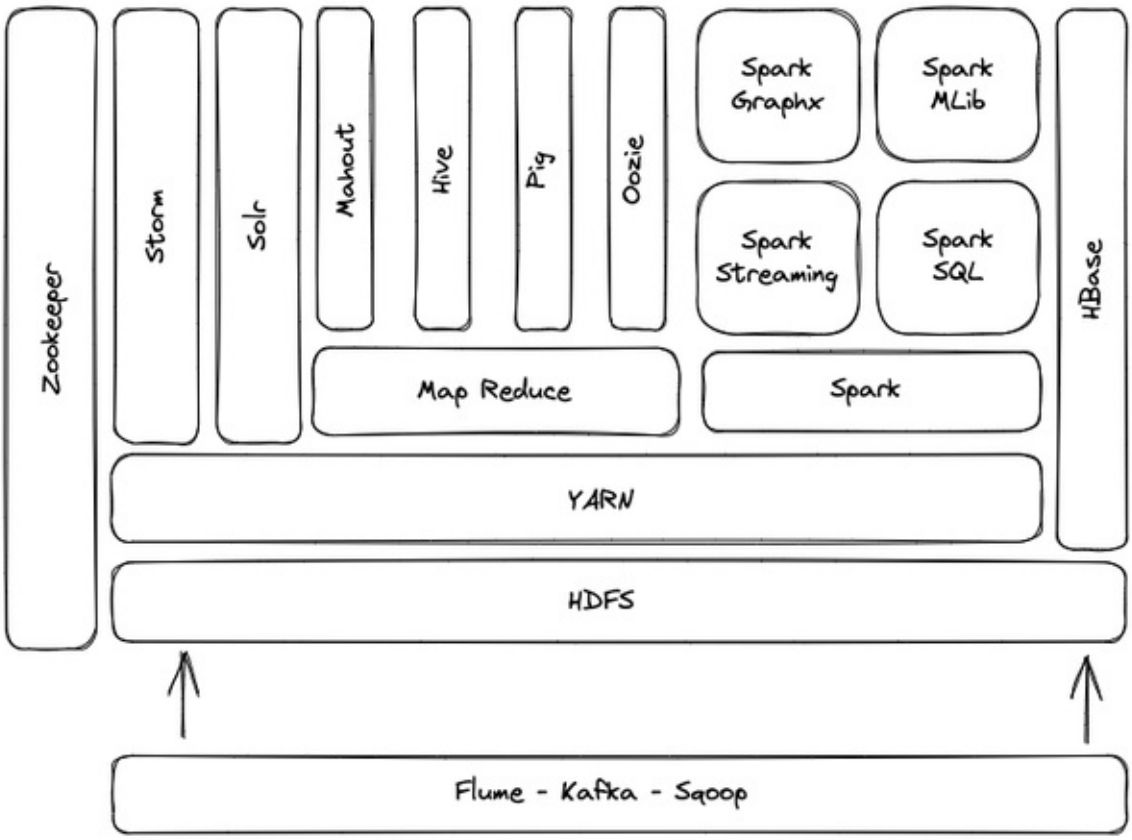


Figure 3-5. Open source Hadoop ecosystem visual representation

The ability to store and process different types of data (structured, semi-structured and unstructured) coupled with the extreme flexibility of the stack has made the Hadoop ecosystem the perfect solution for what is called *Data lake.* Essentially, a data lake is the *home* where you store and analyze all the different data that an enterprise has. The term was firstly coined by James Dixon, CTO of Pentaho, who wrote in his blog[6] that "*If you think of a data mart as a store of bottled water—cleansed and packaged and structured for easy consumption—the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples." and* Dixon's description is appropriate: within a data lake the information can be stored in its RAW format (no need to pre-elaborate it before consumption) and it can be leveraged by several different users (i.e. business analysts, data engineers, data scientists, etc.).

From an architectural perspective there are several ways to implement a data lake and different companies are adopting various approaches, but you can

generally think of storage organization as divided into 3 main *zones* (see Figure 3-6) where the data have different formats and elaboration stages:

- *RAW/ Landing/ Bronze* zone where the data is ingested in its original format.
- *Staging/ Dev/ Silver* zone where the more advanced users (e.g. data engineers, data scientists) process and prepare the data for the final users, and move it into the last zone.
- *Production/ Gold* zone where the data is cleaned, processed and ready to be used.

In some cases you could even have a fourth zone to keep sensitive data that is basically interconnected with all the other stages: the goal of this additional layer is to facilitate the governance of data access to make sure it complies with company and government regulations.

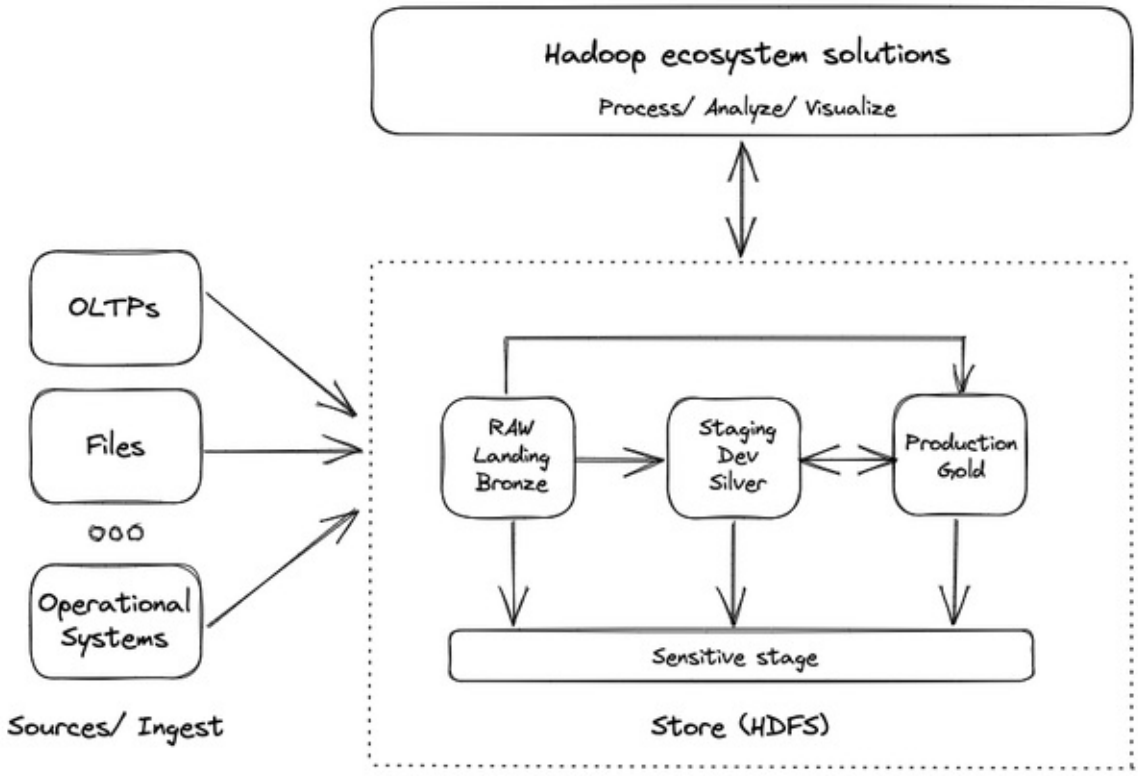


Figure 3-6. Data lake architectural organization

Having so many services available to handle the data creates the need to orchestrate various activities, and it is here where workflow orchestration

solutions (e.g. Apache Oozie) come into play. They are sometimes referred to as workflow automation or business process automation and their objective is to define, schedule, coordinate and manage data processing actions. Every data elaboration process usually involves multiple steps: for example you may want to leverage Sqoop to import data into HDFS from an external relational database and then you may want to perform some cleaning operations before converting it into a JSON[7] file. Then you would probably execute some query on top of it leveraging SparkSQL and finally you would probably store the result into a Parquet[8] file. Each of these jobs is an *action* that the system has to perform and it has to be:

- *Scheduled*: you may want to execute the action in a specific moment of the day or multiple times a day or when a specific event happens
- *Coordinated*: you may want to execute a specific task only when the previous task has been completed
- *Managed*: you may want to keep track of what is happening (e.g. failure or success) and log all the different activities to understand the time spent on each of them to identify, for example, possible bottlenecks

Usually the series of actions of a workflow are defined via a Directed Acyclic Graph (DAG) as shown in Figure 3-7. DAGs allow users not only to configure in detail each single task but even how the tasks are interconnected and what are the different possible paths (e.g. sequence execution, parallel execution, branches, etc.).

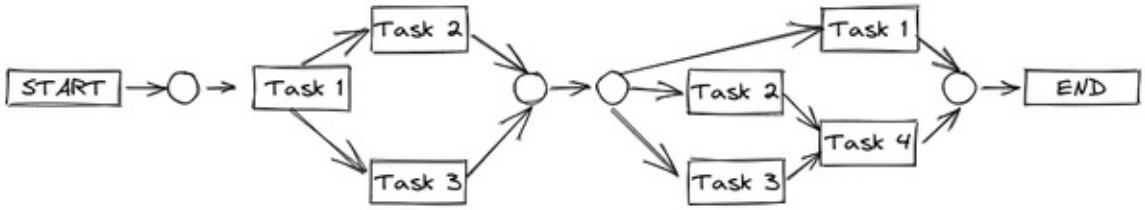


Figure 3-7. DAG example with branches and conditions (rounded circles in this image) and parallel execution

One of the advantages of workflow orchestration is its ability to simplify complex tasks into simple reusable steps. Another important aspect is related to the ability to leverage orchestration system features to avoid having users add extra logic to their applications. From a general standpoint orchestration tools

support metadata and data lineage and, most importantly, out of the box integration/ connectors with various software even outside the Hadoop ecosystem. You need to be able to manage dependencies, monitor the status of the various workflows from a centralized location while generating reports, restart failed workflows and eventually roll back even a single task in case of failure.

In this section you have seen how the organizations have developed their approach on data leveraging Data Warehouse and Data Lakes. In the following section we will discuss the related security and governance implications.

Governance and security

With the volume of data continuously increasing and with the need to give access to a broader set of people within the company, *data governance* and related *security* are assuming a central role within data platform architectures. The concept of data governance is extremely broad. You can address it both at macro level (where it takes a political meaning and it is related to how the data can flow across borders in a controlled and secure way), and a micro level (where it assumes a data management connotation and it is related to the function that ensures the integrity, security, usability and quality of the data collected by a company).

The goal of data governance is to enable companies to make the data available, in a secure and controlled manner, to all users in the organization; this is why it must be broadly applied during the entire data life cycle. A good approach on data governance results in better overall security: knowing for example what data an organization has, where it is, and who has access to what. It fosters privacy management and cybersecurity prevention that, nowadays, are an important agenda item for executives.

Let’s have a closer look at the tools available to govern and secure a data platform.

The role of metadata

One fundamental component is the *catalog* where all the information about the

data will reside. That kind of information takes the name of *metadata* which as we've stated, is essentially data that documents other data and that facilitates not only the searchability of the information but even its understanding. There are two levels of metadata:

- *Technical*: statistical information gathered from datasets (e.g. cardinality, density, range, mean, etc.) all together with the data defining their structures (i.e. table names, field names, field types, etc.)
- *Business*: information that helps business users to understand and correctly interpret the meaning behind the data stored in a particular system (e.g. a field in a table represents the name of the customer or a product identification). Generally that information is organized in glossaries, taxonomies and ontologies.

The catalog solution needs to have all the capabilities to collect this kind of information and, to achieve that, usually it leverages a *tagging* approach where users can manually assign appropriate terms, concepts and technical info to the various datasets. Since the volume of this data can be very huge, it is generally impossible to defer this activity to a single person within a company – this is why there are different approaches that can speed up and scale the collection of information:

- *Crowdsourcing*: the modern concept of tribal knowledge where people within the company (usually Subject Matter Experts or SMEs) comment/ rank datasets to provide a score that will be available to the other people when searching for data and that should be useful to understand the value and the quality behind the data.
- *Automatic cataloging*: leveraging the data made available by SMEs and applied to other datasets, it is possible to adopt AI and ML techniques to automatically infer information about new datasets like for example meaning behind table names or interpretation of field name abbreviations. This technique scales very well with the volume of datasets and can be continuously improved via data curation activities where SMEs check the automatically generated results and mark them as appropriate or not: results will be fed back to the algorithm to make it learn from the errors.

There are of course several solutions available to the market that enable such as activity (i.e. Collibra and Informatica just to mention a few) and you need to identify the best product that suits your needs focusing on services offered by the different vendors not only in terms of technical solution (i.e. features of the product must match your requirements) but even in terms of consulting and support.

Data lineage to foster security

When we talk about data governance and metadata collection via manual or automatic approach, we are inherently identifying the resources (e.g. entire datasets or some fields within a table) that have to be considered sensitive and need to be protected by design. It is pivotal not only to prevent access to information that should be considered secret within a company but even be ready to correctly manage the data in light of government regulations compliance that in some cases are becoming more and more stringent (e.g. GDPR in Europe, HIPAA in US, etc.). It is important to note that when talking about security and compliance your focus should not be limited to who is accessing what (that can be generally addressed via a fine-grained Access Control Lists (ACLs) policy management approach and data encryption/ masquerading techniques); it it should also be on:

- The origin of the data
- The different transformations that have been applied before its consumption
- The current physical data location (e.g. DataLake in Germany or DWH in the UK).

The tracking of this sort of metadata is called *data lineage* and it helps ensure that accurate, complete and trustworthy data is being used to drive business decisions. Looking at data lineage is also helpful in situations where you need to guarantee data locality because of some local legislation (e.g. telecommunication metadata in Germany): if you can track *where* the data is during its lifecycle, then you can put in place automations to prevent the access, usage and movement of that information by people who do not meet the minimum requirements.

As you have seen, metadata plays a central role in helping companies organize their data and govern its access. It is also crucial for evaluating the *quality* of the collected data in terms of accuracy, completeness, consistency and freshness: poor-quality data might lead to wrong business decisions and potential security issues. There are several techniques, and related tools, that can support data quality activities but the most common ones are:

- *Standardization*: the process of putting data coming from different sources into a consistent format (e.g. fix inconsistencies in capitalizations, characters, updating values in the wrong fields, data formats, etc.).
- *Deduplication*: the process of identifying duplicate records (leveraging a similarity score) and then proceed with the deletion of the duplicated values.
- *Matching*: the process of finding similarities or links between data sets to discover potential dependencies.
- *Profiling* and monitoring: the process of identifying a range of data values (e.g. minx, max, mean) revealing outliers and anomalies that could bring for example to a data fix or to a schema evolution.

Governance and security are broad topics that require an ad hoc treatment. Here we just scratched the surface but it is fundamental to have clear in mind the basic concepts especially when dealing with huge amounts of data as you will see in the next section.

Moving to the public cloud

As we have discussed in the previous chapters, data is exploding. Nowadays it has become commonplace to see organizations capturing information not only from standard sources like Enterprise Resource Planning Systems (ERPs), enterprise communication systems, business and commercial applications etc. but also from mobile devices, social media, and edge devices. There is an increase:

- In the data volume (according to IDC and Seagate should become 175 Zettabyte in 2025[9])

- In the data variety (with an increase of unstructured information such as text, audio and video)
- In the data velocity (according to IDC should be 30% streaming nature in 2025[10])

These are ongoing trends that show no sign of stopping and it will be even more amplified once 5G technology becomes mainstream and connects billions of objects that may generate and process data in real time. In this section we will have a closer look at the main challenges and the new standards that are bringing organizations to adopt public cloud services to handle their data.

Handling data explosion

In such a situation of increasing volume, velocity and variety, it is critical that the data platform is backed by an infrastructure that is capable of scaling without being overwhelmed by costs, that can guarantee consistent performances to multiple users in parallel within the same organization, and that can handle eventual peaks with no service disruption in an elastic fashion. It is very challenging to achieve this kind of characteristic within an on-premise environment and this is why many companies are increasingly leveraging public cloud services (Azure, AWS, and Google Cloud to name the most popular ones) that can offer advanced analytical services ecosystems to deploy any kind of workload to manage enterprise data. There are several benefits in switching to a public cloud environment and the most relevant are:

- *Cost effectiveness*: from a general point of view public cloud vendors provide the flexibility to access their services in multiple fashion with various models where the company could adopt a pay-per-use model to pay for only what they are using or they can sign medium/ long-term commitments to leverage flat rate usage of the services.
- *Fast setup*: setting up a cloud environment even from scratch is generally fairly simple and not require a lot of effort and, most importantly, it does not require a huge upfront investment. Once the organization decides to deploy an environment, this is usually se tup in a very short amount of time (minutes generally). Organizations can even leverage Infrastructure As A Code (IAAC) solutions like Ansible or Terraform to automate the process

of infrastructure setup.

- *Elasticity and agility*: organizations need to move fast and they cannot wait for weeks to purchase new servers or to patch their machines. They need to rapidly update and scale their architecture in case of increased demand or unexpected events without introducing latency or disruption.
- *Low maintenance*: the majority of the services are usually offered in a fully managed fashion which means organizations do not need to staff huge teams managing the underlying infrastructure and instead can focus its staff on other activities (e.g. automation, CI/CD, analysis, etc.)
- *Economies of scale*: since the infrastructure costs are shared across multiple users, the cloud providers typically optimize the hardware needs of their data centers and offer the services at lower costs.
- *Security and compliance*: all cloud providers generally have recognized compliance certifications (e.g. ISO 27001, SOC1/2/3, CCPA, etc.) that have to be continuously renewed and they usually have extended global security teams that invest in research to provide the platform continuously with the best in class security system.
- *Ability to be global from day 0*: no matter what the business of the organization is (e.g. a new e-commerce startup or a strongly recognized financial institute), there might be the need to provide a 24/7 service to customers who are globally distributed. Public cloud providers come with a massive network of connected data centers that allow organizations to operate basically anywhere and still respect the compliance rules they need to follow (e.g. data needs to reside in the European Union). They usually even provide global SME support that operates 24/7/365 and is able to assist customers in case of problems.

As you will see better in the following chapters, public cloud services are the foundations of the modern data platforms and even if there are scenarios where the adoption of private data centers are still required, their adoption will be more and more reduced in the near future.

Real time analytics as a new standard

Scalability is not the only characteristic the infrastructure has to provide users with: there is another aspect you need to be consider when designing a modern data platform – its ability to handle real-time data collection and analysis. Note that many technologies that claim to perform real time analysis of their data are not doing so – the majority of them are adopting a technique called micro-batching. It is a practice of collecting data in small chunks called batches with the aim to process each batch all together. The processing phase occurs more frequently and it can provide the illusion of a continuous processing if micro batches are executing one after the other with very limited time separations between the batches (e.g. seconds or double digit milliseconds). It is important to note that data is collected based on a predefined threshold or frequency before any processing occurs. Fluentd, Logstash and Apache Spark Streaming are common frameworks that leverage the micro batching approach.

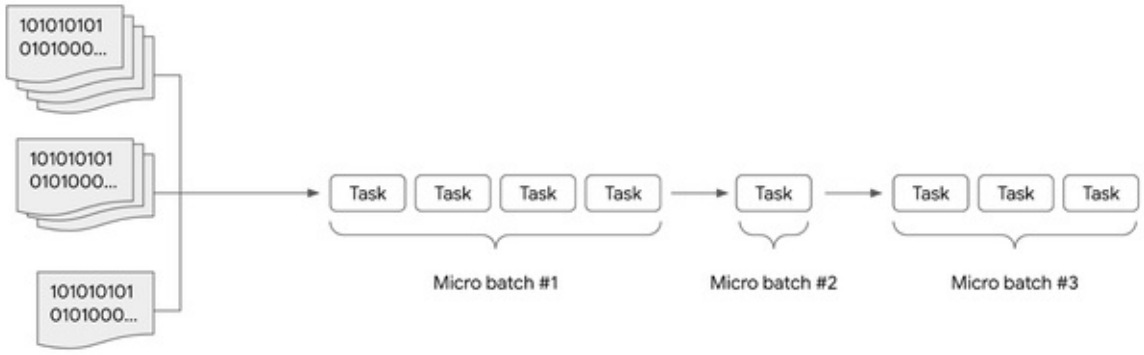


Figure 3-8. Micro batch processing

The problem with micro-batching is that it is impossible to react to an individual event as it happens – you can only react at periodic intervals (even if those periods are vanishingly small). This limits its application in use cases such as fraud prevention. With the increase of the data velocity we discussed earlier, we have seen the rise of some use cases for which there is the need to process data immediately as it is generated with single digit millisecond delay. Some examples are:

- *Clickstream analysis*: analyze the interactions users have with an app or with an e-commerce website to optimize the behavior, the pricing of products, promotions and inventory
- *Fraud detection*: analyze accounts activity to identify fraudulent actions and generate immediate alerts or disable some functionalities (e.g. ability to

perform issue payments) in advance

- *Logs analytics*: collect and analyze business and/ or technical information of your applications to monitor the overall status and to automate some actions to be taken in case of issues (e.g. scale the front end server cluster if the number of users navigating the app is increasing)

Modern data platforms leveraging public cloud services are today able to provide solutions for real time processing and can handle data at scale: no matter if you are generating thousands of millions of events per second, the platform should be always able to manage the load and eventual spikes without any relevant problems.

Easy integration of AI/ML capabilities

As you have seen, the advent of public cloud enabled companies to handle huge volumes of data and opened the door to real time analysis. A capability that is intrinsically related is the opportunity to develop and leverage AI/ML techniques to unlock the hidden potential of such data, getting access to insights that were very difficult to extract before and implementing use cases that were unthinkable in the past. Programs that generate real time captions in videos, tools that automatically recognize if a machine in a plant is going to have a failure in the next hours or virtual assistants that can handle human interactions via natural language are within every company's grasp. The implementation of these solutions requires notable computational power: data need to be collected, cleaned, processed and inputted into a ML model that has to be trained and validated in a process that could have several iterations before it's successful.

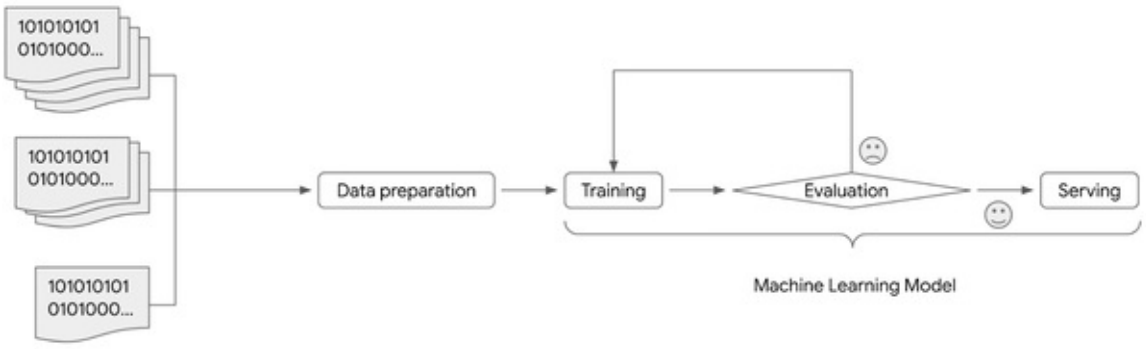


Figure 3-9. ML workflow

Then, once ready, the model has to be served to the final user: generally this can be done via the exposure of an API or it could be embedded directly into the code of the final application. In any case it is code to be executed that can potentially receive even millions of calls per seconds depending on the use case (think about fraud detection in real time) or the success of the app (think about a chatbot published in the main page of a global e-commerce website).

Implementing such applications in a standard on-premise environment is somewhat challenging because it is very difficult to match what is available in a public cloud environment nowadays: the latest versions of GPUs can speed up the linear algebra computations that are the basis of ML code. All together with this there are even several software solutions that facilitate the access of these technologies enabling people with limited AI/ML skills to implement working algorithms without writing any line of code: they just need to select the data used for the training, tweak some configurations and wait for the model to be trained and then served. It seems like a dream but in fact it is something that you can use already today.

Now that you have seen why public cloud fixes the data platform modernization problem, let’s look next at why you need to focus not just on single products but rather on the overall architecture.

Modernizing Data Workflows

In this section we will explain the importance of having a comprehensive vision when dealing with data modernization.

Job to be Done

When we think about a modernization program, our mind gravitates to nouns, to the things we have. We think about the pain we are experiencing and resolve to modernize the tools we use. We decided that we want to upgrade our database and make it globally consistent and scalable, so we modernized it to Spanner or CockroachDB. We decided that we want to upgrade our streaming engine and make it more resilient and easier to run, so we modernized Flink or Dataflow. We decided we are done tuning clusters and queries on our data warehouse, so we modernized to BigQuery or Snowflake.

These are all great moves. You should definitely upgrade to easier-to-use, easier-to-run, much more scalable, much more resilient tools whenever you can. However, if you do only like-for-like tool changes, you will end up with just incremental improvements. You will not get transformational change from such upgrades.

Indeed, modern society and most of the innovation we take for granted is obtained by transformative changes in the verbs—in the way that we solve problems. These are of course made possible by the thousands of incremental changes in the tools we use to solve those problems. But the best way to benefit from improved tools is to forget the individual tools.

Instead, determine the key job that your users wish to accomplish, and approach it from first principles. Why?

It's all about dependencies

When we embark on a data modernization program, we tend to put the focus on solutions: we focus for example on the Oracle databases and Teradata instances we need to migrate. We find the corresponding cloud product and embark on a migration journey.

Of course, you can't just migrate the data. If you only need to transfer the data and schema from Teradata to BigQuery or Snowflake or Redshift, you'd be done in a few hours. There is an automated transfer service available.

The problem is that you have to migrate the dependencies too. What dependencies?

ETL pipelines that used to populate the Teradata instance now have to insert records into BigQuery instead. Those ETL pipelines have to be moved to the cloud too. And all the data sources and libraries that these pipelines depend on.

Your simple project is now a tangle of dependencies that have to be ordered and moved.

Modernize workflows

To avoid this trap, when you start your data modernization journey, force yourself to think of verbs not nouns. You are not modernizing your database or

your data warehouse. You are modernizing your data workflow. The data workflow is the job to be done. Again, approach it from first principles.

What does it mean to modernize a data workflow? Think about the overall task that the end-user wants to do. Perhaps they want to identify high-value customers. Perhaps they want to run a marketing campaign. Perhaps they want to identify fraud. Now, think about this workflow as a whole and how to implement it as cheaply and simply as possible.

Avoid the sunk-cost fallacy. Just because you have an ETL pipeline that knows how to ingest transactions doesn't mean that you have to use that ETL pipeline. Throwing it away completely, or salvaging small scraps from it, is often the smartest option. Porting it to a "modern" platform ought to be only a stop-gap until you can completely modernize the workflow.

The way to identify high-value customers is to compute total purchases for each user from a historical record of transactions. Figure out how to make such a workflow happen with your modern set of tools. When doing so, lean heavily on automation:

- **Automated data ingest**. Do not write bespoke ELT pipelines. Use off-the-shelf EL tools such as Datastream or Fivetran to land the data in a data warehouse. It is so much easier to transform on the fly, and capture common transformations in materialized views, than it is to write ETL pipelines for each possible downstream task. Also, many SaaS systems will automatically export to S3, Snowflake, BigQuery, etc..
- **Streaming by default**. Land your data in a system that combines batch and streaming storage so that all SQL queries reflect the latest data (subject to some latency). Same with any analytics — look for data processing tools that handle both streaming and batch using the same framework. In our example, the lifetime value calculation can be a SQL query. For reuse purposes, make it a materialized view. That way, all the computations are automatic and the data is always up to date.
- **Automatic Scaling**. Any system that expects you to pre-specify the number of machines, warehouse sizes, etc. is a system that will require you to focus on the system rather than the job to be done. You want scaling to be automatic, so that you can focus on the workflow rather than on the tool.

- **Query rewrites, fused stages**, etc. You want to be able to focus on the job to be done and decompose it into understandable steps. You don't want to have to tune queries, rewrite queries, fuse transforms, etc. Let the modern optimizers built into the data stack take care of these things.
- **Evaluation**. You don't want to write bespoke data processing pipelines for evaluating ML model performance. You simply want to be able to specify sampling rates and evaluation queries and be notified about feature drift, data drift, and model drift. All these capabilities should be built into deployed endpoints.
- **Retraining**. What should you do if you encounter model drift? 9 times out of 10, the answer is to retrain the model. So, this should also be automatic. Modern ML pipelines will provide a callable hook that you can tie directly to your automated evaluation pipeline. Automate retraining too.
- **Continuous training**. Model drift is not the only reason you might need to retrain. You want to retrain when you have a lot more data. Maybe when new data lands in a storage bucket. Or when you have a code check-in. Again, this can be automated.

Templatized setup

As you can see, once you get to a fully automated data workflow, you are looking at a pretty templatized setup that consists of a connector, data warehouse, and ML pipelines. All of these can be serverless, so you are basically looking at just configuration, not cluster management.

Of course, you will be writing a few specific pieces of code:

- Data preparation in SQL
- ML models in a framework such as Tensorflow or Pytorch
- Evaluation query for continuous evaluation

The fact that we can come down to such a simple setup for each workflow explains why an integrated data and AI platform is so important.

Transform the workflow itself

You can make the workflow itself much more efficient by making it more automated using the modern data stock. But before you do that, you should ask yourself a key question: “is this workflow necessary to be precomputed by a data engineer?”

Because that’s what you do whenever you build an ELT or ETL pipeline — you are precomputing. It’s an optimization, nothing more.

In many cases, if you can make the workflow something that is self-serve and ad-hoc, you don’t have to build it with data engineering resources. Because so much is automated, you can provide the ability to run any aggregation (not just lifetime value) on the full historical record of transactions. Move the lifetime value calculation into a declarative semantic layer to which your users can add their own calculations. This is what a tool like Looker will allow you to do.

Once you do that, you get the benefits of consistent KPIs across the org, and users who are empowered to build a library of common measures. The ability to create new metrics now lies with the business teams where this capability belongs in the first place.

Summary

Using the lens of the data lifecycle, it should now be clear why organizations need to modernize their data platform. In this chapter, and you have seen some design patterns to follow throughout the journey. The key takeaways are:

- Organizations are looking at their data globally to have a clearer understanding of their business, better serve their customers, develop smarter products and be more efficient. To achieve all these goals it is pivotal to transform raw data into insights and then into wisdom. This journey, made by several stages, is called the data lifecycle.
- The data lifecycle has 5 stages: collect, store, process, analyze/ visualize and activate.
- The architectures aimed to extract value from the data have evolved: from the RDBMS systems we have seen the rise of the Data Lake and Data

Warehouse.

- Data governance and security are becoming more crucial because data is exploding, architectures are becoming more complex, the number of users accessing and leveraging the data is increasing, and legislators are introducing more stringent rules to handle the data.
- Traditional on-premises solutions suffer from the inability to cope with modern volume, velocity and variety. Public cloud architectures with their separation of compute and storage, ability to handle high throughput streaming feeds, and ML capabilities offer technical solutions to address these issues.
- Focus on data workflow modernization rather than on upgrading individual tools. Rather than choosing just a single tool, choose the right tool for the right workload, because this would reduce operational cost, license cost and use the best of the tools available. At the same time, use the right team setup for different workloads. Let the deciding factor be driven by Business Requirements: each Business Unit or team would know the applications they need to connect with to get valuable business insights. This, coupled with the data maturity of the organization, are key to choosing the right data processing tool for your needs.

The following chapter will give you more insights on how to put the focus on developing and fostering a data culture within the organization.

---

1 *https://online.hbs.edu/blog/post/data-driven-decision-making*

2 *https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers*

3 *https://www.sfexaminer.com/archives/how-the-warriors-use-data-analytics-to-engineer-more-wins/article_c2565040-6887-5f50-a6f4-47ec0efc4758.xhtml*

4 Usually we talk about 5Vs including even Veracity and Value but they are not relevant at this stage

5 See Chapter 10 in *The Self-Service Data Roadmap* (O'Reilly 2020) for more about data virtualization.

6 *https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/*

7 *https://www.json.org/json-en.xhtml*

8 *https://parquet.apache.org/*

9 *https://www.seagate.com/files/www-content/our-story/rethink-data/files/Rethink_Data_Report_2020.pdf*

10 *https://www.zdnet.com/article/by-2025-nearly-30-percent-of-data-generated-will-be-real-time-idc-says/*

Chapter 4. Designing your data team

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the fourth chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

When designing a data platform there are several technical aspects to take into consideration: performance, cost, operational overhead, operational excellence, integration of new analytical and ML approaches, etc. However, these technical aspects will fall to the wayside if you don't address the culture of the company — new technologies require a willingness from employees to change their mental models and ways of working. Another key aspect to keep in mind is the skills that existing employees currently possess and will need to pick up. In some cases, employees who learn new skills and change their way of working will end up in a different role than the role they had before the data platform was in place.

In this chapter we explore how organizations can plan and orchestrate these changes in mental models, workflows, technical skills, and roles. Every organization is unique, and so building a data platform will involve devising a granular plan for each division and employee in it. In this chapter, we describe what such a granular plan would look like for different types of organizations.

Classifying data processing organizations

Organizations can succeed by employing different strategies based on their talent. Just as each sports team has a different game plan, each organization must decide the best strategy for its workforce.

When dealing with data, organizations need to decide on the best strategy for their workforce: do they need a small, but highly capable (and expensive) group of data engineers? Or should they leverage a large, but relatively inexpensive workforce of data analysts to enrich and transform data that can be acted on? How much domain knowledge do these workers have? Would training the current workforce to carry out higher value work be realistic? Or should the organization invest in generative AI or no-code tools and make such foundational pieces of technology available to the larger workforce? The best technology approach will often vary within the organization — the composition of the workforce will vary between the sales team and the factory floor. So, the granular plan involves detailing the best technology approach for each business unit. Technology-wise, the plan will also make choices between an approach based on standard ETL (*hard skills on ETL tools required)* and a modern one based on ELT (*more general SQL skills required)*?

The notion that there is a "*one size fits all*" or "*best*" approach is simply incorrect. If a sports team has a strong defense, they should try to win by playing defense, not by copying the offensive strategies of a team with skilled offensive players who are capable of rapid attacks. Similarly, if an organization has a strong team of data analysts, they should focus on their people rather than attempting to transform into an organization full of data engineers.

Consider the traditional persona value chain as sketched in Figure 4-1. You can see that every data user in an organization has a small and specialized set of technical skills. If an organization wants to increase the scope of their data analysis team, they also have to scale the size of their data engineering and data science teams to make sure enough people have the right technical skills to support the data analysts.

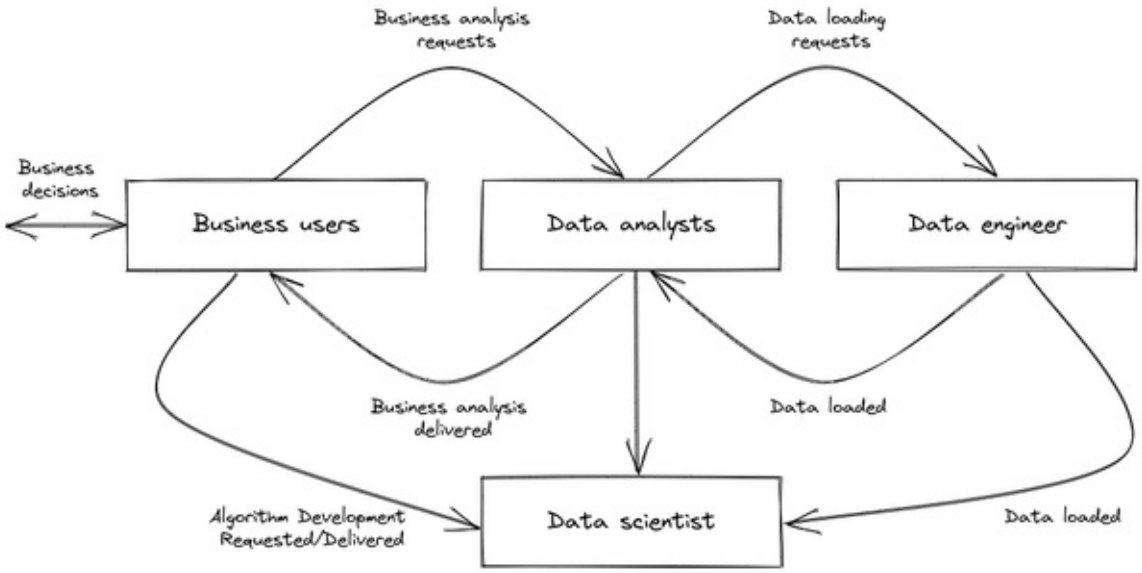


Figure 4-1. Data processing: traditional persona value chain

The new paradigms provided by the public cloud have opened new options for how data processing, data analysis, and algorithm development get done. Cloud technologies facilitate access to new ways of working — they enable analysts to carry out batch or real-time data processing tasks that used to be managed by data engineers while also allowing them to experiment with off-the-shelf data science solutions. Furthermore, the potential surface area of any given persona - *the capabilities they possess and the responsibilities that fall to them* - has expanded. As data technology becomes more accessible, data workers are able to take on new tasks and address the data value chain without the bottlenecks associated with the traditional persona value chain. This is leading to a blending of skills across roles, allowing existing teams to more easily scale to additional scope. The distinction between data analysts focused on solving problems using an ELT approach with SQL code and data engineers/ data scientists more aligned with an ETL approach and general purpose code (e.g. Python, Java, Scala, etc.) is becoming less clear. Blended approaches (as depicted in Figure 4-2) are becoming more common because they can take advantage of the best of both ETL and ELT patterns. The majority of the organizations we see today fall into this blended model, though the balance of roles and how much of the data processing has to be done either via ETL or ELT varies depending on the type of the organization.

In simpler terms, data workers are now able to do more with the data they have, and they are able to do it more efficiently. This is because the technology is more

accessible, and because data workers are able to blend their skills with those of other data professionals. This is leading to a more efficient and effective way of processing data.

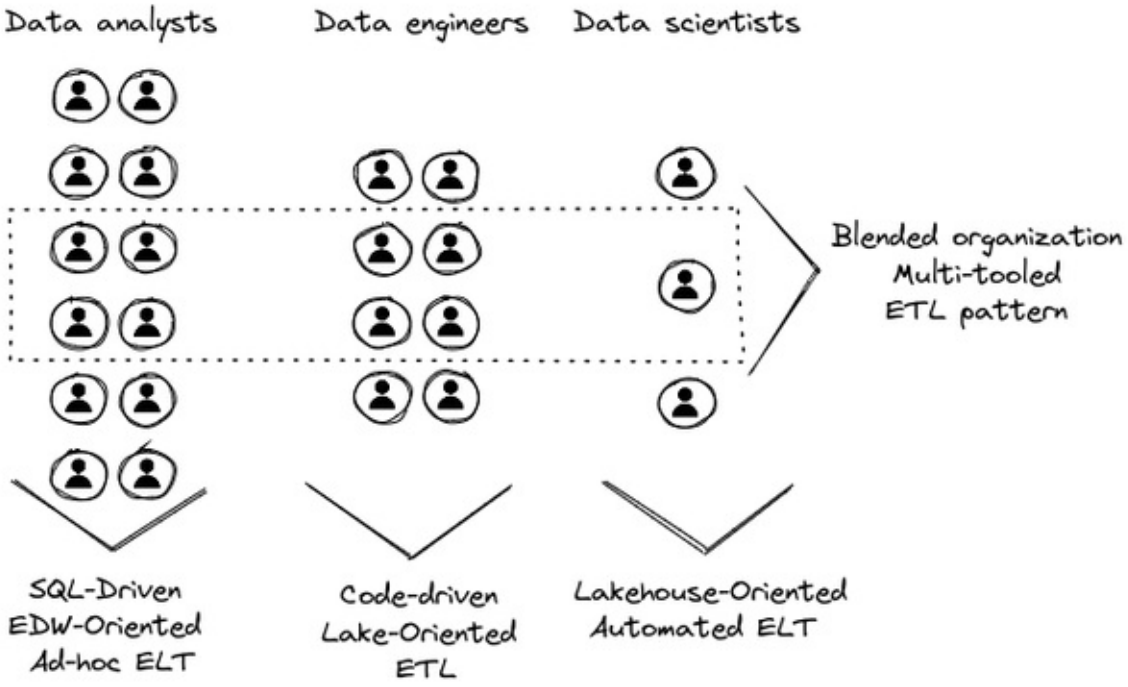


Figure 4-2. Data processing: persona framework

Organizations can be broadly classified under 3 types: *data analysis driven*, *data engineering driven*, and *data science driven*. In the following sections we will cover an idealized way of building a data processing organization for each of these types. In reality, companies will consist of different divisions or business units that fall into these categories, and so will find themselves applying all the strategies. Some teams will be a combination of roles and so, the transformation will involve a blended approach.

Data-analysis driven organization (DADO)

A data analyst-driven organization is one in which data analysts play a central role in decision-making. It is important to note that whether an organization is analyst-driven is not a black-and-white issue, but rather a spectrum of overlapping characteristics.

- **Mature industry.** At a high level, these organizations are well-known,

established businesses with well-established (and perhaps antiquated!) systems. The industry in which they operate is typically mature and stable. The primary work involves human analysis of a variety of products or situations. Canonical examples of data-analyst driven organizations are the merchandising units of retailers (e.g. Walmart) and commercial loan processing divisions of large banks (e.g. JP Morgan Chase).

- **EDW + Batch ETL.** In technical terms, the central information hub is an EDW that has been built over time with a high level of technical debt and legacy technology. The transformation of data within the data warehouse is carried out through scheduled ETL processes such as nightly batches. This batch process adds to the latency of serving the data.
- **Business Intelligence.** The majority of data professionals in the organization are accustomed to answering business questions by running SQL queries against a centralized data warehouse, creating reports and dashboards using BI tools, and using spreadsheets to access similar data. As a result, the internal talent pool is most comfortable with SQL, BI tools, and spreadsheets.

Note that even in mature industries such as e-commerce and fintech, emerging digital organizations may seek to capture the fastest growing digital areas and customer segments with the greatest potential. Such digital natives may have a different workforce composition than established players (e.g. Etsy vs. Walmart; Paytm vs. JP Morgan Chase); we wouldn't put digital natives in the same category as established players.

Now that you have a clearer view of what we mean by DADO, let's discuss the main levers to pull for the transformation.

The vision

Analysts are familiar with the business and use SQL and spreadsheets to analyze data. The best way to scale the use of a cloud data platform is to ensure that they can do advanced analytics through interfaces that they are accustomed to. This means providing easy-to-use tools for bringing data into the target system and seamless connectivity to analysis and visualization tools.

This was once common practice with traditional EDWs. Data was enriched, transformed, and cleansed using SQL, and ETL tools were used to orchestrate the process. Similarly, materialized views and user-defined functions can be used to enrich, transform, and cleanse data in a modern data warehouse.

However, this assumes that the analyst already has access to all the data sources. Creating complex ingestion pipelines used to be costly and often cumbersome. Therefore, data pipelines were managed outside of the data warehouse due to resource and cost constraints.

This is no longer the case with new cloud data warehouses. The role of ingestion is now simply to bring data close to the cloud, and the transformation and processing part moves back to the cloud EDW. This leads to data being staged in a storage bucket or on a messaging system before being ingested into the cloud EDW.

All of this not only reduces the time required to make data available, but it also frees up data analysts to focus on looking for data insights using tools and interfaces that they are used to. Therefore, in the new world, ELT should replace ETL tooling — data analysts can use SQL orchestration tools such as dbt or Dataform to string together SQL statements to carry out ELT. Ingesting data directly from a source or staging area allows analysts to exploit their key SQL skills and also increases the timeliness of the data they receive. They don’t need to wait for a swamped data engineering team to implement ETL pipelines.

In conclusion, the best way to scale the use of a cloud data platform is to provide analysts with easy-to-use tools (such as dbt) and interfaces (such as Excel or Tableau) that they can easily become proficient in. This will enable them to do advanced analytics without having to wait for data engineering teams to implement complex ETL pipelines.

Once the data is available in the cloud EDW, it is time to begin the analysis. In the past, much data analysis was done using spreadsheets, but spreadsheets often struggle to handle the volume of data that needs to be analyzed in the new world. Even though Google Sheets and Excel have capabilities to connect live to data warehouses, it is still somewhat clunky. We recommend that analysts be provided access to create visualizations and reports using modern business intelligence tools that are capable of handling large datasets.

The personas

Focusing on the data department, the main roles and processes in these types of organizations can be summarized as follows:

- *Data analysts* who receive, understand, and fulfill requests from the business, and make sense of the relevant data.
- *Business analysts* who put the information into context and act on the analytical insights.
- *Data engineers* who focus on the downstream data pipeline and the first phases of data transformation, such as loading and integrating new sources. They also manage data governance and data quality processes.

Data analysts aim to meet the information needs of their organizations. They are responsible for the logical design and maintenance of data. Some of their tasks may include creating layouts and designs for tables to meet business processes, as well as reorganizing and transforming data sources. Additionally, they are responsible for generating reports and insights that effectively communicate trends, patterns, or predictions that the business requests.

To build the mission for the DADOs, it is necessary to expand the experience and skill set of the data analyst community in two ways. First, it is critical to promote the trend of data analysts learning about the business. Data analysts need to acquire a deep knowledge of business domains. Second, data analysts need to acquire the technical skills to analyze and depict data regardless of its volume or size. Fortunately, this is now possible using SQL and BI tools. The expansion of the skills of data analysts into the business and into big data is depicted in Figure 4-3.

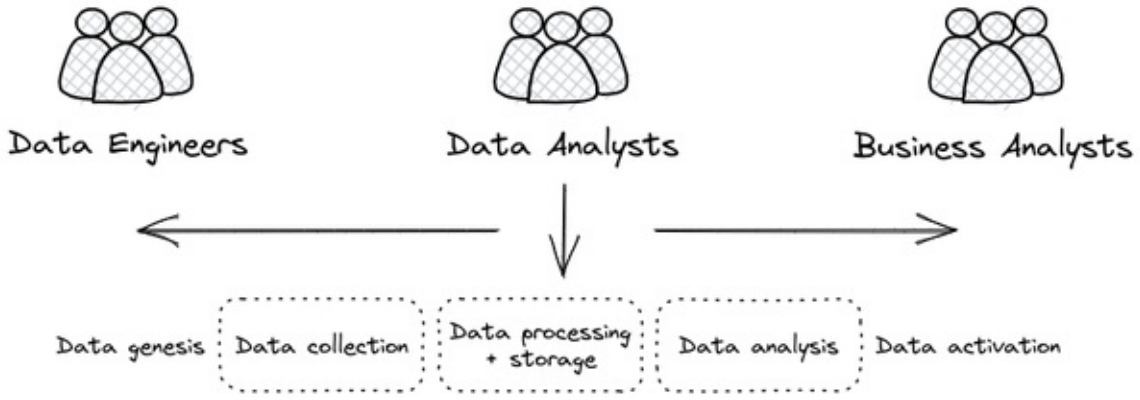


Figure 4-3. Data analysts domain expansions for the development of a data-driven strategy

Cloud-based data warehouses and serverless technologies have expanded the responsibilities of data analysts. This is because analysts can now focus on adding value to the business by analyzing data, rather than wasting time on administrative and technical management tasks. Additionally, the volume and type of data that can be stored in a data warehouse is no longer a limitation, so analysts can now go deeper into the business to find insights.

The data warehouse can function as both a landing area for data and a system-of-record for both structured and semi-structured data. This means that data analysts have all the data they need to analyze in one place. They can use SQL to query the data, and they can also use SQL to process and transform the data. This allows analysts to take on some of the work that data engineers typically do, such as data integration and enrichment.

Overall, cloud-based data warehouses and serverless technologies have made it possible for data analysts to be more productive and to add more value to the business.

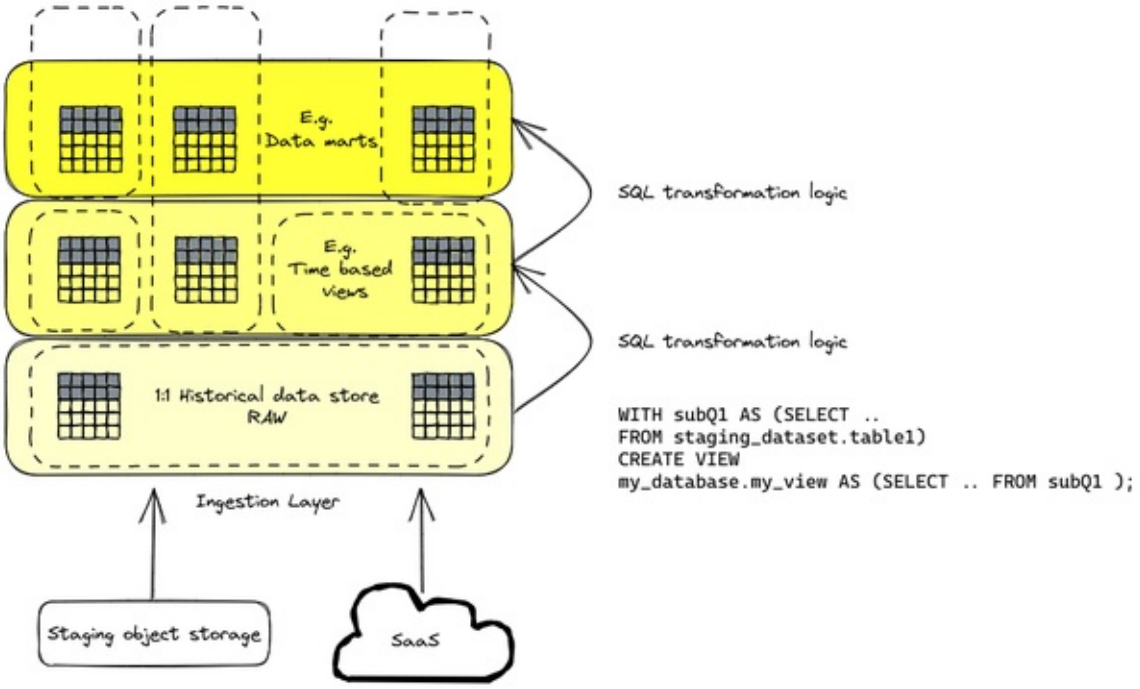


Figure 4-4. ELT paradigm - a SQL-first approach to data engineering

DADOs embrace the concept of ELT rather than the traditional ETL. The main difference is the common data processing tasks are handled after the data is loaded to the data warehouse. ELT makes extensive use of SQL logic to enhance, cleanse, normalize, refine, and integrate data and make it ready for analysis. There are several benefits of such an approach: it reduces time to act, data is loaded immediately, and it is made available to multiple users concurrently. Therefore, a change management strategy for a DADO has to focus on aspects such as SQL, views, functions, scheduling, etc.

