Data Engineering with Apache Spark, Delta Lake, and Lakehouse

Create scalable pipelines that ingest, curate, and aggregate complex data in a timely and secure way

Manoj Kukreja

Foreword by Danil Zburivsky

Senior Consultant at Sourced Group Data Engineering and Cloud Infrastructure Specialist, Ottawa, Ontario, Canada

Data Engineering with Apache Spark, Delta Lake, and Lakehouse

Create scalable pipelines that ingest, curate, and aggregate complex data in a timely and secure way

Manoj Kukreja

Packt

BIRMINGHAM—MUMBAI

Data Engineering with Apache Spark, Delta Lake, and Lakehouse

Group Product Manager: Kunal Parikh

Publishing Product Manager: Sunith Shetty

Senior Editor: Davud Sugarman

Content Development Editor: Joseph Sunil

Technical Editor: Sonam Pandey

Copy Editor: Safis Editing

Project Coordinator: Aparna Ravikumar Nair

Proofreader: Safis Editing

Indexer: Tejal Daruwale Soni

Production Designer: Prashant Ghare

First published: October 2021

Production reference: 1170921

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-80107-774-3

`www.packt.com`

In memory of my two fathers, my mother, and my father-in-law, whom I lost recently. Hope you keep showering your blessings from far and beyond. To my darling wife Roopika, a simple thank you is not good enough. To my daughter Saesha who inspired me to write this book, and to my other daughter Shagun who is living my other dream.

I love you all.

– Manoj Kukreja

Foreword

I don't think it is required anymore to explain the importance data plays in the modern world. Many of the online services we use today are powered by data systems of varying complexity. Machine learning emerged as a powerful tool for certain tasks in big part thanks to modern data platforms that can collect, process, and distribute large volumes of data. These data platforms also provide a strong foundation for data analytics and help drive real world decisions in healthcare, business, and education.

One of the challenges that I have faced when I have started working in data space, which I'm sure is familiar to many of you, is the lack of good books and other educational resources on the topic. There is a lot of documentation on specific tools that you can use, but knowledge of what good architectures look like and which patterns work in which specific cases only resides with a few experts who have done this type of work before. This is especially true when it comes to emerging cloud technologies.

Manoj is one of such experts since he has been implementing data platforms for companies in different industries for the last 10 years. In this book, he will share his knowledge and experience on architecting and implementing modern data platforms using Microsoft Azure, one of the leading cloud providers for data workloads. You will learn practical skills on implementing data lake architectures, implementing data pipelines, and running them in a production environment. Books like this are useful because they give you a download of the knowledge and experience that Manoj has accumulated over the years and you know that recipes and advice he gives will work in a real-life implementation because he has implemented similar systems many times before.

Getting practical data engineering skills is a great investment in your career and knowing patterns that you can rely on will spare you lots of trial and error and will help you get results much faster. Hope you enjoy it!

Danil Zburivsky,

Senior Consultant at Sourced Group

Data Engineering and Cloud Infrastructure specialist

Ottawa, Ontario, Canada

Contributors

About the author

Manoj Kukreja is a Principal Architect at Northbay Solutions who specializes in creating complex Data Lakes and Data Analytics Pipelines for large-scale organizations such as banks, insurance companies, universities, and US/Canadian government agencies. Previously, he worked for Pythian, a large managed service provider where he was leading the MySQL and MongoDB DBA group and supporting large-scale data infrastructure for enterprises across the globe. With over 25 years of IT experience, he has delivered Data Lake solutions using all major cloud providers including AWS, Azure, GCP, and Alibaba Cloud. On weekends, he trains groups of aspiring Data Engineers and Data Scientists on Hadoop, Spark, Kafka and Data Analytics on AWS and Azure Cloud.

About the reviewers

Rajeesh Punathil is currently a Data Architect with a prominent energy company based in Canada. He leads the digital team in building cloud data analytics solutions aimed to help businesses arrive at right questions and answers.In prior roles he has designed data analytics pipelines in AWS and Azure stack for major companies like Sunlife, CIBC , City National bank, Hudsons Bay etc. He holds a B Tech in Electronics and Telecom Engineering from University of Calicut and has over 15 Years of IT experience. He consider himself as a forever student, and his constant striving for knowledge has made him an expert and leader in the field of data analytics and big data. He has completed AWS Big Data Analytics certification to his credit.

Dang Trung Anh is a principal engineer and software architect with over 15 years of experience in delivering big software systems for various customers across many areas. He has spent most of his career writing many software systems that support millions of users. He has had various roles from an ordinary full-stack developer, through software architect, to director of engineering. He is the author of many viral articles related to software engineering on Medium.

Table of Contents

Preface

In the world of ever-changing data and ever-evolving schemas, it is important to build data pipelines that can auto-adjust to changes. This book will help you build scalable data platforms that managers, data scientists, and data analysts can rely on.

Starting with an introduction to data engineering, along with its key concepts and architectures, this book will show you how to use Microsoft Azure cloud services effectively for data engineering. You'll cover data lake design patterns and the different stages through which the data needs to flow in a typical data lake. Once you've explored the main features of Delta Lake to build data lakes with fast performance and governance in mind, you'll advance to implementing the lambda architecture using Delta Lake. Packed with practical examples and code snippets, this book takes you through real-world examples based on production scenarios faced by the author in his 10 years of experience working with big data. Finally, you'll cover data lake deployment strategies that play an important role in provisioning cloud resources and deploying data pipelines in a repeatable and continuous way.

By the end of this data engineering book, you'll have learned how to effectively deal with ever-changing data and create scalable data pipelines to streamline data science, ML, and **artificial intelligence** (**AI**) tasks.

Who this book is for

This book is for aspiring data engineers and data analysts who are new to the world of data engineering and are looking for a practical guide to building scalable data platforms. If you already work with PySpark and want to use Delta Lake for data engineering, you'll find this book useful. Basic knowledge of Python, Spark, and SQL is expected.

What this book covers

Chapter 1, The Story of Data Engineering and Analytics, introduces the core concepts of data engineering. It introduces you to the two data processing architectures in big data – Lambda and Kappa.

Chapter 2, Discovering Storage and Compute Data Lake Architectures, introduces one of the most important concepts in data engineering – segregating storage and compute layers. By following this principle, you will be introduced to the idea of building data lakes. An understanding of this key principle will lay the foundation for your understanding of the modern-day data lake design patterns discussed later in the book.

Chapter 3, Data Engineering on Microsoft Azure, introduces the world of data engineering on the Microsoft Azure cloud platform. It will familiarize you with all the Azure tools and services that play a major role in the Azure data engineering ecosystem. These tools and services will be used throughout the book for all practical examples.

Chapter 4, Understanding Data Pipelines, introduces you to the idea of data pipelines. This chapter further enhances your knowledge of the various stages of data engineering and how data pipelines can enhance efficiency by integrating individual components together and running them in a streamlined fashion.

Chapter 5, Data Collection Stage – The Bronze Layer, guides us in building a data lake using the Lakehouse architecture. We will start with data collection and the development of the bronze layer.

Chapter 6, Understanding Delta Lake, introduces Delta Lake and helps you quickly explore the main features of Delta Lake. Understanding Delta Lake's features is an integral skill for a data engineering professional who would like to build data lakes with data freshness, fast performance, and governance in mind. We will also be talking about the Lakehouse architecture in detail.

Chapter 7, Data Curation Stage – The Silver Layer, continues our building of a data lake. The focus of this chapter will be on data cleansing, standardization, and building the silver layer using Delta Lake.

Chapter 8, Data Aggregation Stage – The Gold Layer, continues our building a data lake. The focus of this chapter will be on data aggregation and building the gold layer.

Chapter 9, Deploying and Monitoring Pipelines in Production, explains how to effectively manage data pipelines running in production. We will explore data pipeline management from an operational perspective and cover security, performance management, and monitoring.

Chapter 10, Solving Data Engineering Challenges, lists the major challenges experienced by data engineering professionals. Various use cases will be covered in this chapter and a challenge will be offered. We will deep dive into the effective handling of the challenge, explaining its resolution using code snippets and examples.

Chapter 11, Infrastructure Provisioning, teaches you the basics of infrastructure provisioning using Terraform. Using Terraform, we will provision the cloud resources on Microsoft Azure that are required for running a data pipeline.

Chapter 12, Continuous Integration and Deployment of Data Pipelines, introduces the idea of **continuous integration and deployment** (**CI/CD**) of data pipelines. Using the principles of CI/CD, data engineering professionals can rapidly deploy new data pipelines/ changes to existing data pipelines in a repeatable fashion.

To get the most out of this book

You will need a Microsoft Azure account.

Software/hardware covered in the book	**Operating system requirements**
Azure	Windows, macOS, or Linux

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Do ensure that you close all instances of Azure after you have run your code, so that your costs are minimized.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Data-Engineering-with-Apache-Spark-Delta-Lake-and-Lakehouse`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots and diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781801077743_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

Code in text: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "In the case of the `financial_df` DataFrame, the index was auto-generated when we downloaded the dataset with the `read_csv` function."

A block of code is set as follows:

```
const df = new DataFrame({...})
df.plot("my_div_id").<chart type>
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
...
var config = {
            displayModeBar: true,
            modeBarButtonsToAdd: [
...
```