In a DADO, the number of data engineers and data scientists will be small because the data analyst teams will be largely self-sufficient and capable of building simple data pipelines and ML models. This is not to say that there will be no data engineers or data scientists in the organization!

Even in DADO, data engineering teams generally control extraction of data from source systems. While it can be made easier through the use of SQL-based tools, enabling data analysts to do some of that work, you still need a solid data engineering team. There are batch jobs that would still require creating data pipelines that would be more suitable for ETL. For example, bringing data from a mainframe to a data warehouse would require additional processing steps: data types need to be mapped, COBOL books need to be converted, and so on.

In addition, for use cases like real time analytics, the data engineering teams will configure the streaming data sources such as Pub/Sub or Kafka topics. The way that you deal with generic tasks is still the same -- they can be written as generic ETL pipelines and then reconfigured by the analysts. For example, applying data quality validation checks from various source datasets to the target environment will follow a template set up by a data engineer.

Similarly, while data analysts can do no-code and low-code ML models, they will struggle with more complex workflows that involve ML on natural language text. They will also not have the skills to implement sophisticated data science algorithms such as for ranking or recommendations. Therefore, you will still need a data science team.

The technological framework

To define a reference high-level architecture, we will first define the fundamental principles from which we will derive the components and their relationships.

SQL as a standard

> Technology should be tailored to the current organizational culture. Components that offer a SQL interface should be prioritized, regardless of where they are in the data processing pipeline.

From EDW/ Data Lake to a structured data lake

> Information systems infrastructure and its data should be integrated to expand the possibilities of analytical processing on new and diverse data sources. This may involve merging a traditional data warehouse with a data lake to eliminate silos (we will dig on that in Chapter 8)

Schema-on-read first approach

> Due to the low cost of storage, your organization no longer needs to impose rigid rules on data structures before data is received. Moving away from a schema-on-write to a schema-on-read model allows for real-time access to data. Data can be kept in its raw form and then transformed into the schema that will be most useful. Additionally, the data platform can manage the

process of keeping these copies in sync (for example, using materialized views, CDC, etc.). Therefore, do not be afraid to keep multiple copies of the same data assets.

Combining these principles we can define a high-level architecture like the one shown in Figure 4-5. This informational architecture satisfies the three principles listed above and supports a few key data analysis patterns:

- The “traditional” Business Intelligence workloads, such as creating a dashboard or report.
- An ad hoc analytics interface that allows for the management of data pipelines through SQL (ELT).
- Enabling data science use cases with ML techniques.
- Real-time streaming of data into the data warehouse and processing of real-time events.

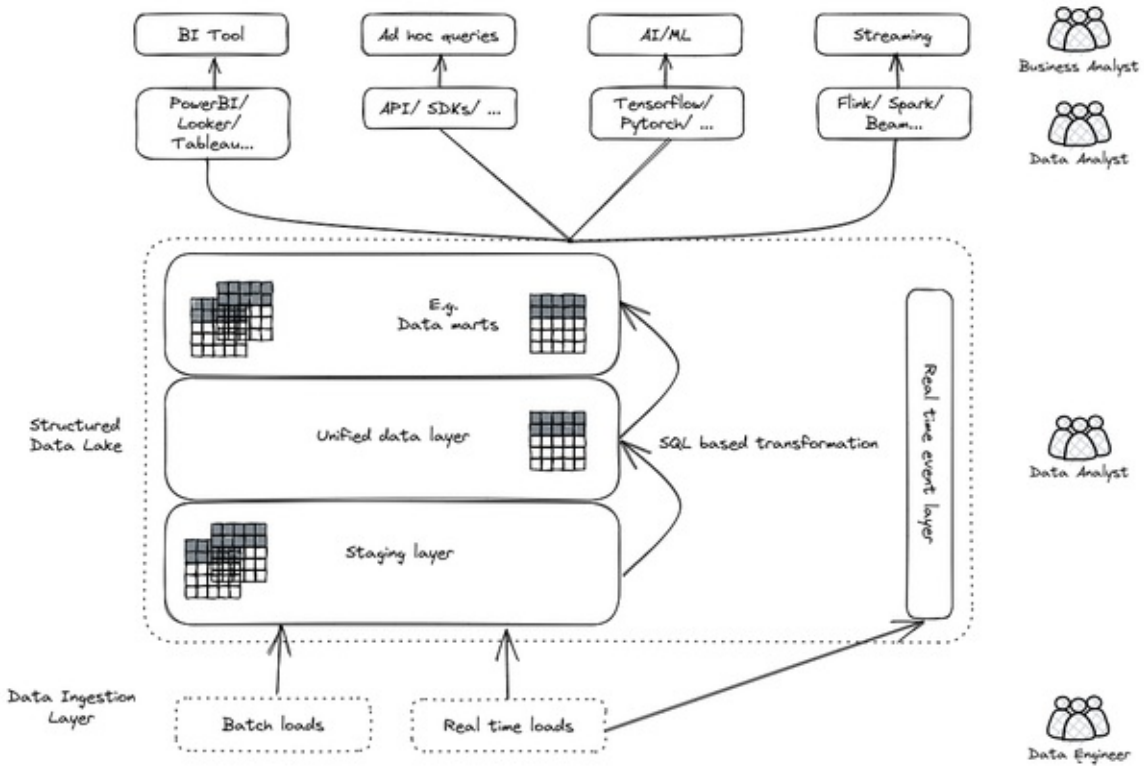


Figure 4-5. A high-level informational architecture for the DADO

The first two patterns are quite similar to the traditional SQL data warehousing

world, but the last two present innovations in the form of SQL abstractions for more advanced analytical patterns. For example, in the realm of ML, RedshiftML or BigQueryML allows data analysts to execute ML models on data stored in the data warehouse using standard SQL queries. SQL streaming extensions such as Dataflow and K-SQL enable aggregating data streams with unbounded, real-time sources such as Pub/Sub or Kafka. This technology enables a world of possibilities, even without the need to invest in new profiles and/or roles.

Data preparation and transformation are key considerations when choosing between ELT and ETL for a DADO. ELT should be used whenever possible, as it allows data to be transformed in the Structured Data Lake using SQL. This approach offers the same functionality as extensive data integration suites, but without the need to sacrifice data quality or operations monitoring. Products such as dbt bring a software engineering approach to data modeling and building data workflows. Dbt effectively allows building ELT workflows similar to ETL workflows built by code, but instead using SQL. This allows data analysts (who are not systems programmers) to carry out reliable and sophisticated data transformations.

In simpler terms, ELT is a better option than ETL for DADOs because it allows for more flexibility and control over data transformation. Additionally, dbt provides a software engineering approach to data modeling and building data workflows, which can help to improve the reliability and sophistication of data transformations.

Data-engineering driven organization (DEDO)

A DEDO is focused on data integration. There are companies (e.g. Plaid in fintech) whose business is to build integration pipelines for an entire industry. More commonly, this is a team that is part of a larger business. For example, an investment firm might have a team whose job is to discover, reformat, and ingest financial (e.g. stock market, companies' SEC filings, etc.) and alternative (e.g. credit card spend, e-receipts, retail footfall, etc.) data from a large number of vendors.

The vision

When your data transformation needs are complex, you need data engineers to play a central role in the company's data strategy and to organize your data architectures in three layers: *business data owners, data engineers,* and *data consumers.*

Data engineers are at the crossroads between data owners and data consumers, with clear responsibilities:

- Transporting data, enriching data whilst building integrations between analytical systems and operational systems (as in the real time use cases)
- Parsing and transforming messy data coming from business units and external suppliers into meaningful and clean data, with documented metadata
- Applying DataOps, that is, functional knowledge of the business plus software engineering methodologies applied to the data lifecycle. This includes monitoring and maintenance of data feeds.
- Deploying ML models and other data science artifacts analyzing or consuming data

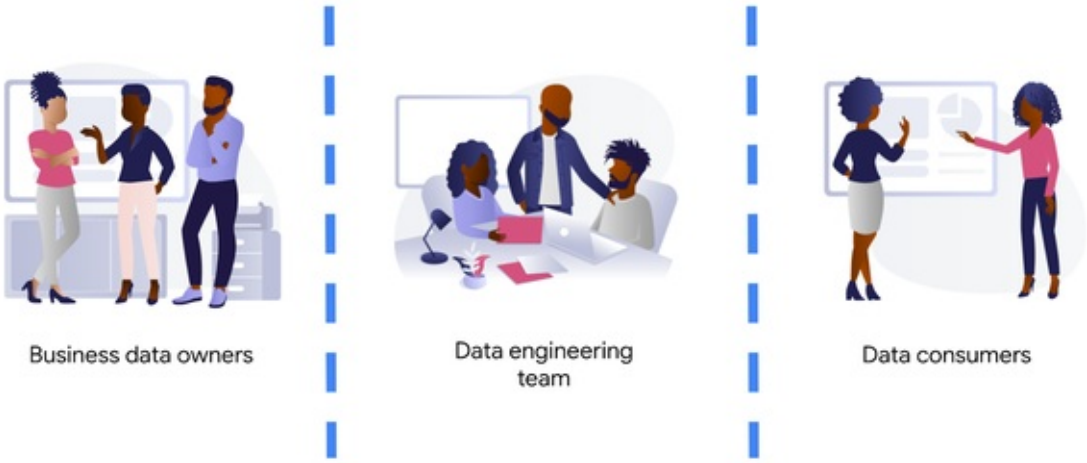


Figure 4-6. Engineering driven organization

Building complex data engineering pipelines is expensive but enables increased capabilities:

- **Enrichment**. Data engineers create repeatable processes and scale the number of sources. Data often does not live in isolation, it needs to be analyzed with other data with some context and then joined appropriately. Hence bringing it in location to query and process with other data you can get more value out of it. Data can be brought with external APIs, or the enrichment process may need to happen using data from external APIs before loading into the target.
- **Training datasets.** The quality of ML models is largely driven by the data used to train those models. By bringing in data from multiple sources and unifying them into datasets ready for training ML models, data engineers can increase the productivity of data science teams.
- **Unstructured data.** When the ML models need to use unstructured data (such as review text or images), there are special challenges. Such data is usually not following a strict schema as a traditional DW would use. In addition, it needs to be scrubbed of PII (such as telephone numbers in chat transcripts), unsafe-for-work content (especially images), and transformed (e.g. using embeddings).
- **Productionisation**. Data engineering work is also required in order to productionize and get value from ad-hoc data science work. Without data engineering, data scientists will be stuck carrying out experiments and producing applications that work for specific use cases but are rarely productionised or generalized.
- **Real-time analytics.** Applications such as anomaly detection and fraud prevention require immediate responses. In such use cases, data consumers need to process information as data arrives on the fly. They have to do so with low-latency. This type of real-time analytics requires transformation done outside of the target data warehouse.

All the above usually requires custom applications or state-of-the art tooling. In reality, there are very few organizations whose engineering capabilities excel to such a degree that they can truly be called engineering organizations. Many fall into what we call a blended organization.

Business data owners are part of cross-functional domain-oriented teams. These

teams know the business in detail, and are the source of data that feeds the data architecture. Sometimes these business units may also have some data-specific roles, such as data analysts, data engineers, or data scientists, to work as interfaces with the rest of the layers. For instance, these teams may designate a business data owner as the point of contact for a business unit in everything that is related to the data produced by the unit.

At the other end of the architecture, you find the *data consumers*. Again, this is cross-functional, but more focused on extracting insights from the different data available in the architecture. Here you typically find data science teams, data analysts, business intelligence teams, etc. These groups sometimes combine data from different business units, and produce artifacts (ML models, interactive dashboards, reports, and so on). For deployment, they require the help of the data engineering team so that data is consistent and trusted.

At the center of these crossroads is the *data engineering team*. Data engineers, as you have read before, are responsible for making sure that the data generated and needed by different business units gets ingested into the architecture. This job requires two disparate skills: functional knowledge and data engineering/ software development skills. This is often coined under the term DataOps (which evolved from DevOps methodologies developed within the past decades but applied to data engineering practices).

Data engineers have another responsibility too. They must help in the deployment of artifacts produced by the data consumers. Typically, the data consumers do not have the deep technical skills and knowledge to take the sole responsibility for deployment of their artifacts.This is also true for highly sophisticated data science teams. So data engineers must add other skills under their belt: ML and business intelligence platform knowledge. Let’s clarify this point, we don’t expect data engineers to become ML engineers. Data engineers need to understand ML to ensure that the data delivered to the first layer of a model (the input) is correct. They will also become key when delivering that first layer of data in the inference path, as here the data engineering skills around scale / HA etc really need to shine.

By taking the responsibility of parsing and transforming messy data from various business units, or for ingesting in real time, data engineers allow the data consumers to focus on creating value. Data science and other types of data

consumers are abstracted away from data encodings, large files, legacy systems, and complex message queue configurations for streaming. The benefits of concentrating that knowledge in a highly skilled data engineering team are clear, notwithstanding that other teams (business units and consumers) may also have their data engineers to work as interfaces with other teams. More recently, we've even seen squads created with members of the business units (data product owners), data engineers, data scientists, and other roles. This effectively creates complete teams with autonomy and full responsibility over a data stream, from the incoming data down to the data driven decisions that impact the business.

The personas

Data engineers are the core of such an organization. They develop and optimize all the processes needed to get access to the data and the solutions needed to perform related analysis and generate reports. This role requires a deep understanding of databases and programming languages, along with certain business skills to work across departments. Regardless of the size of the organization they are working in, data engineers need to have some standard skills to be successful. The most valuable knowledge includes:

- ***Data Warehousing and Data Lake Solutions*** – Cloudera Data Platform, Oracle Exadata, Amazon RedShift, Azure Synapse, Google BigQuery etc. where data engineers can handle huge volumes of data.
- ***Database Systems*** – Database systems like SQL and NoSQL. They should know how to work on and manipulate database management systems (DBMS) for information storage and retrieval.
- ***ETL Tools*** – We extensively talked about this kind of solution throughout the chapter. Common solutions are Informatica, AbInitio, Matillion.
- ***Programming languages***
    - *Python* – The most popular and useful language for statistical analysis and modeling, as well as ETL tasks
    - *Java* – Extensively used in data architecture frameworks
    - *Scala* – A Java extension that is interoperable with Java

- ***Apache Hadoop ecosystem*** – Apache Hadoop is an open-source framework. It is a collection of tools that support data integration. Hadoop allows storage and analysis of a huge chunk of information and is a critical element for data engineering teams to function smoothly (e.g. Spark, Beam, Hive, Ping, Oozie, etc.)
- ***Kafka*** – Kafka is an open-source processing software platform meant to handle real-time data feeds. Data engineers often use Kafka with Hadoop for real-time processing, monitoring, and reporting of data.
- ***Data structures and algorithms*** – Data structures and algorithms allow organizing and storing data for easy access and manipulation, making it an essential skill for data engineers.
- ***Automation and Scripting*** – Data engineers should be able to write scripts to automate repetitive tasks since they have to deal with such huge amounts of data (e.g. Bash, Powershell, Hashicorp Terraform, Ansible, etc.)
- ***Container technology*** - The de facto standard for moving data projects to production. Projects developed on a local machine can be "shipped" to a staging and production cluster (typically) with no issues. Data pipelines and ML models are reproducible and can run anywhere in the same fashion.
- ***Orchestration platforms*** - Because of the proliferation of solutions and tools that need to be integrated to perform data engineering tasks, orchestration tools like Apache AirFlow have become mandatory.

Hard skills by themselves are not enough. There is always a stringent need for more holistic knowledge involving analytical problem solving and industry knowledge. Usually this means certifications. Certifications measure a data engineer's knowledge and proficiency against vendor and/ or industry benchmarks confirming that the individual has the necessary expertise to contribute to your enterprise data strategies. Examples are AWS Certified data analytics, Cloudera Certified Professional (CCP) Data Engineer, Google Professional Data Engineer and Microsoft Certified: Azure Data Engineer Associate.

The technological framework

The number of skills required for the data engineering team is vast and diverse. Don't make their job harder by expecting the team to maintain the infrastructure where they run data pipelines. They should be focusing on how to cleanse, transform, enrich, and prepare the data rather than how much memory or how many cores their solution may require.

The reference architectures presented here are based on the following principles:

- Serverless no-ops technologies
- Streaming-enabled for low time-to-insight

Let's dig into these principles.

By using serverless technology you eliminate the maintenance burden from the data engineering team, and provide the necessary flexibility and scalability for executing complex and/or large jobs. For example, scalability is essential when planning for traffic spikes during black Friday for retailers. Using serverless solutions allows retailers to look into how they are performing during the day. They no longer need to worry about resources needed to process massive data generated during the day.

The team needs to have full control and write their own code for the data pipelines because of the type of pipelines that the team develops. This is true either for batch or streaming pipelines. In batch, the parsing requirements can be complex and no off the shelf solution works. In streaming, if the team wants to fully leverage the capabilities of the platform, they should implement all the complex business logic that is required, without artificially simplifying the complexity in exchange for some better latency. They can develop a pipeline that achieves a low latency with highly complex business logic. This again requires the team to start writing code from first principles.

However, the team that needs to write code should not imply that they need to rewrite any existing piece of code. For many input/output systems, you can probably reuse code from patterns, snippets, and similar examples. Moreover, a logical pipeline developed by a data engineering team does not necessarily need to map to a physical pipeline. Some parts of the logic can be easily reused by using technologies like Dataflow templates, and using those templates in orchestration with other custom developed pipelines. This brings the best of both

worlds (reuse and rewrite), while saving precious time that can be dedicated to higher impact code rather than common I/O tasks. The reference architecture presented has another important feature: the possibility to transform existing batch pipelines to streaming.

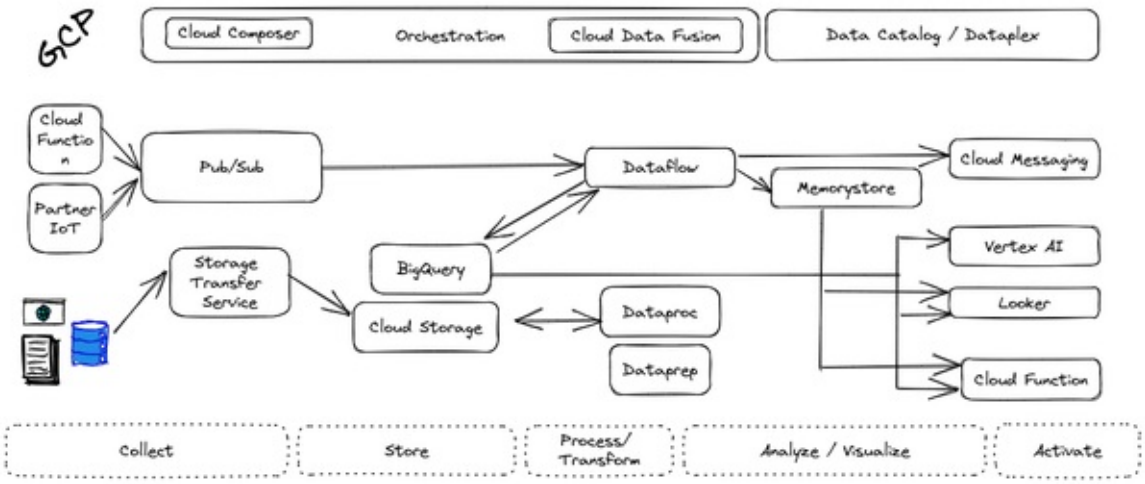


Figure 4-7. Example Data Engineering Driven Architecture on Google Cloud

The ingestion layer consists of Kinesis, EventHub or PubSub for real time and S3, Azure Data Lake or Cloud Storage for batch and does not require any preallocated infrastructure. All solutions can be used for a range of cases as it can automatically scale up with the input workload.

Once the data has been ingested, our proposed architecture follows the classical division in three stages: Extract, Transform, and Load (ETL). For some types of files, direct ingestion into Redshift, Synapse or BigQuery (following an ELT approach) is also possible.

In the transform layer, we primarily recommend Dataflow as the data process component. Dataflow uses Apache Beam as SDK. The main advantage of Apache Beam is the unified model for batch and streaming processing. As mentioned before, the same code can be adapted to run in batch or streaming by adapting input and output.

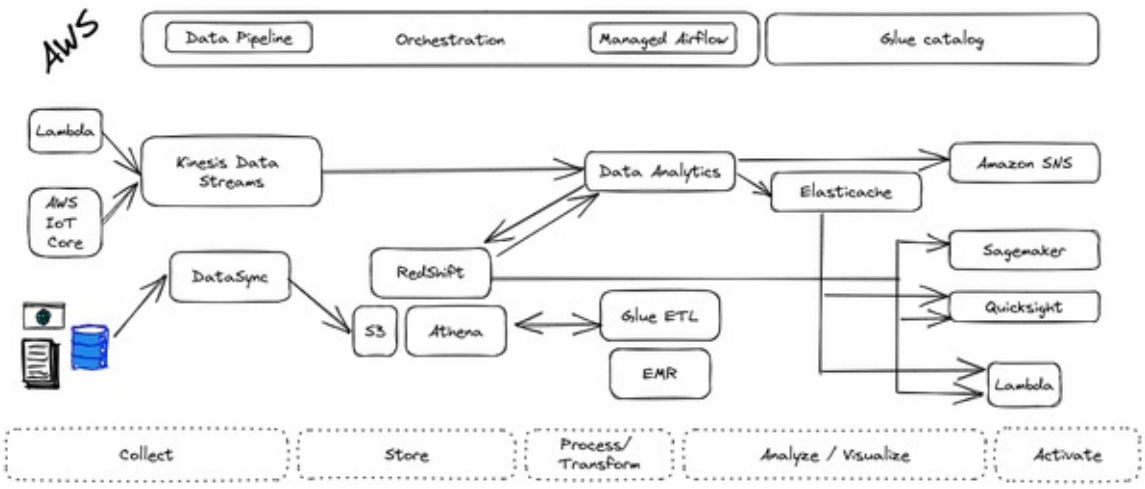


Figure 4-8. Example Data Engineering Driven Architecture on AWS

One of the alternatives to Dataflow in this architecture is using a Spark based solution, there are many choices out there such as Amazon EMR, Databricks, Azure HD Insights, Google Dataproc. However, these solutions are not serverless and running spark clusters idle is a major cost. Having said that, there are also serverless spark alternatives that are provided as part of AWS Sagemaker/Glue and as a service with GCP Serverless Spark. The main use case is for those teams that use large amounts of code in Spark or Hadoop. These Spark solutions enable a direct path to the cloud, without having to review all those pipelines.

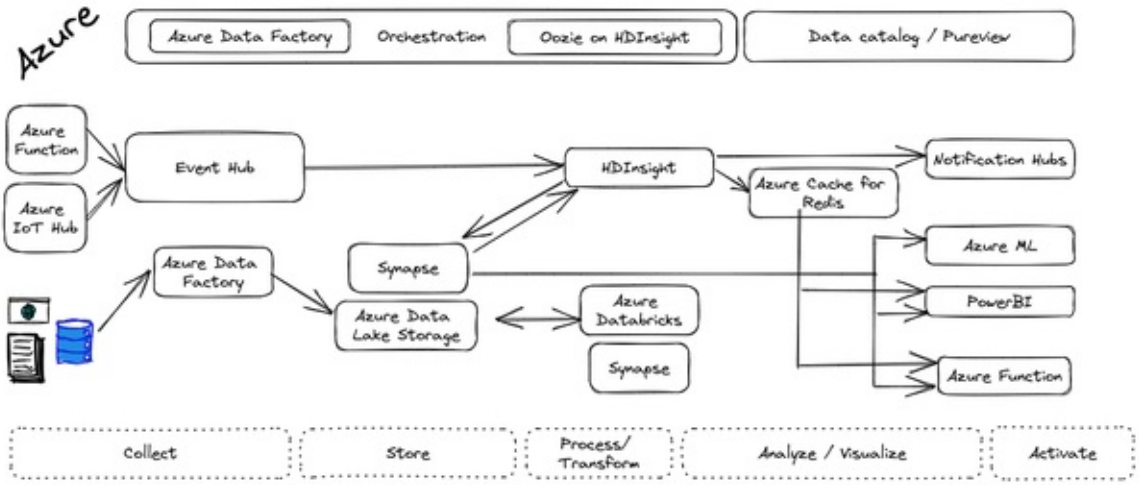


Figure 4-9. Example Data Engineering Driven Architecture on Azure

Finally, we also present the alternative of AWS Glue, Azure Data Factory or GCP Data Fusion, a codeless environment for creating data pipelines using a

drag-and-drop interface. Furthermore, traditional ETL tools such as Informatica, Ab Initio, Talend also run in serverless mode in Cloud with underlying Kubernetes clusters. Some of these tools either use Hadoop/Spark solutions or similar proprietary compute engines behind the scenes, so everything we have mentioned earlier applies also to the case of the ETL tools. If your team prefers to create data pipelines without having to write any code, graphical ETL tools are the right choice.

Data-science driven organization (DSDO)

A DSDO is an entity that maximizes the value from the data available to create a sustainable competitive advantage. In order to do so, a data science organization leans on automated algorithms that often (but not always!) employ ML. Rather than rely on one-off reports and analytics as a DADO would, a DSDO attempts to make decisions in an automated way. For example, a bank that is data analysis driven would have data analysts assess each commercial loan opportunity, build an investment case, and have an executive sign off on it. A data science driven fintech, on the other hand, would build a loan approval system that makes decisions on the majority of loans using some sort of automated algorithm.

The vision

A data-science driven organization is one that extracts the maximum value from its data and uses machine learning and analytics to gain a sustainable competitive advantage. When building such a data science team there are some principles that should be followed:

- **Adaptability**: A platform must be adaptable enough to accommodate all types of users. For instance, while some data scientists/analysts are more inclined to create their own models, others may prefer to use no-code solutions or conduct analyses in SQL. This also encompasses the availability of a variety of ML and data science tools such as TensorFlow, R, Pytorch, Beam, or Spark. The platform should also be open enough to function in multi-cloud and on-premises environments while supporting open source technology when possible to avoid lock-in effects. Finally, resources should never become a bottleneck, as the platform must be able to

scale quickly to meet an organization's needs.

- **Standardization**: Standardization increases a platform's efficiency by making it easier to share code and technical artifacts. This improves communication between teams and boosts their performance and creativity. Standardization also enables data science and ML teams to work in a modular fashion, which is essential for efficient development. Standardization can be achieved by using standard connectors to connect to source/target systems. This avoids "technical debt," which is common in ML and data science workflows.
- **Accountability**: Data science and machine learning use cases often involve sensitive topics such as fraud detection, medical imaging, or risk calculation. As a result, it is critical that a data science and machine learning platform helps to make these workflows as transparent, explainable, and secure as possible. Openness is linked to operational excellence. Collecting and monitoring metadata during all phases of the data science and machine learning workflows is essential to create a "paper trail" that allows you to ask questions such as:
  - Which data was used to train the model?
  - Which hyperparameters were used?
  - How is the model performing in production?
  - Did any form of data drift or model skew occur during the last period?

  In addition, a DSDO must have a thorough understanding of their models. While this is less of a problem for traditional statistical methods, ML models (such as deep neural networks) are much more opaque. A platform must provide simple tools for analyzing such models in order to use them with confidence. Finally, a mature data science platform must provide all the security measures to protect data and artifacts while managing resource usage on a granular level.
- **Business Impact**: According to McKinsey[1], many data science projects fail to progress beyond the pilot or proof-of-concept stage. As a result, it is more important to anticipate or measure the business impact of new

initiatives and choose ROI than to chase the latest cool solution. As a result, it is critical to identify when to buy, build, or customize ML models and connect them together in a single, integrated stack. For example, using an out-of-the-box solution by calling an API rather than building a model after months of development would help you achieve a higher ROI and demonstrate greater value.

- **Activation**: Ability to operationalize models by embedding analytics into the tools used by end users is key to achieve scaling in providing services to a broad set of users. The ability to send small batches of data to the service and have it return your predictions in the response allows developers with little data science expertise to use models. In addition, it is important to facilitate seamless deployment and monitoring of edge inferences and automated processes with flexible APIs. This allows you to distribute AI across your private and public cloud infrastructure, on-premises data centers, and edge devices.

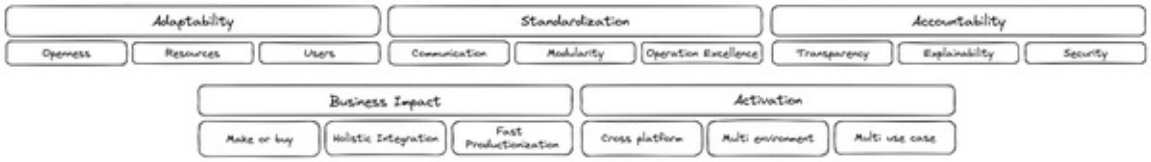


Figure 4-10. Principles of a DSDO

Building a DSDO comes along with several socio-technical challenges. Often an organization's infrastructure is not flexible enough to react to a fast changing technological landscape. A platform also needs to provide enough standardization to foster communication between teams and establish a technical 'lingua franca'. Doing so is key to allowing modularized workflows between teams and establishing operational excellence. In addition, it is often too opaque to securely monitor complex data science and ML workflows.

A DSDO should be built on a technical platform which is highly adaptable in terms of technological openness. Hence, it is critical to enable a wide set of personas and provide technological resources in a flexible and serverless manner. Whether to buy or build a solution is one of the key drivers of realizing ROI for the organization, and this will define the business impact any AI solution would make. At the same time, enabling a broad number of users allows activating more use cases. Finally, a platform needs to provide the tools and

resources to make data science and ML workflows open, explanatory, and secure in order to provide the maximum form of accountability.

The personas

DDSO teams are made up of a variety of people with different skills and experience. However, most teams include four core roles: data engineers, machine learning engineers, data scientists, and analysts. It is important to note that these roles are not always clearly defined and can overlap to some extent. An effective organizational structure for a DDSO team will allow for collaboration and the full utilization of all team members' skills.

- **Data engineers** who are responsible for developing data pipelines and ensuring that the data meets all quality standards. This includes cleaning, merging, and enriching data from multiple sources to turn it into information that can be used for downstream analytics.
- **ML engineers** who create and oversee complete ML models. While ML engineers are the rarest of the four personas, they become essential once an organization intends to run critical business workflows in production.
- **Data scientists** who are a bridge between data and machine learning engineers. Together with business stakeholders, they translate business needs into testable hypotheses, ensure that value is derived from machine learning workloads, and create reports to demonstrate the value of data.
- **Data analysts** who provide business insight and ensure that the data-driven solutions that the business is seeking are implemented. They answer ad hoc questions, and provide regular reports that analyze both historical and recent data.

There are various arguments for whether a company should establish centralized or decentralized data science teams. There are also hybrid models, such as a federated organization in which data scientists are embedded in a centralized organization. As a result, it is more important to focus on how to address these socio-technical challenges using the principles described before.

The technological framework

We strongly recommend standardizing on the ML Pipelines infrastructure of your public cloud provider (i.e. Vertex AI on Google Cloud; Sagemaker on AWS; Azure ML on Azure) rather than cobble together one-off training and deployment solutions. The reference architecture (See Figure 4-11) consists of an ML Pipeline to automate experimentation and training. The trained model is deployed in a pipeline where many of the training steps are repeated through containerization. Use a feature store when features will be too expensive to compute on demand, or have to be injected server-side. Deploy models to endpoints.

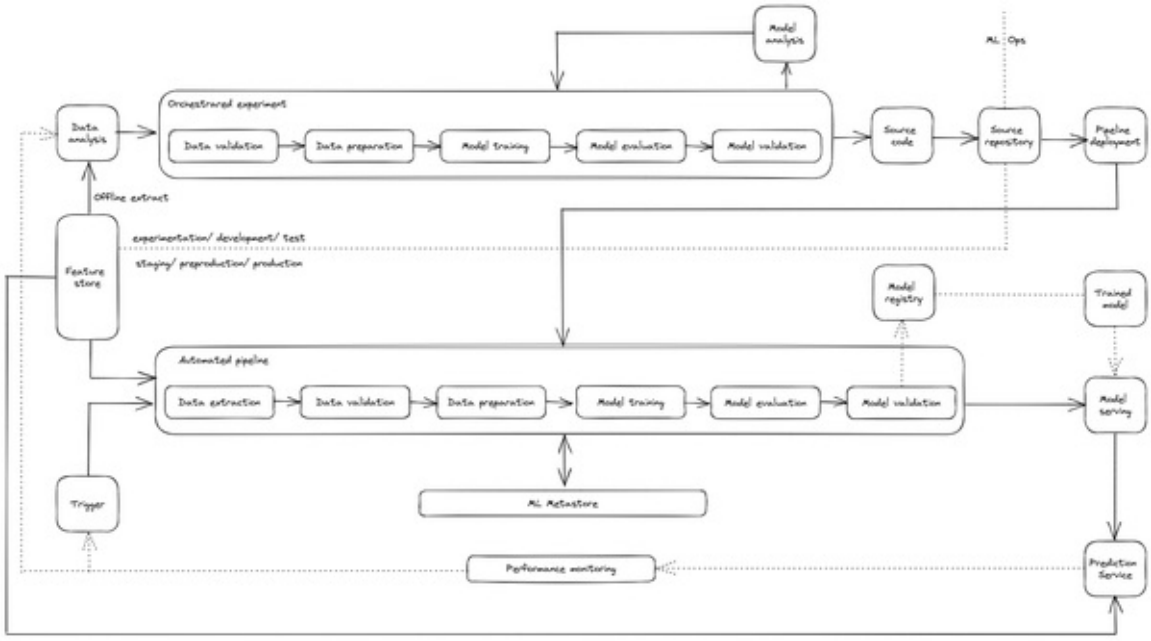


Figure 4-11. Make data science experimentation repeatable and deployments robust using the ML Pipeline product available in the public cloud.

Summary

In this chapter, you have seen different ways of designing your data team to ensure their success in the kind of organization you are in. The best approach is to find the right mix of skills and experience that will complement your existing team and help you achieve your business goals. The key takeaways are:

- Organizations can succeed in defining a data processing organization by employing different strategies based on their talent.
- The notion that there is a "*one size fits all*" or "*best*" approach in defining a

data processing organization simply does not work.

- Cloud technologies facilitate access to new ways of working.
- The potential surface area of any given persona has expanded.
- Data workers are now able to do more with the data they have, and they are able to do it more efficiently.
- You can build a data culture whether your organization consists mostly of data analysts, or data engineers, or data scientists.
- The path towards a data culture and required technologies for each organization type (DADO, DEDO or DSDO) are different.
- You have to decide which organizational classification is right for you, and then start setting the vision, the personas with related skills and the technological framework to support it.

The following chapter will offer you a general technological migration framework that you can apply to migrate from a legacy environment to a modern cloud architecture.

---

1 *https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/global-survey-the-state-of-ai-in-2020*

Chapter 5. A Migration Framework

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the fifth chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

In this chapter, let's examine the process of migration — all the things that you should do when making your journey to a new data platform. We will first present a possible framework you should follow when modernizing the data platform. Then, we will review how an organization can estimate the overall cost of the solution. We will discuss how to ensure that security and data governance are in place even while the migration is going on. Then we will dive into more technical topics like schema, data and pipeline migration (especially when dealing with data warehouses as sources) and we will learn the available options for dealing with regional capacity, networking and data transfer constraints.

A Four-Step Migration Framework

Data modernization transformation should be considered holistically, not as a pure IT or data science project but one where the technological aspect is just a subset of a larger organizational change. Looking at this from a bird's eye perspective, we can identify three main pillars:

- **Business**: It is critical to assess the overall organization's data maturity to identify where the gaps are and where the opportunities sit. It is extremely

important that, before even a single technology decision is made, the project is scoped to the various use cases that the company wants (or is able) to implement. These use cases will have to align to the medium to long term business objectives (typically over a time horizon of 2-3 years) identified by the leadership.

- **People**: Identify the persons or roles (some of these teams may not yet exist) who will potentially gain access to the data. Your goal with data modernization is to democratize data access. Therefore, these teams will need to become data literate and proficient in whatever tools (SQL, Python, dashboards) the modernization end-state technologies expect them to use. Unless these teams know how to read, analyze and use the data, the entire modernization project will be a failure. So, you will have to scope the end-state to the skill set of these users. If they are business users, make sure they can learn how to leverage tools to visualize and analyze the data. If they are data engineers or data scientists, they may need to become experts on the collection, processing, and automation steps of the data lifecycle.
- **Technology**: Elements like target data architecture, data modeling and design, data storage and operations, data security, data integration and operability, documents & contents, reference & master - data, data quality and data governance need to be defined, implemented and deployed in alignment with the overall business strategy and the capabilities of the teams expected to use them.

Starting from these assumptions, you can identify a general and standardized approach that can apply to the various situations you might encounter. This approach is, for the most part, independent of the size and the depth of the opportunity — you can follow it both when you have to modernize the data warehouse of a small company or when you have to develop a brand new data architecture of a multinational enterprise.

Let’s have a detailed look at this process. Imagine yourself as the CDO (Chief Data Officer) of a retail company. You have physical presence thanks to several points of sales geographically distributed in a specific area (let’s assume a couple of countries in EMEA) and you even have a web presence thanks to your new eCommerce application. Thanks to a recent and aggressive marketing campaign,

your company is garnering a lot of traction and the number of users is continuously increasing. The leadership team would love to have more real time insights about what is happening at the various point-of-sale locations and on the web. The intent is to provide customers with ad hoc offers based on their purchase history and their preferences. In addition, the leadership team would love to optimize the supply chain to reduce transportation costs.

They are asking you a lot of information and your team struggles to meet this demand because your current data architecture is not ready for that. You come back to your leadership team and say: "*Folks, we need to get rid of our legacy environment and we need to jump into the future*" and you start explaining how valuable it will be to modernize the data architecture. The CEO (Chief Executive Officer) and the other leaders are enthusiastic about your explanation and they ask you: "*So, what are your suggestions? What is the approach? How long does it take and how much does it cost?*". This is where the four-step migration framework, illustrated in Figure 5-1, comes into play.

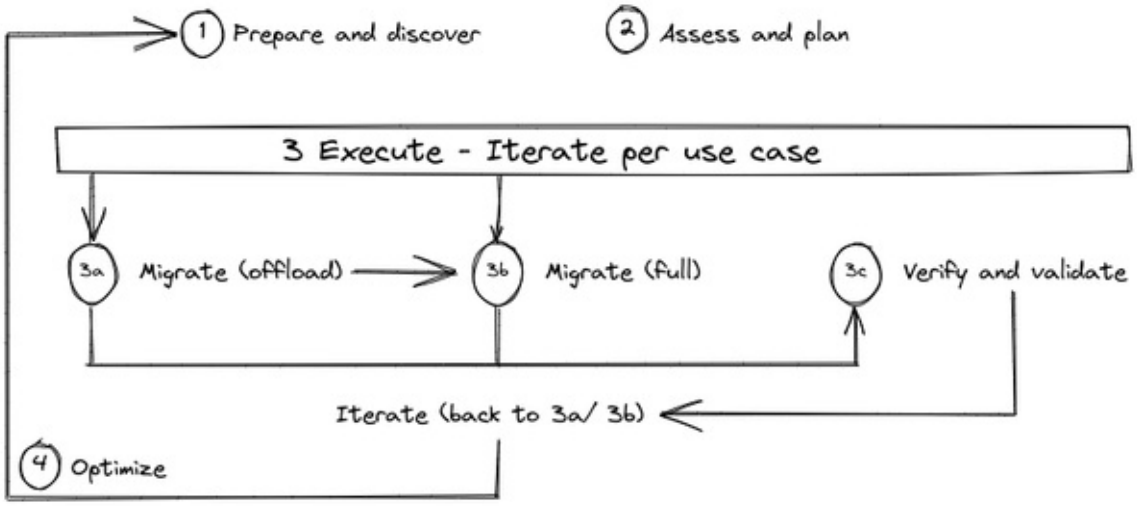


Figure 5-1. Four-step migration framework

One way that organizations can embrace the data modernization journey is to make it part of a cloud adoption project. Data modernization in such a situation can be achieved via a migration/ modernization of the legacy stack to the selected target cloud technology. To be effective in cloud migration, we recommend following a proven methodology that allows organizations to adhere to an organized, structured and measurable series of tasks. This approach is based on the four main steps shown in Figure 5-1:

1. *Prepare and discover*: all the stakeholders involved conduct a first analysis to identify the list of workloads that are available for the migration and collect the first feedback (e.g. inability to scale, process engine cannot be updated, dependencies with solution x,y,z, etc.)
2. *Assess and plan*: assess the information collected at previous stage, define the key measure of success and plan the migration of each component
3. *Execute*: iterate the following steps for each identified use case
    - *Decommission*: once it is identified that a use case is not needed anymore, it can be retired (that is, it will not be migrated)
    - *Migrate (offload)*: migrate only data, schema and downstream business applications (*the scripts, procedures, and business applications that are used to process, query, and visualize the data*)
    - *Migrate (full)*: migrate the use case fully end-to-end. This scenario is an extension of the offload migration step described before but adding the upstream data pipelines (*data sources and pipelines that are feeding the system*).
    - *Verify and validate*: test and validate the migration to double check the Return on Investment (ROI).
4. *Optimize*: Once the process has begun, it can be expanded and improved through continuous iterations as the data platform can evolve in accordance with business requirements. A first modernization step could be quite basic, with the goal of enhancing only the core business (e.g., being able to handle the appropriate data), and it can lead to a broader environment with synergies with the ecosystem (e.g., a retailer that wants to share real-time inventory data with its suppliers to improve the supply chain) or business innovation (e.g., monetizing data as a new business model) as needed.

Let’s go through each of these steps in turn.

Prepare and discover

Prepare and discover is the initial step where we focus on defining the perimeter

of the migration and then on collecting all the information related to the various workloads/ use cases that we have identified to be migrated. The ultimate goal of the journey is to transform current data architecture into something new that will allow the organization to get the most from their data. This encompasses the analysis of a wide range of inputs involving several stakeholders belonging to multiple areas of the enterprise (e.g. business, finance, IT, etc.).

The main activities that should be carried out by the identified stakeholders are to:

- Produce a list of all use cases/ workloads that are in the context of the migration
- Conduct an initial analysis of the expected benefits that they can get with the new system (e.g. query performance, amount of data they are able to handle, streaming capabilities etc.)
- Evaluate solutions available on the market and challenge them based on the business needs
- Perform an initial Total cost of ownership (TCO) analysis to really understand the value of the migration
- Understand the data maturity level within the company and identify what might be a possible path for the future (e.g. ad hoc training, hiring new talents in the market, etc.)

The best approach in collecting all these preliminary insights in a structured way is via questionnaires that should be delivered to all relevant people (e.g. application owners, data owners, selected final users, etc.) . This is a non exhaustive example of questions that can be carried out by the working team:

- What are the use cases?
- What is the value of the use cases?
- How many users are adopting this specific use case?
- What is the amount of data for the use case (e.g. newly generated data every day/ retention period/ etc.)

- What are the dependencies (if any)?
- What do the underneath data models look like?
- What do the data ingesting and transformation pipelines look like?
- What is the current total cost of the use case?
- What are the operational requirements (SLAs) for the use case?
- What are the legal or governmental compliance requirements for the use case?
- What are the downtime and latency sensitivities for accessing the underlying dataset?
- What is the required RPO (Recovery Point Objective) and RTO (Recovery Time Objective)?

Assess and plan

With the information coming from the previous step, it is now time to assess the collected data and plan all the activities that need to be done to accomplish the identified goals. From a general point of view you should follow the following four steps:

1. *Assessment of the current state*: In this phase we need to dig into the details of the current technology footprint in order to be able to identify the correct approaches for each application, workflow, or tool. This activity requires a lot of effort because several elements needs to be collected and analyzed, such as:
    - Server configurations
    - Logs (e.g. DWH, DB, ETL, etc.) to discover details about usage
    - Jobs activity - what is the current list of jobs have been scheduled/ planned
    - Data flow mapping
    - Volumetrics - size of the current data footprint (e.g. 100TB of data)

- Queries - how many queries are currently running/ have been scheduled in the system
- Clusters - how many infrastructure database cluster are present

Because this activity can become very tedious and error-prone as the size of the legacy footprint increases,you should seek to leverage tools (e.g. SnowConvert, Compilerworks, AWS Schema conversion tool, Azure Database Migration service, Datametica Raven, etc.) that automate the entire process from the data gathering up to the analysis and recommendation. General outputs of such tools are:

- Workload breakdown
- Dependencies mapping
- Complexity analysis
- Resource utilization
- Capacity analysis
- SLA analysis
- End to end data lineage
- Recommendations for data model denormalization, query reusability, index and cache tuning
- Suggested optimization for data model, workload and data de-duplication and ETL path optimization

2. *Workload categorization*: Leverage the information collected via the questionnaire used in the "*Prepare and discovery"* step together with the deep insights of the assessment phase. Then, categorize and select the approach for all the identified workloads into one of the following options:

    - *retire*: the workload will initially remain on-prem and will be eventually decommissioned
    - *retain*: the workload will remain on-prem due to technical constraints (e.g. it runs on dedicated hardware) or for business reasons. This may

be temporary until the workload can be refactored; or possibly moved to a co-location facility where data centers need to be closed and services can't be moved

- *rehost*: the workload will be migrated into the cloud environment leveraging its Infrastructure as a Service (IaaS) capabilities often known as a 'lift & shift'
- *replatform*: the workload (or part of it) will be partially changed to improve its performance or reduce its cost and then moved to the IaaS; often known as 'move & improve'. This is where you optimize your lift & shift experience, generally starting from a containerization approach
- *refactor*: the workload (or part of it) will be migrated to one or more of cloud fully managed Platform as a Service (PaaS) solutions (e.g. BigQuery, RedShift, Synapse);
- *replace*: the workload will be completely replaced by a 3rd party off-the-shelf or Software as a Service (SaaS) solution;
- *rebuild*: the workload is completely re-architected using cloud fully-managed solutions and re-implemented from scratch. This is where you rethink your application and you plan how to leverage the best of the cloud native services.

3. *Workload clusterization*: Cluster the workloads that will not be rehosted, replatformed, etc. into a series of groups based on their relative business value and the effort needed to migrate them. This will help to identify a prioritization you can follow during the migration
    - *Group 1*: high business value, low effort to migrate (Priority 0 - quick wins)
    - *Group 2*: high business value, high effort to migrate (Priority 1)
    - *Group 3*: low business value, low effort to migrate (Priority 2)
    - *Group 4*: low business value, high effort to migrate (Priority 3)

In Figure 5-2 you can see an example of clusterization where workloads have

been divided into groups based on the described prioritization criterias. In each group workloads can have different migration approaches.

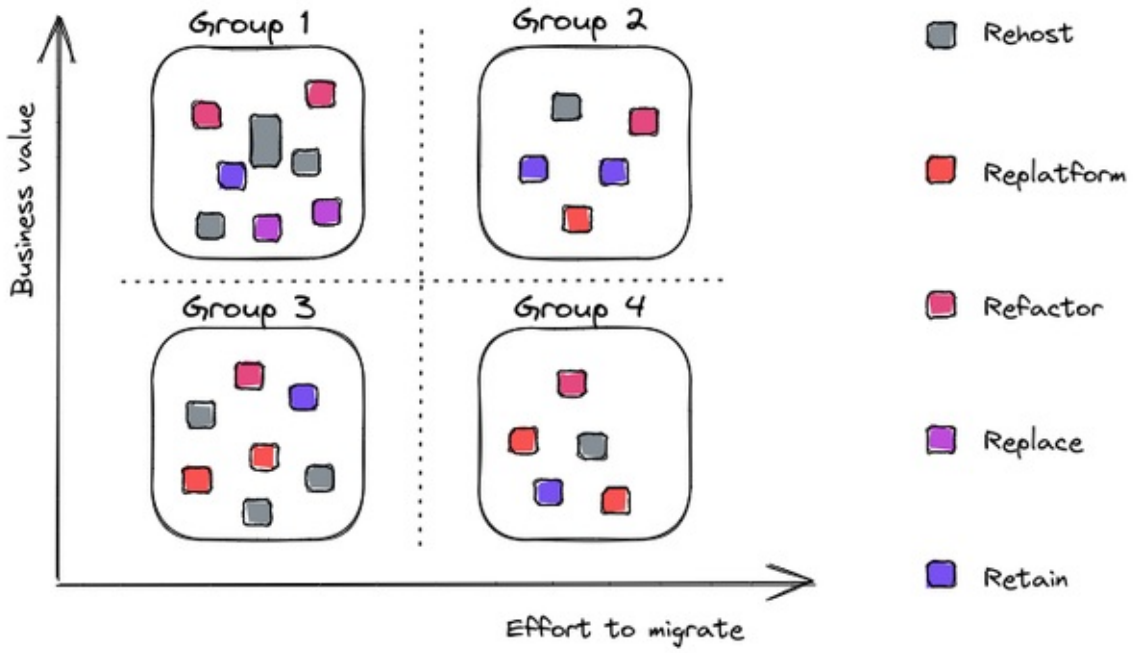


Figure 5-2. Workload categorization and clusterization

Throughout this process, we recommend that you adhere to the following practices:

- *Make the process measurable*: the team now has a list of use cases with related modernization strategy and priority. Make sure to identify how to measure the eventual execution of the modernization by defining some key performance indicators (KPIs) that will allow the stakeholders to evaluate results (and hopefully the success) of the step's execution and, eventually, fix the issues and enhance the overall process.
- *Start with Minimum Viable Product (MVP) or Proof of concept (POC)*: once you have the list of the workloads to be processed and related KPIs to be measured, don't try to bite off everything all at once. Break large jobs down into smaller pieces of work, and ensure that standard templates exist for any work that you are about to carry out. If not, do a Proof of Concept and use that as a template going forward. Be on the lookout for quick wins (Priority 0 workloads) that can be leveraged not only as an example for the other transformations but even as the "*crown jewel*" that can be showcased

within the organization (especially at upper level) to demonstrate the impact that such modernization might introduce.

- *Estimate the overall time needed to complete all the activities*. The last element of this step is, of course, the generation of an overall plan that could give a holistic overview of the entire project in terms of timing and effort. Usually organizations at this stage work closely with vendors or 3rd party consultancy companies/ system integrators to produce a modernization plan defining the time needed to transform every single workload and the related cost—expressed both in terms of FTEs (i.e. people) and in terms of technology running cost.
- *Over-communicate at milestones*. Ensure that stakeholders understand the plan, how long it will take, and what the key components are. Make sure to deliver value and instill confidence along the way with completed cloud projects that people in the organization can actually start to use. Try to identify milestones and send out an ad hoc communication recapping details about the work that have been done.

Execute

Now you have all the information in your hands! For each workload you know

- What will be migrated (the entirety of the workload or just a part)
- Where it will be migrated (IaaS, PaaS or SaaS)
- How you are going to migrate (rehost, replatform, rebuild, etc.)
- How you are going to measure success
- What templates you are going to follow
- How much time it’s going to take
- What milestones you are going to communicate at

You know what will be the first golden example you want to showcase and you have a complete understanding of the overall plan and related costs. You’ve been able to collect all the information your leadership team required! Very well done!

Now it is time to turn the plan described in the previous step into reality; now it is time to *execute*.

Let's now have a look at what are the main activities you have to focus on when executing a migration.

Landing Zone

First you have to build what is called *the landing zone*—the target environment where all the workloads will reside. This activity can assume different levels of complexity based on your current configuration but at the bare minimum you will need to:

- Define the multiple target project and related organization (e.g. Google Cloud Organization Hierarchy)
- Setup a new identity management solution or integrate with a legacy or 3rd party one (e.g. Azure Active Directory or Okta)
- Configure the authorization (e.g. AWS IAM) and auditing systems (e.g Azure Security Logging and Auditing)
- Define and setup the network topology and related configuration

Migrate

The best possible approach to migration—the next step in the process—is to iterate. *You can iterate over* a single or multiple use cases depending on the eventual dependencies and the resource you might make available for the migration (e.g. network bandwidth). It is generally advisable to split a migration into multiple phases, or iterations as reported in figure 5-3, unless the number of workloads to be migrated is very small. This will allow you to gain experience and confidence in how to proceed and deal with challenges and errors as you migrate a subset of the workloads at a time.

For each workload you may have to consider:

- *Schema and data migration*: depending on the use case, you may have to translate the data model before the data migration in order to make it compliant with the target environment (e.g. from a legacy data warehouse into a cloud data warehouse) or you may have to simply think about data

transfer because your use case is mainly leveraging unstructured data or a self-explaining data file like Apache Parquet. When talking about data migration we need to talk about network and bandwidth—we will get back to this topic at the end of this chapter.

- *Query transpile*: in some use cases you may have to translate queries available in your source systems to make them usable in the target system. There could be a situation where the target system does not support all the extensions available in the legacy environment so you may need to perform some refactoring to achieve the same results. You may want to leverage third party tools (Datametica Raven or CompilerWorks for example) to perform this kind of activity at scale, reducing the manual effort and speeding up the overall process.
- *Data pipelines migration*: you have to understand how to treat ETLs/ ELTs jobs that move data from one system to another performing in parallel some useful transformation. These are the core part of a data workload that prepares the data for analysis. We will see in the next chapters what are possible approaches in dealing with such migrations.
- *Business application migration*: once you have migrated the data you need to think about how to migrate the applications that enable users to interact with the data. We are referring mainly to applications for reporting but in some cases even custom developed solutions.
- *Performance tuning*: once migrated, you may find that the workload is not performing as expected so you have to investigate what is not working—maybe the target solution has not been properly configured; the data model you define does not allow you to leverage all the capabilities of the target platform; there is an issue in the transpilation process, etc.

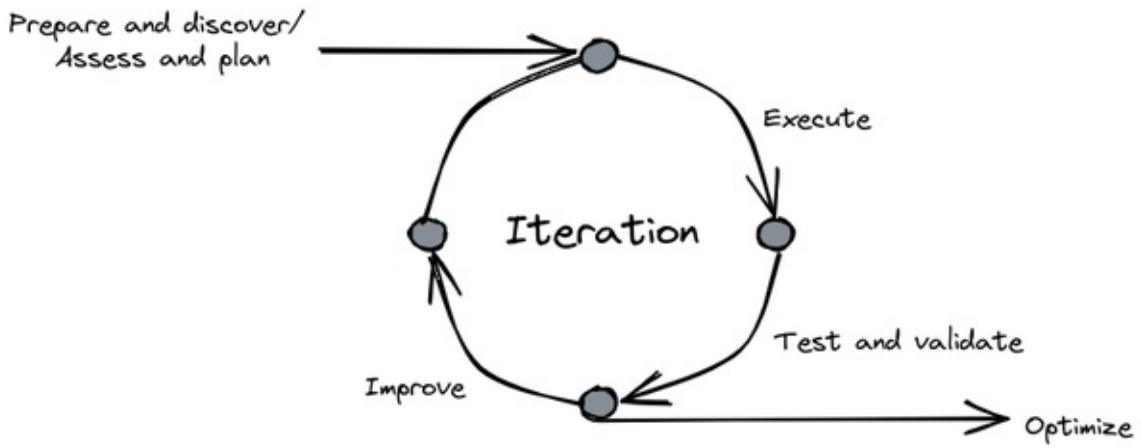


Figure 5-3. Execute iteration workflow

Validate

Once you have completed the first iteration you have to *verify* that everything is aligned with your identified KPIs (e.g. performance, running cost, time to get access to the data, etc.) and validate that all the results you get are compliant with your expectations (e.g. query results are the same that in the legacy environment). Once you are 100% sure that results are aligned with your needs, it is time to move to the second iteration, and then to the third until you have migrated everything.

> **NOTE**
>
> Iterations should be parallelized whenever possible to speed up the overall process.

It is always a good idea at the end of each iteration to note down eventual issues you may have, the time needed to complete all the activities and related lessons learnt in order to improve the process in the subsequent iterations.

Optimize

The last step of the framework is optimization. Here you won’t focus on the performance of every single migrated component. Instead you will consider the new system as a whole and identify potential new use cases to introduce in order to make it even more flexible and powerful. You should reflect on what you got

from the migration (e.g. unlimited scalability, enhanced security, increased visibility, etc.) and what you would potentially do as a next step (e.g. expand the boundaries of data collection to the edges, develop some better synergies with suppliers, start monetizing proper data, etc.). You can start from the information gathered at the "*Prepare and discover"* step, figure out where you are in your ideal journey, and think about additional next steps. It is a never-ending story because innovation, like business, never sleeps and it will help organizations become better and better at understanding and leveraging their data.

Now that you have a better understanding of how to approach a migration by leveraging the four-step migration framework, let us delve into how to estimate the overall cost of the solution.

Estimating the overall cost of the solution

You have just seen a valid and general migration framework that can help organizations define the set of activities that they need to carry out to modernize a data platform. The first question that the CTO, CEO, or CFO (Chief Financial Officer) may ask is: *what is the total cost we will have to budget for*? In this section we will review how organizations generally approach this challenge and how they structure their work in order to get quotes from vendors and third party suppliers. Always keep in mind that it is not just a matter of the cost of technology—there are always people and process costs that have to be taken into consideration.

Audit of the existing infrastructure

As you've already learned, everything starts with an evaluation of the existing environment. If you do not have a clear view of the current footprint, you will surely have challenges in correctly evaluating the pricing of your next modern data platform. Depending of the dimension of your datacenter, this activity can be generally carried out in 3 distinct ways:

- *Manually by the internal IT/ Infra team*: Many organizations maintain a Configuration Management DataBase (CMDB), which can be a file or a standard database that contains all pivotal information about hardware and software components used within the organization. It is a sort of snapshot of

what is currently running within the organization and the related underlying infrastructure, highlighting even the relationships between components. CMDBs can provide a better understanding of the running costs of all the applications and help in shutting down unnecessary or redundant resources.

- *Automatically by the internal IT/ Infra team*: The goal is exactly the same described in the previous point but with the aim to leverage a software that helps collect information in an automatic way (data related to the hardware, applications running on servers, relationship between systems etc.). These kind of tools (e.g. Stratozone, Cloudamize, Cloud Physics, etc.) usually generate suggestions related to the most common target cloud hyperscalers (e.g. AWS, Google Cloud and Azure) like size of machines and optimization options (e.g. how many hours per day a system should be up and running in order to carry out its task).
- *Leveraging a third party player:* A Global System Integrator (GSI), a consulting company, or the cloud vendor themselves can leverage experienced people and automation tools to perform all the activities to generate the CMDB and the detailed reports described in the first two options. This is what we recommend if your organization typically outsources IT projects to consulting companies.

The choice of which approach to take is strictly related to the level of the experience/ maturity of the company, to its human power (*do they have enough resources to perform such an activity that is time consuming?*) and to the financial capabilities. Sometimes the best option ends up being a mixture of all three.

Request for Information/ Proposal and Quotation

While this is the only migration you may do, and so you have to learn as you go along, consulting companies do this for a living and are usually far more efficient at handling migration projects. Verify, of course, that the team assigned to you does have the necessary experience. Some GSIs may even execute assessments and provide cost estimates as an investment if they see a future opportunity.

Identifying the best partner or vendor to work with during the modernization

journey can be a daunting task. There are a lot of variables to consider (e.g. the knowledge, the capabilities, the cost, the experience,...) and it may become incredibly complicated if not executed in a rigorous way. This is why organizations usually leverage three kinds of questionnaires to collect information from potential GSIs:

- *Request for Information (RFI)*: questionnaire used to collect detailed information about vendors/ possible partners solutions and services. It has an *educational* purpose.
- *Request for Proposal (RFP)*: questionnaire used to collect detailed information about how vendors/ partners will leverage their products and services to solve a specific organization problem (in this case the implementation of the modern data platform). It serves to *compare* results.
- *Request for Quotation (RFQ)*: questionnaire used to collect detailed information about pricing of different vendors/ possible partners based on specific requirements. It serves to quantify and standardize pricing to facilitate future comparisons.

Let’s dig into details about these different options.

Request for information - RFI

Organizations leverage RFI to seek details about vendors/ possible partners and what products or services they can offer for a specific project. This information may include:

- Answers to questions about specific needs (e.g. how the company would assess the current cost of the current infrastructure)
- The products or services the company offers
- Information on the company’s prices
- Experience working on similar projects
- A timeline for providing products or services

An RFI document should be easy to prepare and typically should include five sections:

1. *General request details* about the company, the owner of the document, and the deadline for the response
2. *The company background* that describes, in detail, the company's mission, its experience in the industry, and the future medium and long term goals the leadership want to achieve
3. *The high level goal* that the organization should achieve with the questionnaire: usually there should be an overview paragraph that describes the information the organization wants to seek for in order to enable eventual receiver to understand if they are interested or not (*e.g.the issuer of the RFI may want to know if the receiver has a proven knowledge of a specific legacy technology and experience in how to modernize it leveraging cloud solutions*)
4. *A project summary* that describes in detail the ultimate goal of the project (i.e. design and implement a modern data platform) with a related timeline and why the organization has that specific plan in mind.
5. *A response section* generally made by a series of questions that the various responders should complete following a standard template (e.g. *Could you provide some examples of customers who have used your solution in the past, the problems that it solved for them, the metrics that they were trying to improve, and the quantitative results that they achieved?)*

The benefits of leveraging such an approach are different for everybody, but the ultimate goal is to standardize the information you'll receive from each provider. The RFI also tells the providers that if they want to work with you they have to compete with each other for the best product and offering. The outcome of the RFI is usually leveraged as input to prepare a Request for proposal RFP.

Request for proposal - RFP

Once you've received the responses to the RFI and identified the possible vendors/ potential partners to work with, usually the organizations send out a Request for proposal, or RFP. This document describes the business requirements and project needs in detail. The companies your organization selected from the RFI pool work on a proposal to outline how they will solve the challenges and then bid to win the contract.

A good RFP should contain the following elements:

1. *An Introduction* that describes why the organization is issuing the RFP, describing the expectations of what needs to be achieved. The problem that needs to be addressed has to be clearly described and easy to understand (*we personally participated in so many RFP tenders that were incredibly vague and we spent a lot of time just in identifying what was the ultimate goal of that*). This section usually includes a clear definition of timelines (start of RFP, deadline for questions, deadline for response, deadline for the submission).
2. *A definition of the project and needs*, which should detail the goal of the project and what you're asking the vendor to accomplish (e.g. a full definition of a modern data platform or maybe just the assessment). It is important to define, as per the RFI, a standard in the way vendors/ potential partners should prepare their response: generally there are specific sections where detailed information should be collected:
    a. What are the activities to be carried out
    b. How activities will be carried out
    c. Where activities will be carried out (i.e. on premise/ in the cloud)
    d. When the activities will be carried out
3. *Selection criterias* where you should explain the main drivers of your request (e.g. ability to modernize, experiences with previous companies of the same industry, costs, etc.)
4. *A response section* generally made by a series of questions that the various responders should complete following a standard template *(e.g. describe your hybrid capabilities/ describe the utilities available to manage and monitor active data loads/ what options does your solution provide for data streaming?)*

RFPs are the most critical documents to evaluate because they contain detailed information about how vendors/ potential partners plan to help organizations solve their business problems. This questionnaire tends to be focussed only on the operational part of the delivery and not on the pricing considering–that's why

there is a dedicated request for quotation (RFQ) questionnaire–but it has become a common practice to mix the twos in one single document.

Request for quotation - RFQ

The last questionnaire that we'll cover is the Request for quotation, or RFQ. The RFQ, as we mentioned, can be combined with the RFP as necessary. The goal of this document is to have a clear understanding of the pricing that the vendors/ potential partners can provide to deliver services described in the RFP document. Usually it contains the following requests:

- Name of the product, item or service required
- Quantity of products or schedule of services
- Payment terms required
- Special considerations the company has for selecting a vendor/ potential partner

Once the organization has received all the responses from all the vendors/ potential partners they should have all the information to pick the best path forward. In some cases, especially when the problem to solve can be incredibly fuzzy (i.e. real time analysis that can have multiple spikes during even a single day) it is challenging even for the vendors/ potential partners to provide clear details about costs. This is why sometimes vendors will ask to work on a Minimum Viable Product or Proof of concept to gain a better understanding of how the solution works on a real use case scenario and facilitate the definition of the final pricing.

Proof of Concept (PoC)/ Minimum Viable Product (MVP)

As we've mentioned, there are some situations where the design and the development of a new data platform can be challenging: first of all because the majority of the organizations want to leverage this opportunity to have something more than a mere lift & shift and then because they want to add new features and capabilities that were not available in the old world. Parallel to that, organizations (and therefore vendors) may not have a complete understanding of the final behavior and, most importantly, on the final costs the platform will

incur.

To address this challenge, organizations, as first step after the analysis of the RFP response, usually ask selected vendors/ potential partners to implement an initial mockup of the final solution (or a real working solution but limited in functionality). This mockup provides stakeholders the opportunity to experience how the final solution will behave so they can understand if there is the need for some scoping changes. It also greatly facilitates cost estimation, although it should be noted that we are always talking about cost *estimation,* rather than concrete pricing—having clear and final defined pricing when adopting a cloud model is practically impossible, especially because you want to leverage elasticity, which is one of the main benefits of the cloud, and this elasticity can only be experienced in production.

There are two distinct ways to approach the idea of the mockup:

- *Proof of concept (PoC)*: with this approach we configure the development of a small portion of the overall solution to demonstrate how a final solution might work. The goal is not to touch every single feature to be part of the platform but instead to have an overview of the more important ones to verify feasibility, integrability, usability and potential weakness and things that may need to be rethought/ redesigned. This is a valid exercise even if you need to have a better understanding of the factors that could contribute to the final price (e.g. when dealing with streaming pipelines it is a best practice to develop a PoC of a streaming pipeline varying randomly the amount of data to be processed in order to see the scalability of the system in action and have better data to estimate a final production cost).
- *Minimum Viable Product (MVP):* as its name suggests, the goal of an MVP is to develop a product with a very well defined perimeter having all the features implemented and working like a real, full product that can be deployed in a production environment (e.g. a data mart implemented on a new data warehouse and connected to a new business intelligence tool to address a very specific use case). The main advantage of this technique is that final users can get access to something real very quickly and can provide feedback to improve the result. MVPs can help other people in the transition become a sort of advocate for the new solution. Having users really playing with the solutions will allow designer/ developers to collect

more fine grain information about the usage, which will allow the team to produce better estimations in terms of adoption and related costs.

The common approach is a 2-phase approach—initially the team will develop a general proof of concept with a broader perimeter but with a limited depth (e.g. an end to end data pipeline to collect data needed for example to train a ML algorithm for image classification) and then, based on the first results and first cost evaluation, the focus will move to the development of a MVP that can be seen as the first step toward the implementation of the full solution (i.e. the first iteration of our proposed framework).

Having now gained a better understanding of the common approach adopted by organizations to estimate the overall cost of a solution, let us now delve into how to address security and data governance.

Setting up security and data governance

Even if the ownership and control of the data moves to the business units, security and governance remain a centralized concern at most organizations. This is because there needs to be consistency in the way roles are defined, data is secured, and activities are logged. In the absence of such consistency, it is very difficult to be compliant with regulations such as "the right to be forgotten" whereby a customer can request that all records pertaining to them be removed.

In this section, we'll discuss what capabilities need to be present in such a centralized data governance framework. Then, we'll discuss the artifacts that the central team will need to maintain, and how it comes together over the data life cycle.

Framework

There are three risk factors that security and governance seek to address:

- **Unauthorized data access**. When storing data in a public cloud infrastructure, it is necessary to protect against unauthorized access to sensitive data, whether it is company-confidential information or personally identifiable information protected by law.

- **Regulatory compliance.** Laws such as General Data Protection Regulation (GDPR) and Legal Entity Identifier (LEI) limit the location, type, and methods of data analysis.
- **Visibility.** Knowing what kinds of data exist in the organization, who is currently using it, and how they are using it requires an up-to-date data and functional catalog.

Given these risk factors, it is necessary to set up a comprehensive data governance framework that addresses the full life of the data: data ingestion, cataloging, storage, retention, sharing, archiving, backup, recovery, loss prevention, and deletion. Such a framework needs to have the following capabilities:

- **Data lineage.** The organization needs to be able to identify data assets and record the transformations that have been applied to create each data asset.
- **Data classification.** We need to be able to profile and classify sensitive data so that we can determine what governance policies and procedures need to apply to each data asset or part thereof.
- **Data catalog.** We need to maintain a data catalog that contains structural metadata, lineage, and classification and permits search and discovery.
- **Data quality management**. There needs to be a process for documenting, monitoring, and reporting of data quality so that trustworthy data is made available for analysis.
- **Access management**. This will typically work with Cloud IAM to define roles, specify access rights, and manage access keys.
- **Auditing**. Organizations and authorized individuals from other organizations such as regulatory agencies need to be able to monitor, audit, and track activities at a granular level required by law or industry convention.
- **Data protection**. There needs to be the ability to encrypt data, mask it, or delete it permanently.

To operationalize data governance, you will need to put in place a framework

that allows these activities to take place. Cloud hyperscalers provide these capabilities — tools such as Dataplex on Google Cloud and Purview on Azure provide unified data governance solutions to manage data assets regardless of where the data resides (i.e single cloud, hybrid cloud, or multi-cloud). Collibra and Informatica are cloud-agnostic solutions that provide the ability to record lineage, do data classification, etc.

In our experience, any of these tools can work, but the hard work of data governance is not in the tool itself but in its operationalization. It is important to establish an operating model — the processes and procedures for data governance, as well as a council that holds the various business teams responsible for adhering to these processes. The council will also need to be responsible for developing taxonomies and ontologies so that there is consistency across the organization. Ideally, your organization participates in and is in line with industry standards bodies. The best organizations also have frequent and ongoing education and training sessions to ensure that data governance practices are adhered to.

Now that we have discussed what capabilities need to be present in a centralized data governance framework, let’s enumerate the artifacts that the central team will need to maintain.

Artifacts

In order to provide the above capabilities to the organization, a central data governance team needs to maintain the following artifacts:

- **Enterprise Dictionary.** This can range from a simple paper document to a tool that automates (and enforces) certain policies. The enterprise dictionary is a repository of the information types used by the organization. For example, the code associated with various medical procedures or the necessary information that has to be collected about any financial transaction is part of the enterprise dictionary. The central team could provide a validation service to ensure that these conditions are met. A simple example of an enterprise dictionary that many readers are familiar with is the address validation and standardization APIs provided by the US Postal Service. These are often used by businesses to ensure that any

address stored in any database within the organization is in a standard form.

- **Data Classes.** The various information types within the enterprise dictionary can be grouped into data classes, and policies related to that data class can be defined in a consistent way. For example, the data policy related to customer addresses might be that the zipcode is visible to one class of employees but that information that is more granular than that is visible only to customer support personnel actively working on a ticket for that customer.
- **Policy Book.** A policy book lists the data classes in use at the organization, how each data class is processed, how long data is retained, where it may be stored, how access to the data needs to be controlled, etc.
- **Use Case Policies**. Often, the policy surrounding a data class depends on the use case. As a simple example, a customer's address may be used by the shipping department fulfilling the customer order but not by the sales department. The use cases may be much more nuanced: a customer's address may be used for the purpose of determining the number of customers within driving distance of a specific store, but not for determining whether a specific customer is within driving distance of a specific store.
- **Data Catalog.** This is a tool to manage the structural metadata, lineage, data quality, etc. associated with data. The data catalog functions as an efficient search and discovery tool.

Beyond the data-related artifacts listed above, the central organization will also need to maintain a Single Sign-on capability to provide a unique authentication mechanism throughout the organization. Because many automated services and APIs are accessed through keys, and these keys should not be stored in plain-text, a key management service is often an additional responsibility of the central team.

As part of your modernization journey, it is important to also start these artifacts and have them in place so that as data is moved to the cloud, it becomes part of a robust data governance framework. Do not postpone data governance to after the cloud migration — organizations that do that tend to quickly lose control of their

data.

Let’s now look at how the framework capabilities and artifacts tie together over the life of data.

Governance over the life of the data

Data governance involves bringing together people, processes, and technology over the life of the data.

The life of the data lifecycle consists of the following stages:

1. **Data Creation.** This is the stage where you create/ capture the data. At this stage, you should ensure that metadata is also captured. For example, when capturing images, it is important to also record the time and location of the photographs. Similarly, when capturing a clickstream, it is important to note the user’s session id, the page they are on, the layout of the page (if it is personalized to the user), etc.
2. **Data Processing.** When you capture the data then you usually have to clean it, enrich it, and load it into a data warehouse. It is important to capture these steps as part of a data lineage. Along with the lineage, quality attributes of the data need to be recorded as well.
3. **Data Storage.** You generally store both data and metadata in a persistent store such as blob storage (S3, GCS, etc.), a database (Postgres, Aurora, Alloy DB, etc.), document DB (DynamoDB, Spanner, CosmosDB, etc.) or a data warehouse (BigQuery, Redshift, Snowflake, etc.). At this point, you need to determine the security requirements for rows and columns, as well as whether any fields need to be encrypted before being saved. This is the stage at which data protection is front-and-center in the mind of a data governance team.
4. **Data Catalog.** You need to enter the persisted data at all stages of the transformation into the enterprise data catalog and enable discovery APIs to search for it. It is essential to document the data and how it can be used.
5. **Data Archive.** You can age off older data from production environments. If so, remember to update the catalog. You must note if such archiving is

required by law. Ideally, you should automate archival methods in accordance with policies that apply to the entire data class.

6. **Data Destruction.** You can delete all the data that has passed the legal period that it is required to be retained. This too needs to be part of the enterprise's policy book.

You have to create data governance policies for each of these stages.

People will execute the stages above, and those people need to have certain access privileges to carry out the stages of the data lifecycle. The same person could have different responsibilities and concerns at different parts of the data lifecycle, so it is helpful to think in terms of "hats" rather than roles:

- *Legal*, to ensure that data usage conforms to contractual requirements and government/ industry regulations.
- *Data steward*, the owner of the data who sets the policy for a specific item of data.
- *Data governor*, sets policies for data classes and identifies which class a particular item of data belongs to
- Privacy tsar, who ensures that use cases do not leak personally identifiable information
- Data user, typically a data analyst or data scientist, who is using the data to make business decisions

Now that we have seen a possible migration and security and governance framework, let's take a deep dive into how to start executing the migration.

Schema, pipeline and data migration

You persuaded your leadership team to proceed with the data migration journey (congratulations!!!), you engaged with a system integrator that helped you carry on all the information gathering and plan definition, and now you've decided that the first workload to tackle as an MVP is a use case currently running in your legacy data warehouse. You evaluated different options but in the end you

choose to go for a managed solution in the cloud in order to showcase immediate value via:

- *Improved performance and scalability*: queries should be generally faster especially when the size of the data set is increasing
- *Real time enablement*: streaming data ingestion should be easily setup and configured and data immediately available for analysis (e.g. real time analysis of customer behavior in the eCommerce website for products recommendation)
- *Reduced operational overhead*: people who had to handle 24x7 operational support for issues like resource monitoring and scaling, backup, and space allocation can be focused on other activities since all these activities will be automatically handled by the underlying target platform
- *Accelerated time to value*: immediate access to the new features of the product without downtime or upgrade planning

You started the execution phase and based on your previous analysis you figured out that in order to fully leverage the power of the target system before proceeding with the data migration it would be better to adapt the schema and in parallel to enhance the data pipelines. Let's take a closer look at the elements and approaches you should carefully consider when planning a data migration with a focus on the patterns you can leverage for schema and pipeline migrations and challenges you have to face when transferring the data.

Schema migration

When starting to move your legacy application into the new target system, you might need to evolve your schema to leverage all the features made available by the target system. It is a best practice to first migrate the model as-is into the target system connecting the upstream (*data sources and pipelines that are feeding the system*) and downstream (*the scripts, procedures, and business applications that are used to process, query, and visualize the data*) processes and then leverage the processing engine of the target environment to perform all the changes. This approach will help you to ensure that your solution works in the new environment, minimizing the risk of downtime and allowing you to

make changes in a second phase.

You can usually apply the *façade pattern* here—a design method to expose to the downstream processes a set of views that mask the underlying tables to hide the complexity of the eventual needed changes. The views can then describe a new schema that helps in leveraging ad-hoc target system features, without disrupting the upstream and downstream processes that are then "*protected*" by this abstraction layer. In case this kind of approach cannot be followed, the data has to be translated and converted before being ingested into the new system. These activities are generally performed by data transformation pipelines that are under the perimeter of the migration.

Pipeline migration

There are two different strategies you can follow when migrating from a legacy system to the cloud:

- *You are offloading the workload*: in this case you retain the upstream data pipelines that are feeding your source system and you put an incremental copy of the data into the target system. Finally you update your downstream processes in order to read from the target system. Then you can continue the offload with the next workload until you reach the end. Once completed you can start fully migrating the data pipeline.
- *You are fully migrating the workload*: in this case you have to migrate everything in the new system (all together with the data pipeline) and then you deprecate the corresponding legacy tables.

Data that feeds the workload needs to be migrated. It can come from various data sources and it could require particular transformations or joins in order to make it usable. In general, there are four different data pipeline patterns:

- *ETL - Extract Transformation and Load*: all the transformation activities, along with the data collection and data ingestion will be carried out by an ad hoc system that comes with a proper infrastructure and a proper programming language (the tool can make interfaces programmable with standard programming languages available anyway).
- *ELT - Extract Load and Transform*: similar to ETL but with the caveat that

all the transformation will be performed by the process engine where the data will be ingested (*as we have seen in the previous chapters this is the preferred approach when dealing with modern cloud solutions*).

- *EL - Extract and Load*: this is the simplest case where the data is already prepared and it does not require any further transformations
- *CDC - Change Data Capture*: this is a pattern used to track data changes in the source system and reflect them in the target one. It usually works all together with an ETL solution because it stores the original record before making any changes to the downstream process.

As you saw in the previous section, you could identify different approaches for different workloads migration. The same methodology can be applied to the data pipeline solution. Let's have a look at the available options:

- *retire*: the data pipeline solution is not used anymore because it is referred to an old use case or because it has been superseded by a new one
- *retain*: the data pipeline solution remains in the legacy system because it can potentially be retired very soon. so it is not financially viable to embarque on a migration project. It may also be the case that there are some regulation requirements that inhibits data movement outside the corporate boundaries
- *rehost*: the data pipeline solution is lifted and shifted into the cloud environment leveraging the IaaS paradigm. In this scenario you are not introducing any big modification except at the connectivity level, where an ad-hoc networking configuration (usually a VPN) might need to be set up in order to enable communication between the cloud environment and the on-premises world. If the upstream processes are outside the corporate perimeter (e.g. 3rd party providers, other cloud environments, etc.), a VPN might not be needed since the communication can be established in a secure way leveraging other technologies, like authenticated REST APIs for example. Before proceeding it is necessary to validate with the cloud vendor if there is any technology limitation in the underlying system that prevents the correct execution of the solution and double check eventual license limitations.

- *replatform*: in this scenario part of the data pipeline solution is transformed before the migration in order to benefit from features of the cloud, such as PaaS database or containerization technology. Considerations on the connectivity side highlighted in the *rehost* section are still valid.
- *refactor*: the pipeline solution will be migrated to one or more of cloud fully managed PaaS solutions (e.g. Amazon EMR, Azure HD Insight, Google Cloud Dataproc). When dealing with this approach it is a best practice to re-adopt the same iterative approach you are adopting for the entire migration:
    - Prepare and discover the jobs and potentially organize them by complexity
    - Plan and assess the possible MVP to be migrated
    - Execute the migration and evaluate the result against your defined KPIs
    - Iterate with all the other jobs until the end

    Considerations on the connectivity side highlighted in the *rehost* section are still valid.
- *replace*: the pipeline solution will be completely replaced by a 3rd party off-the-shelf or SaaS (Software as a Service) solution (e.g. Fivetran, Xplenty, Informatica etc.). Considerations on the connectivity side highlighted in the *rehost* section are still valid.
- *rebuild*: the pipeline solution is completely re-architected using cloud fully-managed solutions (e.g. AWS Glue, Azure DataFactory, Google Cloud DataFusion). Considerations on the connectivity side highlighted in the *rehost* section are still valid.

During the migration phase, especially in the integration with the target system, you might find that your data pipeline solution is not fully compatible with the identified target cloud solution right out of the box. You could need a connector, usually called *sink*, that enables the communication between the data pipeline solution (e.g. the ETL system) and the target environment. If the sink for that solution does not exist, you may be able to generate a file as output of the

process and then ingest the data in the following step. This approach will introduce extra complexity to the process but it is a viable temporary solution in case of emergency (while waiting for an ad-hoc connector from the vendor).

Data migration

Now that you have your new schema and pipelines ready it seems you are ready to start migrating all your data. You should focus on thinking about how to deal with data transfer. You may want to migrate all your on-premise data into the cloud, even the one in your old tapes full of dust (maybe one day someone will require that data). You may face the reality that a single FTP connection over the weekend will not be enough to accomplish your task. Unfortunately it is not as easy as it seems because there are several things that you need to consider before starting with the actual migration.

Data transfer is always about planning. First of all you need to identify the stakeholders you need to involve:

- *Technical owners*: people who can provide access to the resources you need to perform the migration (e.g. storage, IT, networking, etc.)
- *Approvers:* people who can provide you with all the approvals you need to get access to the data and to start the migration (E.g. data owners, legal advisors, security admin, etc.)
- *Delivery*: the migration team. They can be people internal to the organization if available or people belonging to 3rd party system integrators.

Then you need to collect as much information as you can in order to have a complete understanding of what you have to do, in which order (e.g. maybe your migration team needs to be allowlisted to a specific network storage area that contains the data you want to migrate), and the possible blockers that you might encounter. Here is a sample of of questions (not exhaustive) that you should be able to answer before proceeding:

- What are the datasets you need to move?
- Where does the underlying data sit within the organization?

- What are the datasets you are allowed to move?
- Is there any specific regulator requirement you have to respect?
- Where is the data going to land (e.g. object store vs data warehouse storage)?
- What is the destination region? (e.g. EMEA, UK/IE, US, etc.)
- Do you need to perform any transformation before the transfer?
- What are the data access policies you want to apply?
- Is it a one-off transfer or do you need to move data regularly?
- What are the resources available for the data transfer?
- What is the allocated budget?
- Do you have adequate bandwidth to accomplish the transfer and for an adequate period?
- Do you need to leverage offline solutions? (e.g. Amazon Snowball, Azure DataBox, Google Transfer Appliance)
- …

Finally you need to define your migration strategy, which usually (use case permitting), will be:

1. Upstream processes feed current legacy data solutions.
2. Downstream processes read from the legacy environment.
3. Historical data can be migrated in bulk into the target environment.
4. Downstream processes can be connected to the target environment in order to perform validation tests.
5. Data can be kept in sync between legacy and target environment, leveraging CDC pipelines until the old environment is fully dismissed.
6. Downstream processes become fully operational, leveraging the target environment.

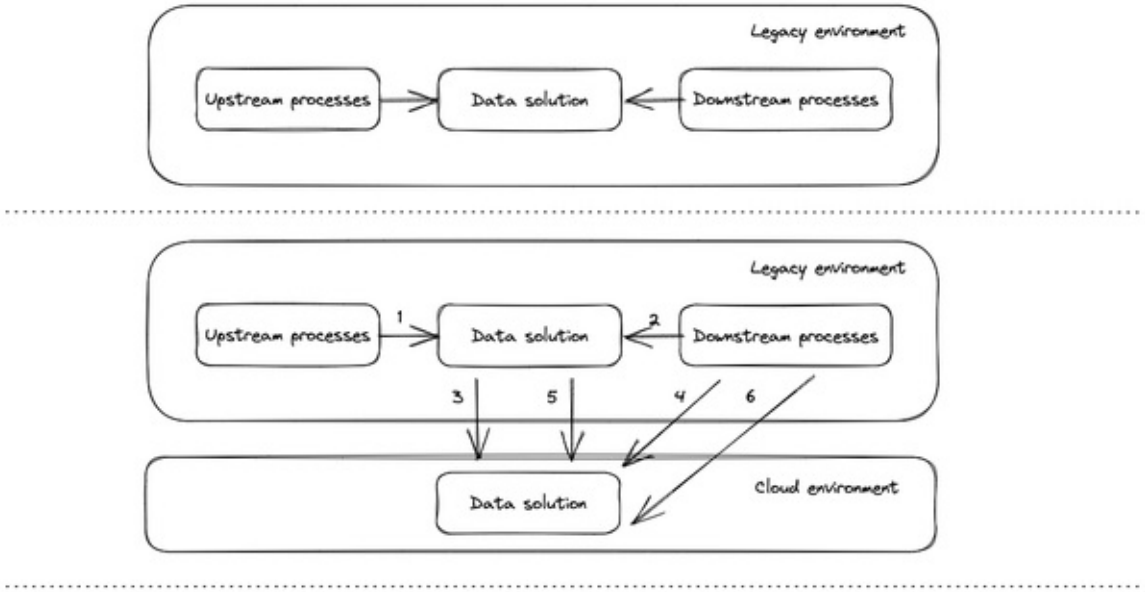


Figure 5-4. Data migration strategy

Now that we have a better understanding of a potential data migration strategy, let's explore the key elements that can affect the reliability and performance of the migration: capacity and network.

Regional capacity and network to the cloud

When dealing with cloud data migration, typically there are two elements that need to be considered carefully:

- Regional capacity
- Networking connectivity

A cloud environment is not infinitely scalable. The reality is that the hardware needs to be purchased, prepared, and configured by the cloud providers and even if they are super-precise in anticipating the needs of your organization and have a lot of data to feed their forecasting models, there is always space for uncertainty and situations beyond their control. Let's think about what happened in 2020 during the COVID pandemic: there was a lack of raw material that slowed down not only the production of the chips needed to craft CPUs and GPUs but even the entire supply chain causing huge delays to every single company in the world. This is why it is pivotal that once you have identified the target architecture and the resources that are needed to handle the data platform, you should also file a capacity regional plan with your selected hyperscaler to be

sure the data platform will have all the hardware needed to meet the usage and the future growth of the platform. It is a fairly easy exercise (especially when you have conducted a very good analysis of the target platform, perhaps via the development of a PoC/ MvP) where you need to know: what are the volume of the data to be migrated and that will be generated once in the cloud, the amount of compute you will need to crunch and elaborate the data, and the number of interactions you will have with other systems. All these components will serve as an input to your hyperscalers to allow them to be sure that the underlying infrastructure will be ready from day 0 to serve all your workloads. In case of stockouts (mainly due to the uncertainty generated by unforeseen events like what occurred in 2020) usually the option is to chose the same service but in another region (when there are not any compliance/ technology implications) or leverage other compute type services (e.g. a IaaS vs PaaS).

Next to the regional capacity of cloud services, another key element is of course the ability to connect to such services. The network, even if it is considered a commodity nowadays, plays a vital role in every cloud infrastructure: if the network does not work or it is not accessible, there is no way to make the system work and the organizations can remain completely disconnected from their business (this is true even when leveraging an on premise environment but in this case we are referring more to a local network instead of an Internet one). When designing the cloud platform some of the first questions you need to think about are: how will my organization be connected to the cloud? Which partner am I going to leverage to set up the connectivity? Am I leveraging standard Internet connection (potentially with a VPN on top of it) or do I want to pay for an extra dedicated connection that will ensure better reliability? All these topics are generally discussed in the RFI/ RFP questionnaires, but they should also be part of one of the very first workshops you have with the vendor/ partner that you've selected to design and implement the platform. While it is outside the scope of this book to discuss the details of connectivity options (e.g. Azure, AWS, Google Cloud), we want to highlight the main three ways to get connected to the cloud to give you a better understanding of what's possible (and these options apply to every cloud hyperscalers):

- *Public Internet connection*: leveraging public Internet networks. In this case organizations usually leverage a Virtual Private Network (VPN) on top of the public Internet protocol in order to protect their data and to guarantee an

adequate level of reliability. Performance is strictly related to the ability of the organization to be close to the nearest point of presence of the selected cloud hyperscalers.

- *Partner interconnect*: this is one the typical connections organizations may leverage for their production workloads, especially when they need to have guaranteed performance with high throughput. This connection is between the organization and the selected partner that then takes care of the connection with the selected hyperscalers. Leveraging the pervasivity of the telco providers, organizations can set up high performance connections with an affordable price.
- *Direct interconnect:* this is the best connection possible where the organization connects directly (and physically) with the network of the cloud provider (this can be possible when both of the parties have routers in the same physical location). Reliability, throughput and general performance are the best and pricing can be discussed directly with the selected hypervisors.

Choosing the right connection is not trivial, because there are several elements involved in the decision: typically during a PoC/ MvP phase the Public Internet connection option is the most valid because it is faster to set up, while Partner interconnect is the most common, especially when the organizations wants to leverage a multi-cloud approach.

Transfer options

Based on the dimension of the data that you want to migrate and the technical limitations you may have to cope with, there could be various transfer options available that have to be evaluated. The key pieces you need to consider here are:

- *Costs*: consider the following potential costs involved in the transfer of data:
    - *Networking*: you may need to enhance your connectivity before proceeding with the data transfer. Maybe you do not have adequate bandwidth to support the migration and you need to negotiate with your vendor to add additional lines.

- *Cloud provider*: uploading data to a cloud provider is always free but if you are exporting data not only from your on-premises environment but also from another hyperscaler you might be charged an *egress-cost* (which usually has a cost per each GB exported) and potentially a read cost as well.
- *Products*: You may need to purchase or rent storage appliances to speed up the data transfer
- *People*: the team who will carry out the migration.

- *Time*: it's important to know the amount of data you have to transfer and the bandwidth you have at your disposal. Once you know that you will be able to identify the time needed to transfer the data. For example, if you have to transfer 200 TB of data and you have only 10 Gbps bandwidth available you will need approximately 2 days to complete your transfer[1]

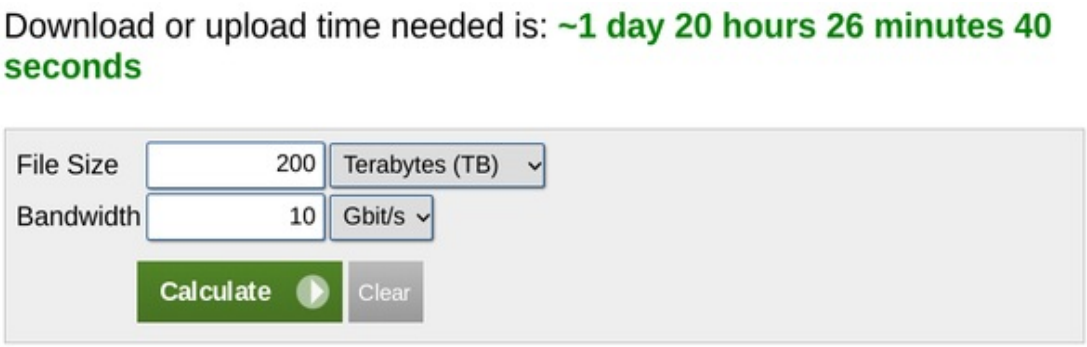


Figure 5-5. Data transfer time estimation

Please consider that we made this estimate considering a bandwidth fully available for your data transfer: this might be impossible because you have other activities to perform in parallel, so do not forget to identify the time of the day when you are allowed to carry out the migration, for example:

- weekdays from 9:00 PM to 5:00 AM when the bandwidth is not used by other activities
- weekend: all day

If during your analysis you discover that you need more bandwidth you might need to work with your Internet Service Provider (ISP) to request an

increase. It is the right time to work with your cloud vendor to implement direct connection. This prevents your data from going on the public internet and can provide a more consistent throughput for large data transfers (e.g. AWS Direct Connect, Azure ExpressRoute, Google Cloud Direct Interconnect).

- *Offline versus online transfer*: in some occasions an online transfer is infeasible so you should select an offline process leveraging a storage hardware. Cloud vendors offer this kind of service (*e.g. Amazon Snowball data transfer, Azure DataBox, Google Transfer Appliance*), which is particularly useful for data transfers from hundreds of terabytes up to petabyte scale. The process is pretty straightforward: you order a physical appliance from your cloud vendor (e.g. AWS Snowball, Azure Databox or Google Cloud Transfer Appliance) that will have to be connected to your network. Then you will copy the data, which will be encrypted by default and then you will request for a shipment to the closest vendor facility available. Once delivered, the data will be copied to the appropriate service in the cloud (*e.g. AWS S3, Azure Blob storage, Google Cloud Storage*) and will be ready to be used. The table in Figure 5-6 will help you make a decision when leveraging the offline service.

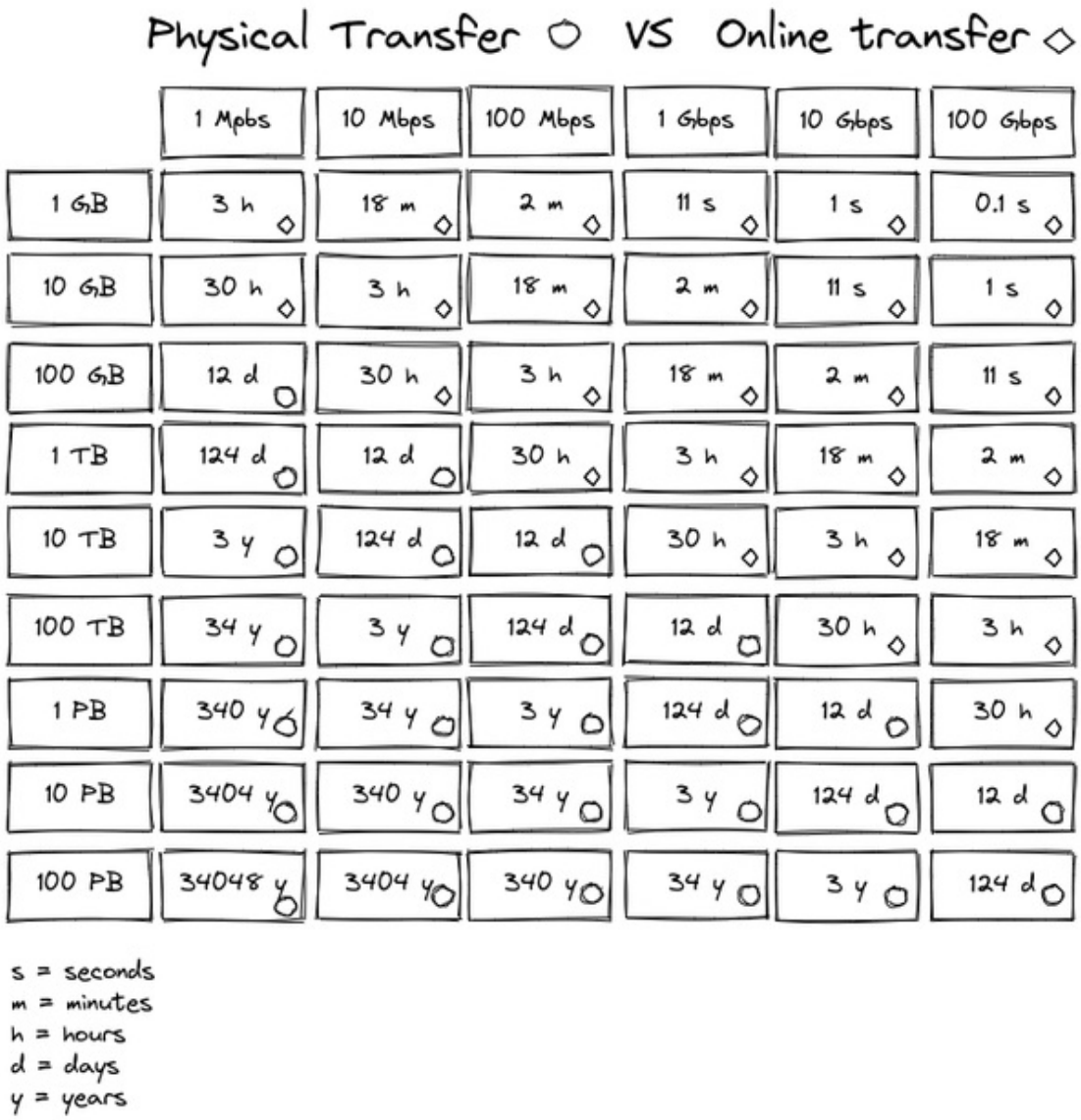


Figure 5-6. - Data transfer Online vs Offline

- *Available tools*: once you have cleared out all the network dynamics you should decide how to handle data upload. Depending on the target system you might want to target (e.g. blob storage, DWH solution, data base, 3rd party application etc.), you generally have the following options at your disposal:
    - *Command line tools* (e.g. AWS CLI, Azure Cloud Shell or Google Cloud SDK) that will allow you to interact with all the services of the cloud provider. You can automate and orchestrate all the processes needed to upload the data into the desired destination. Depending on

the final tool you may have to pass by intermediate systems before being able to upload data to the final destination (e.g. passing by blob storage first before ingesting data into the data warehouse) but thanks to the flexibility of the various tools you should be able to easily implement your workflows for example leveraging bash or Powershell scripts.

- *REST APIs* that will allow you to integrate services with any applications you may want to develop, for example internal migration tools that you have already implemented and leveraged in your experiences or brand new apps that you may want to develop for that specific purpose
- *Physical solutions* for offline migration as discussed in the *Offline versus online transfer* section
- *3rd party Commercial off the shelf (COTS)* that could provide more features like network throttling, custom advanced data transfer protocols, advanced orchestration and management capabilities and data integrity checks.