Any command-line input or output is written as follows:

```
npm install @tensorflow/tfjs
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "In Microsoft Edge, open the **Edge** menu in the upper right-hand corner of the browser window and select **F12 Developer Tools**."

Tips or important notes

Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Section 1: Modern Data Engineering and Tools

This section introduces you to the world of data engineering. It gives you an understanding of data engineering concepts and architectures. Furthermore, it educates you on how to effectively utilize the Microsoft Azure cloud services for data engineering.

This section contains the following chapters:

- *Chapter 1, The Story of Data Engineering and Analytics*
- *Chapter 2, Discovering Storage and Compute Data Lake Architectures*
- *Chapter 3, Data Engineering on Microsoft Azure*

1
The Story of Data Engineering and Analytics

Every byte of data has a story to tell. The real question is whether the story is being narrated accurately, securely, and efficiently. In the modern world, data makes a journey of its own—from the point it gets created to the point a user consumes it for their analytical requirements.

But what makes the journey of data today so special and different compared to before? After all, **Extract, Transform, Load** (**ETL**) is not something that recently got invented. In fact, I remember collecting and transforming data since the time I joined the world of **information technology** (**IT**) just over 25 years ago.

In this chapter, we will discuss some reasons why an effective data engineering practice has a profound impact on data analytics.

In this chapter, we will cover the following topics:

- The journey of data
- Exploring the evolution of data analytics
- The monetary power of data

Remember:

the road to effective data analytics leads through effective data engineering.

The journey of data

Data engineering is the vehicle that makes the journey of data possible, secure, durable, and timely. A **data engineer** is the driver of this vehicle who safely maneuvers the vehicle around various roadblocks along the way without compromising the safety of its passengers. Waiting at the end of the road are **data analysts**, **data scientists**, and **business intelligence (BI) engineers** who are eager to receive this data and start narrating the **story of data**. You can see this reflected in the following screenshot:

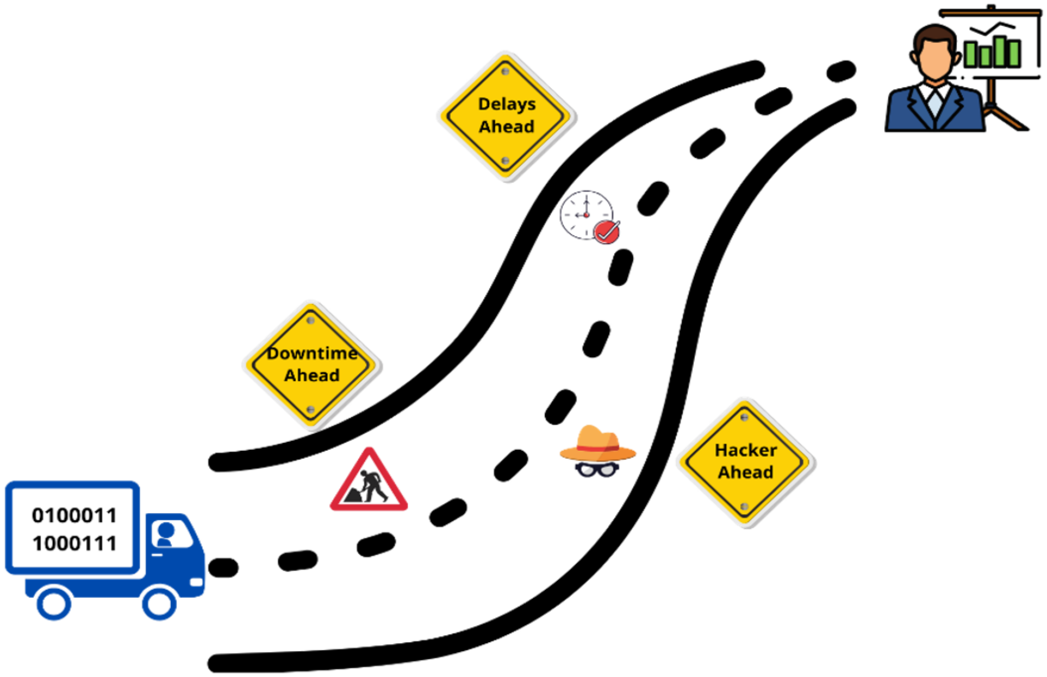


Figure 1.1 – Data's journey to effective data analysis

Traditionally, the journey of data revolved around the typical ETL process. Unfortunately, the traditional ETL process is simply not enough in the modern era anymore. Due to the immense human dependency on data, there is a greater need than ever to streamline the journey of data by using cutting-edge architectures, frameworks, and tools.

You may also be wondering why the journey of data is even required. Gone are the days where datasets were limited, computing power was scarce, and the scope of data analytics was very limited. We now live in a fast-paced world where decision-making needs to be done at lightning speeds using data that is changing by the second. Let's look at how the evolution of data analytics has impacted data engineering.

Exploring the evolution of data analytics

Data analytics has evolved over time, enabling us to do bigger and better. For many years, the focus of data analytics was limited to **descriptive analysis**, where the focus was to gain useful **business insights** from data, in the form of a report. This type of analysis was useful to answer question such as "*What happened?*". A hypothetical scenario would be that the sales of a company sharply declined within the last quarter.

Very quickly, everyone started to realize that there were several other indicators available for finding out *what happened*, but it was the *why it happened* that everyone was after. The core analytics now shifted toward **diagnostic analysis**, where the focus is to identify anomalies in data to ascertain the reasons for certain outcomes. An example scenario would be that the sales of a company sharply declined in the last quarter because there was a serious drop in inventory levels, arising due to floods in the manufacturing units of the suppliers. This form of analysis further enhances the decision support mechanisms for users, as illustrated in the following diagram:

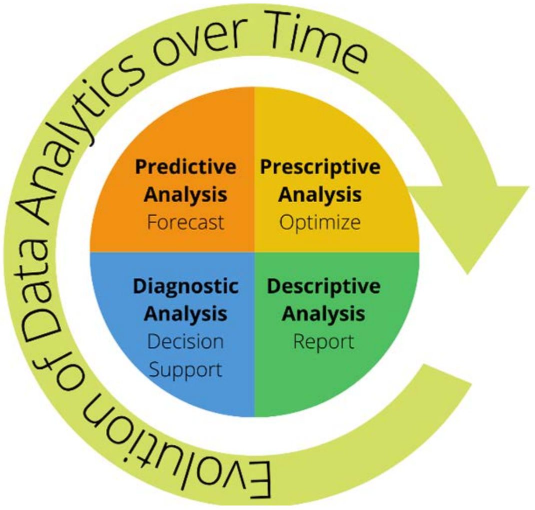


Figure 1.2 – The evolution of data analytics

Important note

Both descriptive analysis and diagnostic analysis try to impact the decision-making process using factual data only.

Since the advent of time, it has always been a core human desire to look beyond the present and try to forecast the future. If we can predict future outcomes, we can surely make a lot of better decisions, and so the era of **predictive analysis** dawned, where the focus revolves around "*What will happen in the future?*". Predictive analysis can be performed using **machine learning** (**ML**) algorithms—let the machine learn from existing and future data in a repeated fashion so that it can identify a pattern that enables it to predict future trends accurately.

Now that we are well set up to forecast future outcomes, we must use and optimize the outcomes of this predictive analysis. Based on the results of predictive analysis, the aim of **prescriptive analysis** is to provide a set of prescribed actions that can help meet business goals.

Important note

Unlike descriptive and diagnostic analysis, predictive and prescriptive analysis try to impact the decision-making process, using both factual and statistical data.

But how can the dreams of modern-day analysis be effectively realized? After all, data analysts and data scientists are not adequately skilled to collect, clean, and transform the vast amount of ever-increasing and changing datasets.

The data engineering practice is commonly referred to as the **primary support** for modern-day data analytics' needs.

The following are some major reasons as to why a strong data engineering practice is becoming an absolutely unignorable necessity for today's businesses:

- Core capabilities of compute and storage resources
- Availability of varying datasets
- The paradigm shift to distributed computing
- Adoption of cloud computing
- Data storytelling

We'll explore each of these in the following subsections.

Important note

Having a strong data engineering practice ensures the needs of modern analytics are met in terms of durability, performance, and scalability.

Core capabilities of storage and compute resources

25 years ago, I had an opportunity to buy a Sun Solaris server—128 **megabytes** (**MB**) **random-access memory** (**RAM**), 2 **gigabytes** (**GB**) storage—for close to $ 25K. The intended use of the server was to run a client/server application over an Oracle database in production. Given the high price of storage and compute resources, I had to enforce strict countermeasures to appropriately balance the demands of **online transaction processing** (**OLTP**) and **online analytical processing** (**OLAP**) of my users. One such limitation was implementing strict timings for when these programs could be run; otherwise, they ended up using all available power and slowing down everyone else.

Today, you can buy a server with 64 GB RAM and several **terabytes** (**TB**) of storage at one-fifth the price. The extra power available can do wonders for us. Multiple storage and compute units can now be procured just for data analytics workloads. The extra power available enables users to run their workloads whenever they like, however they like. In fact, it is very common these days to run analytical workloads on a continuous basis using data streams, also known as **stream processing**.

The installation, management, and monitoring of multiple compute and storage units requires a well-designed data pipeline, which is often achieved through a data engineering practice.

Availability of varying datasets

A few years ago, the scope of data analytics was extremely limited. Performing data analytics simply meant reading data from databases and/or files, denormalizing the joins, and making it available for descriptive analysis. The structure of data was largely known and rarely varied over time.

We live in a different world now; not only do we produce more data, but the variety of data has increased over time. In addition to collecting the usual data from databases and files, it is common these days to collect data from social networking, website visits, infrastructure logs' media, and so on, as depicted in the following screenshot:

Figure 1.3 – Variety of data increases the accuracy of data analytics

> **Important note**
> More variety of data means that data analysts have multiple dimensions to perform descriptive, diagnostic, predictive, or prescriptive analysis.

Naturally, the varying degrees of datasets injects a level of complexity into the data collection and processing process. On the flip side, it hugely impacts the accuracy of the decision-making process as well as the prediction of future trends. A well-designed data engineering practice can easily deal with the given complexity.

The paradigm shift to distributed computing

The traditional data processing approach used over the last few years was largely singular in nature. To process data, you had to create a program that collected all required data for processing—typically from a database—followed by processing it in a single thread. This type of processing is also referred to as **data-to-code** processing. Unfortunately, there are several drawbacks to this approach, as outlined here:

- Since vast amounts of data travel to the code for processing, at times this causes heavy network congestion. Since a network is a shared resource, users who are currently active may start to complain about network slowness.

- Being a single-threaded operation means the execution time is directly proportional to the data. Therefore, the growth of data typically means the process will take longer to finish. This could end up significantly impacting and/or delaying the decision-making process, therefore rendering the data analytics useless at times.
- Something as minor as a network glitch or machine failure requires the entire program cycle to be restarted, as illustrated in the following diagram:

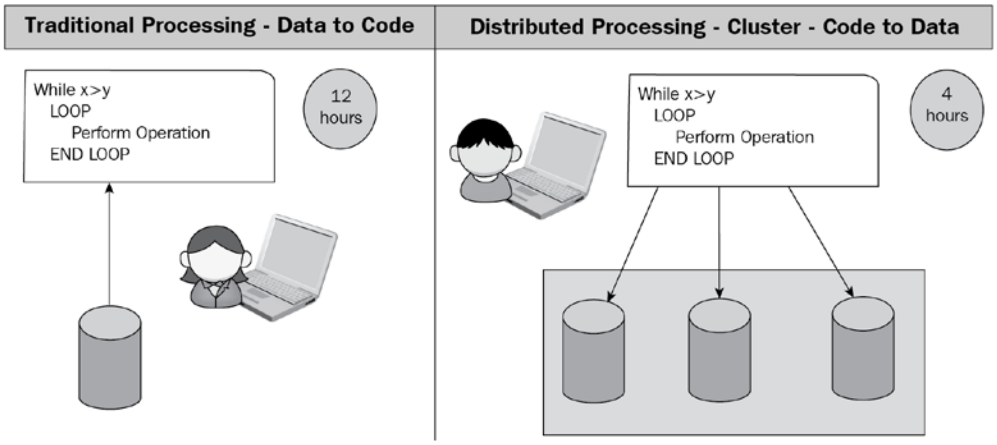


Figure 1.4 – Rise of distributed computing

The distributed processing approach, which I refer to as the **paradigm shift**, largely takes care of the previously stated problems. Instead of taking the traditional data-to-code route, the paradigm is reversed to **code-to-data**.

In a distributed processing approach, several resources collectively work as part of a cluster, all working toward a common goal. In simple terms, this approach can be compared to a team model where every team member takes on a portion of the load and executes it in parallel until completion.
If a team member falls sick and is unable to complete their share of the workload, some other member automatically gets assigned their portion of the load.

Distributed processing has several advantages over the traditional processing approach, outlined as follows:

- The **code-to-data** paradigm shift ensures the network does not get clogged. The entire idea of distributed processing heavily relies on the assumption that data is stored in a distributed fashion across several machines, also referred to as nodes. At the time of processing, only the code portion (usually a much smaller footprint as compared to actual data) is sent over to each node that stores the portion of the data being processed. This ensures that the processing happens locally on the node where the data is stored.

- Since several nodes are collectively participating in data processing, the overall completion time is drastically reduced.
- Program execution is immune to network and node failures. If a node failure is encountered, then a portion of the work is assigned to another available node in the cluster.

Important note

Distributed processing is implemented using well-known frameworks such as Hadoop, Spark, and Flink. Modern **massively parallel processing** (**MPP**)-style data warehouses such as Amazon Redshift, Azure Synapse, Google BigQuery, and Snowflake also implement a similar concept.

Since distributed processing is a multi-machine technology, it requires sophisticated design, installation, and execution processes. That makes it a compelling reason to establish good data engineering practices within your organization.

Adoption of cloud computing

The vast adoption of cloud computing allows organizations to abstract the complexities of managing their own data centers. Migrating their resources to the cloud offers faster deployments, greater flexibility, and access to a pricing model that, if used correctly, can result in major cost savings.

In the previous section, we talked about distributed processing implemented as a cluster of multiple machines working as a group. For this reason, deploying a distributed processing cluster is expensive.

In the pre-cloud era of distributed processing, clusters were created using hardware deployed inside on-premises data centers. Very careful planning was required before attempting to deploy a cluster (otherwise, the outcomes were less than desired). You might argue why such a level of planning is essential. Let me address this:

- Since the hardware needs to be deployed in a data center, you need to physically procure it. The real question is how many units you would procure, and that is precisely what makes this process so complex.
- Order more units than required and you'll end up with unused resources, wasting money.
- Order fewer units than required and you will have insufficient resources, job failures, and degraded performance.

To order the right number of machines, you start the planning process by performing benchmarking of the required data processing jobs.

- The results from the benchmarking process are a good indicator of how many machines will be able to take on the load to finish the processing in the desired time. You now need to start the procurement process from the hardware vendors. Keeping in mind the cycle of procurement and shipping process, this could take weeks to months to complete.
- Once the hardware arrives at your door, you need to have a team of administrators ready who can hook up servers, install the operating system, configure networking and storage, and finally install the distributed processing cluster software—this requires a lot of steps and a lot of planning.

I hope you may now fully agree that the *careful planning* I spoke about earlier was perhaps an understatement. The complexities of on-premises deployments do not end after the initial installation of servers is completed. You are still on the hook for regular software maintenance, hardware failures, upgrades, growth, warranties, and more.

This is precisely the reason why the idea of cloud adoption is being very well received. Having resources on the cloud shields an organization from many operational issues. Additionally, the cloud provides the flexibility of automating deployments, scaling on demand, load-balancing resources, and security.

Important note

Many aspects of the cloud particularly scale on demand, and the ability to offer low pricing for unused resources is a game-changer for many organizations. If used correctly, these features may end up saving a significant amount of cost. Having a well-designed cloud infrastructure can work miracles for an organization's data engineering and data analytics practice.

Data storytelling

I started this chapter by stating *Every byte of data has a story to tell.* Data storytelling is a new alternative for non-technical people to simplify the decision-making process using narrated stories of data. Traditionally, decision makers have heavily relied on visualizations such as bar charts, pie charts, dashboarding, and so on to gain useful business insights. These visualizations are typically created using the end results of data analytics. The problem is that not everyone views and understands data in the same way. Let me give you an example to illustrate this further.

Here is a BI engineer sharing stock information for the last quarter with senior management:

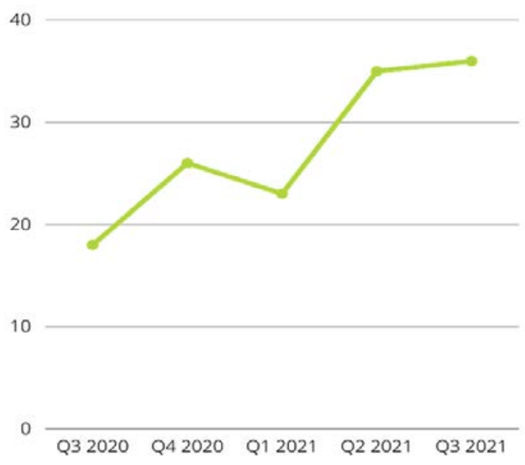


Figure 1.5 – Visualizing data using simple graphics

And here is the same information being supplied in the form of data storytelling:

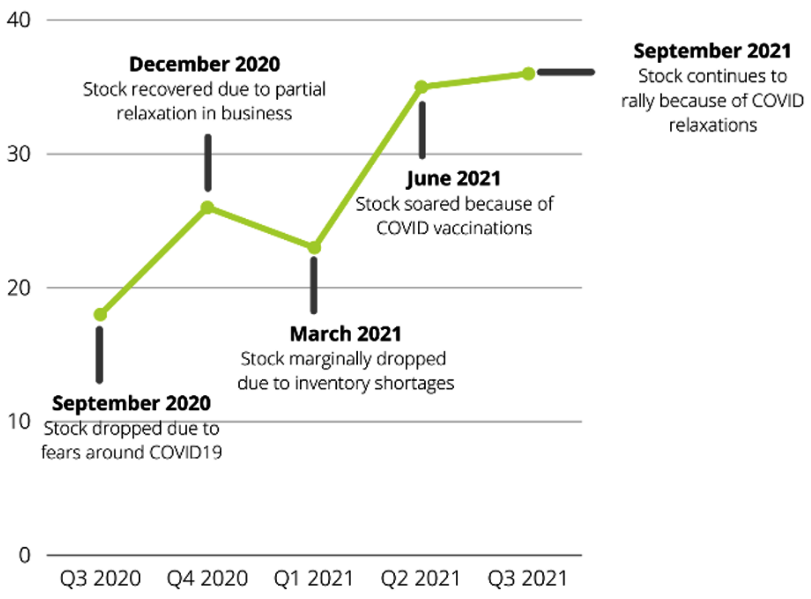


Figure 1.6 – Storytelling approach to data visualization

Important note

Visualizations are effective in communicating why something happened, but the storytelling narrative supports the reasons for it to happen.

Data storytelling tries to communicate the analytic insights to a regular person by providing them with a narration of data in their natural language. This does not mean that data storytelling is only a narrative. It is a combination of narrative data, associated data, and visualizations. With all these combined, an interesting story emerges—a story that everyone can understand.

As data-driven decision-making continues to grow, data storytelling is quickly becoming the standard for communicating key business insights to key stakeholders.

There's another benefit to acquiring and understanding data: *financial*. Let's look at the monetary power of data next.

The monetary power of data

Modern-day organizations are immensely focused on revenue acceleration. Traditionally, organizations have primarily focused on increasing sales as a method of revenue acceleration… but is there a better method?

Modern-day organizations that are at the forefront of technology have made this possible using **revenue diversification**. Here are some of the methods used by organizations today, all made possible by the *power of data*.

Organic growth

During my initial years in data engineering, I was a part of several projects in which the focus of the project was beyond the usual. On several of these projects, the goal was to increase revenue through traditional methods such as increasing sales, streamlining inventory, targeted advertising, and so on. This meant collecting data from various sources, followed by employing the good old descriptive, diagnostic, predictive, or prescriptive analytics techniques.

But what can be done when the limits of sales and marketing have been exhausted? Where does the revenue growth come from?

Some forward-thinking organizations realized that increasing sales is not the only method for revenue diversification. They started to realize that the real wealth of data that has accumulated over several years is largely untapped. Instead of solely focusing their efforts entirely on the growth of sales, why not tap into the power of data and find innovative methods to grow organically?

This innovative thinking led to the revenue diversification method known as **organic growth**. Subsequently, organizations started to use the power of data to their advantage in several ways. Let's look at several of them.

Customer retention