Final considerations

There are some final checks that you should perform to ensure you don't hit bottlenecks or data transfer issues:

- *Perform a functional test*: you need to execute a test to get the validation that your entire data transfer migration is correctly working. Ideally you'd execute this test during the execution of your MVP, selecting a considerable amount of data, leveraging potentially all the tools that you might want to use during the entire migration. The goal of this step is mainly to verify you can operate your data transfer correctly, while at the same time surface potential project-stopping issues, such as the inability to use the tools (e.g. your workforce is not trained or you do not have adequate support from your system integrator) or networking issues (e.g. network routes).
- *Perform a performance test:* you need to verify that your current infrastructure can handle migration at scale. To do so you should identify a large sample of your data (generally between 3% and 5%) to be migrated to

confirm that your migration infrastructure and solution correctly scale according to the migration needs and you are not surfacing any specific bottlenecks (e.g. slow source storage system)

- *Perform a data integrity check*: one of the most critical issues you might encounter during a data migration is that the data migrated into the target system is deleted because of an error or is corrupted and unusable. There are some ways to protect your data from that kind of risk:
  - *Enable versioning and backup* on your destination to limit the damage of accidental delete
  - *Validate your data* before removing the source data

If you are leveraging standard tools to perform the migration (e.g. CLIs or REST APIs) you have to manage all these activities by yourself. However, if you are adopting a 3rd party ad-hoc application it is likely that there are already several kinds of checks that have been implemented by default (we suggest reading the available features in the application sheet carefully before selecting the solution).

Summary

In this chapter you have seen a practical approach to the data modernization journey and have reviewed a general technological migration framework that can be applied to any migration, from a legacy environment to a modern cloud architecture. The key takeaways are:

- Nowadays one of the most important topics in the CxO agenda is *Data Modernization*. Organizations are trying to understand how they can transition from their legacy environments into something new that will allow them not only to cope with the huge amount of data that they are going to collect and generate, but also to become more flexible and provide their users with several open tools that will enable them to adapt to the different needs they might encounter during the entire data lifecycle.
- There are some trends in data modernization that are boosting the needs for data modernization: Tech refresh, cloud as foundation, the central role of data within an organization, the need for real-time analytics and changing in

user habits.

- There are also some barriers that might slow-down the data modernization process: proliferation of data-silos within the organization, hybrid architectures and regulatory requirements.
- A data modernization journey can be based on three main pillars: building the foundations (business, people and technology), execution of the migration and operationalization and optimization.
- A possible data migration framework can be implemented in four steps: prepare and discover, assess and plan, execute, and optimize
- Prepare and discover is a pivotal step where you focus on defining the perimeter of the migration and then on collecting all the information related to the various workloads/ use cases that you've identified to be migrated.
- Assess and plan is the step where you define and plan all the activities that need to be done to accomplish the identified goal (e.g. categorization and clusterization of the workloads to be migrated, definition of KPIs, definition of a possible MVP and definition of the migration plan with related milestones).
- Execute is the step where you iteratively perform the migration, improving the overall process at each iteration.
- Optimize is the step where you consider the new system as a whole and identify potential new use cases to introduce to make it even more flexible and powerful.
- Understanding the current footprint and what will be the total cost of the final solution is a complex step that can involve multiple actors. Organizations usually leverage RFI/ RFP and RFQ to get more information from vendors/ potential partners.
- When defining the new data platform there is an element that requires a lot of attention: data governance and security.
- Governance and security remain a centralized concern at most organizations. This is because there needs to be consistency in the way roles

are defined, data is secured, and activities are logged.

- The migration framework has to work in accordance with a very well defined governance framework that is pivotal throughout the entire life of data.
- When migrating schema of your data, it might be useful to leverage a pattern like the facade pattern to make the adoption of target solution features easier.
- You need to define your data migration strategy according to your needs.
- It's important to understand regional capacity in order to avoid stock out when scaling the final solution
- Connectivity is the most important piece of technology to have in place when utilizing cloud solutions. This is why it is important to work with proper suppliers in order to setup the best configuration possible for your needs.
- An important focus has to be put on the solution and infrastructure needed to perform the data migration: network configuration, bandwidth availability, cost, people, tools, performance and data integrity are the main points that have to be cleared out.
- The life of a CDO is not as easy as you might expect :-)

In the next chapter we will do a deep dive on how to architect modern data lakes.

---

[1] There are several services freely available on the web that can help you with this estimation exercise. For example: *https://www.calculator.net/bandwidth-calculator.htm*

Chapter 6. Architecting a data lake

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the sixth chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

As you learned in Chapter 3, a data lake is part of the central analytics store that captures raw, ungoverned data from across an organization and supports compute tools from the Apache ecosystem. In this chapter, we will go into more detail about this concept, which is important when designing modern data platforms. The cloud can provide a boost to the different use cases that can be implemented on top of it, as you will read throughout the chapter.

We will start with a recap of why you might want to store raw, ungoverned data that only supports basic compute. Then, we discuss architecture design and implementation details in the cloud. Even though data lakes were originally intended only for basic data processing, it is now possible to democratize data access and reporting using just a data lake — because of integrations with other solutions through APIs and connectors, the data within a data lake can be made much more fit-for-purpose. We will finally take a bird's eye perspective on a very common way to speed up analysis and experimentation with data within an organization by leveraging data science notebooks.

Data Lake and the cloud - A perfect marriage

Data allows organizations to make better decisions within a short time frame and even possibly in real-time. Data is the center around which applications, services, people, technology, money, risk, theories, ideas, innovation, processes, and security revolve. More data means more interest and, as a result, a greater need for processing power, which is a typical issue that cloud solutions can solve.

Recalling what we have already discussed in Chapters 1 and 3, companies usually adopt two main approaches to handle their data: Data Warehouse and/or Data Lake. While the first one is more focused on managing structured data, the latter gives more flexibility because it allows users to play with any kind of data (i.e. structured, semi-structured and even unstructured) and to leverage a wide variety of solutions and tools coming from the open source ecosystems.

ROI Challenges with on-premises data lakes

Organizations need a place to store all types of data, including unstructured data (images, video, text, logs, binary files, web contents). This was the main reason that enterprises adopted data lakes — enterprises (via traditional IT) initially believed that data lakes were just the same as having pure raw storage.

Business units, however, had greater plans for the data being stored by IT departments. They wanted to extract insights and value from the data. The traditional way to process large amounts of data involved Hadoop and related technologies. Therefore, thanks to the evolution of the Hadoop ecosystem, data lakes enabled organizations capable of doing Big Data analytics to go beyond the mere concept of storage offload — data lakes brought advanced analytics and ML capabilities within reach. Hadoop and related technologies jumpstarted a massive adoption of data lakes in the 2010s.

However, because of the increase of the volume, variety and velocity of the data, companies struggled to get sufficient return on investments (ROI) from their data lake efforts. The challenges that on-premises Hadoop clusters may encounter can be summarized in:

- **Total Cost of Ownership (TCO)**: resource utilization and overall cost of

management of an on-premises data lakes become unmanageable

- **Scalability:** resource intensive data and analytics processing can lead to missed SLAs
- **Governance**: data governance and security issues open up compliance concerns
- **Agility**: analytics experimentation is slow due to resource provisioning time

With estimates that by 2025, 80% of organizations' data will be unstructured, the on-premises world is no longer able to provide adequate environments at an affordable price. Unfortunately, the explosion of the data led to the development of huge and complex Hadoop clusters that are extremely challenging to maintain. This is why the rise of cloud solutions, as you saw in Chapter 2, allow organizations to first reduce the TCO and then build a platform for innovation because people within the company can focus on business value instead of hardware management.

Cloud data lakes as a perfect habitat

The cloud paradigm is hugely beneficial for data lakes because:

- It is not mandatory to store all the data into a HDFS cluster that is expensive to keep running; we can instead leverage, for the majority of the time and especially for the RAW/ Landing/ Bronze data, an object storage solution (e.g. AWS S3, Azure Blob storage or Google Cloud Storage) that is fully managed, infinitely scalable and that costs a fraction of an Hadoop cluster. When needed (i.e. when there is a need for data processing/ transformation or for executing a production workload) the cluster Hadoop with the underlying HDFS can be instantiated and data immediately synchronized with the one stored in the object store.
- Hadoop clusters, that provide not only storage capabilities but even processing compute power, can be created on-demand in a short amount of time (few minutes or seconds) generating an immediate cost saving due to the fact that they do not need to be always on. These Hadoop clusters can read/ write data directly from object storage — even though such data access is slower than reading/ writing to HDFS, the cost savings of having

ephemeral clusters can make the overall tradeoff worthwhile.

- Hypescalers usually offer capabilities to leverage less expensive virtual machines (spot instances, preemptible instances) for worker nodes. These VMs have the drawback that they can be evicted by the system at any time (generally with a 30-60 seconds notice) but they can easily be substituted with new ones thanks to the underlying Hadoop technology that is worker node failure tolerant. Thanks to this approach, additional costs can be saved.
- Nowadays, the majority of the Hadoop services available on hyperscalers are fully managed and offered in PaaS mode (even though you can build your own Hadoop cluster leveraging a pure IaaS service). PaaS means that you do not need to manage by yourself all the VMs needed for master and worker nodes but that you can instead focus on building processing to extract value from the data.
- On-premises Hadoop clusters tend to generate data silos *(*refer *Chapter 1 - Data silos means data movement tools*) within the organization because of the fact that the HDFS pools were not designed to be prone to data sharing. The fact that instead in the cloud we can have an effective separation between storage (i.e. data in object storage that can be injected in any HDFS cluster) and compute (i.e. VMs created on-demand) gives organizations better flexibility in handling the challenges in data governance.

Cloud is the perfect habitat for data lakes because it addresses all the challenges with on-premises data lakes discussed at the beginning of the chapter by:

- Reducing the TCO via the ability to save money on storage and compute.
- Handling (theoretical) infinite scalability leveraging the power of hyperscalers.
- Driving a fail-fast approach speeding up experimentations thanks to the elasticity of the underlying platform.
- Adopting a consistent data governance approach in the platform across several products to guarantee security, control and adherence to the

compliance requirements.

The market is strongly investing in data lakes and especially in the cloud based ones. We constantly hear about AWS EMR, Azure Datalake and Google Cloud Dataproc, just to name a few, when talking to organizations that are starting their data modernization journey. The majority of them are of course Hadoop based solutions but it is not limited to that: companies like Databricks (the developers behind the open source Spark engine that is part of all major Hadoop distributions) have built an entire business behind the development of a complete data platform that offers not only storage and compute but a full set of features to handle the entire data lifecycle (refer to *Chapter 3 - The data lifecycle - Store - Openness*).

Now that you have seen how the cloud is an accelerator for such solutions, let's have a look at the common architecture design and implementation details of a data lake.

Architecture design and implementation details

When approaching the design of a data lake there are several questions that you'll need to ask yourself:

- What kind of data am I going to manage?
- Where can I find the data that I need within the organization and how can I collect it?
- Is there any specific constraint related to where I can store the data?
- Is the dataset that I found still valid? Do I need to decommission it?
- What are the use cases that I can solve with current architecture?
- What are the technologies that I can leverage both on-premises and in the cloud?

In this section, we will dive into design and implementation details of a data lake to help you answer these questions.

The role of the data catalog

Recall from Chapter 3 (see Figure 3-6) that there are, from a general point of view, four main storage areas in a data lake:

- *RAW/ Landing/ Bronze* where all the RAW data can be collected and ingested directly from the source systems
- *Staging/ Dev/ Silver* where the more advanced users (e.g. data engineers, data scientists) process the data to be ready for the final users and move it into the last area
- *Production/ Gold* where the final data used by the production systems will be kept
- *Sensitive* (optional) where sensitive data resides. It is connected with all the other stages and it facilitates the governance of data access to make sure it complies with company and government regulations.

In reality, data is more scattered than that: think about an organization that is already leveraging other solutions like databases or data warehouses or applications that are leveraging file systems or blob storage to store texts, audio files or images. There will then be multiple sites even outside the data lake where the data can find a home. Hence, the idea to have a centralized data lake containing all the data of the organizations is quite impossible. In this configuration, how can we ensure that data scientists will be able to find all the data they need for the implementation of their brand new ML algorithm and business users will have access to the latest data to update their shining dashboard? We need a magic tool that will allow users to easily search for the data they have to use to perform their tasks: the *data catalog.* A data catalog, as we have introduced in Chapter 3 (refer to *Governance and security* section), is a repository of all the metadata that describe the datasets of the organization and it will help data scientists, data engineers, business analysts and (from a general point of view) any users who may need to leverage data, to find a path toward the desired dataset. In Figure 6-1 you can see a possible high level architecture that describes how the data catalog can be connected to the data lakes and the other solutions of the data platform.

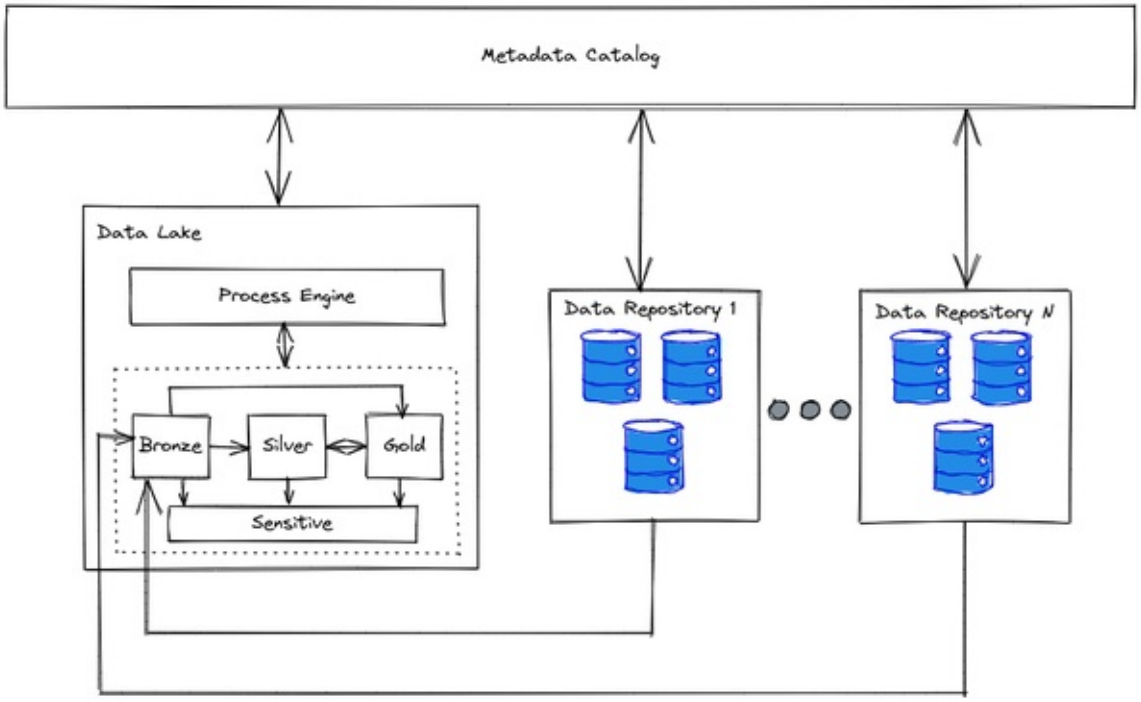


Figure 6-1. Data catalog and data lakes + sparse data repositories

If the data live across several data repositories (one of them being the data lake) and the processing engine and the Gold repository for the analytical workloads are within the data lake, there are two kind of problems that can arise:

- Data *comprehensiveness*: it is important to have access to the entirety of the datasets of the organization
- Data *duplication*: it is important to avoid multiple copies of the same data

The data catalog address these challenges in the following ways:

- Data *comprehensiveness*: the data catalog is the de facto search engine of the data indexing all the datasets of the company (this can be done automatically or manually, as we covered in *Chapter 3 - Governance and security*)
- Data *duplication*: metadata can contain information about the level of the data set (e.g. master or replica). It is important to note that when bringing and transforming a master data set within a data lake, there could be a need for a sync after the computations.

In Figure 6-2 you can see the high level integration with data catalog for data

processing

0. Metadata catalog fulfillment
1. Search for the desired data asset
2. If the data asset NOT already in the data lake, then make a copy of it into the bronze storage area
3. Execute your desired elaboration on the copied data asset
4. Update the original data asset if needed

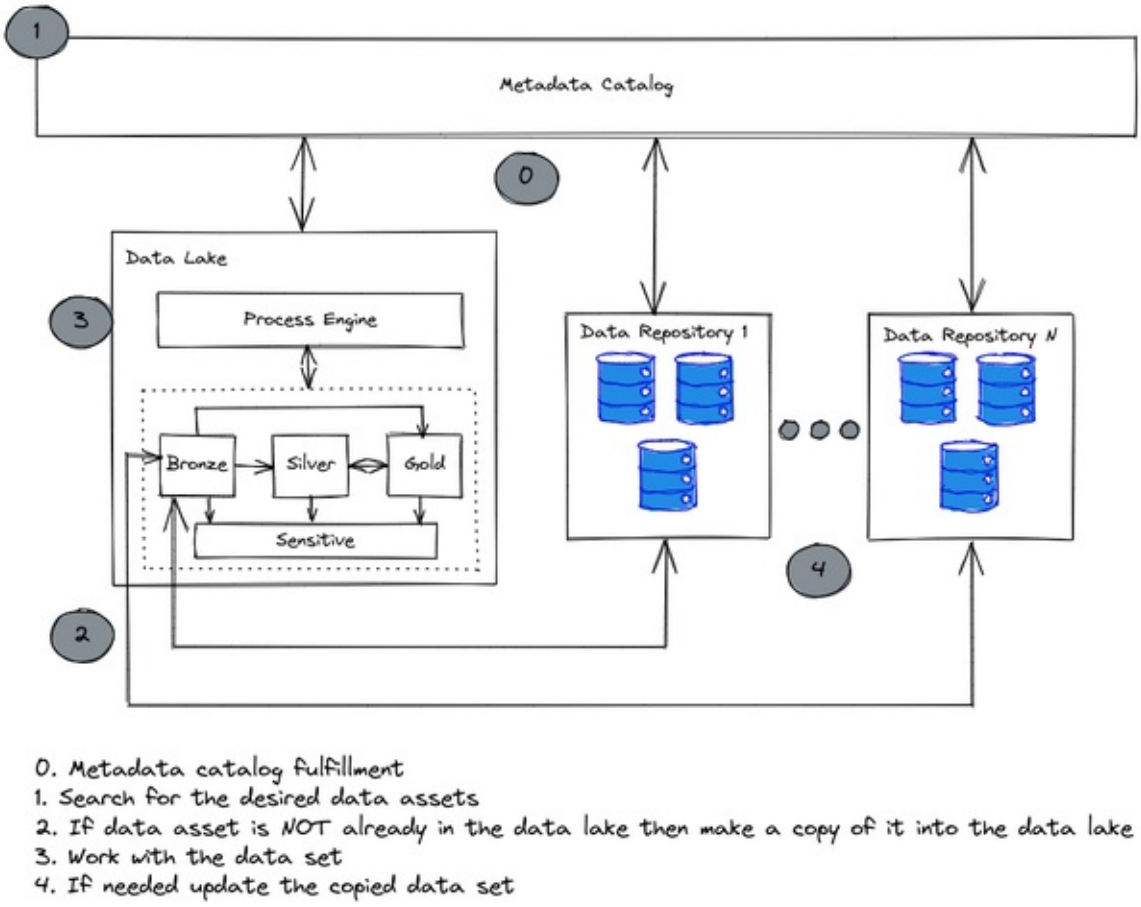


Figure 6-2. From data catalog to data processing

The data catalog is a single point of access for information related to the data, governance activities, and auditing (we can easily find *who is searching for what*). But it can even be leveraged for another goal. As we know, data proliferation within a company is something extremely challenging to eliminate.

A data catalog can help in rationalizing the various datasets an organization may have because it can help in finding:

- Duplicated datasets: copies can be deleted
- Unused datasets: they can be decommissioned
- Similar datasets: they can be merged

This is a critical point, as it will enable organizations to focus on the data that is most relevant to them and avoid using resources to store and process data that is not useful, all of which could lead to cost savings.

Batch, streaming and lambda/ kappa

When analyzing data workloads, the key question to address is about the *age* of data that needs to be processed: is it something that has been already stored somewhere over a period of time or is it something that has just arrived into the system? Based on the response there are typically two approaches on data processing: *batch* or *streaming*. The batch approach has reigned for the last 20+ years (and it is of course still an important approach within the majority of organizations), meanwhile streaming has become popular recently especially with the advent of the cloud—mainly because it requires an elastic infrastructure to be handled at scale.

In 2014, two new architectures were proposed to allow the processing of data at rest and in motion together, at scale: Lambda (by Nathan Marz) and Kappa (by Jay Kreps). The Lambda architecture (shown in Figure 6-3) provides a *general-purpose approach to implementing an arbitrary function (batch processing vs streaming processing) on an arbitrary dataset and having the function return its result with low latency*[1]. It combines what is called *a batch layer* that spans all the (historical) facts and a *speed layer* for the real-time data having, from a general point of view, different technology stacks behind the layers.

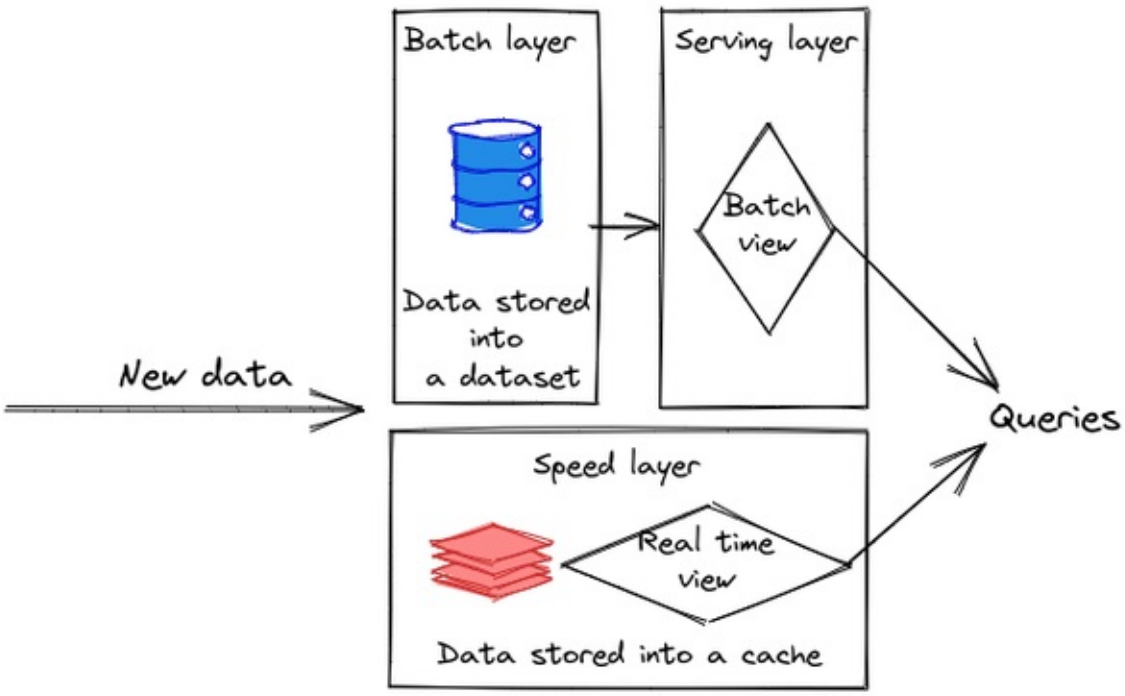


Figure 6-3. Lambda architecture

New data ingested into the system is stored both in a persistent dataset (*batch layer*) and in a volatile cache (*speed layer)*. The first one is then indexed and made available to the serving layer for the *batch views* while the second is exposed by the speed layer via the *real time views.* Both datasets (persistent and volatile) can be queried in parallel or disjointly to answer the needed question. This architecture is generally deployed in an Hadoop environment where HDFS (bronze/ silver and gold layers we have seen before) can be leveraged as a batch layer while technologies like Spark, Storm or Beam can be used for the speed layer. Hives can then finally be the solution for the implementation of the serving layer for example.

The Kappa architecture (shown in Figure 6-4) is a simpler version of the Lambda architecture where the same technology stack is used to handle both real-time stream processing and historical batch processing.

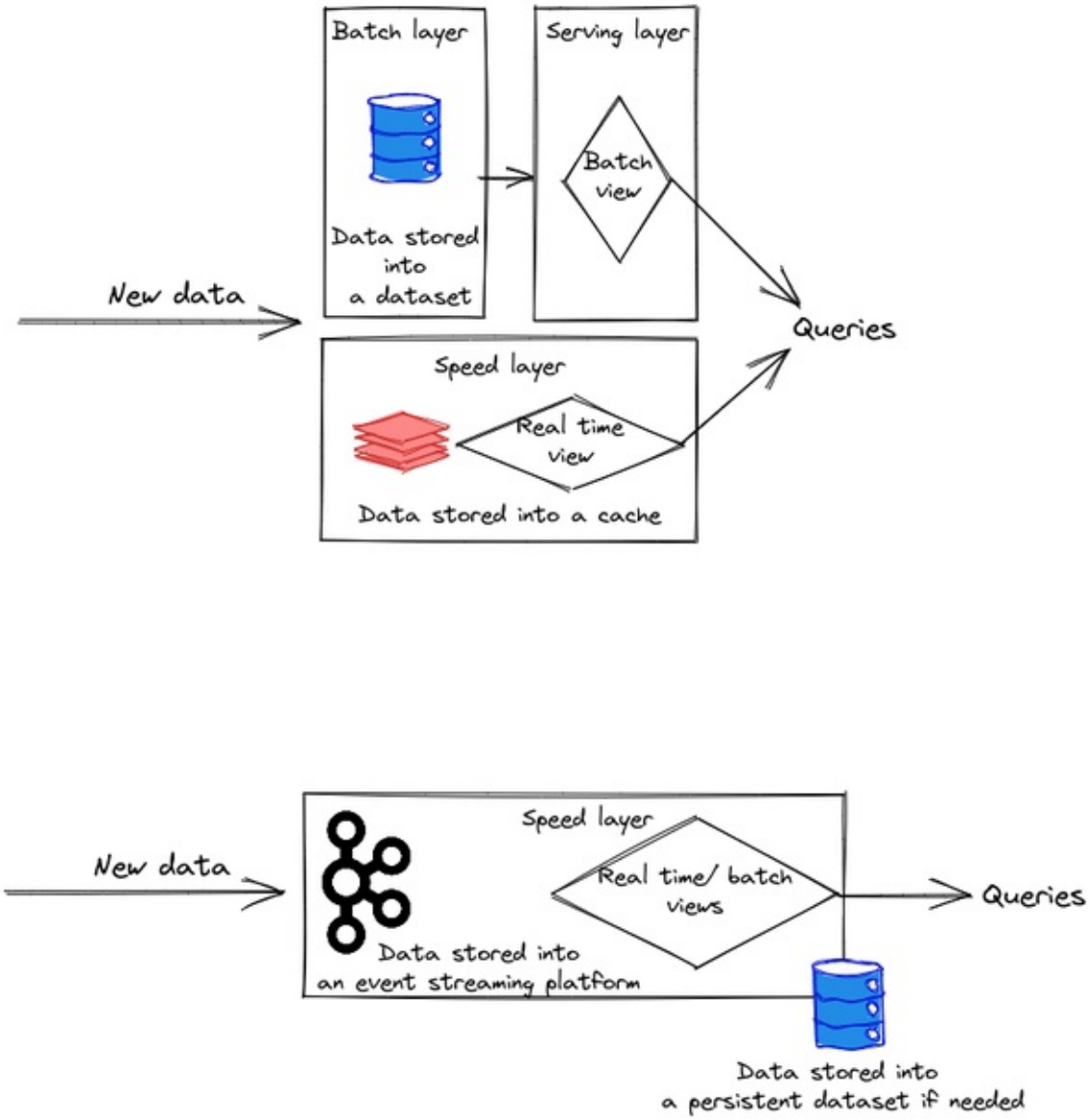


Figure 6-4. Kappa architecture

In the Kappa architecture you can perform both real-time and batch processing with a single technology stack (e.g. Beam/ Spark). The core is the streaming architecture. First, the event streaming platform stores the incoming data. From there, a stream processing engine processes the data in real-time making data available for real time views. At this stage the data can be persisted even into a database to perform batch analysis when needed and leveraging the same technology stack.

Now that we have seen the various approaches that you can leverage when managing and processing data within a data lake, let’s have a look at the

common technology landscape available.

Hadoop landscape

When we talk about data lakes, we are in essence discussing Hadoop technology. Not only because the idea of the data lake started with MapReduce that backed Hadoop technology for a very long time (as you have read in *Chapter 3 Foundational elements - The advent of Hadoop and the birth of Data Lake*), but because it is the de facto standard when adopting this technology both on-premises and in the cloud. The popularity of the open source framework has grown a lot throughout the last 15 years, creating a rich ecosystem of projects to cover every aspect from data ingestion, to storage, to data processing and visualization. The popularity of Hadoop led several companies, like IBM and Cloudera just to name a few, to invest in the development of commercial distribution and, on top of this, in the development of brand new solutions to add to their framework version. In Table 6-1 we have listed some popular tools of the framework split by use case.

Table 6-1. Hadoop solutions per use case

Use Case	Hadoop ecosystem (open source and commercial)
Batch computation	MR2 (MapReduce), Spark
Filesystem	HDFS
Streaming computation	Storm, Spark Streaming, Kafka, Flume,Flink
Governance	Atlas
Graph Processing	Giraph, GraphX
Ingestion	Sqoop, Flume, NiFi
NoSQL	HBase, Cassandra, ELK, Kudu
Security	Rangers, Sentry, Knox

SQL over semi-structured data	Hive, Impala, Presto, Spark SQL, Drill
Workflows	Oozie, Pig, Spark, Airflow, Azkaban

The solutions listed in Table 6-1 are of course available in an on-premise environment but they can be easily deployed even in the cloud leveraging a IaaS approach. Next, the hyperscalers transformed such popular products into fully managed solutions to: reduce the burden of the provisioning and infrastructure management on the user side, augment the inherent scalability and elasticity, and, of course, reduce the cost. Table 6-2 outlines the most popular solutions available in the cloud to facilitate the cloud migration and adoption of the most popular on-premises Hadoop technologies.

Table 6-2.

On-premises	AWS	Azure	Google Cloud Platform
Airflow, Oozie	**Data Pipeline, Airflow on EC2, EMR**	**HDInsight, Data Factory, Databricks**	**Cloud Composer, Cloud Dataproc**
Apache Kafka, MapR Streams	Kinesis, Kinesis Streams, Managed Kafka	Event Hubs	Cloud Pub/Sub, Confluent Apache Kafka
Beam	Beam on EMR, Kinesis Streams	Beam on HDInsight, Stream Analytics	Cloud Dataflow
Drill	Athena, Redshift	Synapse, HDInsight	BigQuery
HBase, Cassandra	DynamoDB	Cosmos DB	Cloud Bigtable
HDFS	EMR	HDInsight, Data	Cloud Dataproc

HDFS	EMR	HDInsight, Data Lake Store	Cloud Dataproc
Hive	Athena, EMR, Redshift	HDInsight, Synapse	BigQuery, Cloud Dataproc
Impala	Redshift, EMR, Athena	HDInsight, Synapse	BigQuery
Sentry, Ranger, Knox	AWS IAM	Azure IAM	Cloud IAM, Dataplex
Spark	EMR	HDInsight, Databricks	Cloud Dataproc, Serverless Spark
Storm	Kinesis Streams, EMR	Stream Analytics, HDInsight	Dataflow, Dataproc

With this information in mind let's review the data lake reference architectures for the three main cloud providers.

Cloud data lake reference architecture

In this section we are going to review some reference architectures for the implementation of a data lake in the public cloud leveraging services of the main three hyperscalers (i.e. AWS, Azure and Google Cloud Platform) to provide you with more insights that you can leverage as reference during your next data migration project. Please note that, as with any cloud architecture, there is no *one-size-fits-all* design. There will always be several different options available that could suit your specific needs.

Amazon Web Service

The managed Hadoop service on AWS is Amazon Elastic Map Reduce (EMR). However, that provides only analytics data processing. AWS recommends that we think of data lakes more holistically and consider that analytics of structured data is better carried out using Amazon Athena[2] or Amazon Redshift[3]. Also, the raw data may not already exist in cloud storage (Amazon S3) and will need to be

ingested.

Therefore, the recommended way to implement a data lake on top of AWS is to leverage AWS Lake Formation. It is a fully managed service that enables the development and the automation of data collection, data cleansing/ processing, and data movement in order to make the data available for analytics and machine learning workloads. It comes with a permissions service that extends the AWS Identity Access Management capabilities and allows for better data governance and security (i.e. fine grade policies, column and row level access control, etc.). Looking at the architecture in Figure 6-5[4] we can identify the main components:

- Data sources, which, in this case, are Amazon S3, Relational and No-SQL databases
- Storage zones on top of Amazon S3 buckets
- Data catalog, data governance, security and process engine orchestrated by AWS Data Lake Formation
- Analytical services such as Amazon Athena, Amazon RedShift and Amazon EMR to provide access to the data

Figure 6-5. AWS Data lake architecture

This architecture should work for any kind of use case managing structured, semi-structured, and unstructured data. Once the data has been prepared and transformed it can easily be made available even to solutions (such as Databricks or Snowflake) outside of the data lake thanks to the pervasive nature of AWS S3

service, which can potentially be connected to any other service of the platform.

Microsoft Azure

On the Azure platform there are several ways to implement a data lake because there are different solutions that can help in designing a possible target architecture. Typically we'll see the architecture shown in Figure 6-6[5], in which we can identify following main components:

- *Azure Data Lake Storage Gen2 (*ADLSG2): object storage optimized for huge volumes of data, fully compatible with HDFS and where users typically implement all the data zones of the data lake.
- *Azure Data Factory*: serveless solution to ingest, transform and manipulate the data
- *Azure Purview*: solution that provides governance for finding, classifying, defining, and enforcing policies and standards across data.
- *Azure Synapse Analytics*: the analytics service used to issue SQL queries against the data stored in ADLSG2
- *Databricks*: complete analytics & ML solution based on Spark processing engine
- *Purview*: data catalog and data governance solution
- *PowerBI*: Business Intelligence (BI) reporting and visualization tool

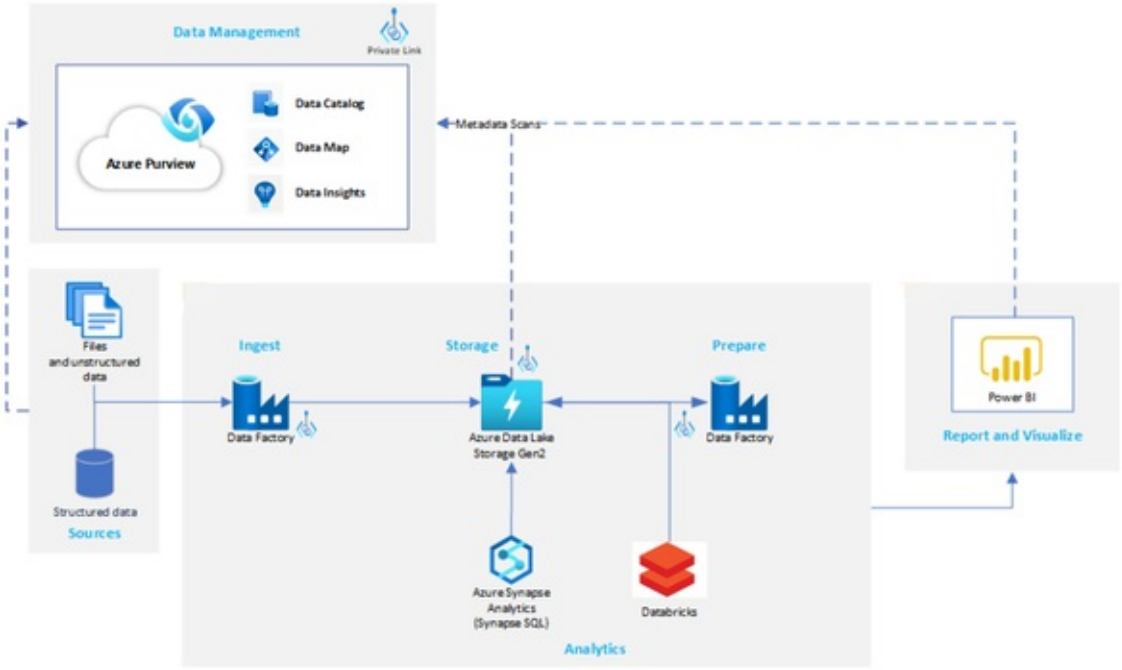


Figure 6-6. Azure Data lake architecture

It has to be noted that Azure Databricks can be interchanged with Azure HDInsight. The main difference between the two is that Databricks is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform while HDInsight is a managed full Apache Hadoop distribution (*i.e. more than just Spark tools, but less tuned to Azure*). If you want to work with the standard Hadoop ecosystem solutions you should leverage HDInsight while if you prefer to leverage a complete analytics & ML solution based on Spark you should go for Databricks.

Google Cloud Platform

The last example we want to present is the one running on top of the Google Cloud Platform (illustrated in Figure 6-7) where the different serverless and fully managed components are communicating with each other via APIs. We can identify following main components:

- *Data Fusion*: a solution to ingest and transform data both in batch and in streaming
- *Pub/Sub*: messaging middleware to integrate the input arriving from Data Fusion and the Hadoop cluster delivered by Dataproc

- *DataProc*: on-demand Apache Hadoop cluster delivering HDFS, Spark, and non-Spark capabilities.
- *Cloud Storage*: object storage solution where data zones are implemented.
- *Bigtable and BigQuery*: analytical and real time data processing engine
- *Looker*/ *Looker studio*: BI and visualization solutions
- *Dataplex*: single pane of glass to handle data governance, data catalog and security
- *Composer*: data workflow orchestration service based on Apache Airflow that empowers users to author, schedule, and monitor pipelines.

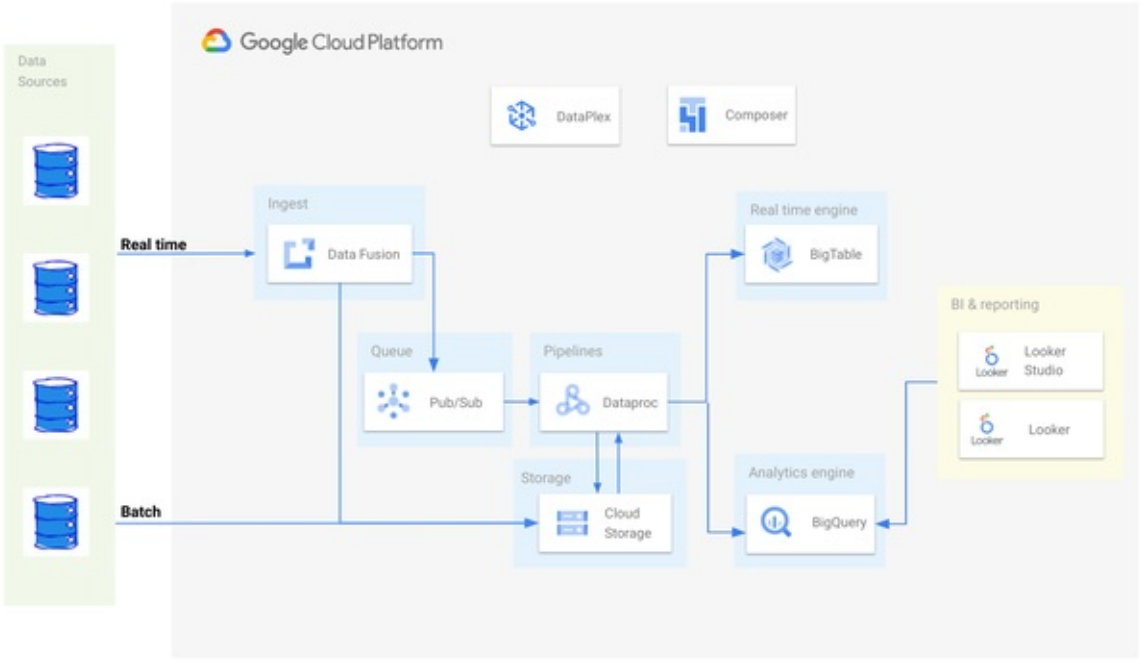


Figure 6-7. Google Cloud Platform data lake architecture

Depending on your use case, Hadoop or a HDFS cluster may not always be the best fit. Having the data in Cloud Storage gives you the freedom to start selecting the right tool for the job at hand, instead of being limited to the tools and compute resources available on the HDFS cluster. For instance, a lot of ad-hoc Hive-SQL queries can be easily migrated to BigQuery which can employ either its native storage or read/query directly off Cloud Storage. Similarly, streaming applications may fit better into Apache Beam pipelines, and these can be run on Dataflow, a fully managed streaming analytics service that minimizes

latency, processing time, and cost through autoscaling and batch processing.

Now that you're familiar with some design patterns and reference architecture of cloud data lakes, let's dive into how we can take it a step further by extending the data lake with third party solutions.

Integrating the data lake: the real superpower

Data lake technology has become so popular due to the fact that they can handle any kind of data—from very structured tabular format to unstructured files like text or images—enabling a myriad of use cases that were not possible beforehand. Discovering the reliability of a supplier simply by performing text analysis on invoices PDF file, having the ability to identify the actors in a film cut scene, or the ease in implementing a real time dashboard to monitor sales on an e-commerce website are just a very small subset of the use cases that can be enabled by a data lake. But the real superpower of a data lake is its ability to connect the data with an unlimited number of process engines to activate potentially any use case you have in mind.

APIs to extend the lake

Everything starts with the data that has to be ingested into the RAW landing zone. This can be done in a batch mode or it can be directly streamed from a real time data source. Once the data has been ingested it has to be processed (and here we could have several passes) before being available for visualization and activation (ref: *Chapter 3 - The data lifecycle*). Every single step can involve several different engines that are part of the data lake itself or are third party products sitting somewhere in the cloud or, in certain cases, even on-premises.

How could you make all these different systems, potentially sitting in hybrid environments, talk to each other and exchange data? You leverage Application Program Interfaces (APIs): pieces of software that enable communications between two or more systems via a shared protocol on top of the standard HTTPS. The most common ones are the REpresentational STate (REST) and the Google Remote Procedure Call (gRPC) but you may find some systems still leveraging old fashioned models like Simple Object Access Protocol (SOAP). The majority of modern applications leverage APIs to integrate services and to

exchange data, and even when they are making available ad hoc connectors, they usually have been built on top of APIs. You can consider APIs as highways where data (the cars) can pass from one system to another: the toll booths are the security measures implemented to protect the data traffic and the speed limits are the rate limits. Thanks to these highways, data can flow among several systems and be processed by any kind of process engine, from a standard ETL to a modern machine learning framework like TensorFlow or PyTorch.

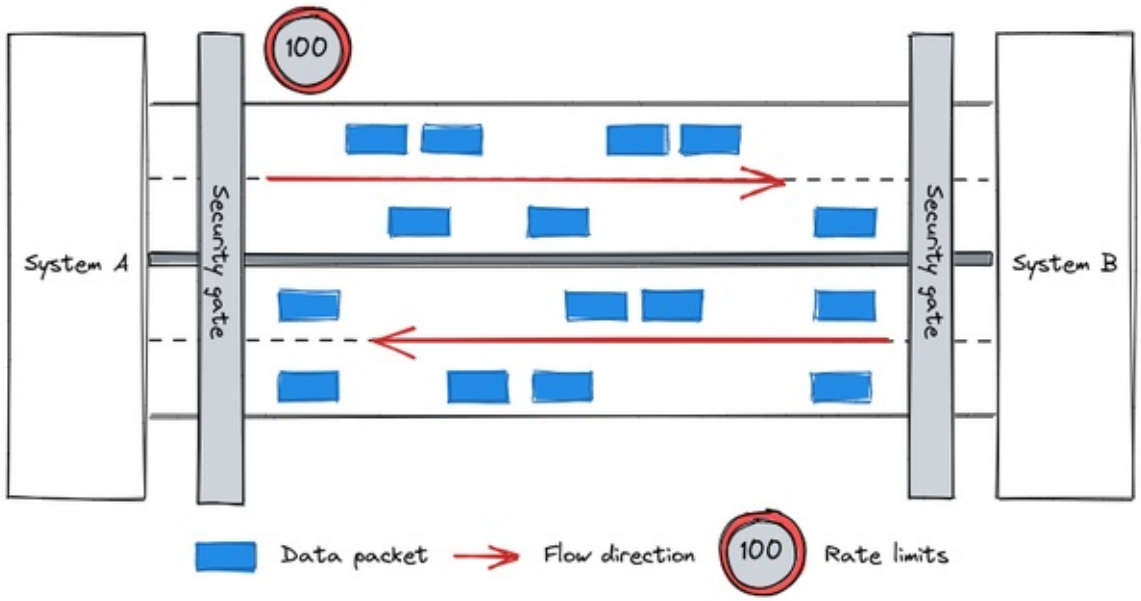


Figure 6-8. API as a highway

Following this approach organizations can evolve their data lake in various ways and where native solutions are not available out of the box yet, they can be implemented with a little bit of system integrations.

It has to be said that APIs are not just useful to integrate the data lake with several solutions but they can even be used to monitor, configure, tweak, automate, and, of course, access and query the lake itself.

The evolution of data lake with Apache Iceberg, Apache Hudi and Delta Lake

The primary goal of integrating the data lake with other technologies is to expand its capability beyond the out of the box features made available by the Hadoop framework. When we consider a standard Hadoop stack (as you saw in

Figure 3-5), there is one element that is missing which is typically handled with other technologies (for example OLTP databases): ACID transactions management. Apache Iceberg, Apache Hudi and Delta Lake are open source storage layers sitting on top of HDFS that have been implemented to address this key aspect. While they each come with a set of different features (e.g. Apache Iceberg support more file format than Apache Hudi and Delta Lake), the set following features are common to every implementation:

- Atomicity, Consistency, Isolation and Durability (ACID) of the data, giving users the certainty that the information they are querying are consistent
- Overcomes the inherent limitation of HDFS in term of file size—with this approach even a small file can work perfectly
- Every single change made on data will be logged, guaranteeing a complete audit if ever necessary and enabling time travel queries.
- No difference in handling batch and streaming ingestions and elaboration
- Fully compatible with Spark engine
- Storage based on Parquet format

These storage solutions can enable several use cases that were usually handled by other technologies (such as the Data warehouse, which we will investigate more in Chapter 7) mainly because of the ability to prevent data corruption, execute queries in a very fast mode, and frequently update the data. There are very specific scenarios where the transactional feature of this new enabled storage (i.e. update/ delete) plays a crucial role — these are related to General Protection Data Regulation (GDPR) and California Consumer Privacy Act (CCPA). In this situation, organizations are forced by these regulations to have the ability to purge personal information related to a specific user in case of a specific request. Performing this operation in a standard HDFS environment could be time and resource consuming because all of the files that pertain to the personal data being requested must be identified, ingested, filtered, written out as new files, and the original ones deleted. These new HDFS extensions simplify these operations, making them easy and fast to execute and, more importantly, reliable.

These open source projects have been broadly adopted by the community, which is heavily investing in their evolution. Apache Iceberg is vastly adopted within Netflix, for example, while Apache Hudi is powering Uber's data platform. Delta lake is a particular case— even though it is a Linux Foundation project, the main contributor is Databricks, the company behind the Spark engine that developed and commercialized an entire suite to handle big data workload based on a vendor proprietary version of Spark and Delta Lake.

In addition to ACID, there are two other features that are evolving the way users can handle data in a data lake:

- *Partition evolution*: the ability to update the partition definition over a file (i.e. table). The partition is the information that allows a file system to split the content of a file into several chunks in order to avoid completing a full scan when retrieving information (e.g. extracting sales figures of Q12022— you should be able to query only data belonging to the first quarter of the year instead of querying the entire data set and then filtering out the data you do not need). Considering the fact that the definition of a partition in a file can evolve because of a business need (e.g. working hours of a device for which we want to collect insights) it is critical to have a file system that is able to handle these changes in a transparent and fast way. HDFS (via Hive) can achieve this from a logical point of view but the computational effort required makes it practically non achievable. Please note that at the time of writing this feature is offered only by Apache Iceberg.
- *Schema evolution* : just like the partition, the schema may need to be updated over time. You may want to add, remove, update columns in a table and the filesystem should be able to do that at scale without the need to re-transform the data (i.e. without an ELT/ ETL approach). Please note that while, at the time of writing, this is fully supported only by Apache Iceberg, all the solutions are able to handle schema evolution when leveraging Apache Spark as the engine.

Now that you have seen how you can expand the capabilities of a data lake and enrich the broad set of use cases that can be solved with them, let's have a look at how to actually interact with data lakes.

Interactive analytics with notebooks

One of the most important things when dealing with data is the ability to interactively get access to it and perform analysis in a very easy and fast manner. When leveraging a data lake, data engineers, data scientists and business users can leverage a plethora of services like Spark, Pig, Hive, Presto etc. to handle the data. A solution that has grown a lot in popularity in several communities, primarily with data scientists, is what we consider the best Swiss army knife an organization may leverage for data analysis: Jupyter Notebooks.

Jupyter Notebook is an open source application that can be used to write *live documents* containing a mix of text, code to be executed, plots and charts. It can be considered a live book where, in addition to the text that you write using a markup language like Markdown, there is code that you execute that performs some activities on data (e.g. query, transformation, etc.) and eventually plots some results generating charts and/ or tables.

From an architectural perspective, you can think of the Jupyter Notebook running on top of a data lake as three different components, as presented in the left-hand side of Figure 6-9:

- *HDFS* : the storage
- *Kernel*: the process engine
- *Jupyter Notebook*: the server that is leveraging the process engine to access the data and, via the browser, provide users with the user interface to write and execute the content of the notebooks

There are several kernels that you can leverage, but the most common one is Spark, which can be accessed using PySpark via code written in Python programming language (as shown in the right-hand side of Figure 6-9).

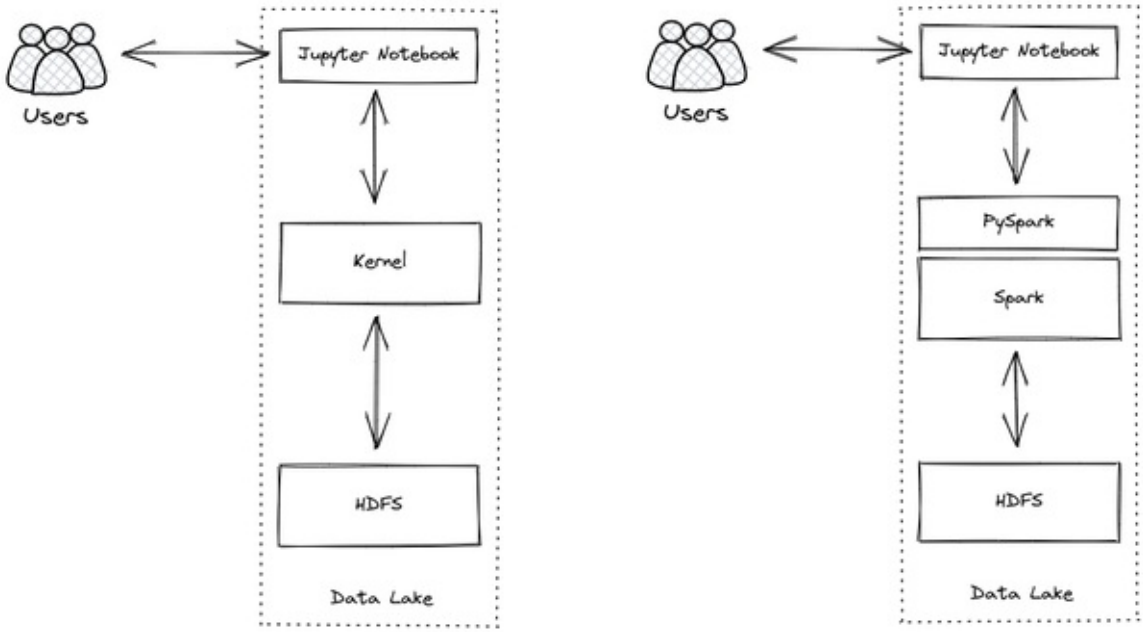


Figure 6-9. Jupyter general notebook architecture & Spark based kernel

Once properly configured, users can start immediately writing text and code directly into the Notebook and interact with datasets as they would while leveraging the Spark command line interface. You can use notebooks to develop and share code with other people within the organization, perform quick tests and analysis or quickly visualize data to get some extra insights as you can see in Figure 6-10. It is important to highlight that results are not *static* because they are *live* documents: this means that if the underlying dataset changes or if the piece of code changes, results will be different when the person with whom the notebook is shared re-executes the notebook. Once you've finalized the analysis, usually there are two different paths that you can take:

- Share the source code of the notebook (Jupyter Notebook produces .ipynb files) with other data scientists, data engineers, developers or anyone who wants to contribute. They can load the file in their environment and re-run the code (it is of course mandatory to have the right access to the underlying storage system and to any other solution integrated via APIs)
- Make results static via the generation of an HTML page or a PDF document that can be shared to a broader set of users.

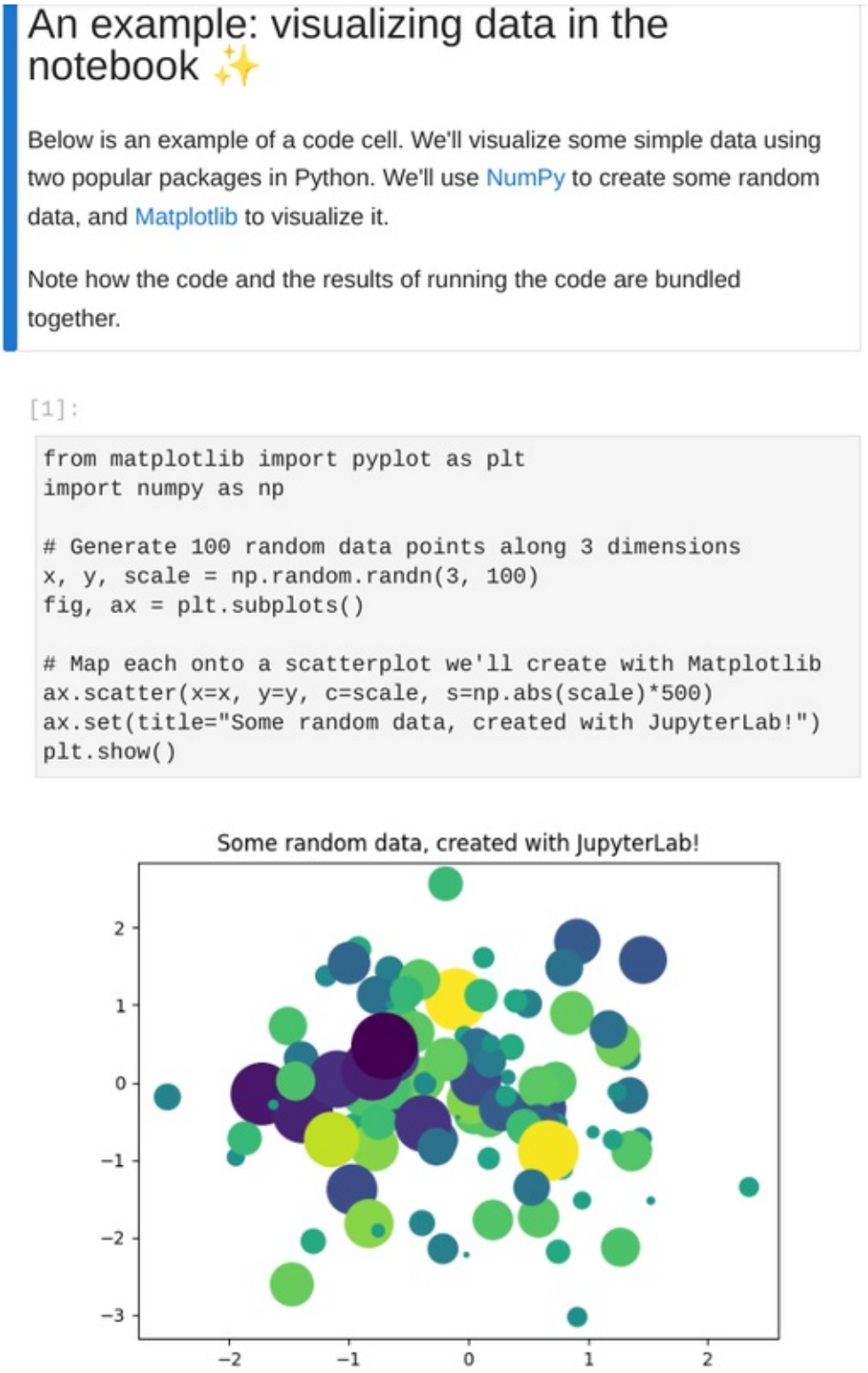


Figure 6-10. Jupyter Notebook in action (from jupyter.org)

Notebooks have become the de facto standard solution for interactive data analysis, testing and experimentation because of their extreme flexibility, both in

terms of programming languages that you can leverage (thanks to the numerous available kernels) and in terms of activity that you can do (e.g. you can generate a chart starting from data in your data lake or you can train a complex machine learning algorithm on a small subset of data before moving that to production).

What we have seen working with several organizations is that the *Notebook approach* is a first step in putting in place a *journey to data democratization* (as we will discuss later in the next section) because it allowed the more technically savvy people to have an immediate and standardized access to the data, fostering an approach to collaborations. While data scientists were the pioneers of notebook use, data engineers and analytical engineers continued to adopt them as well. We have even seen companies developing custom libraries to be included in *notebook templates* with the aim to facilitate the integration with several other solutions (out of the shelf or custom developed) bringing the standardization to another level and reducing the learning curve for the final users (and even the pain). This standardization, thanks to the container technology, has been brought even at compute level: every time users within the company launch a notebook they are, behind the scene, launching a container with an adequate number of computing resources and a set of tools that are immediately at their disposal for their data analysis.

Cloud hyperscalers are making some solutions available for managed versions of Jupyter Notebooks — AWS Sagemaker, Azure Machine Learning studio and Google Cloud Vertex AI Workbench, to name a few—that can help in getting rid of headaches related to the management of the underlying infrastructure thanks to the fact they are fully managed.

Now that you have a better understanding of how you can leverage Jupyter Notebooks to expand the data lake, we'll turn our attention to helping you understand how people within organizations can handle data ingestion up to data visualization reporting, moving from a fully IT model to a more democratic approach.

Democratizing data processing and reporting

The best value that data provides to an organization is that it enables decision makers and users in general to make informed decisions. To that end, data should

be accessible and usable by any authorized individual without the need of ad-hoc expertise and specialism. Even if it is possible to implement democratization of the data access within a company from a technical point of view, it is not sufficient to focus just on the technology. Organizations need to also implement data cataloging and governance operations. Having good metadata describing proper datasets will enable users to find the right information they need and to activate correct data processing and reporting. In this section we will explore the key technologies that can help an organization in switching from a fully IT driven approach to a more *democratic* approach when building a cloud data lake.

Build trust in the data

When one of the authors of the book was working as a consultant for an important retail customer several years ago, he worked to develop solutions to automate the data extraction from the sales database to generate reports to be leveraged by the business users. He needed to be available every time decision makers had questions like

- Where did you get this data from?
- How did you transform that data?
- What are the operations you made before generating the report?

Once he was assigned to another customer, the organization's IT team took over the entire end to end process. While they could fix bugs and monitor the report generation, they were unable to convince the decision makers that the data and reports were trustworthy because they did not have the right level of knowledge to answer these questions.

This trust was clearly a bottleneck that would have been removed by having final users be able to dig into their data in an autonomous way and audit it from ingest to the final reported value. This approach may have been unrealistic in the past, but the fact that people are now more *digitally experienced* helps a lot in shifting this method.

As illustrated in Figure 6-11 there is a clear transition in ownership and responsibility from the old world represented to the left to the new one represented to the right. While the majority of the work used to be carried out by

the IT department, nowadays final users have in their hands tools that enable them to handle the majority of the activities from the data cataloging up to the data visualization with a great level of autonomy.

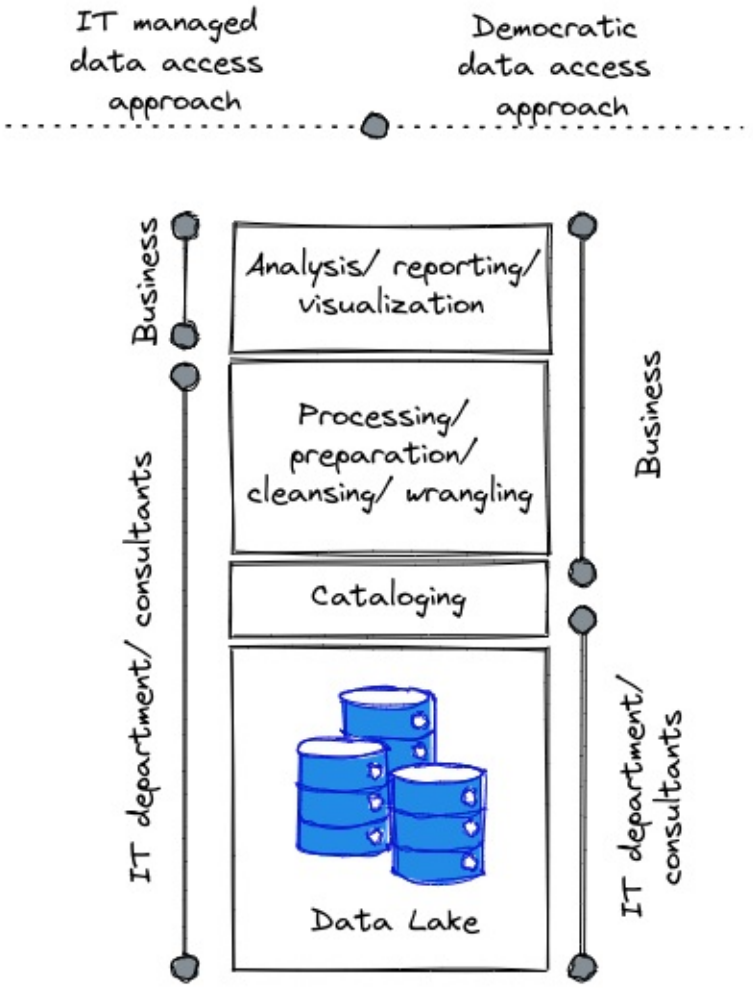


Figure 6-11. Data access approaches (centrally managed vs democratic data access)

Tools like Atlan, Collibra, Informatica, AWS Glue, Azure Purview and Google Dataplex are making the process of metadata collection easier and faster. On one side, a lot of automation has been built to enable data crawling in an automated way, especially thanks to the integration with several database and storage engines (e.g. AWS Redshift and Athena, Azure Synapse, Google BigQuery, Snowflake etc.), and on the other, facilitating data entry activities leveraging rich and easy to use user interfaces and let business people carry out the majority of the work.

Moving up through the chain we find that even the data processing/ preparation/

cleansing and wrangling steps, that have always been for expert users (i.e. data engineers), can be handled directly by the final users. Of course, a subset of the data processing may still need to be fulfilled by data engineers/ data scientists leveraging advanced process engines like Spark. For the activities that require Subject Matter Experts (SMEs), tools like Talend Data Preparation or Trifacta DataPrep have been developed and made available with the clear goal to support non data-eng/ data scientists users: exploration, transformation, and filtering are just a few of the activities that can be achieved leveraging a very intuitive interface that delegates the processing to an underlying engine (e.g. Beam) that can apply all the mutations to vast data sets. This approach is even emphasized by the fact that the access to the underlying data, that sits on HDFS cluster or directly on the object storage like AWS S3, Azure Data Lake or Google Cloud Storage, can be achieved via several different tools offering a broad set of features and controls. This means that different kinds of users can leverage different kinds of tools to deal with the data. Data Engineers, for example, can leverage Spark to develop their data transformation pipelines, data scientists can implement ML algorithms using SciKit learn, Tensorflow or PyTorch, and business analysts can use Hive or PrestoSQL to execute what-if analysis using SQL queries.

The last step is the data visualization/ reporting, which is typically the easiest for business users to carry out themselves. Here there are a ton of solutions that can be leveraged (e.g. Microsoft PowerBI, Looker, Tableau, Qlik just to name a few) that give the users the flexibility they need. Users tend to be naturally autonomous at this stage because the tools are similar in the approach, to some extent, to what they are used to with Excel, so users don’t have a steep learning curve to become proficient. Data visualization, BI and what-if scenarios are the typical analysis that business users can easily carry out.

Even if the raw data is always managed by the IT department, there could be some situations where, thanks to the integrations with third party services, business users can ingest data into the data lake in an autonomous way. Because of this, there is a growing need for trust concerning data content and related quality and correctness of datasets ingested by different business units. The concept of *stewardship* is gaining traction to help with this. Stewardship, which is the process of managing and overseeing an organization’s data assets to ensure that business users have access to high-quality, consistent, and easily accessible

data, can be seen as a combination of three key factors as illustrated in Figure 6-12:

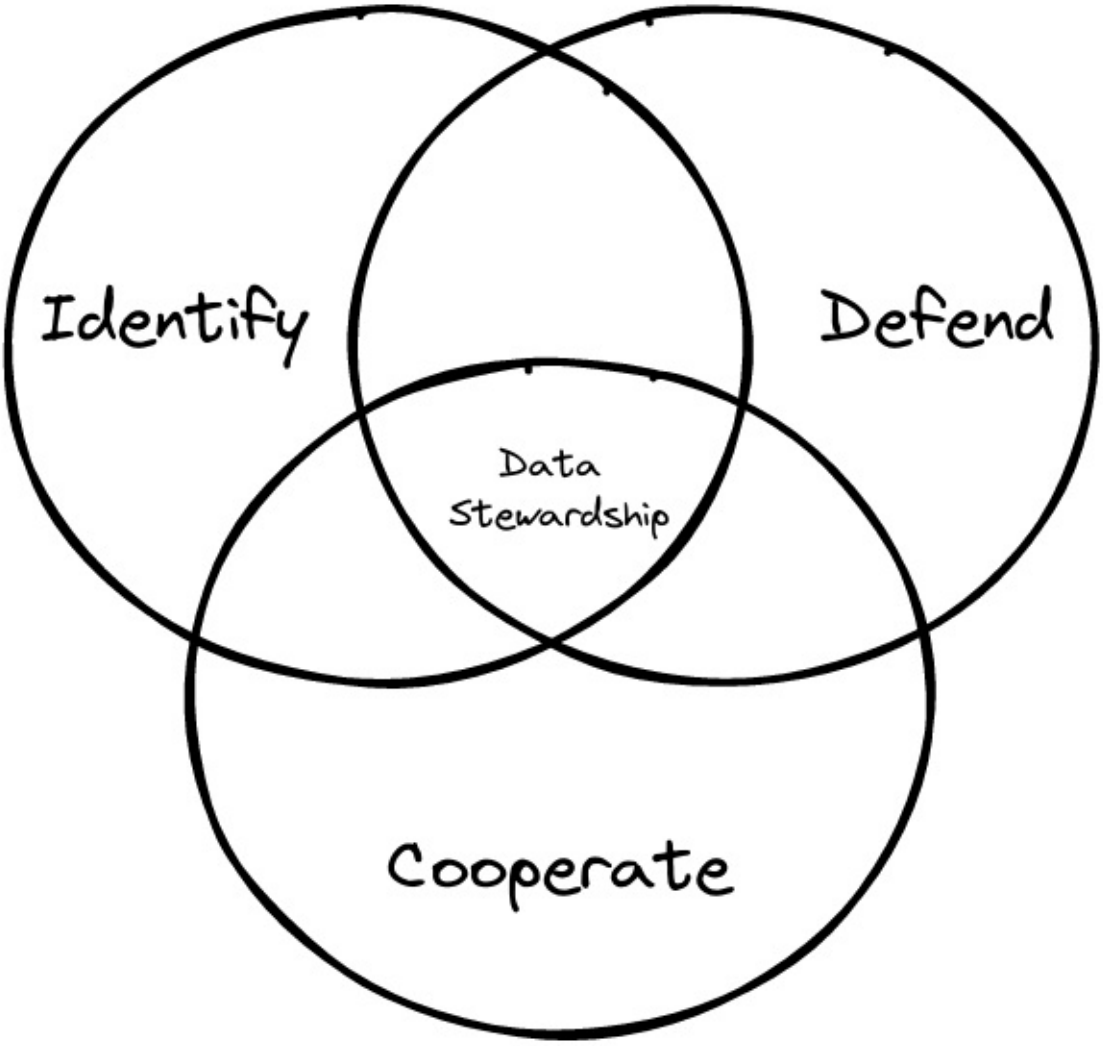


Figure 6-12. Data stewardship

- *Identify* the key stakeholders who have the right information on a timely basis
- *Defend* the data from any kind of exfiltration both internal and external with a focus on personnel
- *Cooperate* with other people inside or outside the company to unlock the value of data

What we are seeing in many companies is that stewardship is not necessarily a role that people are assigned to, but it is more a title that they earn on the field

thanks to their interactions with the tools and the quality of their information that they provide to the internal community. In fact there are several tools (like Informatica Data Catalog for example) that allows users to rate stewards in the same way purchasers can rate sellers on Amazon.

Now that you've seen the various options available to bring the organization toward a more modern and democratic approach, let's discuss the part of the process that is still mainly in the hands of the IT team: the data ingestion.

Data ingestion is still an IT matter

One of the most critical and important steps in a data lake is the data ingestion process (the base of the chart sketched in Figure 6-11). If poorly planned, the data lake may become a "*data swamp*" with a lot of unused data. This typically occurs because organizations tend to load every kind of RAW data into the data lake, even without having a complete understanding of *why* they loaded the data or *how* they will leverage it. This approach will bring the creation of massive *heaps of data* that will be unused for the majority of the time. But even unused data remains in the lake and uses space that should be free for other activities. Stale data also tends to have gaps and inconsistencies, and causes hard-to-troubleshoot errors when it is eventually used by someone. Because of that it is important to follow the following best practices, which you can leverage during ingestion processes:

- *File format*: there are several file formats that can be leveraged that come with pros and cons. If the primary goal is readability CSV (Comma Separated Value), JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are the most common ones. If instead performance is the most relevant goal then Avro, Parquet or ORC are the ones that suit better.
    - Avro (row-based format) works better when intense I/O operations are required (e.g. event based source)
    - Parquet and ORC (columnar based format) are more suitable for query use case
- *File size*: typically in the Hadoop world *larger is better*. Files usually have a

size of dozens of GB and when working with very small files (e.g. IoT use case scenario), it is a good idea to leverage a streaming engine (such as Apache Beam or Spark Streaming) to consolidate the data into a few large files.

- *Data compression*: Data compression is incredibly important to save space, especially in data lakes that are potentially ingesting petabytes of data every single day. Additionally, in order to guarantee performance it is crucial to select a compression algorithm that is fast enough to compress/ decompress the data on the fly, saving a decent amount of space. The standard ZIP compression algorithm is not generally suitable for such applications and, what we are seeing, is that organizations tend to leverage Snappy, an open source algorithm developed by Google that provides performance aligned with data lakes' needs.
- *Directory structure*: this is a very important topic because based on the use case and on the process engine to be leveraged your directory structure can vary a lot. Let's take as an example an IoT use case handled via HBase: imagine you are collecting data from devices globally distributed and you want to extract information of a specific day from messages generated by devices in a specific location. One good example of directory structure might be */country/city/device_id/year/month/day/hours/minute/second*. If instead the use case is a batch operation it may be useful to put the input data in a folder called IN and the output data in a folder called OUT (of course it has to be identified with the best possible prefix to avoid misunderstanding and data duplication).

Depending on the nature of the data (batch vs streaming) and the target of the data (e.g. AWS S3, Azure Lake Storage, Google Cloud Storage just to name a few) it will be possible to leverage several solutions: users can load the data leveraging scripts (running in Powershell or Bash for example) or they can ingest the data directly using APIs or native connectors. Users can stream the data directly into the platform (e.g. AWS Kinesis, Azure EventHub or Google Pub/Sub) or they can upload it as a RAW file into HDFS for future processing. IT departments usually manage these activities because they can involve a lot of automation and at the same time a lot of integration/ data processing that require ad-hoc skills. However, there are solutions like Fivetran that are simplifying the

way users can configure data ingestion into the cloud. For such a solution even less skilled people (E.g. business users) could potentially load data in the lake extending the democratization concept we discussed earlier.

Now that you have seen how to democratize access to the data, let's see from a high level view how data lakes can facilitate the training and prediction of ML algorithms. You will learn more about that in Chapter 12.

Machine Learning in the Data Lake

As we discussed before, it is possible to store any type of data in its raw format in a data lake — this is unlike a data warehouse where data needs to be structured or semi-structured. In particular, it is possible to store unstructured data (images, videos, natural language, etc.) in their typical formats (JPEG, MPEG, PDF, etc.). This makes it extremely convenient to ingest data of all types into the data lake, which can then be used to develop, train and predict ML algorithms especially the ones based on deep learning (refer to *Chapter 1 - Applying AI*).

Training on Raw data

The typical ML framework you would use varies based on the type of data. For structured data you can leverage libraries like Spark, xgboost, and LightGBM. These frameworks can directly read and process data in CSV files, which can be stored as-is in a data lake. For unstructured data, the most commonly used ML frameworks are TensorFlow and PyTorch. These frameworks can read most image and video formats in their native form, which is the form that the raw data will be stored in. Thus, in Figure 6-13, where we walk through the steps in training ML models, the prepared data can be identical to the collected and stored data, and the labels can be part of the data itself — either a column in the CSV files, or based on how images/ videos are organized (for example, all images of screws can be stored in a folder named "screws"). This makes training ML models on a data lake extremely straightforward.

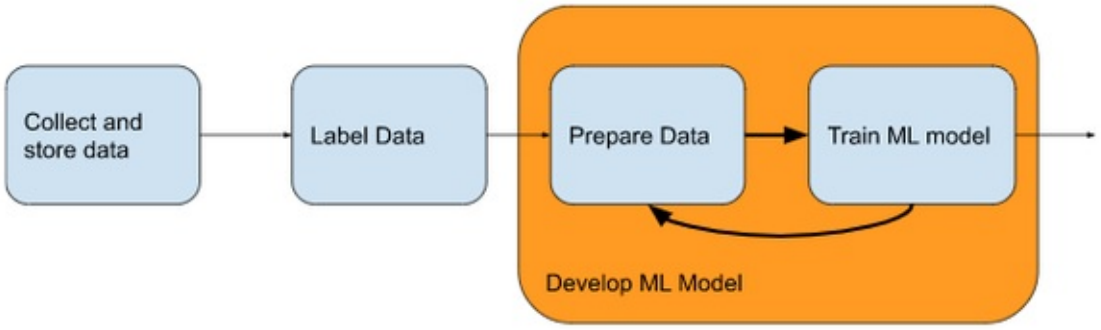


Figure 6-13. Steps of Training ML models

There are some efficiency considerations, however — directly reading JPEG or MPEG files will lead to the ML training programs being I/O bound. Therefore, the data is often extracted and stored in formats such as TensorFlow Records in the data lake. Because these formats optimize for throughput when reading from cloud storage, they help maximize GPU utilization. GPU manufacturers also provide capabilities to read common formats, such as Apache Arrow, directly from cloud storage into GPU memory, thus speeding up the ML process (see Figure 6-14).

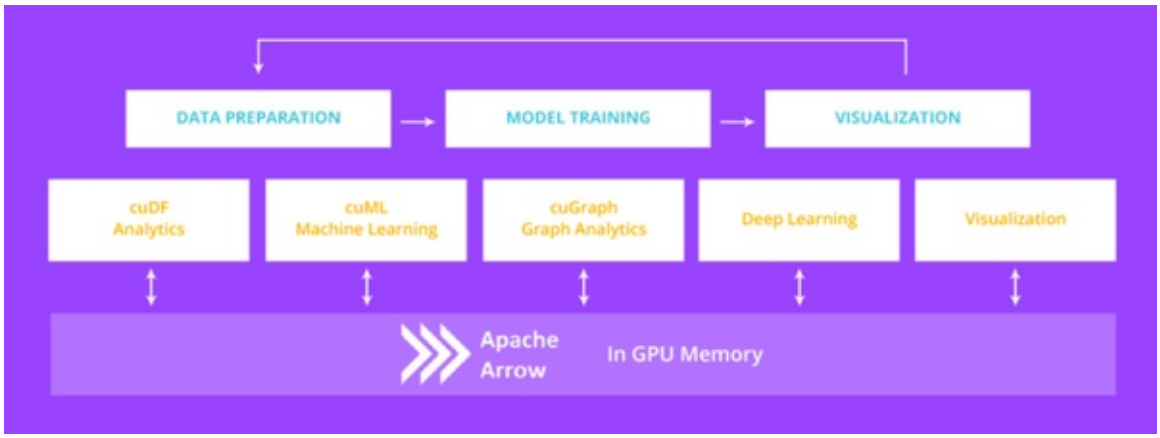


Figure 6-14. In-GPU ML training off a data lake. Image by Nvidia

Predicting in the Data Lake

Because the ML frameworks support directly reading data in the raw formats, invoking models on the raw data is also very straightforward. For example, in Figure 6-15, we show you an image classification workflow on AWS. Note how the image is ingested as-is to the cloud storage bucket, and the ML model is invoked on the uploaded image. Similar capabilities are available in GCP and

Azure as well, which we'll cover more extensively in Chapter 12.

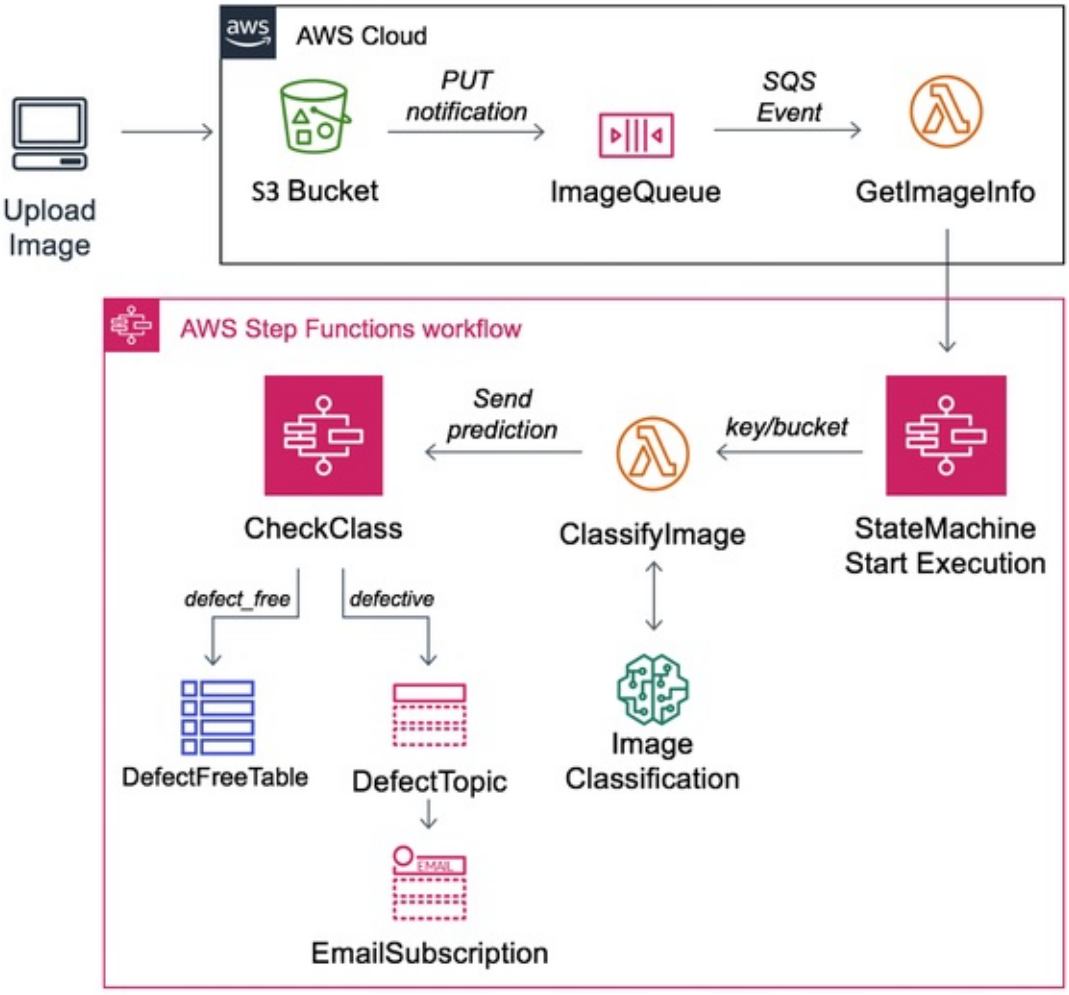


Figure 6-15. ML Inference carried out on an image uploaded into a data lake on AWS. Image from AWS documentation.

If choosing a data lake as a primary platform, also consider using MLFlow, an open-source project, to implement end-to-end machine learning workflows. The data are stored in the data lake, and distributed training is typically carried out using Spark, although integrations with Tensorflow and Pytorch also exist. Besides training and deployment, it also supports advanced features such as Model Registries (See Figure 6-16) and Feature Stores.

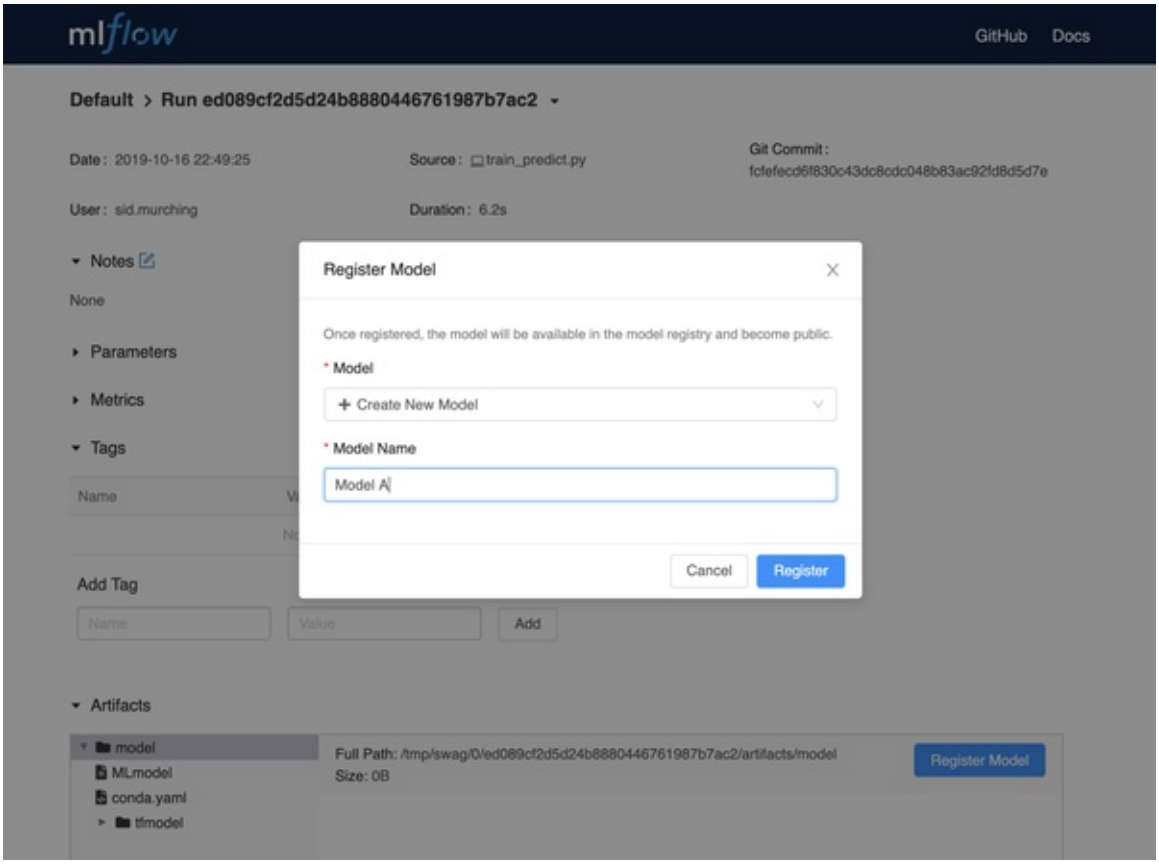


Figure 6-16. MLFlow is an end-to-end platform for machine learning on data lakes. Image from MLFlow documentation.

This capability to directly process raw data in ML frameworks is one of the compelling advantages of data lakes.

Summary

In this chapter you have seen a little more in detail what a real data lake is, what the challenges are, and the patterns that you can adopt to make it the core pillar of the data platform of an organization. The key takeaways are:

- Data plays a key role in every organization to help them make robust decisions within a short time frame and possibly in real-time.
- Companies are mainly adopting two approaches to handling their data: Data Warehousing and/ or Data Lake
- Data lake provides more flexibility because it allows users to play with any kind of data (i.e. structured, semi-structured and even unstructured) and to

leverage a wide variety of solutions and tools coming from the open source ecosystems.

- Organizations started leveraging a lot of the data lake paradigm but because of the explosion of the volume/ velocity and variety they struggled in managing everything in old fashioned on-premises environments.
- Organizations need a new place to store all types of data and cloud environments are a great solution because they allow organizations to first reduce the TCO and then build a platform for the innovation because people within the company can focus on the business value instead of the hardware management.
- The main advantages to cloud adoption are: 1) reduce the TCO via the ability to save money on storage and compute; 2) Handle (theoretical) infinite scalability leveraging the power of hyperscalers; 3) Drive fail fast approach speeding up experimentations thanks to the elasticity of the underlying platform; 4) Adopt a consistent data governance approach in the platform across several products to guarantee security, control and adherence to the compliance requirements.
- The market is strongly investing in data lakes, particularly in those that are cloud based.
- In data lakes there is a repository of all the metadata that describe the datasets of the organization, which will help data scientists, data engineers, business analysts, and any users who may need to leverage data, to find a path toward the desired dataset.
- Data lakes enable batch and streaming operations and facilitate the implementation of lambda/ kappa patterns.
- Data lake can be extended leveraging APIs or integration with third party solutions.
- One common integration is the adoption of Delta Lake on top of HDFS to make the storage ACID compliant and enable other kinds of use cases (mainly those that request a strong consistency in data).
- Jupyter Notebooks are the standard approach used to implement a data

analysis, experimentation and test-based approach within the organizations.

- Data lakes facilitate data democratization access within the organization because the majority of the activity can be carried out on a self-service basis.
- Because data is stored in native formats in a data lake, and because ML frameworks support reading these formats, data lakes facilitate machine learning without any special hooks.

In the following chapter we will adopt a similar approach but for another important component that is the data warehouse.

---

1 Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning's Book by James Warren and Nathan Marz.

2 serverless, interactive analytics service built on open-source frameworks, supporting open-table and file formats

3 Fully managed datawarehouse solution

4 *https://aws.amazon.com/lake-formation/*

5 *https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/azure-purview-data-lake-estate-architecture*

Chapter 7. Innovate with an enterprise data warehouse

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the seventh chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

In the previous chapter we focused on the concept of data lake as a central element in the design of a data platform. In this chapter, you will focus instead on how to approach the same problem but putting at the center a modern data warehouse solution. Data warehouses have been the main data system for many businesses that use them to gain insights from their data. For these organizations, a data modernization journey involves modernizing this central system to make insights available to decision-makers more quickly. Therefore, it is important for you to be familiar with concepts like hub-and-spoke architecture, which are crucial in such modernization journeys. In this chapter you will learn how to land data into the data warehouse through data processing tools or connectors. You will also learn how to provide access to reporting tools and AI/ML systems.

A modern data platform

Whenever you undertake a large technology project, you should first ask yourself

- What business goals are you trying to accomplish?
- What are your current technology challenges?
- What technology trends do you want to leverage?

In this section we will focus on helping you understand how to address these questions when building a modern data platform and how an enterprise data warehouse approach can steer the focus in your data platform design. You may see that some concepts have already been touched in the previous chapters but it is useful to recall them here to help you navigate through the following content and help you connect the various topics.

Organizational goals

In our customer interviews, CTOs repeatedly raised these organization goals as being important:

- **No silos.** Data has to be activated across the entire enterprise because users in one part of the business need access to data that other departments create. For example, a product manager determining how to design next year's product might need access to transaction data created and managed by the retail operations team.
- **Democratization**. The data platform has to support domain experts and other non-technical users who can access the data without going through technical intermediaries but should be able to rely on the quality and consistency of the data.
- **Discoverability**. The data platform has to support data engineers and other technical users who need access to data at different stages of processing. For example, if we have a dataset in which raw, incoming transactions have been reconciled, a data scientist needs to be able to get the reconciled dataset. If they can't discover it, they will rebuild a reconciliation routine. Therefore, it should be possible to discover all these "intermediate" datasets, so that processing steps are not duplicated across the organization.
- **Stewardship**. Datasets should be under the control of teams that understand what they are. For example, financial data should be under the control of

the Finance Department, not of Information Technology (IT).

- **Single-source**. Data should be read-in-place. Minimize the copying and extracting of data.
- **Security**. IT should serve as a broker for data, ensuring that only people with the right permissions can access it.
- **Ease-of-use.** Make reporting easier since there are hundreds of analysts building reports to support a variety of functions.
- **Innovation**. Make data science teams more productive since these roles tend to be expensive and difficult to hire.
- **Agility**. Make insights available to decision-makers faster.

While the relative order of these goals varies between organizations, all of these goals figure in one or another in every organization we spoke to. Therefore, a modern data platform should enable CTOs to achieve these goals.

Technological challenges

What prevents CTOs from accomplishing these goals with the technologies that they have already deployed within the organization? CTOs tend to mention these technological challenges:

- The quantity of data their organization is collecting has dramatically increased over time and is expected to continue to increase. Their current systems are unable to scale and remain within the speed and cost constraints of their business, leading to compromises such as sampling the incoming data or heavily prioritizing new data projects.
- Increasingly, the data being collected is unstructured data – images, videos, and natural language text. Their current systems for managing structured and unstructured data do not intersect. However, there is increasingly a need to use structured (e.g. product catalog details) and unstructured data (e.g. catalog images, user reviews) together in use cases such as recommendations.
- Over time, we have seen the availability of many new sources and sinks of

data to the technology landscape that organizations should and want to leverage. For example, they would love to manage sales information in Salesforce, ads campaigns in Adobe, and web traffic in Google Analytics. There is a need to analyze and make decisions on all this data in tandem.

- Much of the new data is collected in real-time, and there is a competitive advantage to being able to process and make decisions on the data as it arrives. However, organizations do not have a data platform that seamlessly supports streaming.

These are, of course, just a longer elucidation of the traditional Big Data challenges — volume, variety (of data and systems), and velocity.

Technology trends and tools

To enable their organization to achieve these organization and technological goals, a cloud architect can leverage the trends and tools described in the previous chapters. For convenience, they are summarized below:

- **Public cloud providers provide the ability to separate compute and storage**. Data can be stored on blob storage and accessed from ephemeral computational resources. These computational resources consist of software such as Google BigQuery, AWS Redshift, Snowflake, AWS EMR, Google Dataproc, Cloud Dataflow, or Databricks that were custom-built or adapted to take advantage of this separation of computing and distribute data processing over multiple workers. They span both structured data and unstructured data.
- **Cloud computing resources are built to allow multiple tenants.** Therefore, there is no need to create distinct clusters or storage arrays for each department in an organization. Thus, two separate departments can store their data in a single data warehouse and access each other's data from compute resources that they each pay for – each team can spin up their own analytics on the common, shared dataset. Similarly, an organization can use its computing resources to access data from multiple organizations and do analytics on the joined datasets. Unlike traditional Hadoop clusters, it is not necessary to run the compute workload collocated with the data.

- **Cloud Identity and Access Management (IAM) solutions can ensure that a central IT team can secure identities while the owners of the data control access**. In fact, by providing access to groups, it is possible to allow the accessing organization to manage the membership while the data owners manage only business logic of which teams are provided what access. For example, let's think about a department X within an organization that creates a group (let's call it group X) and enables a set of access patterns for anyone in that group. Membership of individuals in that group is the only thing that the business / data owners need to govern.
- **Serverless data warehouses** (and data lakes as we have seen in the previous chapter) allow the architect to **break down data silos even outside the boundaries of the organization** (i.e. leveraging the group concept we mentioned before where people in the group do not need to belong to the same department). Thus, partners and suppliers can share data without having to worry about querying costs or group memberships.
- Tools such as Informatica or Looker make it possible to create a **semantic layer that stretches across hyperscalers** (such as AWS, GCP or Azure), multi-cloud data platforms (such as Snowflake or Databricks), and on-premises environments. The semantic layer can rewrite queries on the fly to provide a consistent and coherent data dictionary (please note that semantic layers are covered in more detail later in the chapter).
- **Data fabric solutions provide a consistent administration experience on the public cloud** no matter where the data is stored, whether in a data warehouse or in data lake formats such as Parquet files on blob storage.
- Tools such as BigQuery Omni provide a **consistent control pane and query layer regardless of which hyperscaler** (AWS, GCP, or Azure) your organization uses to store the data. These are useful if the concern is to ensure that the same tooling can be used regardless of which hyperscaler's storage a particular dataset lives in. The tradeoff is an increased dependency on the GCP control pane.
- **Multi-cloud platforms** such as Snowflake and Databricks provide for the ability to **run the same software on any hyperscaler**. However, unlike in the previous bullet point, the runtimes on different clouds remain distinct.

These are useful if the concern is to ensure that it is possible to move from one hyperscaler to another. The tradeoff is an increased dependency on the single-source software vendor.

- **Data lakes and data warehouses are converging** (we will deep dive on this topic in the next chapter), with federated and external queries making it possible to run Spark jobs on data in the data warehouse and SQL queries on data in the data lake.
- **Enterprise data warehouses** like AWS Redshfit and Google BigQuery **support built-in machine learning** without having to move data out of the data warehouse. Spark has a machine learning library (Spark Mlib) and machine learning frameworks such as TensorFlow are supported in Hadoop systems. Thus, machine learning can be carried out on the data platform without having to extract data to take to a separate ML platform.
- Data warehouses support **streaming SQL**, so that as long as data is landed in the data warehouse in real-time, queries reflect the latest data.
- Tools such as Kafka, AWS Kinesis, Azure EventHub and Google Cloud Pub/Sub, support the ability to **land data into hyperscalers' data platforms in real-time**. Tools such as AWS Lambda, Azure Functions, Google Cloud Run, and Google Cloud Dataflow also support transforming the data as it arrives so as to quality-control, aggregate, or semantically correct data before it is written out.
- Tools such as Qlik, AWS Database Migration Service, and Google Data Stream provide the **ability to capture changes to an operational relational database** (such as Postgres running on AWS RDS or MySQL running on Google Cloud SQL) **and stream them in real-time** to a data warehouse.
- It is possible to use **modern visualization tools** such as PowerBI to **embed analytics into the tools** (mobile phones or websites) that end-users use – it is not necessary to make end-users operate dashboards.

Given these goals and the technological capabilities, there are several architectures that will work. In this chapter, we will discuss the *Hub-and-Spoke architecture*.

Hub-and-Spoke architecture

When leading with a data warehouse as the central core of a modern cloud data platform, the Hub-and-spoke is, in some ways, the ideal architecture. In this architecture, the data warehouse acts as a *hub* that collects all the data needed for business analysis. *Spokes*, which are custom applications, dashboards, ML models, recommendation systems, and so on, interact with the data warehouse via standard interfaces (e.g., APIs). All of these spokes can access data directly from the data warehouse without having to make a copy.

We suggest this approach to startups with no legacy technologies to accommodate, organizations that want a complete do-over to achieve a full-scale transformation, and even large enterprises because it is scalable, flexible, and resilient.

It is scalable because new (modern) data warehouses can easily integrate new data sources and use cases to the existing infrastructure without having to reconfigure the entire system. It is flexible because you can customize the overall architecture to meet the specific needs of an enterprise (e.g., enabling streaming). And it is resilient because it can withstand more failures than other architectures.

A modern cloud-native enterprise data warehouse forms the hub of the Hub-and-spoke architecture and the spokes are either data providers and data consumers as described in Figure 7-1 (*please note that we recall the different components of the diagram in the below description to facilitate your understanding*).

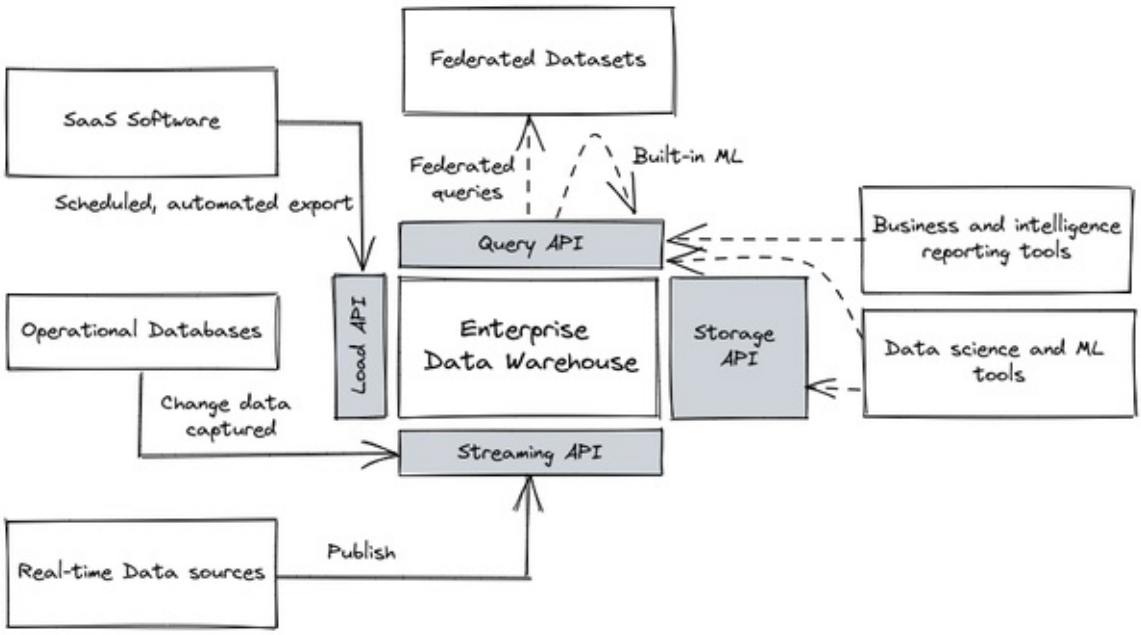


Figure 7-1. Hub-and-spoke architecture. A modern enterprise data warehouse forms the Hub.