Data scientists can create **prediction models** using existing data to predict if certain customers are in danger of terminating their services due to complaints. Based on this list, customer service can run targeted campaigns to retain these customers. By retaining a loyal customer, not only do you make the customer happy, but you also protect your bottom line.

Fraud prevention

Banks and other institutions are now using data analytics to tackle financial fraud. Based on key financial metrics, they have built prediction models that can detect and prevent fraudulent transactions before they happen. These models are integrated within case management systems used for issuing credit cards, mortgages, or loan applications.

Using the same technology, credit card clearing houses continuously monitor live financial traffic and are able to flag and prevent fraudulent transactions before they happen. Detecting and preventing fraud goes a long way in preventing long-term losses.

Problem detection

I was part of an **internet of things** (**IoT**) project where a company with several manufacturing plants in North America was collecting metrics from **electronic sensors** fitted on thousands of machinery parts. The sensor metrics from all manufacturing plants were streamed to a common location for further analysis, as illustrated in the following diagram:

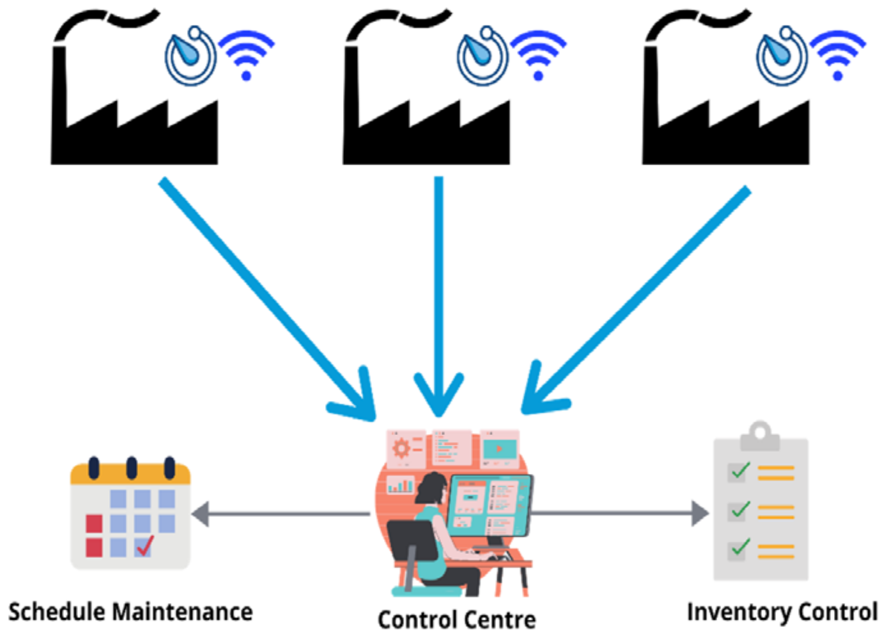


Figure 1.7 – IoT is contributing to a major growth of data

These metrics are helpful in pinpointing whether a certain consumable component such as rubber belts have reached or are nearing their **end-of-life** (**EOL**) cycle. Collecting these metrics is helpful to a company in several ways, including the following:

- The data indicates the machinery where the component has reached its EOL and needs to be replaced. Having this data on hand enables a company to schedule preventative maintenance on a machine before a component breaks (causing downtime and delays).
- The data from machinery where the component is nearing its EOL is important for inventory control of standby components. Before this system is in place, a company must procure inventory based on guesstimates. Buy too few and you may experience delays; buy too many, you waste money. At any given time, a data pipeline is helpful in predicting the inventory of standby components with greater accuracy.

The combined power of IoT and data analytics is reshaping how companies can make timely and intelligent decisions that prevent downtime, reduce delays, and streamline costs.

Data monetization

Innovative minds never stop or give up. They continuously look for innovative methods to deal with their challenges, such as revenue diversification. Organizations quickly realized that if the correct use of their data was so useful to themselves, then the same data could be useful to others as well.

As per Wikipedia, **data monetization** is the "*act of generating measurable economic benefits from available data sources*".

The following diagram depicts data monetization using **application programming interfaces** (**APIs**):

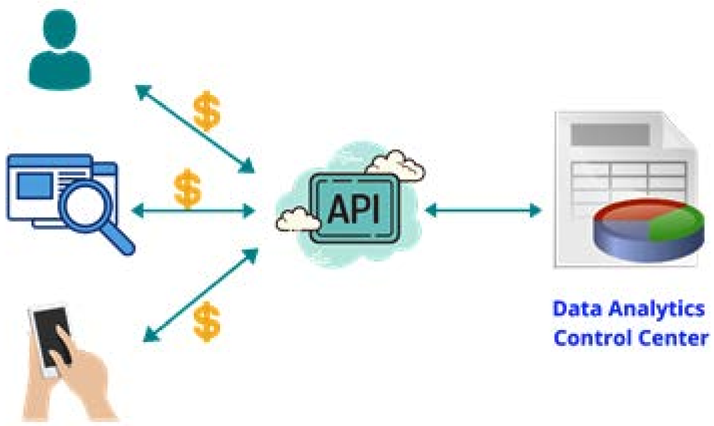


Figure 1.8 – Monetizing data using APIs is the latest trend

In the latest trend, organizations are using the power of data in a fashion that is not only beneficial to themselves but also profitable to others. In a recent project dealing with the health industry, a company created an innovative product to perform medical coding using **optical character recognition** (**OCR**) and **natural language processing** (**NLP**).

Before the project started, this company made sure that we understood the real reason behind the project—data collected would not only be used internally but would be distributed (for a fee) to others as well. Knowing the requirements beforehand helped us design an event-driven API frontend architecture for internal and external data distribution. At the backend, we created a complex data engineering pipeline using innovative technologies such as Spark, Kubernetes, Docker, and microservices. This is how the pipeline was designed:

- Several microservices were designed on a self-serve model triggered by requests coming in from internal users as well as from the outside (public).

- For external distribution, the system was exposed to users with valid paid subscriptions only. Once the subscription was in place, several frontend APIs were exposed that enabled them to use the services on a per-request model.
- Each microservice was able to interface with a backend analytics function that ended up performing descriptive and predictive analysis and supplying back the results.

The power of data cannot be underestimated, but the monetary power of data cannot be realized until an organization has built a solid foundation that can deliver the right data at the right time. Data engineering plays an extremely vital role in realizing this objective.

Summary

In this chapter, we went through several scenarios that highlighted a couple of important points.

Firstly, the importance of data-driven analytics is the latest trend that will continue to grow in the future. Data-driven analytics gives decision makers the power to make key decisions but also to back these decisions up with valid reasons.

Secondly, data engineering is the backbone of all data analytics operations. None of the magic in data analytics could be performed without a well-designed, secure, scalable, highly available, and performance-tuned data repository—a **data lake**.

In the next few chapters, we will be talking about data lakes in depth. We will start by highlighting the building blocks of effective data—storage and compute. We will also look at some well-known architecture patterns that can help you create an effective data lake—one that effectively handles analytical requirements for varying use cases.

2 Discovering Storage and Compute Data Lakes

In the previous chapter, we discussed the immense power that data possesses, but with immense power comes increased responsibility. In the past, the key focus of organizations has been to detect trends with data, with the goal of revenue acceleration. Very commonly, however, they have paid less attention to vulnerabilities caused by inconsistent data management and delivery.

In this chapter, we will discuss some ways a data lake can effectively deal with the ever-growing demands of the analytical world.

In this chapter, we will cover the following topics:

- Introducing data lakes
- Segregating storage and compute in a data lake
- Discovering data lake architectures

Introducing data lakes

Over the last few years, the markers for effective data engineering and data analytics have shifted. Up to now, organizational data has been dispersed over several internal systems (silos), each system performing analytics over its own dataset.

Additionally, it has been difficult to interface with external datasets for extending the spectrum of analytic workloads. As a result, it has been difficult for these organizations to get a **unified** view of their data and gain **global insights**.

In a world where organizations are seeking revenue diversification by fine-tuning existing processes and generating organic growth, a globally unified repository of data has become a core necessity. Data lakes solve this need by providing a unified view of data into the hands of users who can use this data to devise innovative techniques for the betterment of mankind.

The following diagram outlines the characteristics of a data lake:

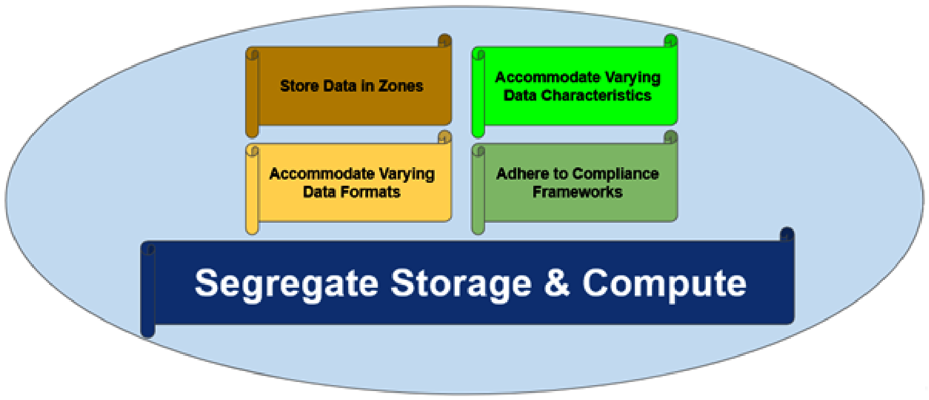

Store Data in Zones
Accommodate Varying Data Characteristics
Accommodate Varying Data Formats
Adhere to Compliance Frameworks
Segregate Storage & Compute

Figure 2.1 – Characteristics of a data lake

You will find several varying definitions of a data lake on the internet. Instead of defining what a data lake is, let's focus on the benefits of having a data lake for a modern-day organization.

We will also see how the storage and computations process in a data lake differs from traditional data storage solutions such as databases and data warehouses.

Important note

In the modern world, the typical **extract, transform, load** (**ETL**) process (collecting, transforming, and analyzing) is simply not enough. There are other aspects surrounding the ETL process—such as security, orchestration, and delivery—that need to be equally accounted for.

Exploring the benefits of data lakes

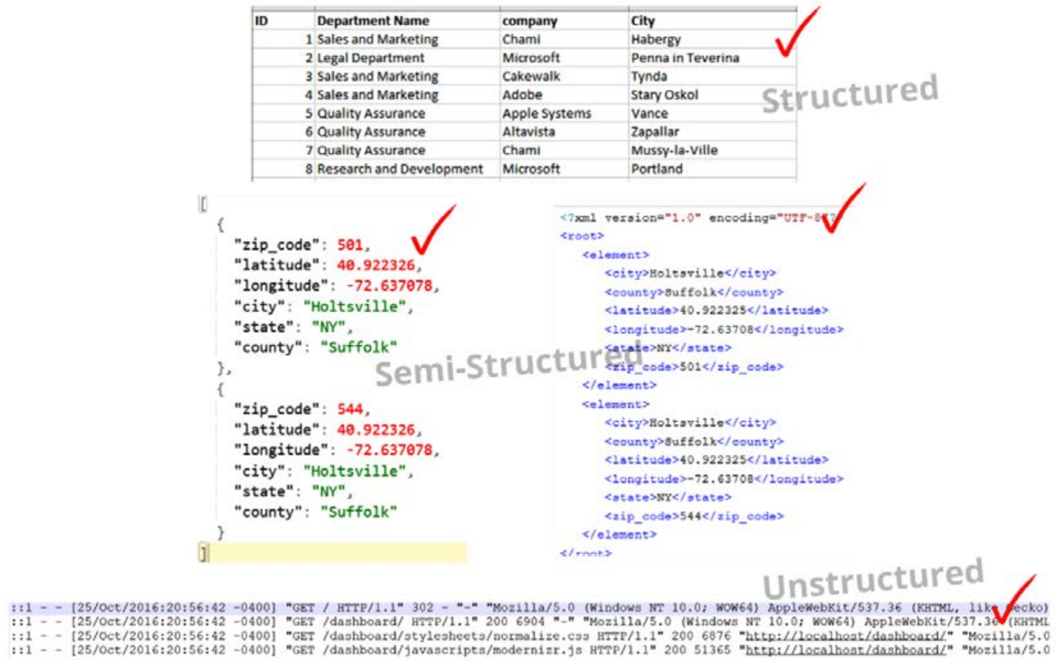
In addition to just being a simple repository, a data lake offers several benefits that make it stand out as a strong data management solution for modern data analytic needs.

Accommodating varying data formats

The most advertised benefit of a data lake is its ability to store **structured**, **semi-structured**, and **unstructured** data on a large scale. The increasing variety of data sources that participate in the data analytics process has introduced a new challenge. Since there has not been an accepted standard around **data exchange formats**, it is pretty much up to the discretion of the **data provider** to choose one for you. Some commonly used data exchange formats and their common uses are listed as follows:

- **Structured**—Database rows
- **Semi-Structured**—**Comma-separated values** (CSV), **Extensible Markup Language** (XML), and **JavaScript Object Notation** (JSON).
- **Unstructured**—**Internet of things** (**IoT**), logs, and emails

These can be better represented as follows:



ID	Department Name	company	City
1	Sales and Marketing	Chami	Habergy
2	Legal Department	Microsoft	Penna in Teverina
3	Sales and Marketing	Cakewalk	Tynda
4	Sales and Marketing	Adobe	Stary Oskol
5	Quality Assurance	Apple Systems	Vance
6	Quality Assurance	Altavista	Zapallar
7	Quality Assurance	Chami	Mussy-la-Ville
8	Research and Development	Microsoft	Portland

```
[
  {
    "zip_code": 501,
    "latitude": 40.922326,
    "longitude": -72.637078,
    "city": "Holtsville",
    "state": "NY",
    "county": "Suffolk"
  },
  {
    "zip_code": 544,
    "latitude": 40.922326,
    "longitude": -72.637078,
    "city": "Holtsville",
    "state": "NY",
    "county": "Suffolk"
  }
]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <element>
      <city>Holtsville</city>
      <county>Suffolk</county>
      <latitude>40.922325</latitude>
      <longitude>-72.63708</longitude>
      <state>NY</state>
      <zip_code>501</zip_code>
   </element>
   <element>
      <city>Holtsville</city>
      <county>Suffolk</county>
      <latitude>40.922325</latitude>
      <longitude>-72.63708</longitude>
      <state>NY</state>
      <zip_code>544</zip_code>
   </element>
</root>
```

```
::1 - - [25/Oct/2016:20:56:42 -0400] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
::1 - - [25/Oct/2016:20:56:42 -0400] "GET /dashboard/ HTTP/1.1" 200 6904 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML
::1 - - [25/Oct/2016:20:56:42 -0400] "GET /dashboard/stylesheets/normalize.css HTTP/1.1" 200 6876 "http://localhost/dashboard/" "Mozilla/5.0
::1 - - [25/Oct/2016:20:56:42 -0400] "GET /dashboard/javascripts/modernizr.js HTTP/1.1" 200 51365 "http://localhost/dashboard/" "Mozilla/5.0
```

Figure 2.2 – Support for varying formats in a data lake

Important note

Data lakes do not differentiate data based on its structure, format, or size.

The **flexibility** of a data lake to store and process varying data formats makes it a one-stop shop for data analytics operations capable of dealing with a variety of data sources in one place. Without a data lake in place, you would need to deal with silos of data, resulting in dispersed analytics without a unified view of data.

Storing data in zones

A data lake is a repository of data that stores data in various zones. Although there are no limits or rules for how many zones are ideally required, three zones are generally considered the best practice: **raw**, **curated**, and **transformed**, as illustrated in the following diagram:

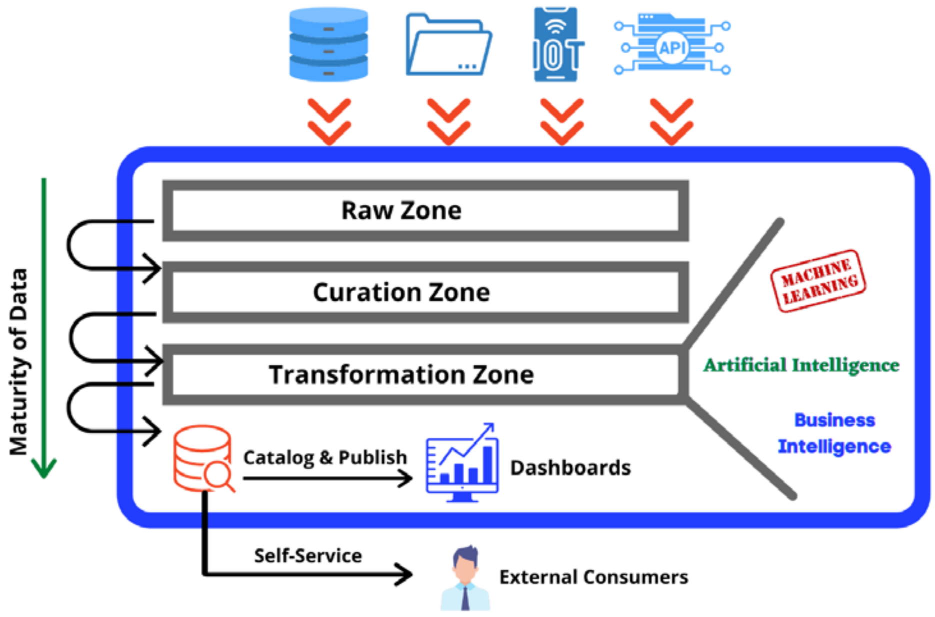


Figure 2.3 – Data lake zones and maturity of data

As data moves through each zone, it increases in maturity, becomes a lot cleaner, and finally achieves its true purpose for which it was originally collected: **data analytics**.

Once data gets ingested from a variety of sources such as databases, files, **application programming interfaces** (**APIs**), and IoT in the raw zone, it passes through a wrangling (cleaning) process to validate, standardize, and harmonize it—in short, bring all data to a common level in terms of format and organizational standards. After cleansing, data is saved to the curation zone.

The next phase of processing is done in a **self-service model**. Various sub-groups within the analytics group—such as data analysts, data scientists, **machine learning** (**ML**) engineers, and **artificial intelligence** (**AI**) engineers—start transforming data as per requirements.

Important note

Each sub-group has different requirements and views data a little differently from the others, but a data lake gives everyone the opportunity to use a unified set of data (in the curation zone) but reach different conclusions.

A typical set of transform operations include a combination of joining datasets, data aggregation, and data modeling. Finally, the results of the transformations are stored in the transformation zone. Now, data is ready to make the final leg of its journey.

Once data is in the transformation zone, it needs to be properly cataloged and published. Very frequently, data gets published in a **data warehouse** where it can be queried by **business intelligence** (**BI**) engineers for creating visualizations—dashboarding, charting, and reporting. The data has now met its final purpose.

Accommodating varying data characteristics

Data lakes can support varying characteristics of data, outlined as follows:

- **Volume**—In the early days of big data, everyone was focused on dealing with the growing size of data. Due to the immense growth of data, resource limits of databases and processing platforms were being tested to the limits. It was at this time that **Hadoop** originated as a distributed storage (**Hadoop Distributed File System**, or **HDFS**) and distributed processing (**MapReduce**) platform capable of dealing with large amounts of data at scale.

- **Variety**—Once organizations started to use Hadoop, the need to store and process large data volumes started to slowly diminish. This was the phase when dealing with large varieties of data—structured, semi-structured, and unstructured—became the top priority. The Hadoop framework was quickly able to deal with this because of its ability to store any format of data in HDFS, and MapReduce was able to read/write data using a variety of **serializers/deserializers** (**SerDes**).
- **Velocity**—The rise of modern data sources such as IoT, logs, and social networking has created the challenge of dealing with incoming data at high speeds. During this phase, the Hadoop framework started to fall short due to its inability to deal with streaming data effectively, and other frameworks such as Spark, Kafka, and Flink started to take its place. We still seem to be in this phase, where a lot of work is going into the adoption of frameworks that are capable of accommodating batch and streaming workloads.
- **Veracity**—As we continue to ingest, store, and process big data, a lot of organizations are getting concerned about the quality of data, asking: *Is my data precise, out-of-date, or biased?* While the other characteristics of data highlighted here are well defined, this one is a little more complex to handle. Organizations are looking for innovative methods to fix data quality, both at technical and operational levels.

Important note

While a *data lake* is a globally accepted term/concept, it may be implemented using a range of tools, services, and frameworks. It can be created using on-premises hardware, but the true power is realized when a data lake is created in the cloud.

Adhering to compliance frameworks

It is becoming a standard practice to incorporate security in every layer of a data lake. It is safe to assume these days that a typical organization is likely to fall under at least one compliance framework or another, such as the **General Data Protection Regulation** (**GDPR**), **Health Insurance Portability and Accountability Act** (**HIPAA**), **Payment Card Industry** (**PCI**), or the **Sarbanes-Oxley Act** (**SOX**).

The three pillars of data security are depicted in the following screenshot:

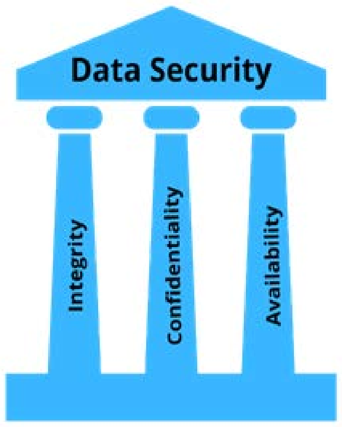


Figure 2.4 – The three pillars of data security

During a brief period of my career, I was heavily focused on data security audits. I still remember something that one of the customers jokingly said over 15 years ago, as his words have resonated several times since. He held a sales report in his hand and said:

"*There will come a time when people will care less about the correctness of data but will care more that it was processed securely.*"

Quite honestly, the sentence did not make too much sense to me at the time, but today, I can fully relate to it.

Important note

For any organization that deals with data, security is no longer a luxury—it is a core requirement.

As per the **CIA** triad, the key pillars of security are **integrity**, **confidentiality**, and **availability**. Following these principles, security in a data lake should be implemented in layers: the greater number of layers, the more protection. Data lakes can comply with the core requirements of most compliance frameworks using the following techniques to cover the key pillars of the CIA triad:

- **Perimeter security**—Access to data lake resources can be limited using a combination of site-to-site **virtual private networks** (**VPNs**), filtering **Internet Protocol** (**IP**) addresses, and blocking access to the internet.
- **Access control**—User access in a data lake can be controlled by a combination of **role-based access control** (**RBAC**), policies, **access control lists** (**ACLs**), **multi-factor authentication** (**MFA**), and **single-sign-on** (**SSO**).

- **Encryption**—A key method of protecting **data-in-motion** (traveling from one point to another) and **data-at-rest** (in storage) is using encryption. Encryption can be implemented not only at the file level in the storage, but also any **personally identifiable information** (**PII**) data within a file can be encrypted.
- **Authorization**—Extreme caution is desired during data distribution. The task becomes even more complex if data distribution includes external users. A data lake can comply with authorization needs by implementing a role-based access mechanism (RBAC) using a combination of table-, row-, and column-based security and data masking.

> **Important note**
>
> The CIA triad is simply a guideline for an organization's security program. The strength of security is directly proportional to the number of layers implemented within it.

Next, we'll see how storage and compute can be segregated within a data lake.

Segregating storage and compute in a data lake

We saved the best for last. The ability of a data lake to segregate (decouple) the storage and compute layers is perhaps one of its most desirable features. But why is decoupling storage and computation such a big deal? To understand why, we should first understand the storage and compute layers.

Storage

The fundamental function of storage is to store data. Briefly, this is how storage has evolved over the last few years:

- **Direct-attached storage** (**DAS**)—As the name suggests, a storage device (**hard disk drive/solid-state drive**, or **HDD/SSD**) is physically attached to the computer. This type of storage mostly exists in personal computers and laptops.
- **Network-attached storage** (**NAS**)—Centralized storage using multiple hard drives connected as an array. NAS allows quick and easy sharing of data between computers using regular ethernet.
- **Storage area network** (**SAN**)—High-performance shared storage using fiber-optic connectivity. A SAN allows for the fast sharing of data between computers.