A business-analyst-first data and ML capability involves loading raw data into a data warehouse (*Enterprise Data Warehouse*) and then carrying out transformations as needed to support various needs. Because of the separation of compute and storage, there is only one unique copy of the data. Different compute workloads (querying (*Query API*), reporting/ interactive dashboards (*Data science and ML tools)*, machine learning (*Data science and ML tools)*) work on top of this data that sits in the data warehouse. Because you can leverage SQL for all the transformations, you can use views and materialized views to carry out elaborations, making ETL pipelines unnecessary. These views can invoke external functions, thus allowing for the enrichment of data in the data warehouse using machine learning APIs and models. In some cases, you can even train ML models using just SQL syntax and you can schedule complex batch pipelines via simple SQL commands. Modern data warehouses support directly ingesting (*Load API*) even streaming data (*Streaming API*), and so you have to leverage on streaming pipelines only when you need to perform low-latency, windowed aggregations analysis on data as it comes in.

The key idea behind the hub-and-spoke architecture is that you land all the data into the enterprise data warehouse as efficiently as possible. When data is coming from *Software As a Service (SaaS)* software (such as Salesforce) you can load it through a scheduled, automated export mechanism. When instead it is

coming from *operational databases* such as Postgres, you can land it in near real-time through *Change Data Capture (CDC)* tools. Meanwhile, *real-time data sources* are expected to publish new data to the data warehouse when new events happen. Some data sources are federated (*Federated Datasets)*, which means that you will dynamically query and treat them as part of the enterprise data warehouse itself. With all the enterprise data now logically part of the data warehouse, data science, and reporting tools can act on the data (*Storage API/ Query API)*.

If you don't have pre-existing ETL pipelines that you need to port or preexisting end-users whose tool choices you have to support, the hub-and-spoke is simple, powerful, fast, and cost-effective. An example manifestation of this architecture on Google Cloud is shown in Figure 7-2.

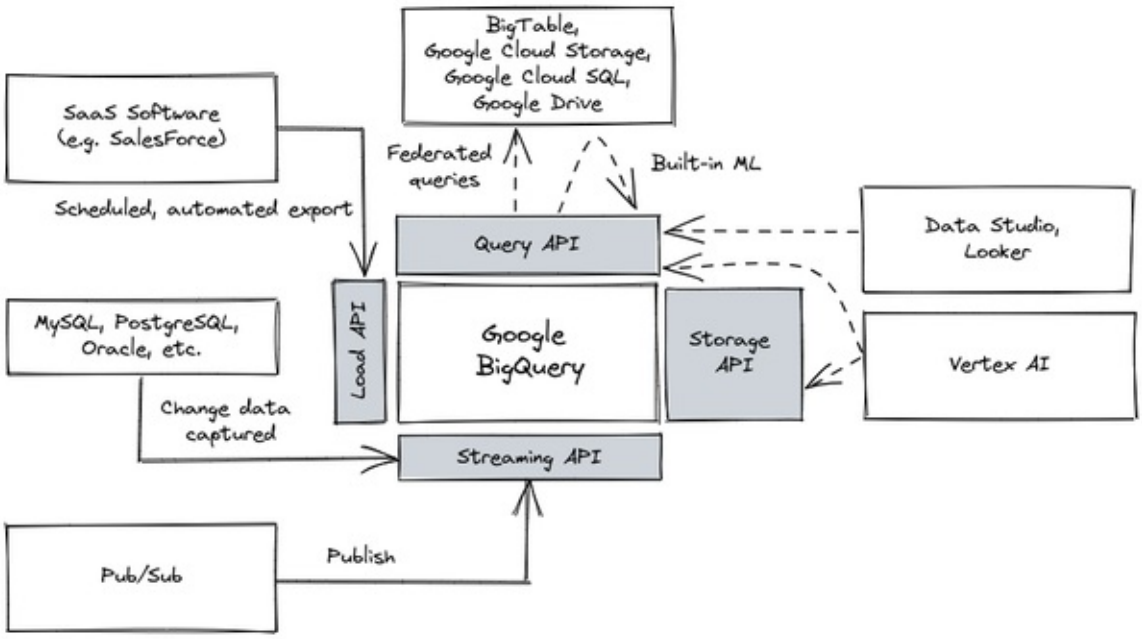


Figure 7-2. Hub-and-spoke architecture as manifested on Google Cloud

In its fully automated form, the hub-and-spoke is a great choice for enterprises without strong engineering skills and for small departments doing shadow IT, because there is very little code to maintain when it comes to data ingestion. Also, you can complete data science and reporting activities with only a knowledge of SQL.

It is worth noting that the capabilities that enable an analyst-first data warehouse are all recent developments. Separation of compute and storage came about with

the public cloud: previously, the BigData paradigm was dominated by MapReduce which required sharding data to local storage. Analytics workloads required the building of Data Marts that were business specific. Due to performance constraints, it was needed to carry out transformations before loading the data into these data marts (i.e. ETL). Stored procedures were data warehouses, not external functions which themselves rely on developments in autoscaling, stateless microservices. ML deployments required bundling and distributing libraries, not stateless ML APIs. ML workflows were based on self-contained training code, not ML pipelines consisting of containerized components. Streaming involved separate codebases, not unified batch and stream processing.

Now that you have seen the basic concepts of the hub and spoke architecture, let's deep dive into its main components: ingest, business intelligence, transformations, and ML.

Data ingest

One set of spokes in the hub-and-spoke architecture (the gray boxes gravitating around the enterprise data warehouse in Figure 7-1) corresponds to various ways to land (or ingest) data into the data warehouse. There are three ingest mechanisms: pre-built connectors, real-time data capture, and federated querying. We will look at each of these respectively, in this section.

Pre-built connectors

Landing data into the data warehouse can be extremely easy when leveraging popular SaaS platforms because they are making available connectors that ingest data with few clicks. Every cloud-native data warehouse (Google BigQuery, AWS Redshift, Snowflake, Azure Synapse, etc.) typically supports SaaS software such as Salesforce, Google Marketing Platform, and Marketo. You can directly import these datasets into the data warehouse using services like the BigQuery Data Transfer Service.

If you happen to be using a software that your choice data warehouse doesn't support, look at whether the software vendor provides a connector to your desired data warehouse – for example, Firebase (a mobile applications platform) can directly export crash reports from mobile applications into BigQuery for

analytics (“crashlytics”).

A third option is to use a third-party provider of connectors such as Fivetran. Their pre-built connectors provide a turn-key ability to integrate data from marketing, product, sales, finance, and other applications (see Figure 7-3).

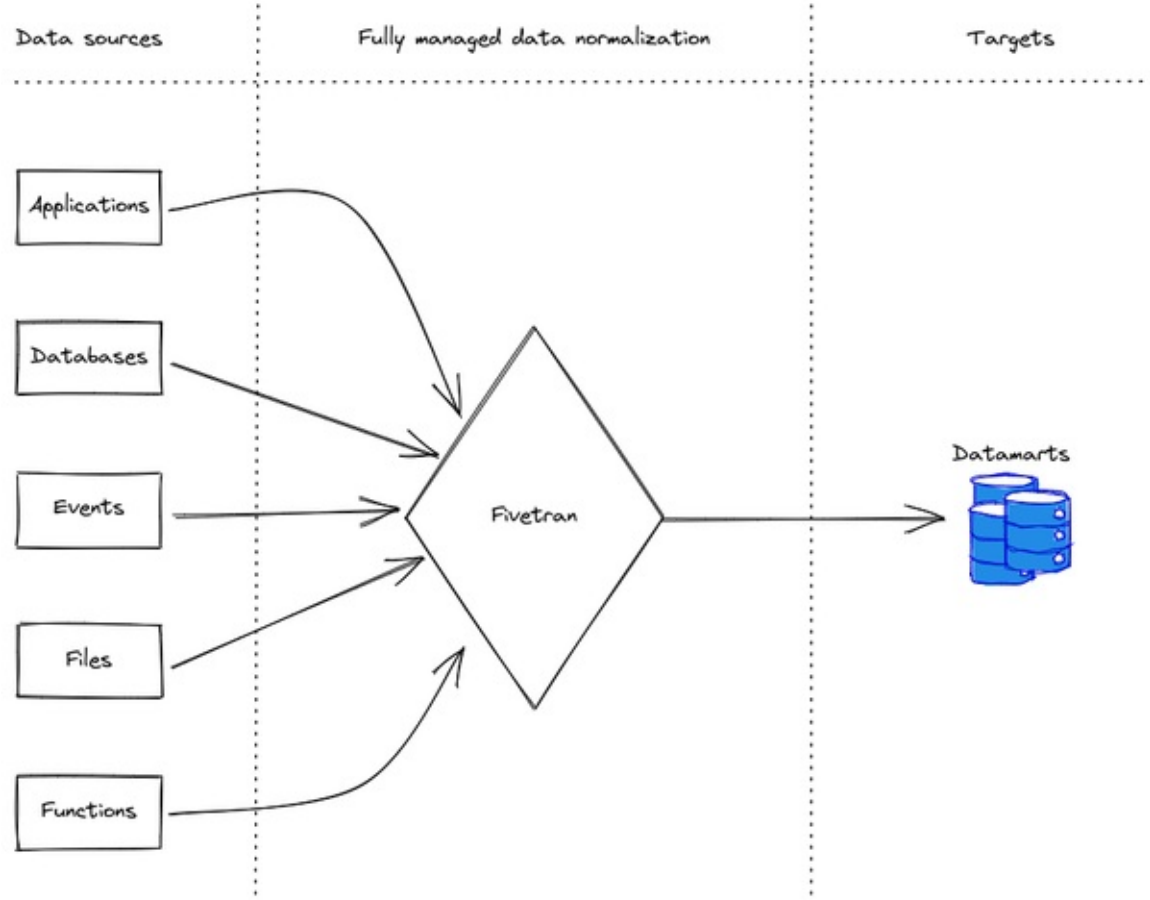


Figure 7-3. Third-party vendors such as Fivetran can automatically handle landing data from a wide range of sources into a cloud data warehouse such as BigQuery, Redshift, or Snowflake. Diagram by Fivetran.

Between the transfer services of the cloud provider, software vendors that support cloud connectors, and third-party connector providers, you can buy (rather than build) the ability to export data routinely from your SaaS systems and load them to the enterprise data warehouse.

Real-time data

What if you want your data warehouse to reflect changes as they happen? You need to leverage a smaller set of tools called Change Data Capture (CDC) tools. Operational databases (Oracle, MySQL, Postgres) are typically part of the

support as are enterprise resource planning (ERP) tools like SAP. Make sure that these tools use the Streaming API of the data warehouse to load data in near real-time. On Google Cloud, Datastream is the recommended CDC tool, and on AWS, it is Data Migration Service (DMS).

If you have real-time data sources, such as clickstream data or data from IoT devices, look for a capability to publish the events as they happen using the Streaming API of the data warehouse. Because the Streaming API can be accessed through HTTPS, all you need is a way to invoke an HTTPS service every time an event happens.

If the provider of your IoT devices doesn't support push notifications, then look for a way to publish events into a message queue (for example, using MQTT) and use a stream processor (Dataflow on GCP, Kinesis on AWS) to write those events into the data warehouse (see Figure 7-4).

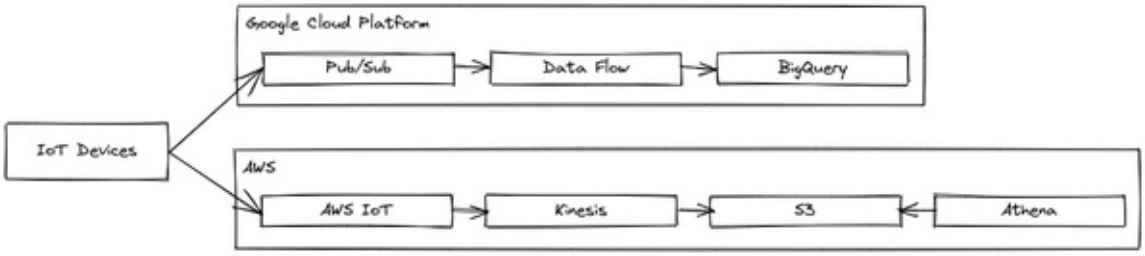


Figure 7-4. Landing real-time data from an IoT device into a data warehouse. Top: on Google Cloud. Bottom: On AWS.

Federated data

You do not need to land into the data warehouse all possible data. Modern cloud data warehouses are able to run queries on datasets in standard formats such as Avro, CSV, Parquet, and JSON-L without moving the data into the data warehouse. These are called *federated* queries and often require either that the data format be self-describing or that the schema be pre-specified. For example, getting Google BigQuery to perform federated queries on Avro files, a self-describing format, involves three steps:

1. Create a table definition file using *bq mkdef --source_format=AVRO gs://filename* and edit the defaults if necessary. For example, you might change a field that, in Avro, is an integer to be treated as a real number.
2. Use the resulting table definition file to create a BigQuery dataset using *bq*

mk --external_table_definition mydataset.mytablename

3. Query the dataset with SQL as normal.

Note that the data remains on Cloud Storage in AVRO format. This is what makes this a federated query. If the data format is not self-describing, the *mkdef* command allows you to specify a schema.

It is possible even to combine these steps and apply a "*schema on read*", so that the schema definition is only for the duration of the query. For example, to have Azure Synapse query Parquet files in an Azure data lake, you can query:

```
SELECT ... FROM
  OPENROWSET(
     BULK 'https://....dfs.core.windows.net/mydata/*.parquet',
     FORMAT='PARQUET'
  ) AS Tbl
```

In the case of federated queries, the querying engine is the data warehouse. It is also possible to use an external querying engine (such as a PostgreSQL relational database) to carry out the queries. Such queries are called *external* queries (see Figure 7-5). For example, to get Amazon Redshift to query a Postgres database follow these three steps:

1. Make sure that your RDS PostgreSQL instance can accept connections from your Amazon Redshift cluster. To do this you need to set up up physical networking and ensure that the Redshift cluster is assigned an IAM role authorized in the PostgreSQL instance.
2. In Redshift, create an external schema using CREATE EXTERNAL SCHEMA FROM POSTGRES passing in the database, schema, host, IAM, and secret.
3. Query the schema as normal.

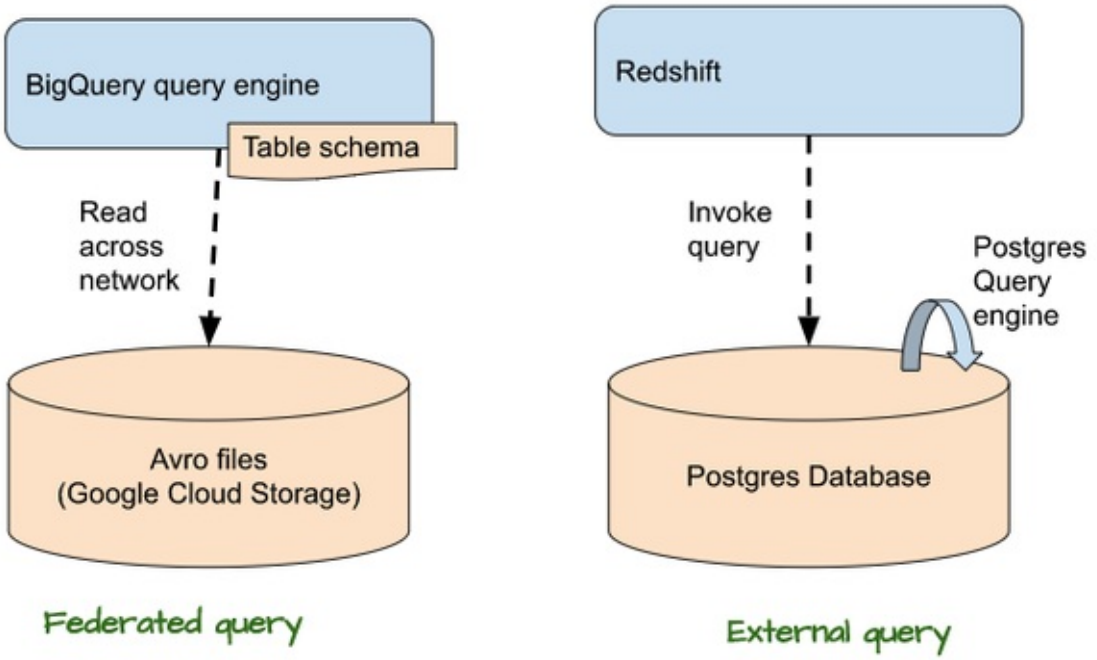


Figure 7-5. In a federated query, the data warehouse's query engine reads data from the federated data source. In an external query, the data warehouse invokes a query that runs within the external relational database. Both GCP and AWS support both types of queries, but an example of each is shown.

In all these instances, the key thing to note is that data remains in place and is queried from there—it is not loaded into the data warehouse. Because the opportunities for optimization are more limited when data remains in place (and can not be partitioned, clustered, etc.), federated and external queries tend to be slower than native queries.

Given that federated and external queries are slower, why use them? Why not simply load the data into the data warehouse and treat the data warehouse as the source of truth? There are a few situations where it can be advantageous to avoid moving the data:

- In some cases, storage within the data warehouse is more expensive than storage outside of it. It may be more cost-effective to keep the data in a federated data source for very rarely queried data.
- If the data is frequently updated in a relational database, it may be advantageous to treat the relational database as the golden source. Doing change-data-capture from the operational database to the data warehouse may introduce unacceptable latency.

- The data may be created or needed by workloads (such as Spark). Because of this, it may be necessary to maintain Parquet files. Using federated/ external queries limits the movement of data. This is the most common use case when you already have a data lake.
- The data may belong to an organization that is different from the one that is querying it. Federation neatly solves the problem. However, suppose you are using a fully serverless data warehouse such as Google BigQuery that is not cluster-based. In that case, it is possible to provide direct access to partners and suppliers even to native storage.

The last situation is one of the reasons that we recommend a fully serverless data warehouse that does not expect you to move data to a cluster, create data extracts, or provide specific applications (rather than specific users) access to the data.

Now that you have a better knowledge of the available options for data ingestion let's deep dive into how to make the data speak by having a look at the various capabilities we can develop on the business intelligence side.

Business intelligence

Data analysts need to be able to rapidly obtain insights from data. The tool they use for this purpose needs to be self-service, support ad-hoc analysis, and provide a degree of trust in the data being used (*Business and reporting intelligence tools* in the hub and spoke architecture). It needs to provide several capabilities: SQL analytics, data visualization, embedded analytics, and a semantic layer, all of which we will cover throughout this section.

SQL analytics

As highlighted in the previous sections, SQL is the main language when querying the data warehouses. SQL is the *lingua franca* of data analysts and business analysts within an organization. These analysts will often carry out ad-hoc queries on the data warehouse to help answer questions that come up (such as, "How many liters of ice cream were sold in Romania during the last heatwave?"). But in many cases, the questions become routine, and users operationalize them in the form of reports leveraging ad hoc tools like PowerBI

or Looker.

These reporting tools, commonly known as *business and intelligence* (BI) tools, aim to provide a view of the entire data estate of the organization by connecting to a data warehouse (*Business and Intelligence reporting tools* in the hub and spoke architecture in Figure 7-1) so that analysts can make data-driven decisions.

Considering that it is not practical to expect an analyst to collect, clean, and load some piece of data at the time that they need it, the data needs to already be there. That's the reason why the hub-and-spoke model with a central enterprise data warehouse is so popular.

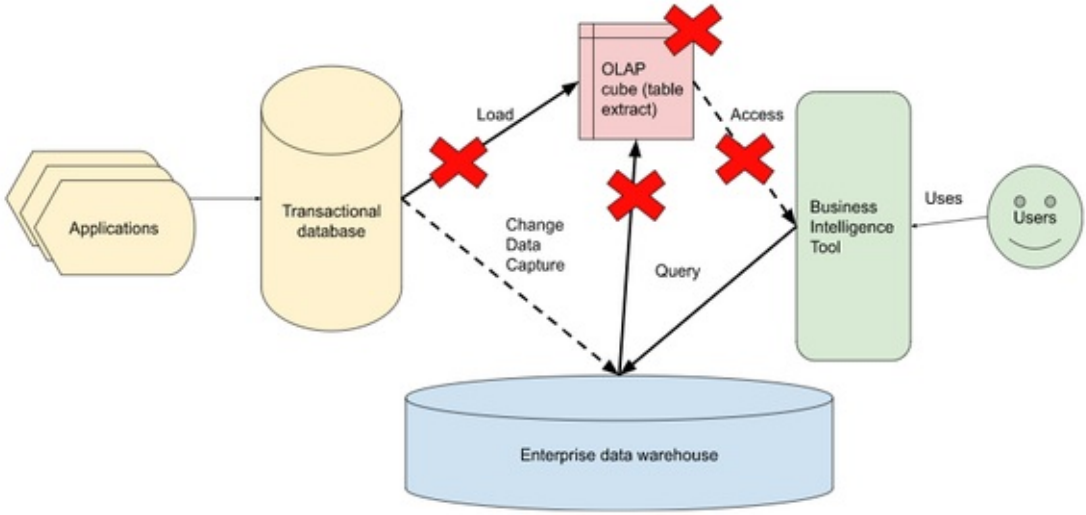


Figure 7-6. Make sure that the BI tool pushes all queries to the enterprise data warehouse and doesn't do them on OLAP cubes (extracts of the database/data warehouse).

However, you should make sure that the BI tool is cloud-ready and capable of dealing with large volumes of fast arriving data. Early generation BI tools would require that the data be extracted into OnLine Analytical Processing (OLAP) cubes for performance reasons (see Figure 7-6). This simply won't do – it leads to a proliferation of stale, exported data or huge maintenance burdens in order to support separate OLAP cubes for every possible use case. You want the BI tool to transparently delegate queries onto the data warehouse and retrieve the results. This is the best way to take advantage of the scale of the data warehouse and the timeliness of its streaming interfaces. SQL engines in the enterprise data warehouse have been upgraded to a level that are able to handle multiple parallel

queries or maintain in memory a huge amount of data that allow organizations to get rid of the OLAP approach.

Visualization

Your SQL queries will produce tables. However, it can be difficult to gain understanding and insights from raw tables alone. Instead, you will often plot your results as graphs, charts, or maps. Visualizing the results of SQL queries is often what leads to insight.

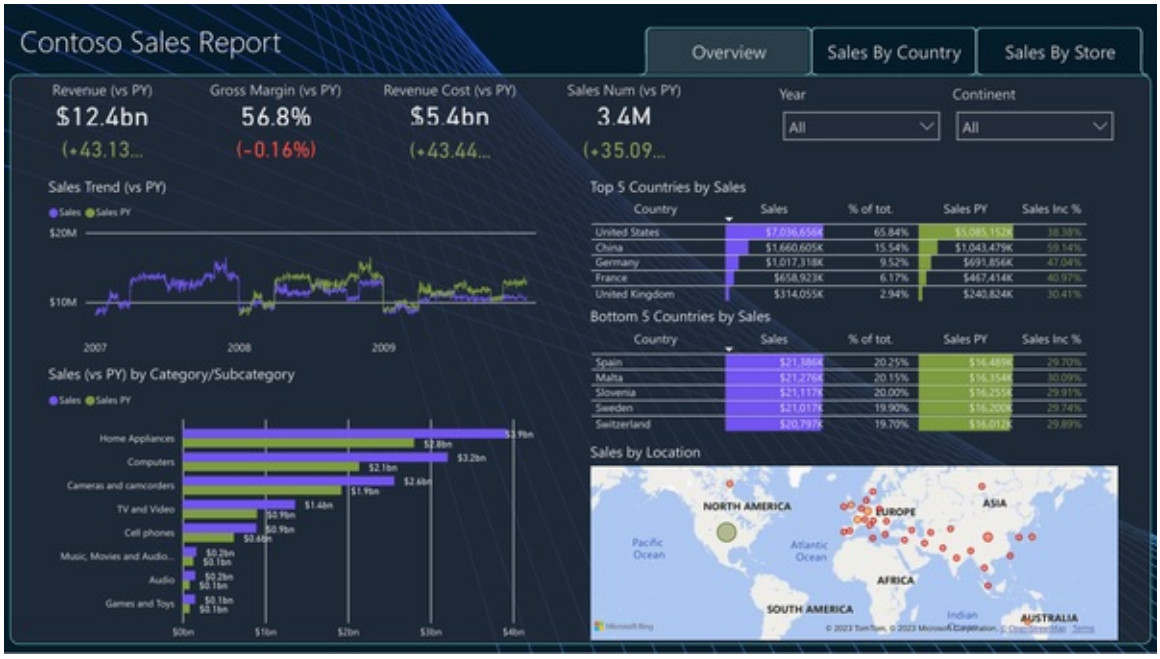


Figure 7-7. An example Power BI dashboard from the Microsoft documentation depicts several important analyses.

In exploratory data analysis, visualization is ad-hoc. However, the visualization has to help frame answers to common questions using common charting elements (see Figure 7-7). This is the remit of dashboarding tools like Tableau, Looker, Power BI, and Looker Studio.

Good dashboards keep the audience in mind and tell stories. They work both as a high-level overview of the current state but also as a launching point for further interactive analysis. They display contextually relevant, important metrics (called Key Performance Indicators or KPIs) using the appropriate form of a chart.

Embedded analytics

Visualization through traditional dashboard tools is sufficient if you wish to share the analytics results with a handful of internal people. Such users will appreciate the rich controls and interactivity that a dashboard provides. What if you are a crafts marketplace or telecom operator, and you want each artist or each kiosk to have access to near-real-time graphs of the performance of their own store (see Figure 7-8)?

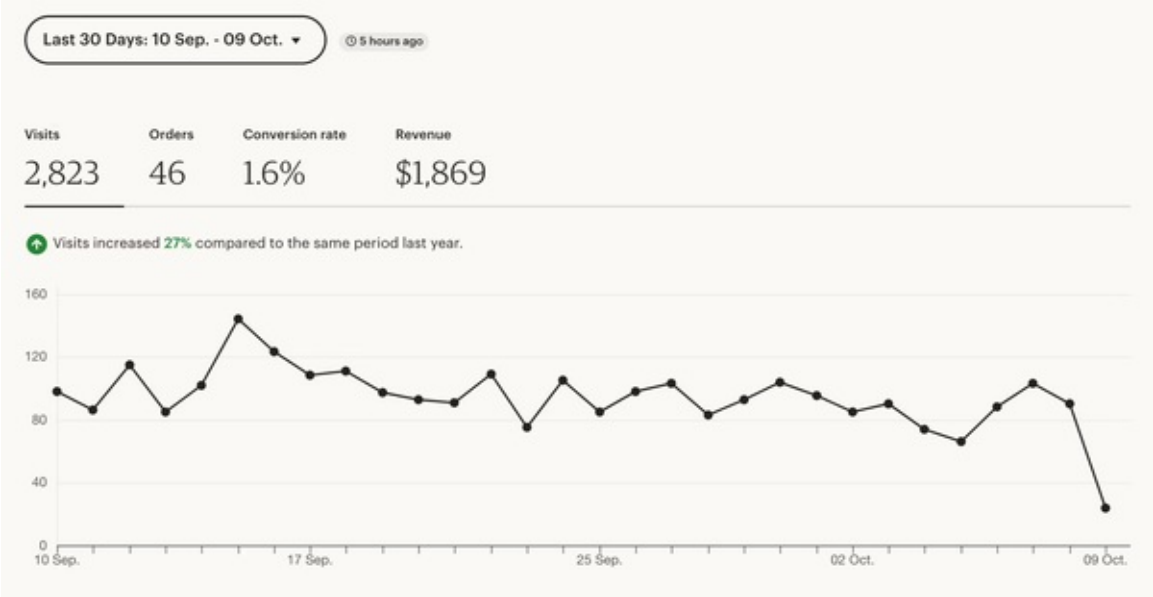


Figure 7-8. Example from Etsy, a crafts marketplace of their Shop Manager. Image from Etsy seller handbook.

When you are providing customized reports to thousands of users, you don't want to provide end-users with a feature-rich dashboard interface that can be hard to support and maintain. Instead, what you need is a lightweight graphics visual layer that can be embedded within the tool that the user is already using. It is common to embed the analytics within the website or mobile app (see Figure 7-9) that artists visit to list items for sale, or that kiosk operations visit to order new items.

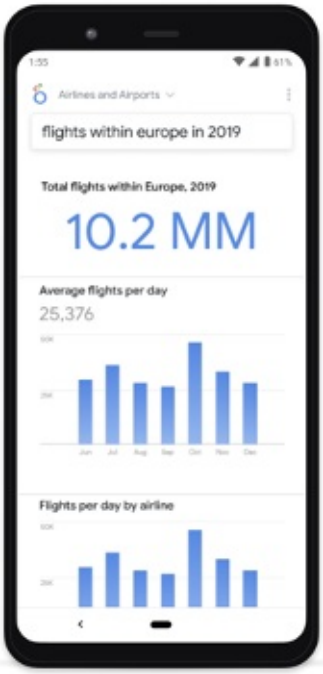


Figure 7-9. Examples of Looker embedded analytics: an example of embedding analytics as a result of exploration within a mobile app.

Providing consistent, live metrics of their store performance can significantly enhance the ability of artists and operators to make money in your marketplace. It is also possible to better connect workflows – for example, using analytics, sellers might be able to change the price of frequently back-ordered goods easily. Providing machine learning capabilities such as forecasting the demand of products might also provide the marketplace with new revenue streams.

Semantic layer

There is a tension between self-service analytics and consistency. It is important to give every business analyst in your business the ability to rapidly create dashboards without waiting on a central IT team. At the same time, it is imperative to maintain consistency in the way dashboards perform calculations (e.g. *shipping costs* for example have to be calculated always in the same way across dashboards). While it is important that business analysts can build complex dashboards themselves, it is also important that analysts reuse existing visualizations as much as possible. The traditional approach to providing consistency and reuse has been to centralize calculations and foundational capabilities within IT. However, such centralized, IT-centric solutions are

typically too brittle and too slow to satisfy business users in a data-driven environment.

Modern business intelligence tools like Looker or Microstrategy offer a *semantic layer* to help address this tension. A semantic layer is an additional layer that allows you to facilitate an end users access to data autonomously using common business terms; it works via a decoupling between the nomenclature adopted by the table creator and the names adopted by the business users. LookML, which Looker calls its semantic model layer, is a data language based on SQL (see Example 7-1). It provides data stewards the ability to create reusable dimensions or measures that business analysts can reuse and extend. These definitions are made available in a data dictionary for easy discovery and management.

Example 7-1. The semantic layer consists of centralized definitions of metrics and dimensions. This example from Looker shows a metric (overall health score).

```
dimension: overall_health_score {
    view_label: “Daily Account Health Scores”
    description: “Weighted average of Usage, User, Support and CSAT
Scores, range from 1-5”
    Type: number
    Sql:
(${usage_score}*2+${user_score}*2+${support_score}+${csat_score})/6 ;;
    Value_format_name: decimal_1
}
```

These semantic layers function as a business intelligence data source in and of themselves. Tableau, for example, can connect to Looker’s semantic layer instead of directly to the data warehouse.

Although business users will typically not see or interact with LookML like tools directly, they get to build dashboards that use the defined dimensions and metrics. This helps foster reuse so that each analyst doesn’t have to define from base table columns every dimension or metric that they employ. Centralizing the definition of a metric also decreases the likelihood of human error, and provides a single point in which definitions can be updated. Having such a centralized definition exist in a text file permits easy version control as well.

You have seen how you can dive into the data with the help of the BI tools and, via a semantic layer, facilitate business users managing the data by themselves. Sometimes this approach is not enough and you need to prepare the data before

starting with the ingestion into the data warehouse. In the next section we will focus on this topic.

Transformations

Suppose you land raw data from ERP systems into a data warehouse. If the ERP is SAP, it's likely that the column names are in German and reflect the application state, not what we would consider data that is useful to persist. We don't want to force all our users to have to transform the data into a usable form every time they need it. So, where should the transformation take place?

One option is to define the way columns have to be transformed in a semantic layer that is part of your BI tool as discussed in the previous section. However, this limits the definitions to dimensions and metrics, and accessing these definitions will be difficult from non-BI tools.

A better approach is to define transformations in SQL and create views, tables, or materialized views. In this section we will have a look at the common options available to handle transformations within a data warehouse. This is another advantage of the hub-and-spoke architecture: when data transformations are implemented in the data warehouse, the results are immediately available to all use cases that require them.

ELT with views

One approach is to load data into the data warehouse as-is and transform it on the fly when reading it through views (see Example 7-2). Because the views carry out the transformation on the fly after the data is loaded into the data warehouse (no transformation is carried out before loading), this is commonly called Extract-Load-Transform (ELT) in contrast to the typical ETL workflow.

Example 7-2. In this example from Azure Synapse Analytics the SQL code creates a view by joining two tables.

```
CREATE VIEW view1
AS
SELECT fis.CustomerKey, fis.ProductKey, fis.OrderDateKey,
    fis.SalesTerritoryKey, dst.SalesTerritoryRegion
FROM FactInternetSales AS fis
LEFT OUTER JOIN DimSalesTerritory AS dst
ON (fis.SalesTerritoryKey=dst.SalesTerritoryKey);
```

Instead of querying the raw tables, users query the view. The SQL that creates the view can select specific columns, apply operations such as masking to the columns, and join multiple tables. Thus, ELT provides a consistent, governed view of the raw data to business users. Because the final query runs the *view query* before it further aggregates or selects, all queries reflect up-to-date information based on whatever data is present in the data warehouse.

Views, however, can become quite expensive in terms of computational resources, time, and/or money. Consider, for example, the view shown in Example 7-2 that joins the Sales orders tables with the sales territory. All sales orders from all the territories over the entire lifetime of the business are being queried in this view. This is the case even if the analyst querying view is interested only in a specific year or region (and in that case it makes a lot of sense to filter out the data that is not relevant before performing the join).

Scheduled queries

If the tables are updated infrequently, it can be far more efficient to extract the data into tables periodically. For example, in Google BigQuery as reported in Example 7-3, we specify the destination table, how often to run the query, and the query itself:

Example 7-3. In this Google BigQuery example , raw data from the Orders table is extracted, aggregated, and aggregated results stored in a destination table every 24 hours.

```
bq query ...
    --destination_table=Orders_elt.lifetime_value \
    --schedule='every 24 hours' \
    --replace=true \
    'SELECT
        customer_id,
        COUNT(*) AS num_purchases,
        SUM(total_amount) AS lifetime_value
    FROM Orders.all_orders
    GROUP BY customer_id'
```

The raw table is queried only once every 24 hours in the example shown. While there is an increase in storage cost associated with the destination table, storage costs are typically orders of magnitude cheaper than computational costs. Therefore, the cost of creating the destination table is quite manageable.

The key advantage of scheduled queries is that analysts query the destination table and not the raw data. Therefore, analyst queries are relatively inexpensive.

The drawback of scheduled queries is the results returned to analysts can be up to 24 hours out of date. The degree to which queries are out of date can be reduced by running the scheduled query more frequently. Of course, the more often the extraction query is scheduled, the more the cost advantages start to dissipate. Another drawback of scheduled queries is that they can be wasteful if the resulting destination table is never queried.

Materialized views

It is clear that the most efficient way to balance obsolete data and cost is to extract raw data into a destination table when you request a view for the first time. Subsequent queries can be fast because they can retrieve data from the destination table without doing any extraction. Meanwhile, the system needs to monitor the underlying tables, and re-extract the data when the original tables change. You can make the system even more efficient by updating the destination table with new rows of raw data rather than requerying the entire table.

While you can build a data engineering pipeline to all this, modern cloud data warehouses support fully managed materialized views out of the box. Creating a materialized view in these systems is analogous to creating a live view (see Example 7-4) and you can query it just like any other view. The data warehouse takes care of ensuring that queries return up-to-date data. The data warehouse vendor charges a few costs for managing the materialized view

Example 7-4. An example, from the Snowflake documentation, of creating a materialized view of data that doesn't change very frequently. Snowflake will transparently keep the materialized view up-to-date.

```
create materialized view vulnerable_pipes
(segment_id, installation_year, rated_pressure)
as
select segment_id, installation_year, rated_pressure
from pipeline_segments
where material = “cast iron” and installation_year < “1980”::date;
```

Be careful, though – some data warehouses (like Amazon Redshift, at the time of writing) do not automatically manage the materialized view for you – you are

expected to set up a schedule or trigger to REFRESH the materialized view. In that sense, what they term a materialized view is actually just a table extract.

Google BigQuery, Snowflake, and Azure Synapse do transparently maintain the materialized view content. The view content is automatically updated as data is added to the underlying tables.

Lineage

Data governance best practice, as we introduced in Chapter 3, is to ensure that an organization keeps track of all data transformations from its ingestion up to its usage. This is achieved via the *lineage* that is nowadays an important capability of a modern data warehouse.

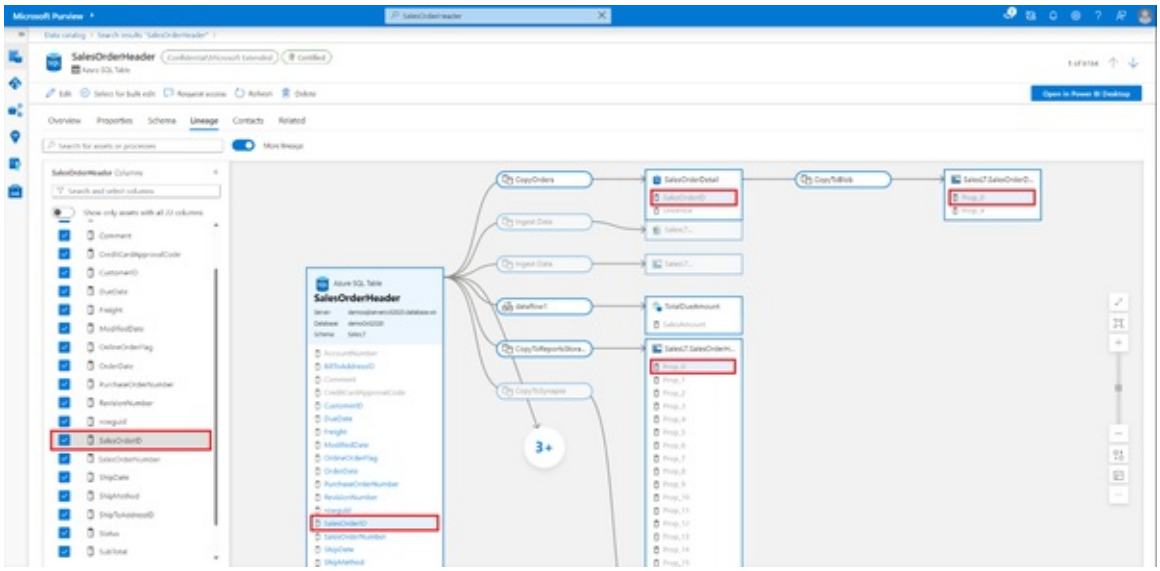


Figure 7-10. Example from Microsoft documentation of following the mapping of a column in the Azure Data Catalog.

If you use the native managed services of a cloud vendor, when you perform data transformations the tools will typically manage and carry along metadata (see Figure 7-10). Thus, if you use for example Data Fusion, views, materialized views, etc. on Google Cloud, the Data Catalog is updated and lineage maintained. If you build your transformation pipeline using Dataflow, you should update the Data Catalog. Similarly, the crawlers will automatically update the Data Catalog on AWS, but you need to invoke the Glue API to add to the catalog if you implement your own transformations.

You have seen how the data warehouse (the *hub* in our architecture) can drive

data transformation to make it available to all use cases you want to implement keeping track at the same time all the metadata you need to maintain a robust lineage of all the elaborations. Now let's have a look at how you can craft the structure of the organization to match the hub and spoke architecture.

Organizational Structure

In many organizations, there are many more business analysts than engineers. Often, this ratio is 100x. A hub-and-spoke architecture is ideal for organizations that wish to build a data and ML platform that primarily serves business analysts. Because the hub-and-spoke architecture assumes that business analysts are capable of writing ad-hoc SQL queries and building dashboards, some training may be necessary.

Roles and responsibilities

In an analyst-first system, the central data engineering team is responsible for (see the filled shapes in Figure 7-11):

- Landing the raw data from a variety of sources into the data warehouse. Many sources can be configured to directly publish to modern data warehouses.
- Ensuring data governance, including a semantic layer and the protection of PII.
- Workloads that cross business units (e.g., activation), involve data across business units (e.g., identity resolution), or require specialized engineering skills (e.g., ML).
- Common artifact repositories such as data catalog, source code repository, secret store, feature store, and model registries.

The business unit is responsible for (see the unfilled shapes in Figure 7-11):

- Landing data from business-specific sources into the data warehouse
- Transforming the raw data into a form usable for downstream analysis
- Populating the governance catalogs and artifact registries with business-

specific artifacts

- Reports, ad-hoc analytics, and dashboards for business decision making

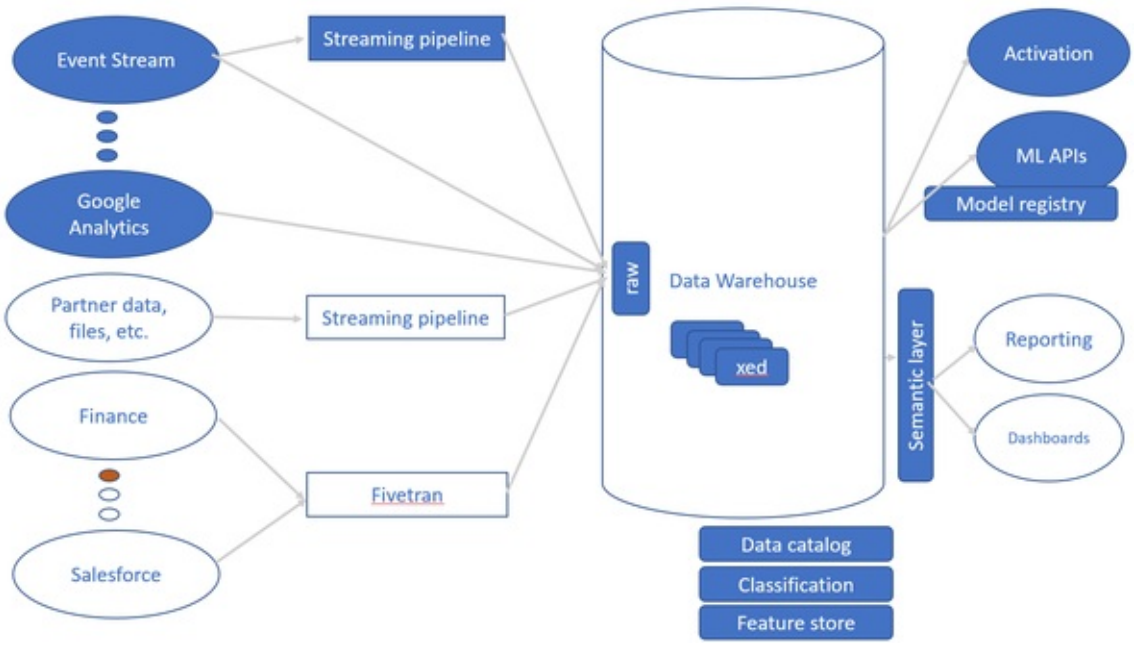


Figure 7-11. The central data engineering team is responsible for the filled shapes while the business unit is responsible for the unfilled shapes.

Figure 7-11 shows Google Analytics, Finance, and Salesforce as illustrative of data sources. In our example, Google Analytics data may be needed across multiple business units and is therefore ingested by the central data engineering team. Finance and Salesforce data are needed only by specific business units and are therefore ingested by that business unit.

Each business unit manages its own deployment. It is the responsibility of the central team to land data into the data warehouse that the business team uses. This often means that the software that the central team uses needs to be portable across different clouds and the data format that is produced by the pipelines is a portable format. For this reason, Apache Spark and Parquet are common choices for building the ETL pipelines.

If two business units choose to use the same data warehouse technology, data sharing across those two business units becomes simpler, so where possible, we recommend using the same data warehouse technology (BigQuery, Snowflake, Redshift, etc.) across the organization. However, in businesses that grow via acquisitions, this is not always possible.

In this section you have seen how you can modernize your data platform via the implementation of a hub and spoke architecture putting at the center your data warehouse. You have understood how multiple spokes can gravitate around the central hub, your modern data warehouse, to enable whatever use case you want to implement both batch and streaming leveraging pure SQL language and be compliant with data governance requirements. In the next section we discuss how the data warehouse can enable data scientists to carry out their activities.

Data Warehouse to enable Data Scientists

Data analysts support data-driven decision-making by carrying out ad-hoc analysis of data to create reports and then operationalize the reports through business intelligence. Data scientists aim to automate and scale data-driven decisions using statistics, machine learning, and artificial intelligence. What do data scientists and data science tools need from a modern cloud data warehouse? As you saw in Figure 7-1 they need to interact with the data warehouse in various ways to execute queries or to simply get access to the low level data. In this section we will have a look at all the most common tools they can leverage to achieve this goal.

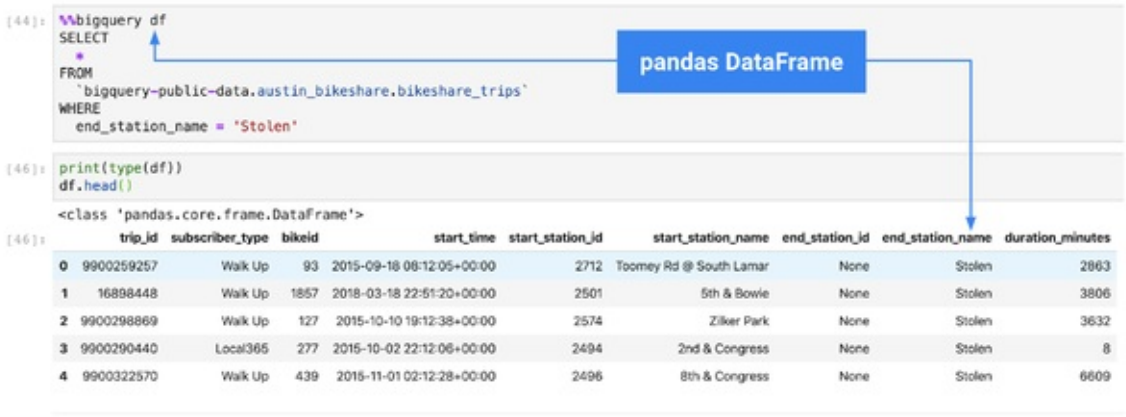
As you saw in the previous chapter, data scientists need to carry out experiments to try out different forms of automation and learn how the automation will work on various slices of historical data. The primary development tools that data scientists use for their experimentation, as we have seen earlier, are *notebooks*. Therefore they need to have ways to access data in the data warehouse efficiently. This has to be for both exploratory data analysis through the query interface of the data warehouse and for operationalization through the storage interface of the data warehouse (please refer Figure 7-1). It is important to ensure that your cloud data warehouse supports both these mechanisms. Let’s have a look at how these mechanisms work.

Query interface

Before automating decision-making, data scientists need to carry out copious amounts of exploratory data analysis and experimentation. This needs to be done interactively. Therefore, there needs to be a fast way to invoke SQL queries from

the notebook.

A Jupyter magic (such as the %%bigquery line in the cell in Figure 7-12) provides a way to invoke SQL queries from the notebook without boilerplate code. The result comes back as a native Python object, called a DataFrame, that can be acted upon using Pandas, a library of data analysis functions.



	trip_id	subscriber_type	bikeid	start_time	start_station_id	start_station_name	end_station_id	end_station_name	duration_minutes
0	9900259257	Walk Up	93	2015-09-18 08:12:05+00:00	2712	Toomey Rd @ South Lamar	None	Stolen	2863
1	16898448	Walk Up	1857	2018-03-18 22:51:20+00:00	2501	5th & Bowie	None	Stolen	3806
2	9900298869	Walk Up	127	2015-10-10 19:12:38+00:00	2574	Zilker Park	None	Stolen	3632
3	9900290440	Local365	277	2015-10-02 22:12:06+00:00	2494	2nd & Congress	None	Stolen	8
4	9900322570	Walk Up	439	2015-11-01 02:12:28+00:00	2496	8th & Congress	None	Stolen	6609

Figure 7-12. From a notebook, you can invoke SQL queries on the data warehouse without boilerplate code and get the result back as an object that is easy to work with. In this example we are invoking BigQuery.

It is important to note that this is done without creating in-memory extracts of the data. The cloud data warehouse carries out the SQL queries in a distributed way. The notebook server doesn't need to run on the same computational infrastructure as the data warehouse.

This combination of using the data warehouse backend for large-scale data processing, and the programmatic and visualization capabilities of the notebook frontend is potent and necessary for data scientists to be productive.

Storage API

While the interactive capabilities of the notebook-data warehouse connection are important for exploration and experimentation, they are not what you need for automation. For automation, the speed of data access is paramount. It should be possible for machine learning frameworks to bypass the query API and directly access the storage layer of the data warehouse. The storage access should support parallel reading from multiple background threads because the common scenario is for the reading of one batch of data to happen while the ML accelerator (GPU or TPU) is carrying out heavy computations on the previous

batch.

Instead of using the query API, therefore, use the storage API that ML frameworks support to read data efficiently and in parallel out of the data warehouse. Reading data from Google BigQuery using the Storage API from Spark and TensorFlow is shown in Example 7-5.