- **Cloud storage**—Cloud storage is a relatively recent addition to this list. In this case, the storage physically exists on the cloud-provider servers. Cloud storage allows users the flexibility to access it from anywhere, as well as an assurance that data is highly available and safe because multiple copies are maintained on the backend.

> **Important note**
>
> Azure Blob storage, Amazon **Simple Storage Service** (**S3**), and Google Cloud Storage are some of the best-known cloud storage solutions.

Block storage

All storage types in the preceding list (except cloud storage) are categorized as file and block storage. This means data is stored on the storage device in small-sized blocks, usually 512 bytes in size. A group of contiguous blocks is deemed a file, and that is how humans relate to data.

Object storage

Cloud storage is categorized as object storage. Object storage does not work like a traditional filesystem. Unlike block storage, every object contains the following:

- **Identifier (id)**—A **globally unique identifier** (**GUID**) for each object in storage
- **Data**—Stored as blobs all in one location (not broken into multiple blocks)
- **Metadata**—Data about the data, such as author, permissions, and timestamps

The differences between block and object storage are better illustrated as follows:

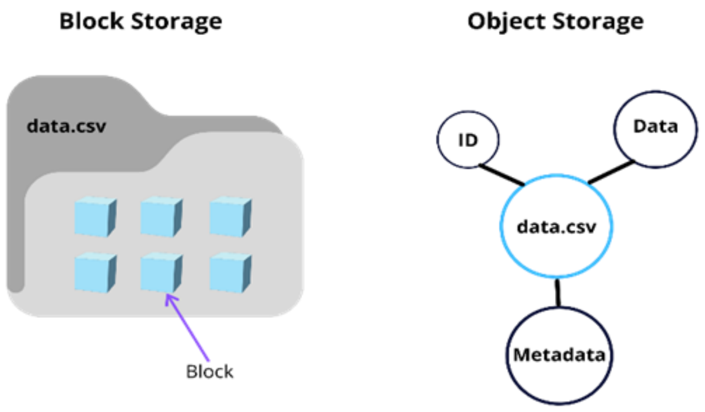


Figure 2.5 – Block storage versus object storage

Object storage is quickly becoming the standard for data engineering and analytics operations due to certain key advantages, listed here:

- Object storage is highly scalable, which means the storage can grow as per the growing data demands within an organization.
- You save money because there is no requirement to book storage capacity in advance.
- The feature of embedding metadata within the object facilitates data classification and analytics workloads.
- Data can easily be accessed via **HyperText Transfer Protocol** (**HTTP**) or APIs.
- Very suitable for storing large files such as video, audio, images, and backups of data.
- Suitable for write-once, read-many-times scenarios such as data analytics.

However, there are a few drawbacks as well, as outlined here:

- Performance of object storage is slower compared to block storage, particularly writing.
- Data cannot be **appended** to objects; any change to data simply amounts to deleting the existing object and recreating it.
- Suitable for workloads that require relatively static data.

Important note

Since each change to an object amounts to deleting it and creating a new version, object storage is unsuitable for **online transaction processing** (**OLTP**) operations where data changes are very frequent. However, it is very suitable for **online analytical processing** (**OLAP**) operations because data changes infrequently.

Compute

In the context of a data lake, compute is the ability to process (read and write) data for analytical operations such as wrangling, joining, and aggregating. In analytical computations, two resources are very critical for smooth operations, outlined as follows:

- **Central processing unit** (**CPU**)—On any given computer, all calculations are performed in the CPU. The speed of the CPU is measured in clock speed—2.8 **gigahertz** (**GHz**), for example. This means that this CPU can perform 2.8 billion computations per second. The greater the CPU speed, the more the computational power. In the world of virtualization, this term is referred to as a **virtual CPU** (**vCPU**).
- **Random-access memory** (**RAM**)—RAM stores active data that is required for any currently ongoing computations. This is done to prevent repeated read and write operations from hard drives, which are typically a lot slower (a ratio of 1:50) to respond compared to RAM. Therefore, it is generally preferred to possess as much RAM as possible.

Containers

In analytical processing terms, resources are allocated to jobs in container units. The idea of a container was introduced in **Yet Another Resource Negotiator** (**YARN**), the distributed processing framework in Hadoop.

In simple terms, a container is a logical grouping of the two key physical resources—a vCPU and RAM. Since YARN works in a cluster, it maintains the total tally of RAM and the vCPU across the cluster.

At processing time, YARN allocates containers based on the requirements of the job(s). If a cluster has enough resources to meet the demand of the job, it starts it immediately, else it puts it in wait mode until the resources free up. The different-sized containers are depicted in the following screenshot:

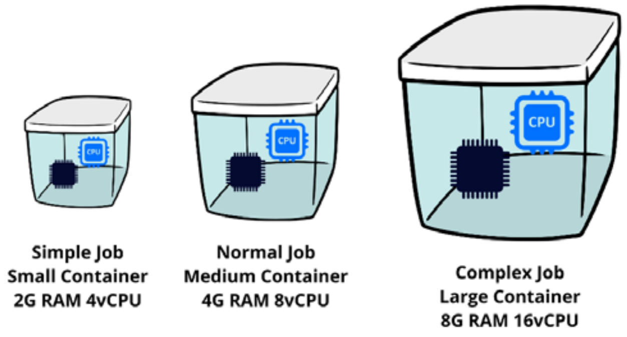


Figure 2.6 – Varying container sizes

The modern era's distributed frameworks are extremely resource-hungry. Apache Spark and Apache Flink perform computations in memory, so at times, you need a lot of it. In the big data world, understanding the internals of how job resources are allocated and managed in a cluster goes a long way in improving the completion times and performance of jobs.

> **Important note**
> The balancing act of allocating the optimal required resources to your jobs is a core skill that every data engineer should carry.

You might be wondering, *Why should I be bothered by all this? The cloud allows us to use practically endless amounts of resources anyway…* However, I assure you that I do have a point here, and a huge one…just wait for it.

Storage is cheap; compute is expensive

Here it is: *Storage is cheap, yet compute is expensive*. Now, what is that supposed to mean?

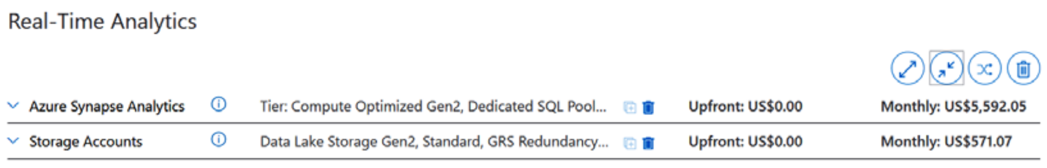
A few pages ago, I started this section by stating that the ability of a data lake to segregate (**decouple**) the storage and compute layers is perhaps one of the most desirable features—particularly if an organization's infrastructure is in the cloud.

I am going to support this theory by doing a very high-level estimate on **Microsoft Azure**. Please note that the estimate that I am going to present here is just an example to highlight the preceding point.

The assumptions are as follows:

- **Storage**—10 **terabytes** (**TB**)/month of hot tier Data Lake Storage Gen2 in the East-US region
- **Compute**—Azure Synapse Analytics for 500 **data warehouse units** (**DWU**)/month in the East-US region

We'll use the Microsoft Azure calculator available at `https://azure.microsoft.com/en-ca/pricing/calculator/` to make this estimate, shown in the following screenshot:


Real-Time Analytics

Azure Synapse Analytics	Tier: Compute Optimized Gen2, Dedicated SQL Pool...	Upfront: US$0.00	Monthly: US$5,592.05
Storage Accounts	Data Lake Storage Gen2, Standard, GRS Redundancy...	Upfront: US$0.00	Monthly: US$571.07

Figure 2.7 – Comparison of storage and compute costs

I hope that proves the point that I have been trying to make—**compute is almost 10 times more expensive than storage**.

Now, let me ask you a question. Put yourself in the shoes of the data engineer. Which layer would you try to optimize the most to keep the costs in check? If your answer is *compute*, then I have managed to get my point across.

> **Important note**
>
> As a data engineer, while choosing storage and compute resources, it is very important to keep cost optimization in mind. It is a rare skill to have but is something that your customers will thank you for the most.

In the pre-cloud era, big data platforms such as Hadoop were mostly installed on physical machines/**virtual machines** (**VMs**). In this case, the storage layer was tightly coupled with the compute layer, as illustrated in the following diagram:

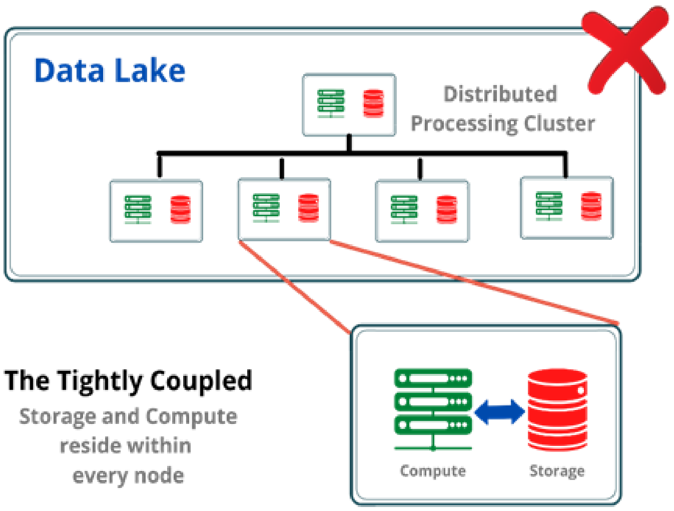

Data Lake
Distributed Processing Cluster
The Tightly Coupled
Storage and Compute reside within every node
Compute
Storage

Figure 2.8 – A tightly coupled data lake

There are a few drawbacks with this setup, outlined as follows:

- The required capacity for compute and storage needs to be **pre-purchased**. The purchased capacity is typically based on the requirements for the most complex job. Assuming the most complex job only runs for a fraction of time during the day, the unused capacity is underutilized for the rest of the day.
- You need the cluster to stay up 24/7 (whether you are actively using it or not). Costs to keep the cluster operational, such as air conditioning, real estate, electricity, and administrative costs, all add up.

In short, you are paying for something that you are not using… and no one likes to do that.

In many respects, moving toward cloud computing has been a game-changer. Having a data lake in the cloud allows us to achieve something that was not possible before: we can finally decouple storage and compute!

The following diagram shows a loosely coupled data lake:

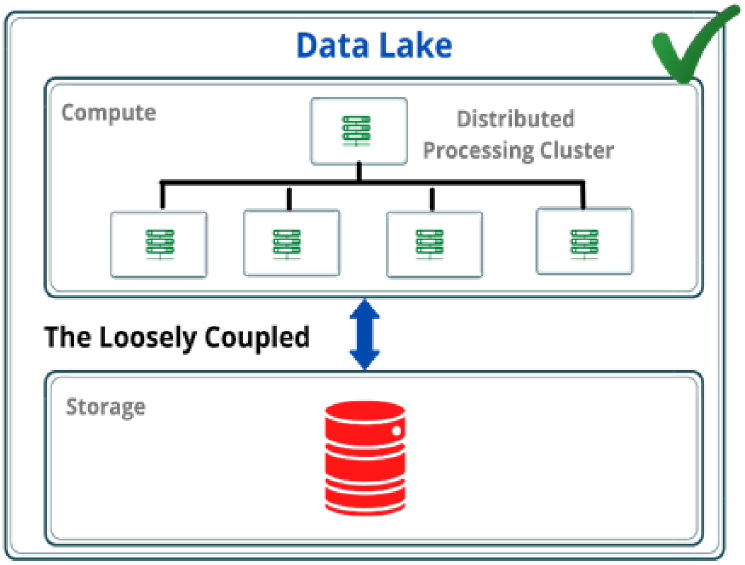


Figure 2.9 – A loosely coupled data lake

Decoupling storage from compute offers several advantages, outlined as follows:

With a **loosely coupled** data lake, the cheap (storage) layer is **permanent**, whereas the expensive (compute) layer is **temporary**. How does it work this way?

Typically, during batch processing, computations are performed only for a brief period each day (usually just a few hours). Having your infrastructure on the cloud allows you to create a distributed processing cluster (Hadoop/Spark/Flink) every day, just before batch processing starts; all the jobs are then run and, finally, the cluster is deleted right after all the jobs finish. Since the storage portion exists outside the compute cluster, there is no data loss if the cluster is destroyed. This process is depicted in the following diagram:

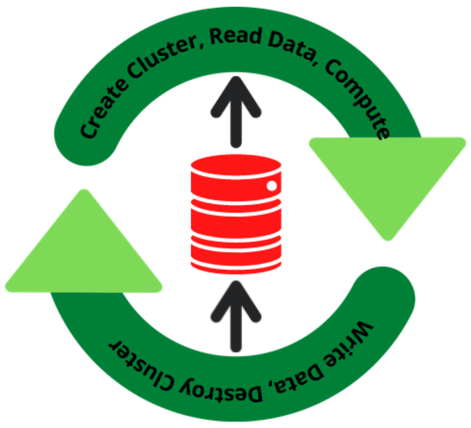

Create Cluster, Read Data, Compute
Write Data, Destroy Cluster

Figure 2.10 – Preserving compute costs

By doing this, you make sure that the compute costs are fully optimized… you **only pay for what you use**.

In *Chapter 1*, *The Story of Data Engineering and Analytics*, we discussed the journey of data and the reasons why a data engineering practice is becoming a core necessity for organizations that are serious about data analytics. The question is, how should they start this journey? In the next section, we will deep dive into the planning stages of creating a data lake, starting with understanding some well-known data lake architectures.

Discovering data lake architectures

Before we start the journey of data lake creation, it is important to understand how we undertake a normal journey in real life. Before taking a journey—a vacation, for instance —we often start by doing some preparations, such as these:

- **Plan**—Shortlist destinations that we want to visit during our trip.
- **Reserve**—Book the best airlines and hotels that fall within our **budget**.
- **Visit**—Hopefully, our preparations will make the trip both **comfortable** and **secure**.

Preparing for the journey of data lake creation is no different, as we can note here:

- **Plan**—Choose the correct data lake architecture that meets the objectives of the organization's analytical needs (**destination**).
- **Reserve**—Procure resources (computer and human) required to create a data lake within the financial restrictions (**budget**).

- **Create**—Create the necessary infrastructure and data pipelines based on a chosen architecture that helps decision makers (**comfort**) and distributes as per individual clearances (**security**).

I hope it is evident from this comparison that without the correct choice of data lake architecture, all subsequent steps will not meet the objectives of the organization.

A properly designed architecture should be capable of meeting the diverse needs of data storage and data processing, including aggregating, joining, and querying—not only in a batch mode but, more importantly, in a streaming mode as well.

Important note

In batch processing, analysis is performed using data that has been collected and stored over a large period (typically a few hours), whereas in stream processing, analysis is performed using data as soon as it is collected. After processing, this data may or may not be permanently stored.

In this section, we will focus on four well-known data lake architectures that are very commonly used in the data lake world, listed as follows:

- Traditional architecture
- Lambda architecture
- Kappa architecture
- Lakehouse architecture

Before we understand the various architecture types, we should know about a famous theorem applicable to distributed computing platforms.

The CAP theorem

In the era prior to distributed computing reaching maturity, the **scalability** of a system was increased by adding more hardware resources. This is likely done by adding more memory, disk, and CPU (or a combination of) to the pre-existing nodes. This type of scaling is commonly referred to as **vertical scalability**, as shown in *Figure 2.11*. But there are a couple of huge drawbacks with this type of scaling, as detailed here:

- Adding new hardware is limited to the maximum expandable in the node—for example, there are only a limited number of slots available on the motherboard for expanding memory.

- An existing node will eventually reach its expiration period and may need to be replaced entirely. Therefore, the cost of adding new hardware may not be justified.

In the modern era of computing, commonly referred to as **distributed computing**, the scalability of the system is achieved laterally, as shown in *Figure 2.12*. This type of scaling is commonly referred to as **horizontal scalability**. It works on the principle of *just keep adding nodes*.

Vertical scalability can be depicted as follows:

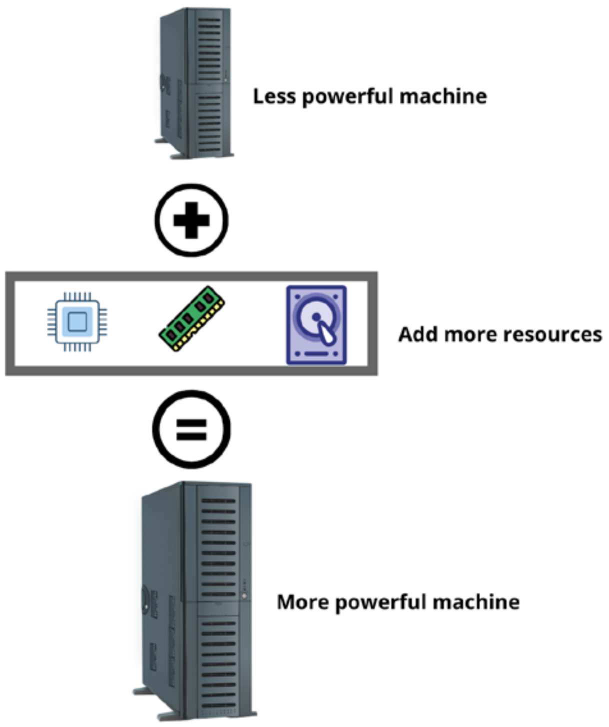


Figure 2.11 – Vertical scalability

Important note

Horizontal scaling is not achieved by adding hardware to existing nodes but by adding new nodes whenever more power is required.

Horizontal scalability can be depicted as follows:

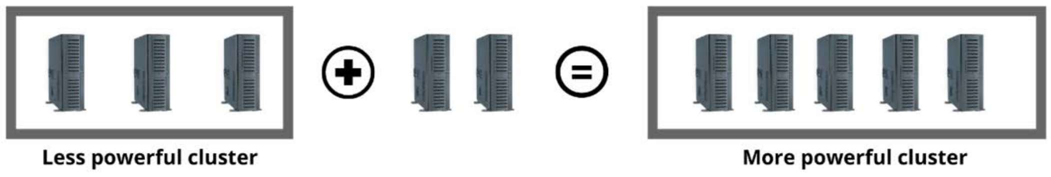


Figure 2.12 – Horizontal scalability

The CAP theorem is based on three key characteristics of a distributed computing system, outlined as follows:

- **Consistency**—All nodes should contain the same data. This means that a write request should successfully propagate data to all other replicated nodes before it becomes available for read requests. In other words, a read request should receive the same data irrespective of which node served it.
- **Availability**—Users should be able to access the system even if a node goes down. This means they should always receive a **success** response to their read/write requests. However, this does not mean that the data that has been read is indeed the most recent.
- **Partition tolerance**—The system should be able to tolerate network delays and interruptions. This means that the data replication arrangement in the distributed system should be done taking network failures in mind. In Hadoop, this concept is implemented using rack awareness.

As per the CAP theorem, it is impossible for a distributed computing system to simultaneously provide all three guarantees at the same time. It can provide a maximum of two—you need to sacrifice the third. The concept is illustrated in the following diagram:

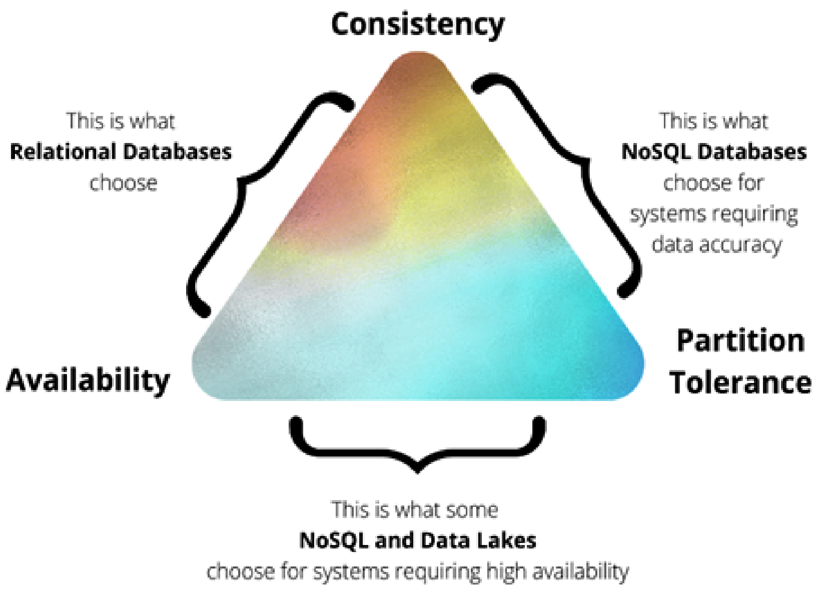


Figure 2.13 – The CAP theorem

The CAP theorem is important to understand in the data lake world because it makes you aware of trade-offs that you may need to make in your case scenarios.

The trade-offs

As per *Figure 2.13*, for data lake operations we usually opt for **AP** (**availability** and **partition** tolerance) and sacrifice consistency. This choice covers most data analytics scenarios where data accuracy can be sacrificed for the greater good. If the data is aggregated over a large period, a few inconsistent/incorrect values are not going to affect the correctness to a greater degree.

However, consider a case in IoT analytics where we are trying to monitor the temperature of a room using sensors in the possibility of a fire event. In this case, a missing or incorrect reading from a sensor could be catastrophic. Therefore, it may be advisable to choose **CP** (**consistency** and **partition** tolerance) for this situation. This is the reason why using a NoSQL database is a better choice for mission-critical streaming analytics.

Traditional architecture

Several years ago, Hadoop was the most popular engine for distributed data storage and processing systems—it was the phase of **batch processing**. Using Hadoop, a typical batch workflow functioned like this:

- **Orchestration**—Using preconfigured tools such as **Oozie** to invoke workflows (pipelines) that perform the following set of operations.
- **Batch collection**—Collecting and ingesting data every few hours. Initially, a window of 24 hours was the norm, but due to growing demands for real-time data, the windows started to get smaller. Data collection was performed using a variety of Hadoop ecosystem tools such as **Sqoop** and **Flume**. The incoming data was stored in HDFS for further processing. Every time new data was collected, it was appended to the previous dataset, usually by creating a timestamp partition.
- **Batch processing**—Once the data had been saved to HDFS, processing was performed using the **MapReduce** engine (distributed processing framework). This was done using a variety of methods such as writing programs in languages such as **Java**, **Python**, or **Perl**, or by using frameworks such as **Pig** and **Hive**.

I must admit that in those times, my customers were simply thrilled by the fact that I managed to ingest and process their data from various sources in Hadoop, even though it came with **high latency**.

The following diagram represents the state of **SALES** data 24 hours apart:

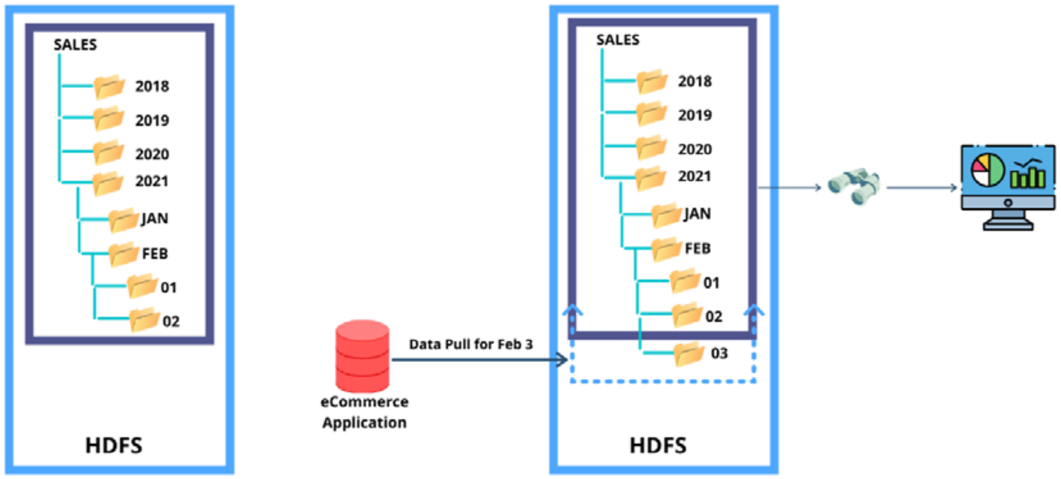


Figure 2.14 – State of data 24 hours apart

Sales report created on February 2—Includes all data from start (2018) until February 2, 2021.

Sales report created on February 3—Includes all data from start (2018) until February 3, 2021.

Now, here is the problem with traditional architecture:

If the sales report is produced somewhere between the usual scheduled report generation times on February 2 and February 3, the report will be missing sales that are happening on February 3. In other words, no matter what a report generated during an interim period, this will always only be partially inaccurate. To address this problem, we started to bring the batch window down to 12/8/4 hours, but no matter what, there would always be a fraction of data missing in the report.

As times changed, customers started to request data analytics over near-real-time data, paving the way for a better architecture that could address their growing demands.

Important note

The inability of MapReduce to process data in real time paved the way for other distributed frameworks such as Spark and Flink to come into existence.

Lambda architecture

Lambda architecture came into existence to address the need for hybrid processing that can account for both batch and streaming data at the same time. Lambda architecture was introduced by **Nathan Marz**, who created the famous real-time processing engine named **Apache Storm**. Lambda architecture is comprised of three layers, detailed next.

Batch layer

The batch layer stores the master dataset. The master dataset includes all historical data collected since the collection process started. In the example that we covered previously, the dataset in the screenshot represents the **SALES** master dataset.

These are some of the characteristics of the master dataset:

- It is immutable. Immutability of data means that existing data cannot be changed, and new data can only be appended.
- It can be used by the serving layer to publish data.
- It can incrementally merge changed data on a regular basis.

You can see the file structure of the master dataset here:

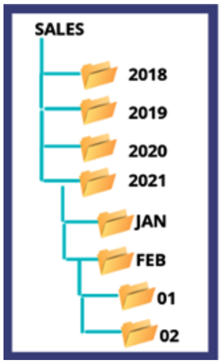


Figure 2.15 – Master dataset

Let's now have a brief introduction to the term **change data capture**, or **CDC** in short. Successfully capturing and processing changing data over time makes it one of the top three challenges that I have faced in my career. We will dedicate a whole section to CDC in a future chapter, titled *Solving Data Engineering Challenges*.

A few well-known tools of the trade for creating the batch layer are **Apache Hadoop**, **Apache Flink**, and **Apache Spark**.

Serving layer