Example 7-5. Reading data directly from Google BigQuery using Spark (above) or TensorFlow (below) without going through the query layer.

```
df = spark.read \
  .format("bigquery") \
  .load("bigquery-public-data.samples.shakespeare")
def read_bigquery():
  tensorflow_io_bigquery_client = BigQueryClient()
  read_session = tensorflow_io_bigquery_client.read_session(
      "projects/" + PROJECT_ID,
      "bigquery-public-data", "samples", "shakespeare",
      ...,
      requested_streams=2)

  dataset = read_session.parallel_read_rows()
  transformed_ds = dataset.map(transform_row)
  return transformed_ds
```

Query interface and storage API are the two methods that data scientists use to access the data in the data warehouse in order to carry out their analysis. Now there is a new trend to consider—the ability to implement, train and use a ML algorithm directly in the data warehouse without having to pull out the data. In the next section we will have a look at how it works.

Machine learning without moving your data

Some modern cloud data warehouses (BigQuery and Redshift, at the time of writing) also support the ability to train machine learning models on data in the data warehouse without having to first extract data out. They do this by training simple models in SQL, and more complex models by delegating the job to Vertex AI and Sagemaker respectively. Let’s have a look at how you can leverage both training and serving (known as *activation*) of your machine learning algorithm directly from your data warehouse.

Training ML models

Let’s imagine you want to train a machine learning model to predict whether or not a user will churn given the characteristics of an account and charges on the account: it is possible to do everything leveraging the historical data as shown in the Example 7-6. The types of models trained can be quite sophisticated.

Example 7-6. Develop and train a classification ML model in AWS RedShift leveraging historical data you already have in the data warehouse.

```
CREATE MODEL customer_churn_auto_model FROM (SELECT state,
             account_length,
             area_code,
             total_charge/account_length AS average_daily_spend,
             cust_serv_calls/account_length AS average_daily_cases,
             churn
      FROM customer_activity
      WHERE  record_date < '2020-01-01'
     )
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE 'arn:aws:iam::XXXXXXXXXXX:role/Redshift-ML'SETTINGS (
  S3_BUCKET 'your-bucket'
);
```

You can train a recommender system using historical data on what visitors actually purchased, as shown in Example 7-7.

Example 7-7. Develop and train a recommendation ML model in Google BigQuery leveraging historical data you already have in the data warehouse.

```
CREATE OR REPLACE MODEL bqml_tutorial.my_implicit_mf_model
OPTIONS
 (model_type='matrix_factorization',
  feedback_type='implicit',
  user_col='visitorId',
  item_col='contentId',
  rating_col='rating',
  l2_reg=30,
  num_factors=15) AS
SELECT
 visitorId,
 contentId,
 0.3 * (1 + (session_duration - 57937) / 57937) AS rating
FROM bqml_tutorial.analytics_session_data
```

The Example 7-8 shows an anomaly detection system written using just two SQL statements.

Example 7-8. Anomaly detection model developed in SQL in BigQuery.

```
CREATE OR REPLACE MODEL ch09eu.bicycle_daily_trips
OPTIONS(
    model_type='arima_plus',
    TIME_SERIES_DATA_COL='num_trips',
    TIME_SERIES_TIMESTAMP_COL='start_date',
    DECOMPOSE_TIME_SERIES=TRUE
)
AS (
    SELECT
        EXTRACT(date from start_date) AS start_date,
        COUNT(*) AS num_trips
    FROM `bigquery-public-data.london_bicycles.cycle_hire`
    GROUP BY start_date
);

SELECT *
FROM ML.DETECT_ANOMALIES(
    MODEL ch09eu.bicycle_daily_trips,
    STRUCT (0.95 AS anomaly_prob_threshold))
ORDER BY anomaly_probability DESC
```

Being able to do machine learning using just SQL without having to set up data movement opens up predictive analytics to the larger organization. It democratizes machine learning because data analysts leveraging typical BI tools can now implement predictions. It also dramatically improves productivity – it is more likely that the organization will be able to identify anomalous activity if it requires only two lines of SQL than if it requires a six-month project by data scientists proficient in TensorFlow or PyTorch.

ML training and serving

Training an ML model is not the only ML activity that modern data warehouses support. They also support the ability to:

- Export trained ML models in standard ML formats for deployment elsewhere.
- Incorporate ML training within the data warehouse as part of a larger ML workflow.
- Invoke ML models as external functions.
- Load ML models directly into the data warehouse to invoke ML predictions efficiently in a distributed way.

Let’s look at all four of these activities, why they are important, and how a modern data warehouse supports them.

Exporting trained ML models

ML models trained on data in the data warehouse can be invoked directly on historical data using ML.PREDICT in a *batch* mode. However, such a capability is not enough for modern applications that require real-time results (e.g. an e-commerce application that based on the connected user has to decide which ads to display in the page). It is necessary to be able to invoke the model *online*, i.e. as part of a synchronous request for single input or a small set of inputs.

You can accomplish this in Redshift by allowing the model to be deployed to a Sagemaker endpoint and in Google Cloud by exporting the model in SavedModel format. From there, you can deploy it to any environment that supports this standard ML format. Vertex AI endpoints are supported, of course, but so are Sagemaker, Kubeflow pipelines, iOS, and Android phones, and Coral Edge TPUs. Vertex AI and Sagemaker support deployment as microservices and are often used in server-based applications, including websites. Deployment to pipelines supports use cases such as stream data processing, automated trading, and monitoring. Deployment to iOS and Android supports mobile phones and deployment to Edge TPUs supports custom devices such as the dashboards of cars and kiosks.

Using your trained model in ML pipelines

It is rare that a data scientist’s experiment consists of just training an ML model. Typically, an experiment will consist of some data preparation, training multiple ML models, doing testing and evaluation, choosing the best model, creating a new version of the model, setting up an A/B test on a small fraction of the traffic, and monitoring the results. Only a few of these operations are done in the data warehouse. Therefore, the steps that are done in the data warehouse have to be part of a larger ML workflow.

ML experiments and their workflows are captured in ML pipelines. These pipelines consist of a number of containerized steps, a few of which will involve the data warehouse. Therefore, data preparation, model training, model evaluation, etc. have to be invokable as containers when they are part of an ML pipeline.

Pipeline frameworks offer convenience functions to invoke operations on a cloud data warehouse. For example, to invoke BigQuery as a containerized operation from Kubeflow pipelines, you can use a BigQuery operator.

Invoking external ML models

Machine learning is the common nowadays way to make sense of unstructured data such as images, video, and natural language text. In many cases, pre-trained ML models already exist for many kinds of unstructured content and use cases – for example, pre-trained ML models are available to detect whether some review text contains toxic speech. You don't need to train your own toxic speech detection model. Consequently, if your dataset includes unstructured data, it is important to be able to invoke a machine learning model on it.

In Snowflake, for example, it is possible to invoke Google Cloud's Translate API using an `EXTERNAL FUNCTION`. You can do this by creating a API integration through a gateway and then configuring Snowflake as demonstrated in Example 7-9:

Example 7-9. Snowflake 3rd party APIs integration example

```
create or replace api integration external_api_integration
    api_provider = google_api_gateway
    google_audience = '<google_audience_claim>'
    api_allowed_prefixes = ('<your-google-cloud-api-gateway-base-url>')
    enabled = true;
create or replace external function translate_en_french(input string)
    returns variant
    api_integration = external_api_integration
as 'https://<your-google-cloud-api-gateway-base-url>/<path-suffix>';
```

Given this, it is possible to invoke the translate API on a column from within a `SELECT` statement:

```
  SELECT msg, translate_en_french(msg) FROM ...
```

Loading pre-trained ML models

Invoking ML models using external functions can be inefficient because the computation does not take advantage of the distributed nature of the data warehouse. A better approach is to load the trained ML model and have the data warehouse invoke the model within its own computing environment.

In BigQuery, you can do this by first loading the TensorFlow model and then invoking it by using ML.PREDICT from within a SELECT statement as presented in Example 7-10:

Example 7-10. BigQuery loading TensorFlow model stored in Google Cloud Storage and predicting results

```
CREATE OR REPLACE MODEL
  example_dataset.imported_tf_model
OPTIONS
  (MODEL_TYPE='TENSORFLOW',
    MODEL_PATH='gs://cloud-training-
demos/txtclass/export/exporter/1549825580/*')
SELECT *
FROM ML.PREDICT(
  MODEL tensorflow_sample.imported_tf_model,
  (SELECT title AS input FROM `bigquery-public-
data.hacker_news.stories`))
```

Because `ML.PREDICT` does not require an external call from the data warehouse to a deployed model service, it is usually much faster. It is also much more secure since the model is already loaded and can not be tampered with.

Summary

In this chapter we focused on describing how a data warehouse can be the central core of a modern data platform analyzing how to leverage a hub and spoke architecture to enable data analysis and machine learning training and activation. The key takeaways are:

- A modern data platform needs to support making insights available to decision-makers faster
- The Hub-and-Spoke architecture is the ideal architecture for organizations with no legacy technologies to accommodate. All data is landed onto the enterprise data warehouse in as automated a manner as possible.
- Data from a large number of SaaS software can be ingested using pre-built connectors.
- A data warehouse can be made to reflect changes made in an OLTP database or ERP system using a Change Data Capture (CDC) tool.

- It is possible to federate data sources such as blob storage, ensuring that some data does not need to be moved into the data warehouse.
- It is possible to run external queries on SQL-capable relational databases.
- To assist business analysis, invest in a SQL tool that pushes operations to a database instead of relying on extracting OLAP cubes.
- When you have thousands to millions of customers, it’s more scalable to provide a lightweight graphics visual layer that can be embedded within the tool that the user is already using.
- Build a semantic layer to capture Key Performance Indicators (KPIs), metrics, and dimensions to foster consistency and reuse.
- Views provide a way to make the raw data landed in the data warehouse more usable. Scheduled Queries to extract view content to tables can be more cost-effective. Materialized views provide the best of both worlds.
- Data scientists need to be able to interactively access the data warehouse from notebooks, run queries without boilerplate code, and access data in bulk directly from the warehouse’s storage.
- Being able to do machine learning without any data movement and have the resulting ML model be usable in many environments helps foster the use of AI in many parts of the business.
- Organizationally, some data engineering and governance functions are centralized, but different business units transform the data in different ways, and only as and when it is needed.

In the next chapter we will focus on the convergence of the data lake and data warehouse worlds and we will see how modern platforms can leverage the best of the both paradigms giving to the final users the maximum of flexibility and performance.

Chapter 8. Converging to a Lakehouse

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the eighth chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

As you know by this point, there are two main approaches that organizations can take when designing their data platform: following a data lake or a data warehouse paradigm. Both approaches come with pros and cons but the question is: is it possible to make both technologies coexist to have a convergent architecture? In this chapter we will explore this topic starting with a brief motivation for this idea and then we will analyze two broad variants of the convergent architecture—known as the lakehouse architecture—and help you decide how to choose between them. The lakehouse concept is becoming increasingly popular because, as you will read in this chapter, it allows for a more flexible and scalable way to store and analyze structure, semi-structured and unstructured data at scale. As a result, it is becoming an important paradigm you should consider when designing a modern data analytics architecture. At the end of this chapter we will describe how to evolve towards the lakehouse architecture to give you a more consistent overview of the art of possible.

The need for a unique architecture

From spreadsheets, to data warehouses and data lakes, as you have seen in the previous chapters, there are different solutions developed over the time that allow you to cope with the exponential growth of data being generated with higher and higher frequencies and in formats that have become increasingly varied. Different roles started to work with data, and products were created that were appropriate for the skillsets of these users. The move to the public cloud provides new capabilities that help mitigate some problems with siloed data warehouses and data lakes. In this section you will learn more about the different user personas you may find interacting with the solution and what is the most important blocker that slowed down the adoption of such approach before the advent of cloud.

User personas

As you know, some key differences between a data lake and a data warehouse are related to the type of data that can be ingested and the ability to land unprocessed (raw) data into a common location.

These core differences explain the changes around the personas using the two platforms:

- Traditional data warehouse users are BI analysts who are closer to the business, focusing on driving insights from data. Data is traditionally prepared by the ETL tools based on the requirements of the data analysts. These users are traditionally using the data to answer questions. They tend to be proficient in SQL.
- Data lake users, in addition to analysts, include data engineers and data scientists. They are closer to the raw data with the tools and capabilities to explore and mine the data. They not only transform the data to make it accessible by the business (i.e. data that can be transferred to the data warehouses) but also experiment with it and use it to train their ML models and for AI processing. These users not only find answers in the data, but they also find the questions that are relevant for the business and prepare the data to make it useful to other users. They tend to be proficient in a coding language such as Python, Java, or Scala.

As a result of these different needs, we often see different IT departments with

individual teams traditionally managing the data warehouse and the data lake separately. However, this split approach has an opportunity cost: organizations spend their resources on operational aspects rather than focusing on business insights. As such, they cannot allocate resources to focus on the key business drivers or on challenges that would allow them to gain a competitive edge.

Additionally, maintaining two separate systems with the same end goal of providing actionable insights from data, can cause data quality and consistency problems. By not aligning on the storage and transformations of the data, there may end up being two different values for what is ostensibly one record. With the extra effort required to transform data that have standardized values (such as timestamp) many data users are less compelled to return to the data lake every time they need to use data. This can lead to *data puddles* across the enterprise, which are datasets stored on individual's machines, causing both a security risk and inefficient use of data.

For example, if an online retailer spends all their resources on managing a traditional data warehouse to provide daily reporting which is key to the business, then they fall behind on creating business value from the data, such as leveraging AI for predictive intelligence and automated actions. Hence, they lose competitive advantage as they have increased costs, lower revenues and higher risk. Effectively, it is a barrier to gain a competitive edge.

Anti-pattern: Duplicated data

If you think about an enterprise standard modern data architecture that leverages the data lake and data warehouse concepts we introduced in Chapters 6 and 7, you may end up with the architecture represented in Figure 8-1, where the data lake and the data warehouse coexist to enable self-service analytics and machine learning environments.

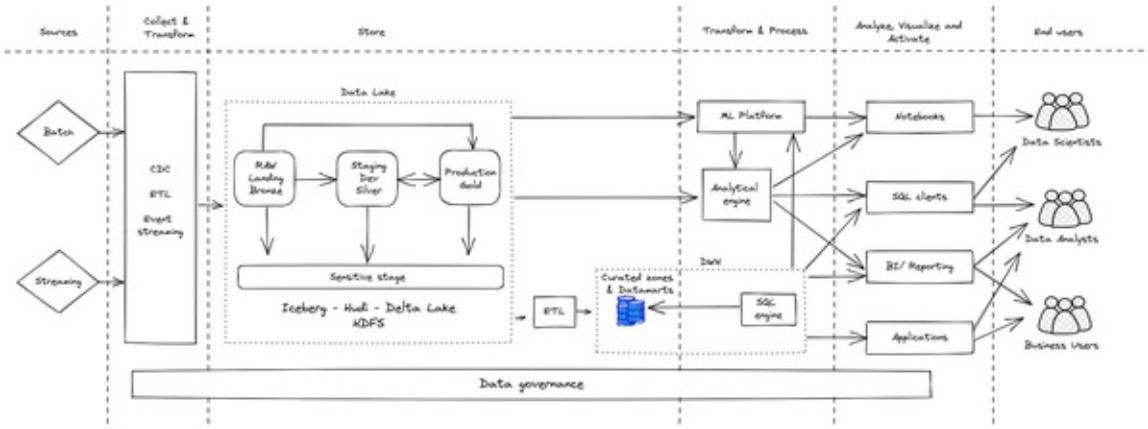


Figure 8-1. Data lake & DWH coexistence

Note that in this architecture the data that is *centrally managed* by the platform is the data that resides within the data lake. All the other data (e.g. Data Warehouse, BI/Reporting tools, Notebooks, etc.) can be considered a duplicated/ transformed version of the original data that from one side helps in getting better performance and facilitating analysis (e.g. materialized views) but, as you have already seen in deep in the previous chapters, it generates a lot of drawbacks because it leads to the generation of data silos.

Let’s have a look at the main building blocks of the data journey:

1. Data from various sources (batch and/ or streaming) is *collected* and *transformed/ wrangled/ prepared* (even on the fly) to be *stored* in the data lake where it will pass by all the different stages (bronze/ silver and gold). Since the beginning the data is subject to security, data quality and data governance checks.
2. Data can now be *processed* via the *analytical engine* solutions (e.g. Spark) that interacts with SQL clients, batch reporting and notebooks.
3. The data in parallel can be transformed and ingested into data marts (refer *Chapter 3 - Foundational elements*) in the data warehouse to handle BI workloads. Maintaining data marts is a complex and costly activity because it requires a lot of ETL engineering and it is challenging to keep all the needed data up to date. And this has to be done in a perpetual manner. Furthermore all the data governance activities have to be repeated in the DWH.

4. The data warehouse powers the BI tools and SQL clients, but sometimes it is not able to give the adequate level of performance requested by the final users so people tend to download the data, create local copies and work with that.
5. Machine learning solutions (e.g. SciKit Learn, Tensorflow, Keras), generally handled via notebooks, can leverage the data coming from the data lake but at the same time they can be connected with the DWH and the analytical engine.

An architecture like this typically includes a series of drawbacks:

- *Proliferation of the data* in various forms and, potentially, even outside the boundaries of the platform (i.e. local laptops), which may cause *security and compliance risks*, *data freshness issues*, or *data versioning issues*.
- *Slow down in time to market*: companies could spend even a couple of weeks just to implement a small change in a report for leadership because they have to request that data engineers implement modification to the ETLs in order to get access to the data needed to accomplish the task.
- *Limitation in data size*: users may not have access to all the data they need to carry out their analysis because, for example, data marts in the DWH could include just a subset of the necessary info due to performance needs.
- *Infrastructure* and *operational cost* could increase due to the complexity of ETLs that need to be put in place.

It's bad practice to have duplicated/ transformed data like this and you should minimize it as much as possible. Ideally you will get rid of duplicates and self-managed data with the goal of having all the data in a platform-managed mode that is available to all of the actors for analysis (i.e., in the right side of the architecture diagram depicted in Figure 8-1).

At this point you may ask yourself why organizations are leveraging this kind of architecture. The reason for this is pretty straightforward: because, as recently as 2018, this was the only way to give to the many different users all the solutions needed to meet their needs.

Technology evolves and, especially thanks to the advent of the cloud, new

solutions have been developed to bring better possibilities in terms of data handling. Let’s have a look at how we can improve on the previous architecture.

Converged architecture

By this point it’s clear that a full convergence between data lake and data warehouse can help final users get the most from their platform and hence from their data. But what is the architecture for such an end-state, and what is the path to get there?

The key factor in the converged lakehouse architecture is that the data warehouse and data lake share a common storage. Two different compute engines — a SQL engine (for data warehouse use cases) and a distributed programming engine (for data lake use cases) read and process the data without moving it around.

This common storage can take one of two forms:

1. The data is stored in an open-source format (Parquet, Avro, etc.) on cloud storage and you can leverage Spark to process it. At the same time you can use SparkSQL to provide interactive querying capability (the Databricks solution). In addition, data warehouse technologies (e.g. BigQuery, Athena, DuckDB, Snowflake, etc.) can support you in running SQL directly on the datasets without any data copying or movement. You can use open-source formats combined with a technology such as Delta Lake or Apache Iceberg that improves the underlying performance .
2. Alternatively, the common storage can be a highly optimized data warehouse format (e.g. the native format within BigQuery, Snowflake, etc.) where you can of course natively leverage SQL. In addition, these data warehouses support the use of compute engines such as Spark directly on the native data.

Both of these are compromises, so we recommend making the choice based on who your primary users are. A SQL engine operating on cloud storage (the first option) loses many of the optimizations that make data warehouses interactive and suitable for ad-hoc querying. A data lake running SQL (the second option) loses the schema-free data processing capability that makes data lakes so

flexible. So, choose the form of hybrid architecture based on which type of user and workload you want to support very well, and which one you want to support in a compromised way.

The first option is best for users who are proficient programmers. Build a lakehouse with the data lake (i.e. cloud storage) as your storage if the majority of your users are programmers who do a lot of data wrangling in code, such as to write ETL pipelines and to train ML models. The main advantage of data lakes is that they permit flexible data processing by programmers.

The second option is best for users who want to interact with the data to gain insights from it. Build a lakehouse with the data warehouse as your storage if the majority of your users are analysts rather than programmers, by which we mean that they can either write SQL or use a dashboard tool (like Tableau, PowerBI, or Looker) that will generate SQL. The main advantage of data warehouses is that they permit self-service, ad-hoc querying by business users.

Now that you have a better understanding of the different alternatives you can use to implement your lakehouse, let’s take a closer look at both options to better understand how they work behind the scenes.

Lakehouse on cloud storage

Separation of compute and storage provides the perfect habitat for a convergence of the data lake and data warehouse, with the benefits of having both. The cloud provider is able to enable distributed applications for interactive queries by bringing compute to the storage. This allows the cloud provider to allocate storage and compute independently to an organizations’ jobs, something that is hard to do on-premises where machines have to be procured months in advance. The amount of data and the compute required to analyze it can scale dynamically from warm pools of compute clusters. When storage is decoupled from the compute, you can utilize it for many different use cases. The same storage that was once file-based data lakes for structured data, can now be the same storage and data for the data warehouse. This key convergence enables you to store data once, utilizing views to prepare the data for each specific use case.

Let’s have a look at what are the foundational elements of such architecture, hot to get there and why it is something it can help you even in the future.

Reference architecture

An example of the storage convergence would be utilizing the same underlying storage for both the data warehouse that serves BI reporting and for the data used by a Spark cluster. This enables Spark code, that data lake teams spent years perfecting, to take advantage of the more performant storage that is often used as part of a distributed computing system. It allows the compute to move to the data, rather than the data to have to be shuffled among local disks. The high throughput of cloud storage unlocks better speed and performance. Many cloud providers offer this as a managed service, further abstracting the required management of the infrastructure.

Convergence of the data lake and data warehouse (see Figure 8-2) is about simplifying and unifying the ingestion and storage of the data, and leveraging the correct computing framework for a given problem. For structured and semi-structured data, writing all of the data as it is streamed into tables using a change data capture tool enables users to use simple SQL to transform the data and build logical views to query the information in a way that aligns with the business use cases. Because views are heavily leveraged in this pattern, there can be column elimination, partitioning, and logic to optimize the speed of the queries while maintaining a historical ledger of the data streamed into the tables.

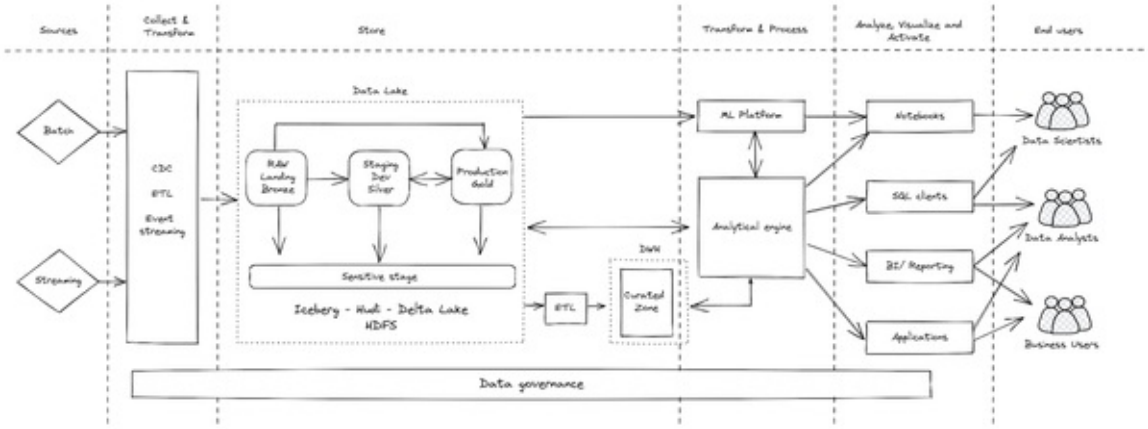


Figure 8-2. Data lakehouse reference architecture - Cloud storage approach

Conversely, data that is ingested via a batch pipeline can use a similar approach by which all of the data is written to a table, and SQL is used to create views with the most recent view of each record. Like the streaming data, a historical ledger is maintained in the raw tables, allowing data scientists to use all of the data for building and testing ML models. In this architecture, users can leverage

scheduled queries or an event-based lambda architecture for data ingestion.

In contrast to the architecture in Figure 8-1, which has two different engines, the reference architecture in Figure 8-2 has a single analytical engine that can access both data lake and data warehouse data. This is important to note because it allows for a more streamlined and efficient data analysis process.

The main goal of data lakehouse solutions like Dremio and Databricks is to enable high performance data processing while also supporting analytics and BI directly on the data lake storage. They usually provide a semantic layer that allows you to abstract the underlying data and provide to the analytical engine a view on the data that enables fast query processing without the need to implement data marts.

The "*Production - Gold*" layer of the project (see Chapter 6) can be the business-driven views or materialized tables that are governed and purpose-built. The underlying logic and storage provides access to end users and applications alike, allowing you to use the converged data platform for Hadoop, Spark, analytics, and ML.

It no longer matters if the data is stored within the data warehouse or within the freely floating cloud bucket. Behind the scenes it is the similar distributed storage architecture but data is structured differently. For example, data lakes would move the data from HDFS to the same object storage outside the cluster. This is the same as what cloud EDW would use as the backbone of its storage system. As a result, data is easily accessible and managed by both data lake and data warehouse architectures in one place. Therefore, organizations can now apply governance rules around data residing in the lake and the same data accessed by the data warehouse. Thanks to that, you can break down silos, not just by putting data into a central repository, but by enabling processing and query engines to move to wherever that data is as described in Figure 8-2. This leads to data warehouses and data lake convergence allowing them to use the same metadata store and governance, enabling data engineers, data scientists and data analysts to work together in collaboration, rather than in siloed systems.

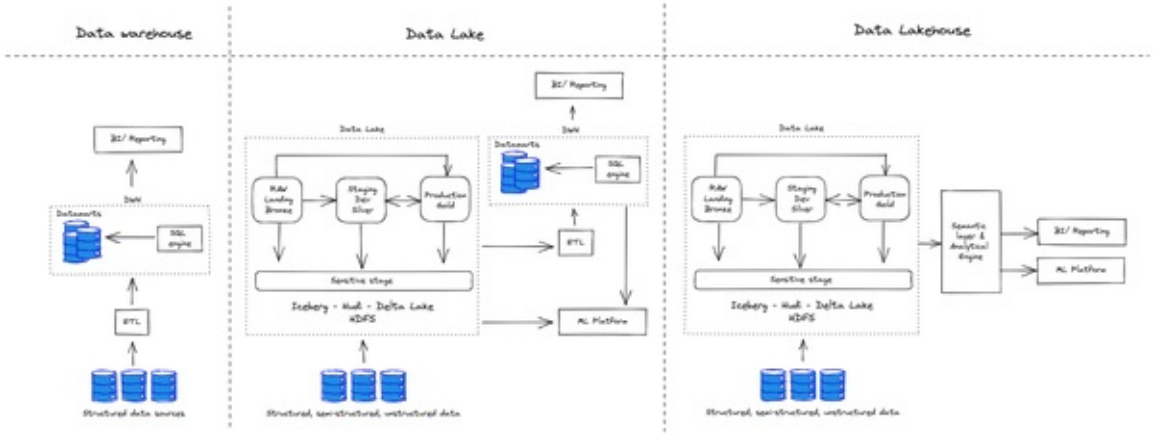


Figure 8-3. Data warehouse, Data lake and Data lakehouse approaches

Data engineers, data scientists and even business users will be able to perform self-serving queries on any kind of data they may have, leveraging one single point of access. And even in terms of development the work will be easier because there is only one system to interact with instead of a plethora of different tools. Another point in favor of this approach is the centralization of security and governance that is made stronger via the elimination of self-managed copies of the data.

Migration

Moving from the architecture in Figure 8-1 to the one in Figure 8-2 is an iterative process that initially starts with the introduction of a single analytical engine to take care of all the data processing, getting rid as much as possible of the Data Warehouse SQL engine. You can achieve this through various sub steps, following the process described in Chapter 5. Based on the solution that you've identified, your first action is to set up the new analytical engine connecting it with the Data Lake and the Data Warehouse storage as represented in Figure 8-4.

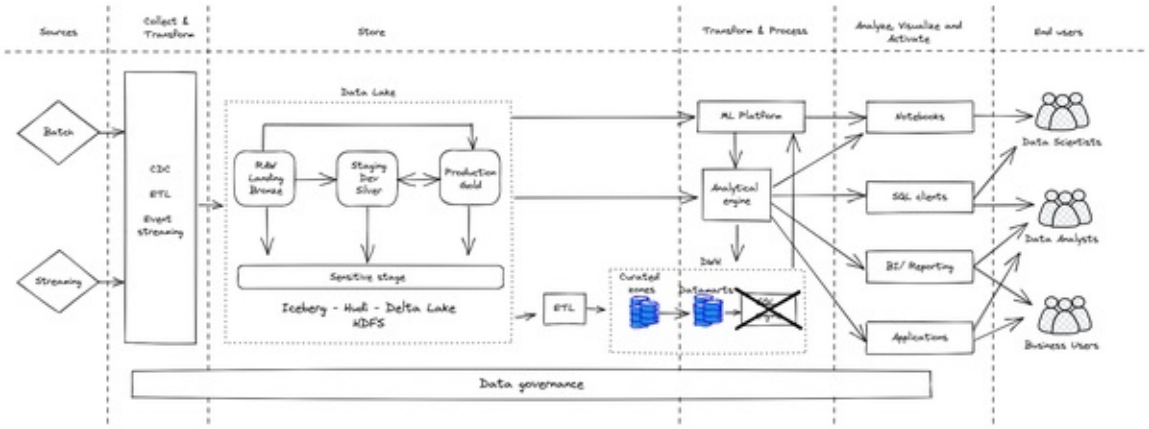


Figure 8-4. Journey to data lakehouse - Central analytical engine

Since this process can take several repetitions, it is important to start with by identifying a quick win use case that could demonstrate the benefits of the new solutions to create consensus within the organization. From there it will be possible to continue expanding the footprint of the new solution until it becomes the “de facto” analytical engine of the entire platform. A very good place to start with is generally the world of interactive queries for data exploration: users can interact with the new engine performing data analysis on a variety of data that span across the data lake and data warehouse. They can carry out queries directly with the new tool or via the integration with BI and reporting solutions. Once the benefits of the change become tangible, the old Data warehouse engine can be slowly decommissioned in order to save costs, reduce complexity and limit the need to download the data for offline elaboration, as the users will have now access, based on the organizations’ data governance rules, to the entire data of the company. Of course brand new workloads should only be implemented leveraging the new analytical engine.

Future-proofing

Once fully deployed, the interaction with data sets sitting mainly in the data lake and in the data warehouse (*only the curated ones*) will be bi-directional: that means that the analytical engine will be able to read and write directly on the underlying storage systems that are usually storing the data in an open format like Apache Parquet or Apache Avro. This is a great benefit because the underlying technology can be seen as a consistent and common medium across different types of storage systems. The analytical engine will be flexible enough to use either a schema on read approach (that is typical of Data Lake pattern) or

a more structured approach like the SQL-based data system. Another important benefit that becomes out of the box when adopting a lakehouse architecture, is the ease of streaming adoption. As you will see in the next chapter, real time access to the data is becoming increasingly important for every kind of business and the fact that you can treat streaming data in the same way as standard stored data is surely something that can bring additional value. In fact, in this architecture streaming data pipelines can read and write data in real time, knowing that the underlying technology can ensure full ACID transactions, bringing consistency within the system and always handling the most recent and up to date data.

SQL-first lakehouse

The main goal of SQL-first lakehouse solutions is to enable high performance analytics and BI while also supporting flexible data processing with Spark directly on the data warehouse storage. An advantage of this architecture is that business users can carry out orchestration and machine learning.

Let us examine the fundamental elements of such architecture, how to achieve it, and why it can be beneficial to you in the future.

Reference architecture

Using a data warehouse as a data lake of course requires that the data warehouse solutions are able to handle not only standard SQL queries on tables but also native integration with Spark based environments, machine learning capabilities and streaming features. Modern data warehouses like BigQuery, Athena, Synapse and Snowflake support these capabilities to varying extents that will, no doubt, have improved by the time you are reading this. In Figure 8-5 you can see a reference architecture where the data warehouse acts as a central analytical engine point for the platform. As you can see from the diagram, the data flows from the original sources (in various forms and velocity) through both the data lake and the native data warehouse storage. From here you can identify four main storage areas:

- The data lake storage that is equal to what we have seen in the previous section

- The data warehouse storage split in three dimensions
  - *RAW*: the data coming as-is from the various sources (batch or streaming)
  - *Enriched*: RAW data with a first layer of transformation
  - *Curated:* enriched data ready for final transformations

Spark (ETL) or SQL (ELT) can carry out all the transformations.

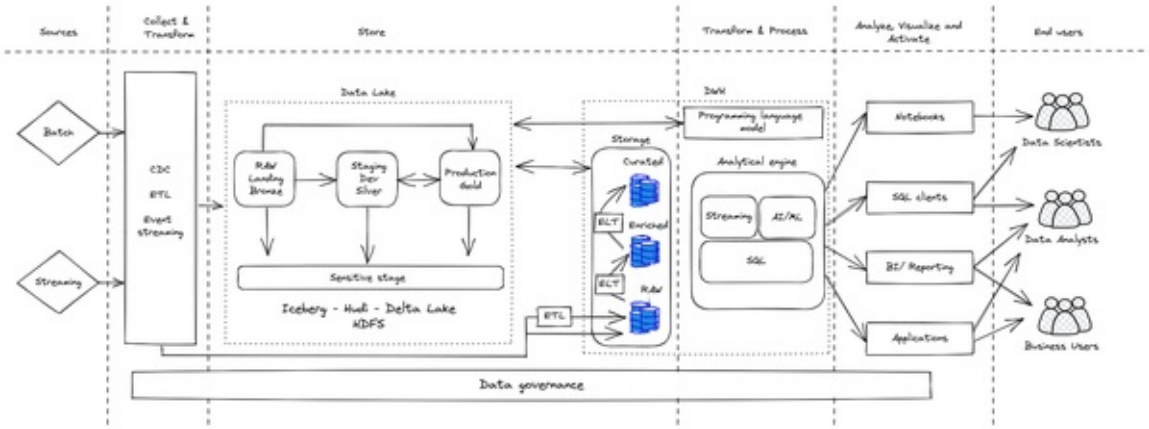


Figure 8-5. Data lakehouse / SQL-first approach

As you know by now, using SQL is the preferred approach to handling and transforming your data. A SQL-first approach leverages the highly optimized, interactive querying capabilities of the data warehouse to keep costs down, and open data-driven decision making to business users.

When you need to tackle more advanced data processing, you may be able to leverage a structured programming language like Spark. This is especially true when working with ML algorithms, be it structured data (e.g. boosted tree regression/classification) or unstructured data (e.g. natural language processing). This is due to the fact that these kinds of operations are not flexible to implement in SQL (inflexible, but not impossible: BigQuery and Redshift SQL have some ML functions today, and other data warehouses will no doubt support them shortly). Utilizing Spark also allows you to avoid having to rewrite legacy data processing jobs that may have been written in Spark in SQL.

Modern data warehouse solutions come with the ability to execute Spark jobs directly from their engine and, most importantly, without moving or copying

data outside the data store where it resides. BigQuery for example, achieves this by providing the ability to write stored procedures in Spark, executed in an ad-hoc serverless spark environment, that can be called in a SQL statement. Snowflake provides Snowpark, a built-in Python engine that is capable of running Spark (and other programming languages). Athena achieves this by operating off standard-format files in formats such as Parquet so that the same data can be read from managed Hadoop environments like EMR. Also, data lake technologies such as Databricks support directly reading and writing to the native warehouse storage through optimized bulk APIs such as the BigQuery Storage API.

One of the key advantages of the data lakehouse paradigm is that it has urged vendors (and the industry in general) to develop increasingly powerful data warehouses to make several complex solutions very easy to use for a broad set of users. This is particularly true for ML. Business users can train and deploy machine learning models leveraging standard SQL code and they can leverage open source solutions, like dbt, to easily automate and productionize data preparation. It is pretty common nowadays to see business analysts who are, for example, able to autonomously forecast, with a high level of accuracy, customer demand to put in place an efficient inventory management or to predict the best price based on historical data.

Having said that, the majority of the tasks related to ML technology are in the hands of data engineers and data scientists who are happy to have multiple screwdrivers available to play with data, while at the same time are more keen in leveraging tools they are more familiar with. Especially when dealing with the development of deep learning models on unstructured data, you need to work with massive data sets leveraging open source languages to manipulate tabular data known as *data frames* to generate models that have to be served to final users. This is why the native integration among the various data sources is playing a pivotal role: the ability to leverage the Spark support is a key feature of the paradigm that gives a high level of freedom to users.

With the proliferation of the development of ML models, another category of operations that the platform has to handle arose—Machine Learning Operation (MLOps). In MLOps users want to keep track of the data versioning, data lineage (especially important for audit purposes) and model updates leveraging

the same approach adopted in the development of software (i.e. DevOps). Solutions like MLFlow, Google VertexAI or Amazon Sagemaker are natively connected with modern data warehouse and the data lake, giving final users a unified experience when dealing with the ML model lifecycle.

Migration

As with the lakehouse-on-cloud-storage approach, the transition to a SQL-first lakehouse is an iterative process that requires several steps in order to be fully operational as represented in Figure 8-6. The first step is to enable data ingestion directly into the data warehouse: the data sources have to be directly connected not only with the data lake but especially with the three types of data warehouse storage (i.e. RAW/ Enriched and Curated). Once the data is fully ingested into the central data repository, both in batch and in streaming mode, it is time to cut out external analytical engines, elevating the SQL engine of the data warehouse as the main one. Here it is pivotal to pay attention to how you're going to move data lake workloads into the data warehouse leveraging the built-in programming language module (i.e. Python or Spark engines/ connectors) to operate directly on the native storage format. The ML workloads, that initially are still handled outside the data warehouse, will be insourced as the last iteration of the process.

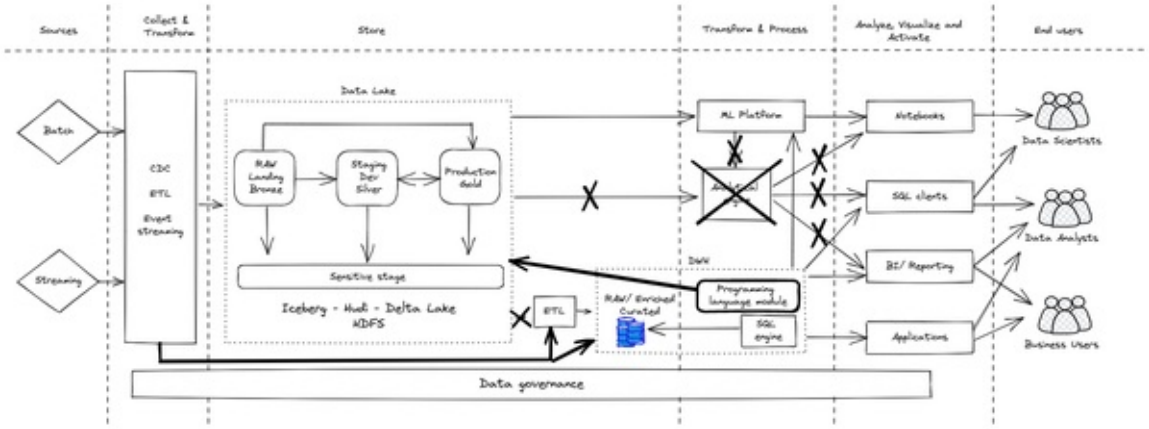


Figure 8-6. Journey to data lakehouse - SQL-first approach

During the phases of the migration, following the SQL-first approach, organizations will discover that a huge number of pipelines will have to be written in SQL.

In this architecture, it is more performant to implement data processing in SQL

(although ETL in Spark remains a possibility). This often requires a new type of skill within your organization, termed analytics engineering. You may be able to easily find workers skilled in analytics engineering within your organization (refer *Chapter 4 - Data analysis driven organization*) since SQL is a widespread language (more so than Spark or other programming languages) and can be learned rapidly. In fact this is a point in favor of democratization because adopting this paradigm will make data access easier for a huge number of employees.

Analytics Engineers will be responsible for much of the data enrichment and curation, and will be able to bring in their domain knowledge and require only SQL skills. The different types of datasets (raw, enriched, and curated) will likely each be utilized by different types of users. In general most end users, analytical teams and applications will utilize the curated data, while data engineers and data scientists will probably prefer to access the RAW/ data lake and Enriched data.

In this scenario it is important to note that streaming and machine learning capabilities are included in the Analytical Engine: this means that they are natively available to everyone, even to people who are not advanced in writing code in TensorFlow, PyTorch, Keras or SciKit Learn. This is a great achievement in enabling democratization of data and tool access to employees within the organization, and it will enable more and more users to achieve more with their data. Finally it is important to note that data governance is handled in a federated and transversal way, creating a unified and centralized place to manage, monitor and govern the data making it accessible to a variety of users and tools.

Future-proofing

Once the migration has been completed, and the organization has developed an architecture as outlined in Figure 8-6, the majority of interactions will be SQL-based. It is possible to leverage other programming languages for use cases where SQL is difficult, or off-the-shelf libraries are available, making the non-SQL option more attractive.

The benefit of a SQL-first lakehouse architecture is huge because it provides a single place for data storage and democratizes access to it via a standard and widely-known programming language (SQL) that is supported by a large number

of tools used by business users. The major benefit, compared with the cloud storage based data lakehouse, is exactly this ability to bring a broader set of users closer to the data and allow them to innovative solutions (i.e. ML)—the more people who the ability to employ the data (in a governed way) the more innovation your organization will see.

The benefits of convergence

The concept of lakehouse is just at the beginning of its journey but we can already identify several benefits that this paradigm can immediately bring to you if you will decide to embrace it. Leveraging the main two approaches you have learned so far, you can achieve following immediate benefits:

- **Time to Market**: You can ingest and immediately use data straight away whether it is batch or real-time data sources. Rather than employing complex ETL pipelines to process data, data is "staged" in either a messaging bus or through object storage. Then it is transformed within the converged data warehouse / data lakes that enables users to act as the data is received.
- **Reduced Risk:** You can continue leveraging existing tools and applications without rewriting them. This reduces the risk and costs associated with change.
- **Predictive Analytics:** Moving away from the traditional view of data marts and data mining to real-time decision making using fresh data increases business value. This is only possible because the governance and strictness around DWs have come down reducing barriers to entry.
- **Data Sharing:** Converged environment is now the one-stop shop for all the types of users (i.e. data analyst / data engineer / data scientist) you may have. They can all access the same managed environment getting access to different stages of data when they need it. At the same time different roles can have access to the same data through different layers and this is governed by platform wide access rights. This does not only increase the data governance but also allows simpler access management and auditing throughout the data ecosystem.

- **ACID transactions**: In a typical data warehouse the data integrity is maintained, and multiple users reading and writing the data see a consistent copy of the data. Although ACID is a key feature in the majority of the databases, traditionally this has been rather difficult to provide the same guarantees when it comes to traditional HDFS-based data lakes. There are schemes such as Delta Lake and Apache Iceberg which try to maintain ACID semantics (refer *Chapter 6 - The evolution of data lake with Apache Iceberg, Apache Hudi and Delta Lake*); they store a transaction log with the aim of keeping track of all the commits made to a data source.
- **Multi-modal data support:** Semi-structured and structured data are key differentiators with the data warehouses and data lakes. Semi-structured data has some organizational properties such as semantic tags or metadata to make it easier to organize, but data still does not conform to a strict schema. In the converged world this is accommodated with extended semi-structured data support. On the other hand, for unstructured use cases, data lakes are still required apart from edge cases.
- **Breakdown silos and ETL pipelines:** Traditionally different tools and environments, usually orchestrated by ETLs, manage data capture, ingest, storage, processing and serving. In addition, processing frameworks such as Spark, Storm Beam, etc provide built-in ETL templates to enable organizations to build ETL pipelines. However, with capable Cloud EDWs and integrated Cloud tools this pipeline is now all handled by a single environment. ELT does most of the traditional ETL tasks such as cleanse, de-dupe, join and enrich. This is made possible at different stages of the Data Lake implementation within the DWH. Furthermore, with the support of core data warehouses you can have access through a united environment to concepts such as stored procedures, scripting, and materialized views.
- **Schema and governance:** In reality, business requirements and challenges evolve in time. As a result, associated data changes and accumulates, either by adapting to new data or by introducing new dimensions. As the data changes, applying data quality rules becomes more challenging and requires schema enforcement and evolution. Furthermore, PII data governance becomes more important as new data sources are added as well. There needs to be a data governance solution allowing organizations to have a

holistic view of their data environment. In addition, it is paramount to have the ability to identify and mask PII data for different purposes and personas.

- **Streaming analytics:** Real-time analytics enables immediate responses and there would be specific use cases where extremely low latency anomaly detection application is required to run. In other words, business requirements would be such that it has to be acted upon as the data arrives on the fly. Processing this type of data or application requires transformation done outside of the warehouse.

As you have seen there are a lot of benefits you may get adopting a lakehouse architecture especially because it guarantees that different user personas can use the complete and most recent dataset for business intelligence, data analytics, and machine learning. Having a single system to manage simplifies the enterprise data infrastructure and allows users to work more efficiently.

Summary

In this chapter we focused on describing how you can mix data lake and data warehouse technology in a brand new mixed architecture called lakehouse. We presented the two most common architectures and how to get there. The key takeaways are:

- Technology evolved a lot over the years and we can see the same for the volume, variety and velocity of the data each organization deals with.
- What used to be possible to achieve with a spreadsheet application, nowadays would be impossible: because of this organizations are asking for more evolved systems to help them store and derive value from the collected data.
- There are two main approaches that organizations have followed in designing their data platform: data lake or a data warehouse paradigm
- Both approaches come with pros and cons but there’s a new option—convergence in the form of a lakehouse that allows you to have the best of both worlds

- Cloud can accelerate the idea of lakehouse because organizations can leverage fully managed environments whereby most of the operational challenges are resolved by the services provided
- Convergence of the data lake and data warehouse is about simplifying and unifying the ingestion and storage of the data, and leveraging the correct computing framework for a given problem
- Data is easily accessible and managed by both data lake and data warehouse architectures in one place.
- Organizations can apply governance rules around data residing in the lake and the same data accessed by the data warehouse.
- It is possible to break down silos, not just by putting data into a central repository, but by enabling processing and query engines to move to wherever that data is.
- There are two possible approaches to take when opting for a lakehouse.
- One approach to a lakehouse is to converge data lake and data warehouse through an interactive process that starts with the identification of a common analytical engine (generally Spark based). Over time, this will lead to the decommission of the data warehouse engine.
- The second lakehouse option is to move data lake workloads into the data warehouse and use built-in Python engines or Spark connectors to operate directly on the native storage format.
- There are several benefits of the lakehouse approach: time to market, ability to easily implement predictive analytics, breakdown silos and ETL pipelines, schema and governance and streaming analytics are just a few.

In the following chapter, you will discover why the industry trend is unstoppably shifting from batch to streaming and you will learn what are the main streaming architectures you may want to leverage.

Chapter 9. Architectures for Streaming

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the ninth chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *vwilson@oreilly.com*.

In this chapter, you will learn why the industry trend is inexorably away from batch and into streaming. We will discuss different streaming architectures, and how to choose between them. We will also do a deeper dive into two of these architectures—micro-batching, and streaming pipelines—and discuss how to support real-time, ad-hoc querying in both these architectures. Finally, sometimes the reason to do streaming is to autonomously take some action when certain events happen, and we will discuss how to architect such automated systems.

The value of streaming

Businesses along the entire technology maturity spectrum, from digital natives to more traditional companies, across many industries are recognizing the increasing value of making faster decisions. For example, consider business A that takes 3 days to approve a vehicle loan. Business B, on the other hand, will approve or deny a loan in minutes. That increased convenience will lead business B to have a competitive advantage.

Even better than faster decisions is being able to make decisions *in context*. Being able to make decisions while the event is proceeding (See Figure 9-1) is significantly more valuable than making the decision even a few minutes later. For example, if you can detect a fraudulent credit card when it is presented for payment, and reject the transaction, you can avoid a costly process of getting reimbursed.

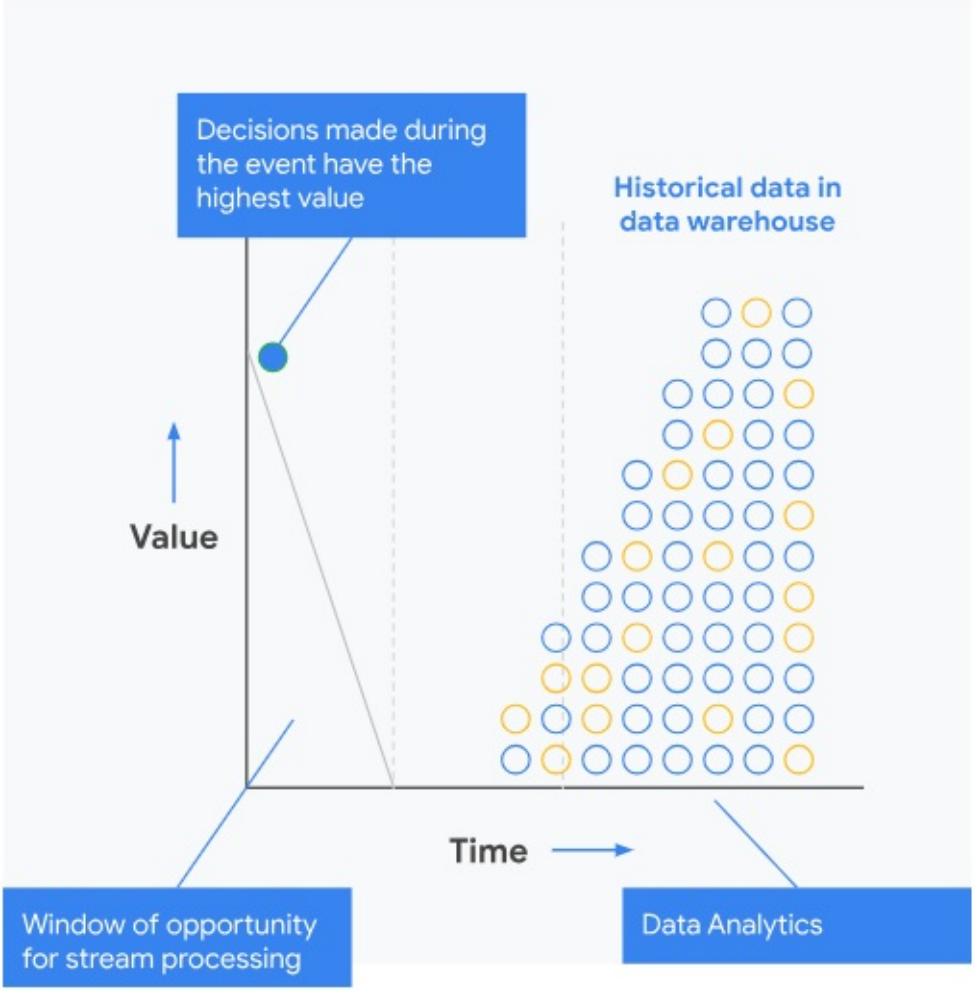


Figure 9-1. The value of a decision typically drops with time. Stream processing allows an organization to make decisions in near real-time.

Industry use cases

Whether it is fraud detection, transaction settlement, smart devices, or online gaming, industry after industry has started to adopt streaming.

In healthcare, we see streaming being used for real-time patient monitoring, alerting on falls or self-harm, providing personalized care with IoT medical

devices, and optimization of drug and supply and inventory in hospitals.

In financial services, we see streaming being used to detect and prevent fraud, predict and analyze risk, identify transactions that run afoul of compliance regulations, and deliver personalized offers to customers.

In retail, we see streaming being used for personalized marketing in websites, providing real-time inventory across multiple channels (website, mobile, physical stores), alerting on fulfillment issues, dynamic pricing, product recommendations, and omnichannel customer visibility.

In media and entertainment, we see streaming being used to generate personalized content, deliver targeted ads, minimize customer churn, and prevent subscriber fraud. In telecommunications, we see similar use cases around customer churn and subscriber fraud. In addition, streaming is used to improve network reliability and optimize network capacity planning.

Streaming use cases

Think of streaming as data processing on unbounded datasets. Technologically, the challenge with streaming is two-fold. One is that the dataset is infinite and never complete. Because of this, all aggregates (such as maxima) can be defined only within time-windows. Secondly, the data is in motion and held in temporary storage. This makes it difficult to apply conventional programming techniques and concepts such as file handles to read and process the data. Because of this complexity, there is a huge benefit to dividing streaming use cases into four categories in increasing order of complexity and value:

1. Streaming ingest is when you care only about keeping up with the stream of data and landing it in a persistent store.
2. Real-time dashboards are useful when you wish to visualize data as it comes in. You may also be interested in statistics and charts of time-windowed aggregates of the data.
3. Stream analytics is when you wish to carry out computations on the data as it arrives. Usually, this is to alert human operators about threshold exceedance or abnormal patterns.

4. Continuous intelligence is the automation of stream analytics so that actions can be taken without any human intervention.

In the sections that follow, we will look at the architecture for each of these. You will see that the architecture is quite modular, and you can get away with architecting simple systems and adding complexity as and when additional needs arise. It is not necessary to build the final, most complex, autonomous system to get value from streaming. However, this does assume that you carry out your data processing in frameworks like Apache Beam which will allow you to seamlessly move from batch to stream.

These categories build on each other, so the first one is fundamental to all four types of use cases. To make faster decisions, you need to ingest data in near real-time, as the event happens. This is called *streaming ingest*, which we will cover next.

Streaming Ingest

Streaming ingest can be done in two ways:

1. You could aggregate events and write only the aggregates (such as hourly averages) to the persistent store. This is called *streaming ETL* because the aggregation is a transformation (the T in ETL) and comes between Extraction and Loading into the persistent store.
2. Alternatively, you could ingest (Load) the data directly into a data lake or data warehouse and expect clients to transform the data at the time of analysis. This is called *streaming ELT*.

Let's take a more detailed look at these two approaches in the following subsections.

Streaming ETL

Streaming ingest is sufficient if your goal is to provide the business with more timely data. First of all you need to focus on *where* the data is ingested in the cloud because it matters a lot. You need to store the data in a location where you can access it for processing or querying. In the latter case, it is important to

ensure that any queries that are carried out on the ingested data will reflect the latest data.

To achieve the goal of real-time insights, therefore, the ingest has to happen into a system that allows real-time ingest and querying. Modern data warehouses such as Google BigQuery, AWS Redshift, and Snowflake have this capability. Therefore, commonly, you will carry out streaming ETL into such a data warehouse as described in Figure 9-2.

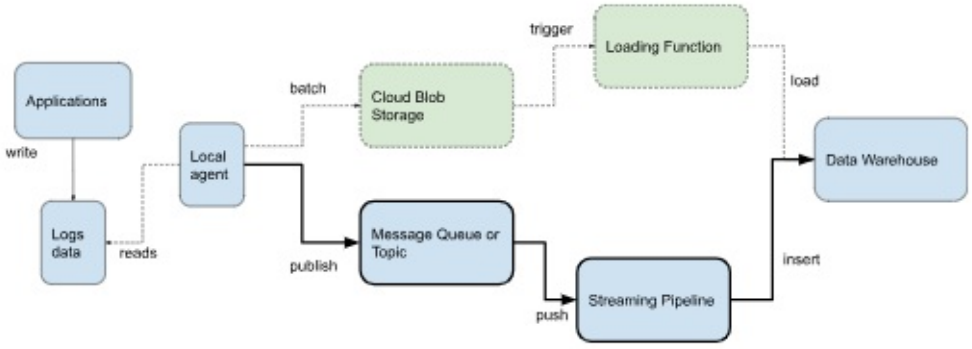


Figure 9-2. Two options for Streaming ETL of logs data: micro-batching (shown with dashed lines) or streaming pipeline (shown with thick, solid lines).

The goal of streaming ETL, therefore, is usually to land the data in a data warehouse. Log data written from multiple applications are read by a local agent that is responsible for making the log's data available for real-time monitoring and querying. You can adapt this same architecture for other types of real-time data, whether from IoT devices or from online games, by recasting what the data is, and by using an appropriate local agent.

The local agent can make the data available for real-time querying in one of two-ways (see Figure 9-2):

- **Micro-batching**. Data corresponding to, say, the last 5 minutes is written to a file on cloud blob storage. The arrival of a new file in the cloud blob storage triggers an loading function (such as Google Cloud Functions, Google Cloud Run, AWS Lambda, Azure Functions, etc.). The function can process the data and then load the file into the final destination. Micro-batching involves some latency.
- **Streaming Pipeline.** The local agent can publish an event corresponding to each log event to a message queue (such as Kafka) or a topic (such as

Cloud Pub/Sub). These events are pushed to a streaming pipeline (such as AWS Glue or Google Cloud Dataflow) that processes the events and inserts them into the final destination. This makes the event immediately available for querying.

The role of the loading function or streaming pipeline is to process the data to make it much more usable by downstream users and applications. Typically, you would leverage data processing in order to:

- Convert event data into the schema required by the data warehouse table.
- Filter event records to keep only the data specific to the business unit that operates the data warehouse.
- Interpolate or otherwise fill in missing values in events (e.g. you would use the most recently valid value).
- Connect events across time (e.g. you would leverage identity resolution methods to connect events into user sessions, and even across sessions).
- Aggregate events into more consumable chunks, for example, writing out time averages (e.g. total page visits over the past minute rather than every page visit) or per-session values (e.g, one record noting which buttons were clicked in a session rather than separate events for every button click).
- Enrich the data with additional sources (e.g. currency conversion).
- Mask, encrypt, or otherwise protect sensitive fields.

If you do not need to perform any of the aforementioned activities and the loading function is simply a pass-through, consider whether you can use the streaming ELT approach in the next section.

When leveraging the micro-batching approach, It is helpful to understand where the latency comes from. The obvious culprit is that data loaded into the data warehouse could be 5 minutes old in our example. But that is usually not the big problem. If a 5-minute latency is unacceptable, you can go down to 30 seconds. The real problem is that the data warehouse load jobs are queued, and so may not happen immediately. Check the SLA of your data warehouse for what this delay could be. Usually, this loading delay is what makes micro-batching in

stream ingest unacceptable for many use cases. It is, however, a valid architecture if the batches consist of, say, daily data and an hour's latency in loading is acceptable.

The micro-batching approach is usually less expensive than the streaming pipelines one. Some data warehouses (notably Google BigQuery) do not even charge for loading data. Even in data warehouses like Snowflake that charge for both loading and inserts, inserts are more expensive and require a higher tier. Additionally, you only pay the computational cost of loading while the function is running. Especially if you are loading data once a day, the computation cost of micro-batching can be quite minimal. Autoscaling in AWS Glue and Cloud Dataflow do close the gap as the frequency of micro-batching increases.

We've covered the first of the two approaches mentioned earlier, now let's move onto the second: Streaming ELT.

Streaming ELT

The streaming ETL approach you have learnt before assumes that you can anticipate the kind of cleanup, aggregation or enrichment that the consumer of the data will need. After all, you are transforming the raw data before writing it to the data warehouse. This could be a lossy operation. As the number of consumers of the data grows, and it becomes impossible to anticipate the kinds of transformation different consumers need, many organizations switch their data pipelines from streaming ETL to streaming ELT. Streaming ELT is also a better fit than Streaming ETL if the transformations require business-specific knowledge and is better carried out by business users rather than programmers.

In streaming ELT (see Figure 9-3), the local agent directly loads or inserts the raw data into the data warehouse.

Consumers of the data can apply whatever transformations they require. In some cases, you can provide logical or materialized views of the data to make it convenient to data consumers.

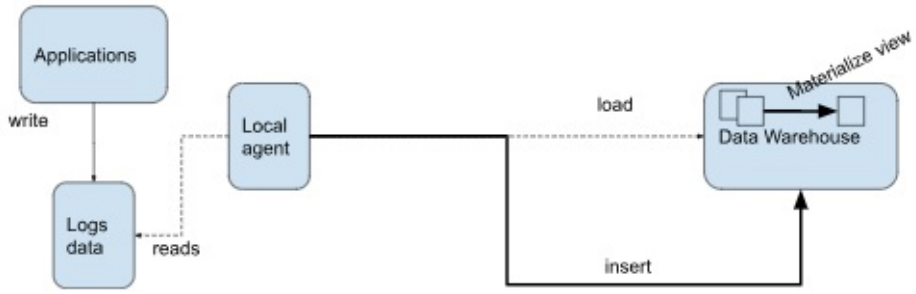


Figure 9-3. Streaming ELT can also be done in microbatches (dashed line) or as events happen (solid line).

The big drawback of streaming ELT is that the amount of data being written to the data warehouse can be large – the transformation step in many ELT pipelines significantly thins the data and writes only aggregate data to the data warehouse. Therefore, streaming ETL vs ELT is very much a decision that you make based on business value and cloud costs.

Counterintuitively, the more data you have, the more cost-competitive streaming ELT becomes. The larger the data volume gets, it makes more sense to process the data *more* frequently. To see why, imagine that a business is creating a daily report based on its website traffic and this report takes 2 hours to create. Now suppose that the website traffic grows by 4x. Then, the report will now take 8 hours to create. How do you get back to 2 hours? Well, fortunately, these are embarrassingly parallel problems. So, autoscale the job to 4x the number of machines. What if, instead, we consider an approach that makes the reports more timely?

- Compute statistics on 6 hours of data 4 times a day
- Aggregate these 6 hourly reports to create daily reports
- You can now update your “daily” report four times a day.
- The data in the report is now only 6 hours old.

This is, of course, the micro-batching idea. The computational cost of both these approaches is nearly the same. Yet, the second approach reduces latency, increases frequency, spreads out the load, and handles spikes better. Plus, the organization gets more timely, less stale reports which can provide huge business benefits for almost no extra cost. The more data you have, the more sense it makes to go from 6-hour reports to hourly updates, to minute-by-minute updates,

to real-time updates. Once you need near-real-time updates to multiple consumers, streaming ELT becomes a very attractive option.

Streaming Insert

In Figures 9-2 and 9-3, we assumed that we needed a local agent that would look at available data and load or insert the data into the persistent store. It is not necessary to have this intermediary – if the data warehouse provides a streaming API, a cloud-native application may disintermediate the local agent and use a cloud client library to do the insert itself (see Figure 9-4).

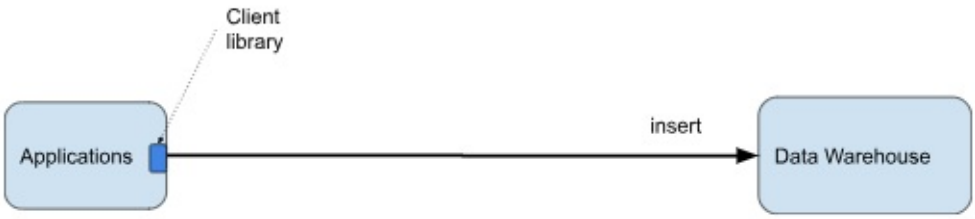


Figure 9-4. Streaming ELT in a cloud-native application can take advantage of client libraries to directly insert the data.

For example, in BigQuery, a streaming insert involves a REST API call and can be accomplished in Python using:

```
client = bigquery.Client()
…
errors = client.insert_rows_json(table_id, rows_to_insert)
```

In Redshift and Snowflake, at the time of writing, this isn't possible and you'd have to use a pass-through transformation step in Kinesis or Snowpipe respectively. It will have probably changed by the time you are reading this, so we encourage you to consult the documentation for your data warehouse.

Some messaging systems (e.g. Google Pub/Sub) also offer specific subscription types that load events into the data warehouse as they happen, so applications can simply publish events to the messaging topic or queue and have the events show up in the data warehouse in real-time.

Streaming from Edge Devices (IoT)

In Figure 9-2, we assumed that events in Streaming ETL would be published to a

general-purpose message queue such as Kafka from a custom local agent.

In the case of Internet of Things (IoT), cloud providers usually have a more targeted solution (see Figure 9-5). Azure provides IoT Devkit for edge software and IoT Hub for the remote cloud component. On AWS, the local agent will use software that has pre-built AWS IoT Greengrass components and the remote queues will be managed by AWS IoT Core. On Google Cloud Platform, the edge components might consist of Coral devices and TensorFlow Lite software while the remote cloud component might be Clearblade IoT core.

Figure 9-5. For streaming data from IoT devices on the edge, make use of IoT-specific functionality.

You can use the provided edge software SDK and devices to do local processing, data management, machine learning inference, etc. Leveraging the provided remote cloud component, you can activate bespoke functionalities such as device management, and you can transparently handle situations such as unreliable networks, device restarts, etc.

Regardless of the cloud provider, the edge software SDKs will support standard protocols such as MQTT in addition to proprietary ones. Choose a standard protocol to retain for yourself the ability to deploy software to different clouds – especially in the case of edge devices, it is quite common to end up having to support devices from other clouds or processing software on different clouds due to partnerships and acquisitions. If using Greengrass on AWS, for example, you might want to consider using the MQTT broker Moquette to ensure portability.

Streaming Sinks

In Figures 9-2 and 9-3, we assumed that we needed to land the streaming data into a data warehouse to support interactive querying. This is not necessarily the case. There are two exceptions – unstructured data and high-throughput streams.

If you are streaming video, then disregard all the above and use a framework meant for video. On the edge, your video camera will support a live streaming protocol such as Real-time Streaming Protocol or WebRTC. Use this to send the live video feed to the cloud.

On Google Cloud, Cloud Video Intelligence will convert from the live streaming protocol to a decodable video stream and write the stream to files on cloud storage.

On AWS, Kinesis Video Streams provides a similar integration with AWS Sagemaker and publishes ML inference results to Kinesis Data Streams. Similarly, Azure Video Indexer allows you to extract insights from video.

Data Warehouses are a general purpose answer for persistent storage and interactive, ad-hoc querying. However, data warehouses are not a good solution for high-throughput and/or low-latency situations. Typical throughput supported by data warehouse streaming inserts are of the order of a GB/second and typical latencies are on the order of seconds. If you want to stream terabytes of data per second or want millisecond latencies, then data warehouses are not a good choice. Use Cloud Bigtable on Google Clodu or DynamoDB on AWS instead. Of course, these are no-SQL databases and so you are trading off the convenience of SQL for the real-time ingest and querying.

It is also possible to stream to files on cloud blob storage if immediate querying is not a concern or if your architecture consists purely of a data lake rather than being a data lakehouse (see Chapter 8). Apache Beam, for example, can be used to store unstructured or semi-structured data on Google Cloud Storage. When doing so, it is important to decide how to shard the files — stream processing frameworks will automatically shard the files once the files reach a certain size, but these will be essentially based on timestamp (since records are getting streamed out) and this may not be suitable for your needs.

Real-time Dashboards

Regardless of whether you are storing aggregate data or landing real-time data into the data warehouse, you may want to provide decision makers with the ability to visualize the data as it is coming in. You can do this through *real-time dashboards*.

Live Querying

The dashboard tools periodically query the data warehouse into which events are

landed and update their graphics. This architecture requires that the dashboard push down the query to the cloud data warehouse (see Figure 9-6). In other words, all queries are “live” and reflect the latest data in the data warehouse.

Figure 9-6. Dashboards periodically query the data warehouse using SQL.

Earlier approaches such as data cubes and data marts – which involve materializing the subset or aggregate of data warehouse data used by the graphics – are no longer necessary. Dashboard tools like Tableau have the capability to create and maintain extracts. However, it is preferable to query the data warehouse live – modern cloud data warehouses can handle it.

If your data warehouse offers convenience or performance features tailored for dashboards (such as caching of datasets in memory, time-bound caching of queries, optimizations to return previous query results if the data has not changed, etc.), you should turn them on. For example, in Google BigQuery, turn on BI Engine. In AWS Redshift, use dense compute nodes because these are tailored for the heavier compute that dashboards will impose.

Materialize Some Views

It is preferable that you do not have complex SQL querying code in the dashboard tool. Create views that retrieve the required data and have the dashboard tool query the view. Reuse query logic among views through user-defined SQL functions.

If you discover that some views are accessed very often, convert the view to be a materialized view (see Figure 9-7). This provides the performance benefit of dashboard-maintained database extracts without the extra governance burden posed by datasets floating around the organization.

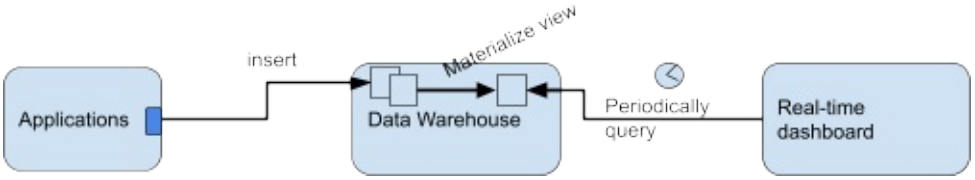


Figure 9-7. Use materialized views for frequently requested extracts.

Do not go overboard, however: minimize the use of materialized views, and allow most queries to happen live. Materialized views add storage costs, and add a lot of unnecessary expense if the particular extract is rarely displayed.

Stream Analytics

When implementing a dashboard you may want to not simply display data. You would probably display alerts, useful to the decision makers, that are based on automatically extracted insights and predictions. This is called stream analytics. You can determine if an alert is warranted via the computation of analytics on an event-by-event basis, or on a time-based schedule. Doing it as the event arrives is better, and to do this, you will need a streaming pipeline. If you choose to do it on a time-based schedule, micro-batching is sufficient.

Streaming analytics is useful in these situations:

- Time-series analytics, to track assets, predict impact of events, and do predictive maintenance.
- Click-stream analysis to make real-time offers, create dynamic websites, and optimize a customer journey.
- Anomaly detection, to predict equipment failure, prevent fraud, and monitor system health.

We’ll look at each of these respectively as we progress through this section. The architecture for these situations can serve as a template for other streaming analytics use cases that you may have – you will end up writing to both topics and dashboards as in time-series analytics, using a backfill pipeline as in clickstream analytics, or using multiple time windows as in anomaly detection.

Time Series Analytics

The most common application of stream analytics is to validate data values periodically or to compute time-based averages.

For example, suppose a physical asset (such as a delivery vehicle) streams its location to the cloud and that we would like to analyze the location of the asset and alert if:

- The asset moves out of some predetermined geographic area.
- The asset's speed is above some predetermined limit for the location that it is in.

The architecture of this use case is shown in Figure 9-8.

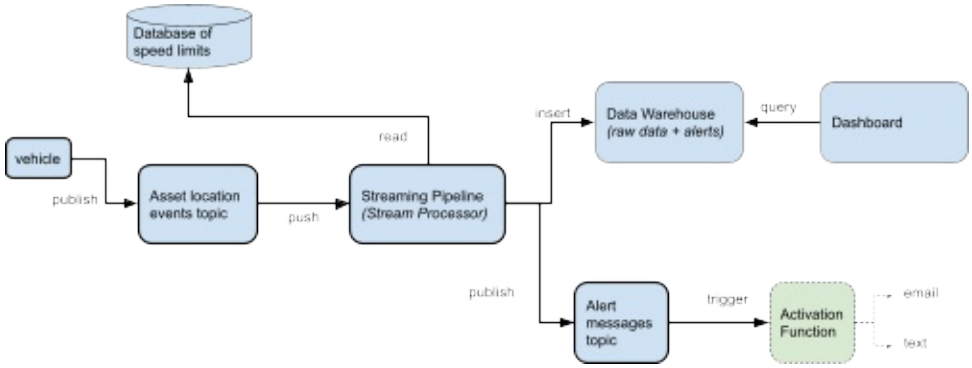


Figure 9-8. Architecture for time-series analytics.

You can land the location data in real-time into the data warehouse through an ETL pipeline. The streaming pipeline (implemented in technology such as AWS Glue, Google Cloud Dataflow, Apache Flink, etc.) processes the real-time stream, validates the location, and writes alerts to a special topic. An activation function then is triggered on new events to this topic, and takes care of sending the alerts via email, text, etc. to interested parties

The stream processor is capable of applying time windows to the incoming event stream in order to compute statistics like the average speed. It is also capable of obtaining static values from a database (static values such as the speed limit in any specific location). Therefore, it is also capable of carrying out the second computation and alerting on it as well.

It is a best practice to write alert messages not only to the alert topic, but also to the enterprise data warehouse. It is better to provide users control over what

alerts they receive, and because of this, it is better to not have the streaming pipeline directly send emails or text messages. This separation of responsibility also allows you to build dashboards that can show the frequency of such alerts and permit users to explore the alerts and hopefully recognize a pattern.

Clickstream Analytics

A clickstream consists of the sequence of events (such as button clicks, page views, etc.) carried out by visitors within an application or website. To collect this data, organizations instrument their websites so that user activities invoke a web action that, in turn, ends up in a data warehouse. As a lot of business has moved online, it has become possible for organizations to gain insights into customer behavior based on the clickstream.

While it is possible to write custom JavaScript code for such instrumentation and collect and process the data in a custom streaming pipeline, it is much more common to use prebuilt tooling such as Google Marketing Platform or Salesforce Marketing Cloud. Google Marketing Platform consists of Google Tag Manager, which is how you instrument your website, and Google Analytics which collects this information and provides a way to export clickstream data into Google BigQuery, from where you can transfer it to any other data warehouse. You can use connectors from companies such as FiveTran to similarly export data from Salesforce Marketing Cloud to the desired data warehouse.

Once the data is in a data warehouse, you can analyze it using SQL. Clickstream data is used for A/B testing, tracking changes in item popularity, and identifying sales friction. You can do all these through appropriately crafted SQL. However, the processing code will handle situations such as:

- Customers who start on one device and finish the transaction on another. The automatic session tracking may not capture this unless the customer is logged in on both devices. In general, user identification is a challenge since only a small subset of users will be logged in. Every other mechanism (cookies, device ids, IP addresses, etc.) fails quite frequently and can be improved upon if you have more data.
- Privacy compliance will often require that collected data is appropriately

anonymized, and user data aggregated in such a way that individual actions can not be identified. It is quite common that you will have to redact information from text fields filled out by users.

- Activity by automated agents such as spiders (search bots) and bad actors looking for security vulnerabilities.

This kind of processing is hard to do in SQL. Because of these situations, even when prebuilt tooling is used, it is common to build a post-processing "backfill" pipeline to handle cases that are unique to your organization (see Figure 9-9). This streaming pipeline might be able to do a better job of identity stitching, privacy aggregation and bot detection than the more general-purpose code provided by the prebuilt tools. At the same time, the pipeline might be able to enrich the clickstream feed based on other data sources within the organization (such as information about customers, products, prices, inventory levels, etc.). It is this backfilled, post-processed data that you can use for further analysis.

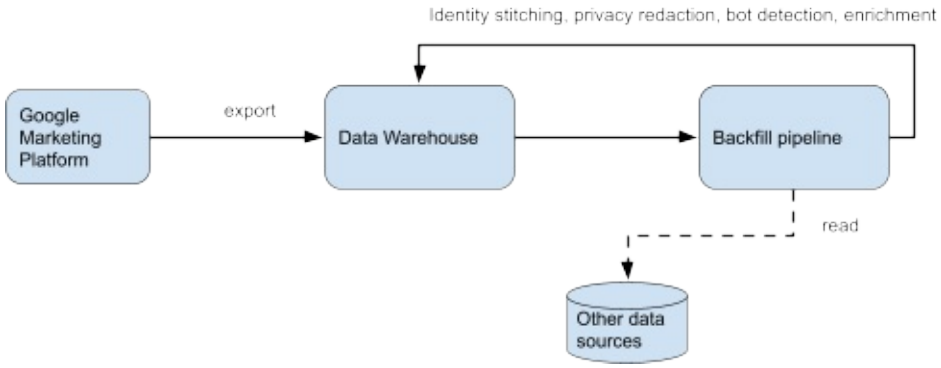


Figure 9-9. Use a backfill pipeline to enrich clickstream data and improve identify stitching, privacy redaction, and bot detection.

If you use the clickstream data to build personalization or recommendation systems, it is necessary to also take actions while the customer is on the website. This will fall under the continuous intelligence use case which we cover later in this chapter.

Anomaly Detection

Anomaly detection involves identifying unusual patterns as the data is arriving. In any situation where you have a lot of data on what "usual" behavior looks like, but it is hard to write specific rules on what abnormal activity looks like

(because bad actors keep changing their attack mechanism, or because the environment is subject to change), anomaly detection can be very useful. Anomaly detection is used to detect mispriced items (all of a sudden, the popularity of an item shoots up), detect overloaded equipment, bot activity in online games, security threats, etc.

A signature-based pattern is the primary technique used by many organizations to identify viruses and other security threats. In a signature based pattern the system leverages repositories of viruses detected in the past against new menaces. However, it is difficult to detect new attacks using this technique because no pattern or signature is available. When using anomaly detection, you usually cluster incoming data over, say, the past three days (see Figure 9-10). Any events that are far from existing clusters are assumed to be suspect. This allows for the environment to adapt (since clustering is done only on the most recent 3 days of data), and allows you to identify abnormal patterns as long as they are not already too widespread.

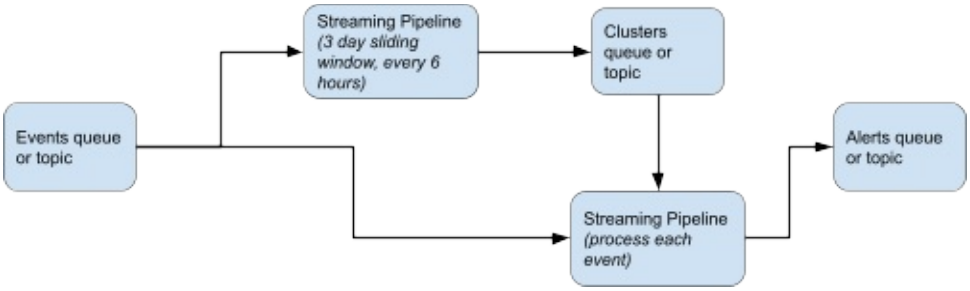


Figure 9-10. Anomaly detection involves two streaming pipelines. The first pipeline applies a sliding window and computes clusters within that sliding window. The second pipeline compares incoming events against the current set of clusters and raises alerts.

Resilient Streaming

When ingesting and processing streaming data, there will be malformed data that you receive, or unexpected data values that you do not know how to process.

In batch processing, it is common to simply throw an exception and expect the programmer to find the logic error, fix it, and rerun the batch job. However, in stream processing, the pipeline needs to continue running. However, you also don’t want to just ignore errors.

Therefore, it is crucial that every streaming analytics job sets up a dead-letter queue to store any events that were unable to be processed. You can periodically

inspect these dead-letter queues, and (as with batch jobs) fix the logical errors.

Once you have fixed the logic error, you have to update the currently running streaming pipeline without any data drops. Tools like Cloud Dataflow provide the ability to *update* a running pipeline in place or to *drain* the event queue and seamlessly transfer processing to a new pipeline.

Updating a running pipeline keeps in-flight data and resumes processing within the new pipeline. To do this, it reuses the persistent store from the old pipeline for the updated pipeline. As a result, the two streaming pipelines must meet certain compatibility requirements. If you are in a situation where compatibility is maintained, you should follow this approach because you can achieve exactly-once semantics. Events will be processed exactly once (i.e., either the old one or the new one), and aggregates will be accurate.

If the pipelines are not compatible (perhaps because the bug fix changed the data transmitted between steps), the next best approach is to drain the existing pipeline before starting the new pipeline (see Figure 9-11). The existing job stops pulling from its input sources and completes processing of all in-flight and buffered data, causes triggers to emit the contents of open windows and sends new events to the new pipeline. Draining pipelines is essential to ensure at-least-once processing semantics – this is not as good as exactly-once, but it is better than simply canceling a running pipeline and throwing away data.

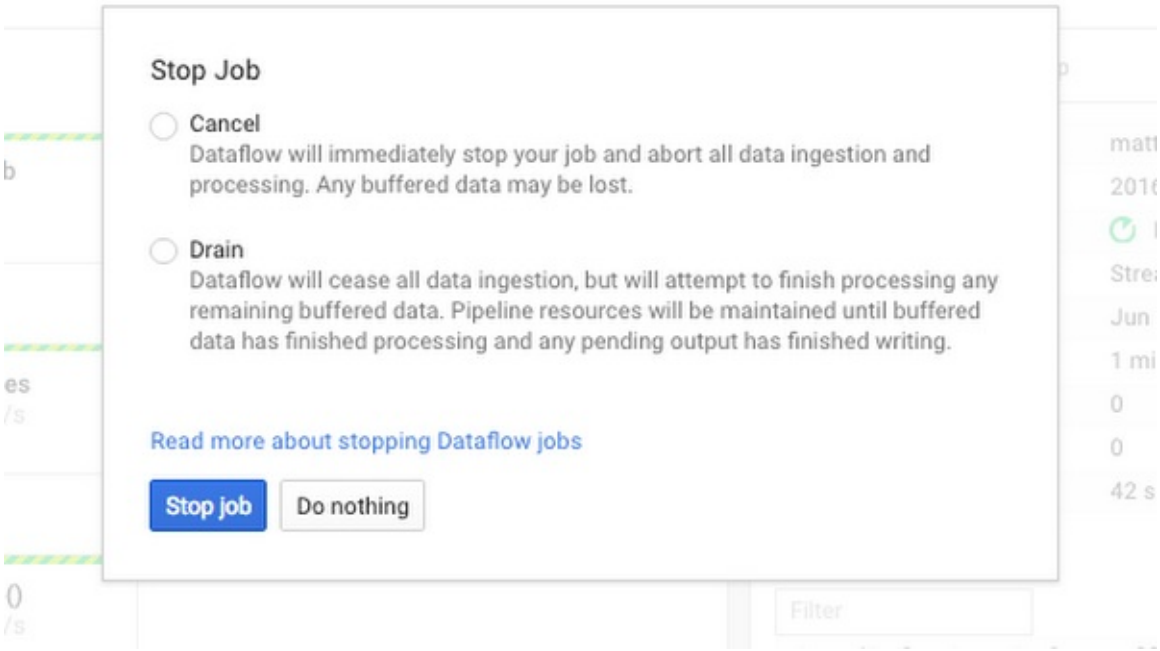


Figure 9-11. Some stream processing engines provide the ability to replace a running pipeline with another with exactly-once and/or at-least-once semantics.

Continuous Intelligence through ML

It is not necessary to have a human in the loop to make decisions. As the amount of data grows, it is very common to move to a system of "*human over the loop*". In this situation, actions are automatically carried out in response to real-time insights, alerts, and predictions. A human supervisor can override an action that would otherwise be automatically applied, but a system is designed to operate autonomously without any human intervention. This is known as continuous intelligence.

To enable the system to operate autonomously, you will need to automate the following:

- Train a machine learning (ML) model on historical data, and retrain the model on subsequent data if necessary.
- Invoke the trained ML model as events come in. This is called *inference*.
- Take actions based on the prediction of the ML model.

Let’s look at some considerations for each of these steps.

Training model on streaming data

ML models are trained on historical data. How much data should you train the model on? This depends on the problem at hand. The general guidance is that you should only train the model on data that is similar to what it will encounter once it is put into production. Moreover, data from several years ago is often from a completely different context (see Figure 9-12) and may capture trends and relationships that are not relevant anymore today.

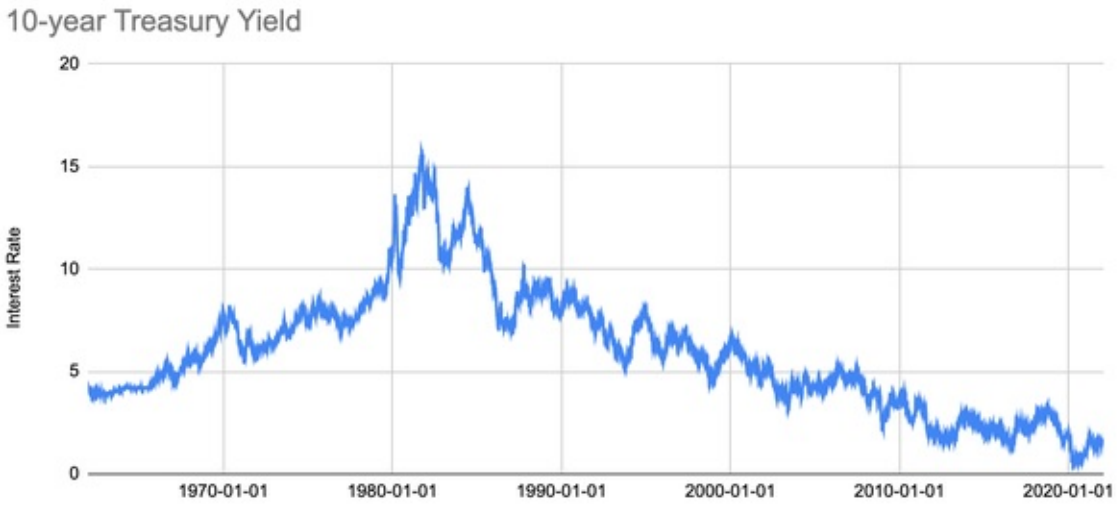


Figure 9-12. In many short-term financial predictions, data from the 1980s is typically not valid anymore because the operating context (the interest rate environment) was very different.

Therefore, when training an ML model on streaming data, you are unlikely to train the model on the entire historical archive. Instead, you are interested in training on recent data where “recent” refers to data that has the same characteristics as data that you are about to receive. “Recent” in this context could be the past 3 days (as in our anomaly detection example, shown in Figure 9-10) or the past 3 years in unchanging environments.

Windowed Training

If you are training very frequently and the training data consists of relatively small time periods, you can use a sliding window streaming pipeline as in Figure 9-13. This is extremely common when you are extrapolating time series

(such as doing demand prediction solely on the historical demand cycle).

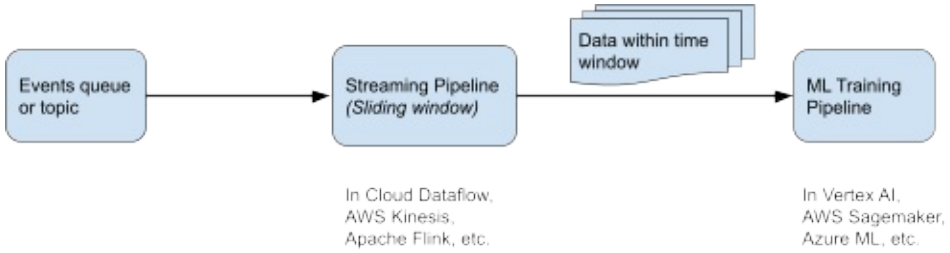


Figure 9-13. Training on events within a sliding time window.

What you need for this is:

- A streaming pipeline to create a dataset consisting of data within the time window
- An automated training pipeline in Google Cloud Vertex AI, AWS Sagemaker, Azure ML, etc. that is parameterized in terms of where it obtains the training data.
- The training pipeline also deploys the model to an endpoint, as we will discuss in the "*Streaming ML inference*" section

Note that the word pipeline here refers to different things. The streaming pipeline involves processing of data in motion whereas the training pipeline consists of ML operations (preprocessing data, training model, evaluating model, deploying model, etc.). We will discuss in greater detail ML Pipelines in Chapter 12.

Scheduled Training

For situations where a trained model will be valid for a moderately long time period (on the order of days to weeks), you can use a scheduled job to initiate training as described in Figure 9-14. The training job will retrieve data from the data warehouse or other persistent store corresponding to the past month, for example.

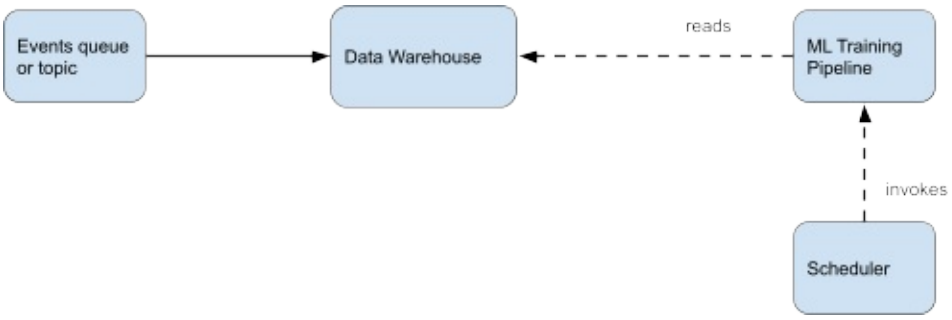


Figure 9-14. Scheduled Training.

You can schedule a Vertex AI ML training pipeline on Google Cloud using Google Cloud Scheduler. On AWS, scheduling a Sagemaker pipeline is a supported target in AWS EventBridge. On Azure, Azure ML Pipelines support scheduled triggers (as a pipeline setting), and so don't need an external scheduler to invoke it.

We strongly discourage leaving models going more than a few weeks without changing the model in a streaming pipeline. If you believe the model will continue to remain valid, validate your intuition by continuously evaluating the model and retraining it if necessary. We will discuss this in the next section.

Continuous Evaluation and Retraining

The most complex situation is using a model until you determine it no longer fits for purpose. To determine that the model has *drifted* in performance, you will need to employ *continuous evaluation*. For example, if you have a model to predict whether a user will buy an item, you could verify a few days later if the user has bought the item in question. You can then carry out a weekly evaluation of the model based on predictions made two weeks ago and for which the true answer is now available. When the evaluation metric drops below some preset threshold, the model can be retrained (see Figure 9-15).

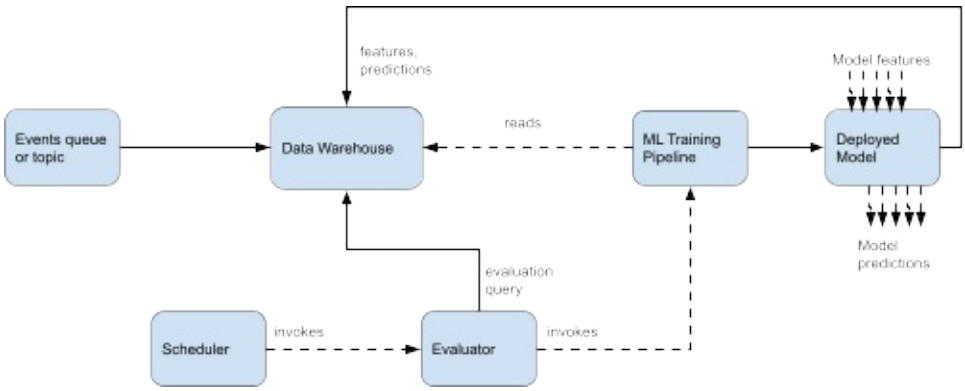


Figure 9-15. Continuous Evaluation to automatically initiate retraining.

You can also extend the continuous evaluation approach to detect *feature drift* – if the distribution of any of the inputs to the ML model changes (for example, if the number of repeat purchases was 10% of all purchases, but has now increased to 20% of purchases), you might want to retrain the model.

At the time of writing, only Vertex AI supported setting up continuous evaluation queries and detection of feature drift on deployed models. To set this up in Vertex AI, you will need to define an evaluation query and turn on the capability of deployed models to write out a sample of features and corresponding predictions to the data warehouse. Periodically run the evaluation query, and use the resulting metric to determine whether the pipeline needs to be invoked.

Consult your cloud provider documentation on whether this scenario is now supported. If it is, the mechanism is likely to be somewhat similar. If not, you will have to build the corresponding pipelines and capabilities in a bespoke way.

Streaming ML Inference

Normally, you invoke the trained machine learning model as events come in and obtain ML predictions for those events.

It is possible to load the model object into the streaming pipeline and call the prediction signature of the model. This is usually the most efficient way to invoke ML predictions. However, it doesn't scale beyond small models and projects.

For example, the ML prediction may be required by non-Python code, and by programs that are running on hardware that doesn't have GPUs. To handle these situations, the model is usually deployed to an endpoint so that it can be invoked as a microservice. The model will then be invoked by sending a HTTP request to it with the input to the ML model being the payload.

ML inference is not efficient if it is invoked one event at a time. Because modern machine learning frameworks are based on matrix operations, they are much more efficient if you pass in a small batch of events for inference. This is what is shown in Figure 9-16.

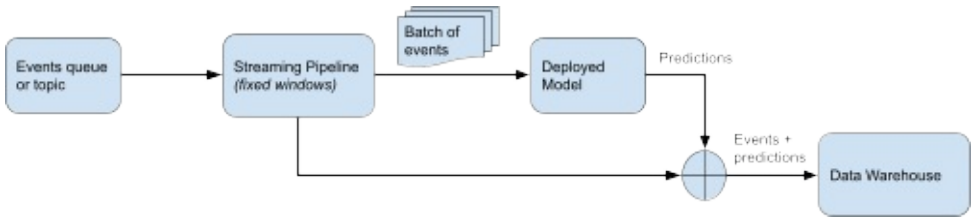


Figure 9-16. Streaming Inference typically involves accumulating a batch of events and sending them to a deployed model for inference.

Automated Actions

If all that is required is that human users be able to view the predictions of the ML model and make decisions, it is sufficient to land the predictions in a data warehouse, as shown in Figure 9-16. However, what if you need the system to autonomously take some action based on the prediction of the ML model?

You could use an ML model for a task that requires automation, such as to automatically issue a coupon to the person who's likely to abandon the shopping cart, or to create a ticket to change an about-to-fail part on a machine. You can invoke these actions in a serverless way through cloud functions like Lambda, Fargate, Google Cloud Functions, Google Cloud Run, or Azure Functions.

To support this, you will need to write out a subset of the enriched events meeting alerting criteria to a place from which you can trigger the cloud function (see Figure 9-17).

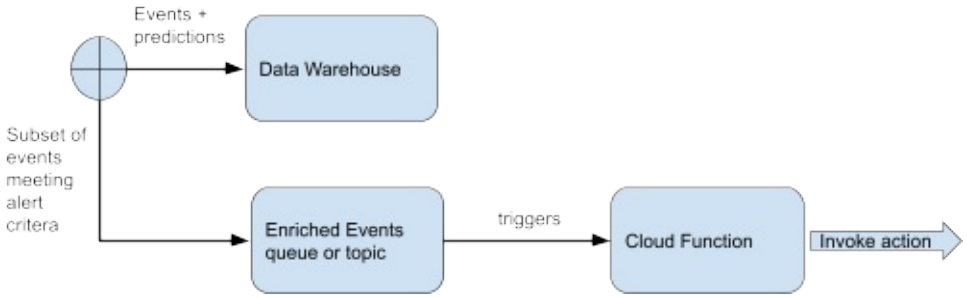


Figure 9-17. Supporting automated actions - this diagram is a continuation of Figure 9-16.

We have looked at a number of streaming use cases and scenarios and how they can be architected and implemented using cloud-native technologies. Compose the architecture for your own streaming solution based on what your needs are.

For example, if you will be doing machine learning on streaming data, you will choose one of the training architectures in Figure 9-13, 9-14 and 9-15 and

combine it with the inference architecture of either Figure 9-16 or 9-17. If you will be doing only analytics, do not complicate the architecture – see if you can get away with Figure 9-8, and add a backfill architecture (Figure 9-9) only if necessary.

Summary

This chapter focused on cloud-native streaming architectures where you understood that they are highly modular and allow you to start small and add capabilities only when necessary. The key takeaways are:

- In many industries, being able to make decisions while the event is proceeding is significantly more valuable than making the decision even a few minutes later.
- Streaming architectures are quite modular, and you can get away with architecting simple systems and adding complexity as and when additional needs arise.
- Streaming ingest is sufficient if the goal is to provide the business with more timely data.
- Micro-batching is usually less expensive than streaming pipelines but streaming pipelines make the event immediately available for querying whereas there is latency (greater than the chunking frequency) involved in micro-batching.
- As the number of consumers of the data grows, and it becomes impossible to anticipate the kinds of transformation different consumers need, many organizations switch their data pipelines from streaming ETL to streaming ELT.
- If the data warehouse provides a streaming API, a cloud-native application may disintermediate the local agent and use a cloud client library to do the insert itself.
- For streaming data from IoT devices on the edge, make use of IoT-specific functionality such as edge software SDKs, edge hardware devices, and IoT

Core.

- Data warehouses are not a good solution for high-throughput and/or low-latency situations. In such situations, use no-SQL analytics stores such as Cloud Bigtable or DynamoDB.
- Have your dashboard tool push down the query to the cloud data warehouse, create views for reuse, and materialize some views to optimize performance/cost.
- In time-series analytics, write alerts to an alert topic and to the data warehouse to support both autonomous actions and human exploration.
- Use a pre-built tool for clickstream analytics, but supplement it with a backfill pipeline to enrich clickstream data and improve identify stitching, privacy redaction, and bot detection.
- Anomaly detection involves two streaming pipelines, one to compute clusters over long time periods, and the other to compare incoming events to the most recent clusters and raise an alert.
- For resilient streaming, make sure that you update running pipelines. If updating is not possible, drain them. Do not simply cancel a running production pipeline.
- If you are training very frequently and the training data consists of relatively small time periods, use a sliding window streaming pipeline to supply training data to the ML training pipeline.
- For situations where a trained model will be valid for days to weeks, a scheduled job can be used to initiate training.
- To determine whether a deployed model has drifted in performance, you will need to employ continuous evaluation.
- Because modern machine learning frameworks are based on matrix operations, they are much more efficient if you pass in a small batch of events for inference.
- To support autonomous actions, you will need to write out a subset of the enriched events (events + predictions) meeting alerting criteria to a topic on

which a cloud function is triggered.

In the following chapter you will see an overview of approaches and techniques for distributed architectures focusing on edge computing, a pattern that can reduce latency, increase security, and lower costs.

About the Authors

Marco Tranquillin is leading a Principal Architect and Customer Engineering team at Google Cloud who helps Italian financial and insurance firms to adopt and leverage cloud data technologies to solve business problems. In the past he led the European Data Analytics practice within Google Cloud and has more than 10 years of experience working in complex IT cloud projects for many global firms.

Valliappa Lakshmanan works with management and data teams across a range of industries to help them employ data and AI-driven innovation to grow their businesses and increase value. Prior to this, Lak was the Director for Data Analytics and AI Solutions on Google Cloud and a Research Scientist at NOAA. He is a co-author of *Data Science on the Google Cloud Platform*, *BigQuery: The Definitive Guide*, and *Machine Learning Design Patterns*, all published by O'Reilly.

Firat Tekiner is an adjunct professor at the University of Manchester and a Senior Product Manager in Google Cloud. Firat has over 20 years of experience in designing and delivering bespoke information systems for some of the world's largest research, education, telecommunications, finance and retail organizations. Following roles within National Supercomputing Services and National Centre for Text Mining, he has over 30 publications in the areas of Parallel Computing, Big Data, Artificial Intelligence and Computer Communications.