The serving layer makes the master data available to the query process using batch views. The batch views are often the results of precomputed aggregations. Due to the data being largely precomputed, these views provide low latency access to the users.

The following screenshot shows a batch view that aggregates **SALES** by quarters. In this view, the aggregations of quarters in 2018 till 2020 will be accurate, because the data is not changing for those partitions anymore. However, data for **2021 > Q1** is still evolving and changing by the second due to currently ongoing sales:

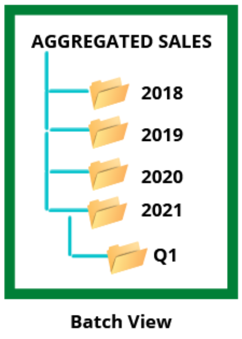


Figure 2.16 – Aggregated batch view

Here is how we can achieve an accurate result for 2021-Q1:

2021-Q1 from batch view + Aggregate (February 3 SALES)

The latter summation of data is missing, and this is where the speed layer will help us.

The tools of the trade for the serving layer are **Apache Hive** and **Presto**.

Speed layer

The speed layer stores the most recent dataset (**incremental**) that has not yet been merged to batch views due to the latency of the merge process. This data is made available to the query process as **real-time views**, often created using stream processing. You can see a depiction of a real-time view here:

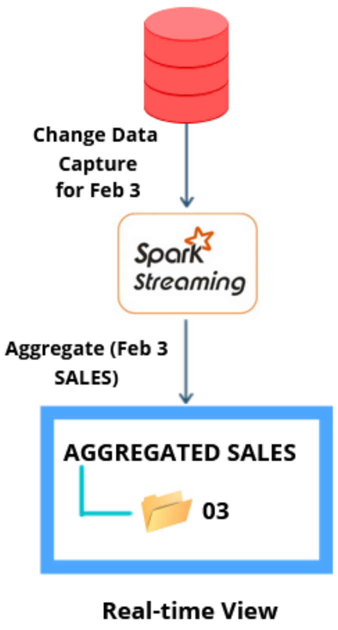


Figure 2.17 – Aggregated real-time view

In traditional architecture, this is exactly the layer that could not be handled by Hadoop MapReduce, and therefore advancement to newer frameworks was required.

The tools of the trade for the speed layer are **Apache Spark Streaming**, **Apache Storm**, and **Apache Flink**.

This is how everything comes together in Lambda architecture:

- Incremental data from the source is concurrently pushed to the batch and speed layers.
- The batch layer keeps accumulating the data over a certain period and refreshes the batch view after the period expires.

You can see the various layers of a Lambda architecture here:

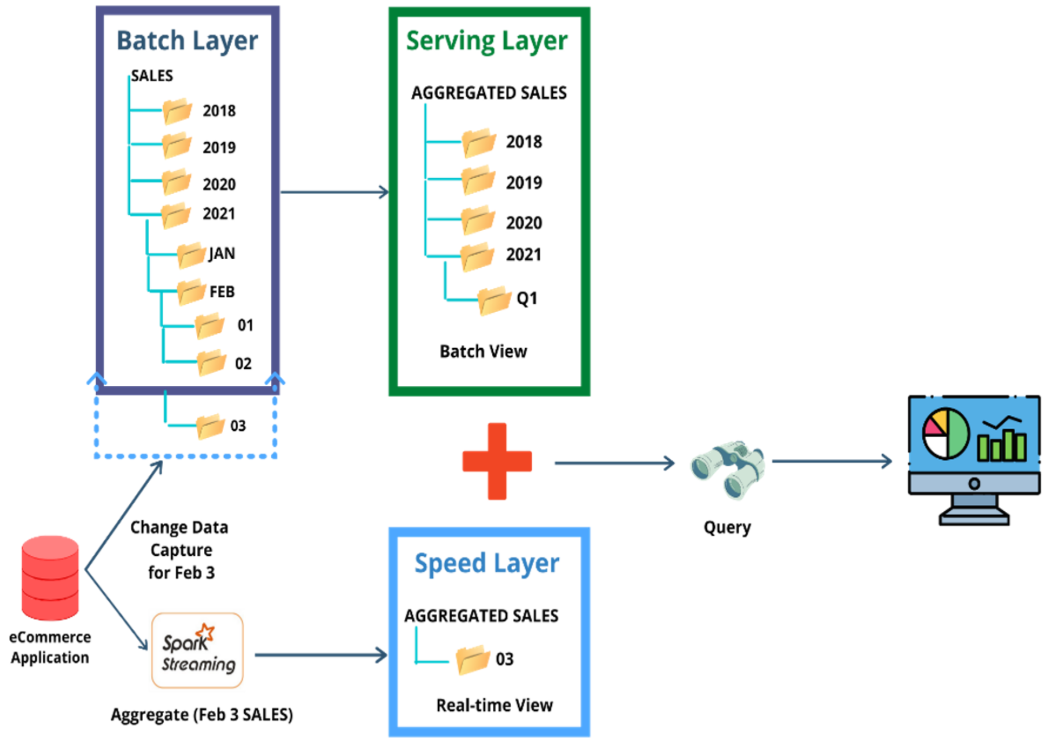


Figure 2.18 – Various layers of a Lambda architecture

- Incremental data is pushed to the speed layer where it is aggregated in a streaming fashion as a real-time view.
- At query time, the results from the batch view are consolidated with the results from the real-time view to get the most recent and accurate figures.

Lambda architecture has several benefits, including low latency due to precomputed views, and data accuracy due to near-real-time access to data.

Here are some case scenarios where you might use a Lambda architecture with great success:

- **Access to near-real-time data**—There are scenarios in which the decision-making process is detrimentally impacted if access to the latest data is not available.
- **Large historical datasets**—There are situations in which the historical data is extremely large, so aggregations can take a long time to finish and cannot be performed frequently.
- **Dynamic nature of analytics (ad hoc)**—There are cases in which the case scenarios for data analytics are very dynamic and vary quite frequently. In this case, the batch and real-time views can be created or adjusted without affecting the underlying master data.

> **Important note**
>
> A particular strength of the Lambda architecture is its strength to be easily migrated from traditional architecture. I have done several of these migrations in the past with great success.

Lambda architecture provides a nice balance of performance and data freshness. But in recent times, requirements for streaming analytics have grown quite sharply. Use cases such as IoT monitoring and anomaly detection require a fast real-time analytics engine able to support millions of incoming requests per second.

Despite all the advantages that Lambda architecture has, there are a few drawbacks, as outlined here:

- In a few cases, it is sometimes complex to implement due to the multiple layers. More layers also translate to more costs, in resources, administration, and monitoring.
- Each layer requires a different set of tools, services, and programming. Therefore, infrastructure and code management become a challenge.
- In some scenarios, implementing a Lambda architecture may be overkill due to the nature of the analytics.

Quite recently, a simpler architecture was introduced, the Kappa architecture, which many call the simplified version of the Lambda architecture.

Kappa architecture

The Kappa architecture was introduced in 2014 by **Jay Kreps**. This architecture eliminates the need for the batch layer altogether. Instead, all data is stored in a real-time layer that maintains an immutable stream of events stored in a stream-processing platform.

The major components of the Kappa architecture are described in the following subsections.

Streaming layer

The **streaming layer** stores all raw data in the form of sequential events (messages) sorted by the creation timestamp. Since data is not stored on a regular filesystem, the chosen streaming platform should include built-in features such as **fault tolerance** (**FT**) and **high availability** (**HA**).

The tools of the trade for the streaming layer storage are **Apache Kafka**, **Azure Event Hubs**, **Amazon Kinesis**, and **Google Pub/Sub**.

The stream-processing engine reads event data, performs required computations, and publishes the transformed data as views in the serving layer.

The tools of the trade for streaming-layer processing are **Apache Spark Streaming**, **Apache Flink**, and **Kafka Streams**.

Serving layer

The serving layer makes event data available to the query process using precomputed views. These views can be recomputed at regular intervals by relaunching the view build code.

The tools of the trade for the serving layer are **relational databases** and **data warehouses**, **cloud data warehouses**, and **cloud data storage**.

Here is how the Kappa architecture functions:

The stream-processing engine performs aggregations and publishes the results to the serving layer as precomputed views. As per *Figure 2.19*, at time **t**, the stream-processing code computes data from 2018 until the latest event in the sales topic.

The computed data is published as the **AGGREGATED SALES** view, which is then queried for analysis.

You can see a diagram of the process here:

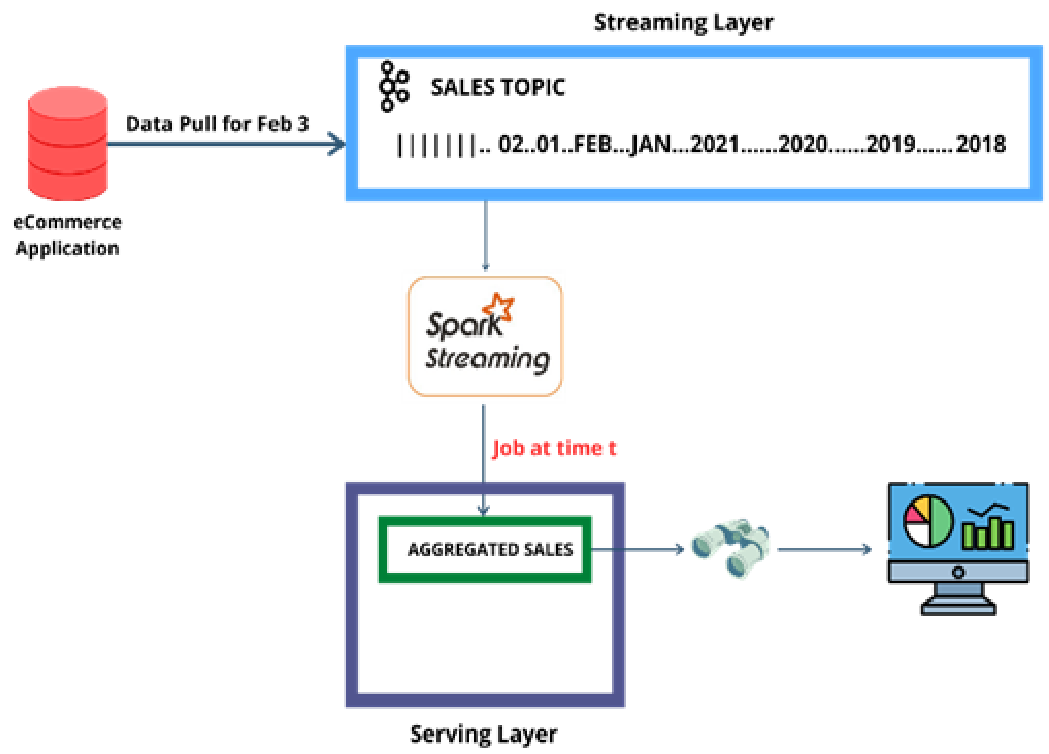


Figure 2.19 – State of aggregate view at time t

At time **t + 1**, the stream-processing code recomputes data from 2018 up to the most recent event within the sales topic. Notice that new events have been added to the queue since the last time the computations were performed. The recomputed data is published as follows:

- The previous version of the **AGGREGATED SALES** view is dropped.
- A new version of the **AGGREGATED SALES** view is computed to take its place.
- Queries are pointed to the recomputed view.

You can see a diagram of the process here:

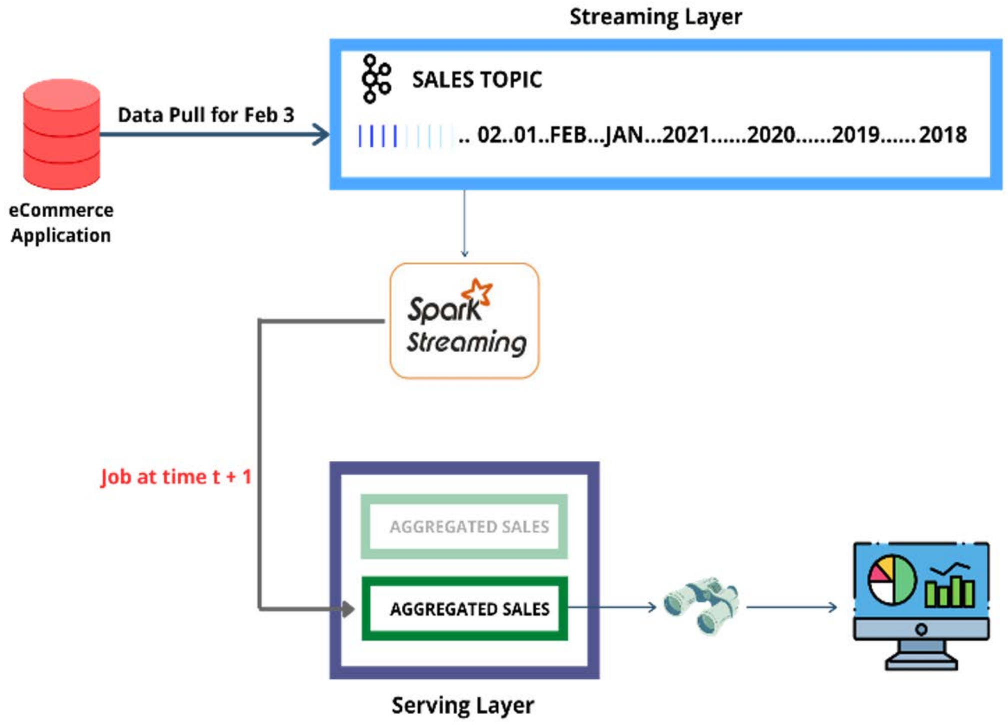


Figure 2.20 – State of aggregate view at time t+1

Now we know how the Kappa architecture generally functions, let's outline some of the pros and cons of it, as follows:

- The Kappa architecture works extremely well for use cases around IoT monitoring and real-time analytics with limited dependency on historical data.
- Unlike Lambda architecture, a single code base is sufficient for data processing. This hugely simplifies infrastructure deployment and management requirements.
- Since Kappa stores all data in the streaming layer for larger datasets, typically in the TB range, the architecture becomes extremely expensive.

Let's compare the costs at a very high level, as follows:

- Storing 32 TB of data on Microsoft Azure Premium SSD Managed Disks will cost you roughly **US dollars** (**USD**) $4,194/month.
- Storing 32 TB of data on Microsoft Azure Blob storage will cost you just $682/month.

Many experts feel the Kappa architecture is a substitute for the Lambda architecture, although I see things a bit differently. To me, each one of these architectures solves a different purpose. Here are some case scenarios where I have used the Kappa architecture with great success:

- **Real-time monitoring**—Monitoring of IoT sensors is a perfect use case for the Kappa architecture. In almost all monitoring cases, having access to historical data is not required.
- **Live dashboarding**—I am sure you have seen dashboards that visualize data in a live fashion. The visual is refreshed every few seconds based on incoming data.
- **Real-time ML**—Many companies are now exposing their ML models as microservices for use cases such as fraud detection.
- **Real-time data monetization**—We previously discussed data monetization as a means for revenue diversification. More and more companies are now exposing their data to external companies for consumption in real time.

Important note

Before choosing a suitable architecture, data engineers should carefully evaluate the strength of each one based on customer requirements, the urgency of near-real-time analysis, and resource costs.

Both Lambda and Kappa architectures are layered—data is collected in the storage/streaming layer, processed, and finally published to a warehouse. I have done several projects in the past using these architectures but, truthfully, implementing either one of them comes with a few challenges, such as the following:

- Keeping data fresh and up to date in each tier is difficult. Ever-changing data requires a carefully designed merge process.
- Infrastructure costs in both architectures can be steep at times.
- More layers mean more chances of failure. Dealing with a failure is not only costly but also untimely because it delays the ongoing data analytics.
- If the data is being published externally for data monetization purposes, any failures or inaccuracies can cause revenue loss and customer dissatisfaction.

Some of the previously mentioned deficiencies of the Lambda and Kappa architectures have forced the experts to find a new way of thinking.

Generally, in any data lake operation, we ingest data to storage, process it as per requirements, and finally publish the aggregated data to a warehouse so that it is available for analytics.

Lately, a new architecture is being adopted at a fast rate that combines the power of a data lake and a data warehouse in one, as illustrated in the following diagram:

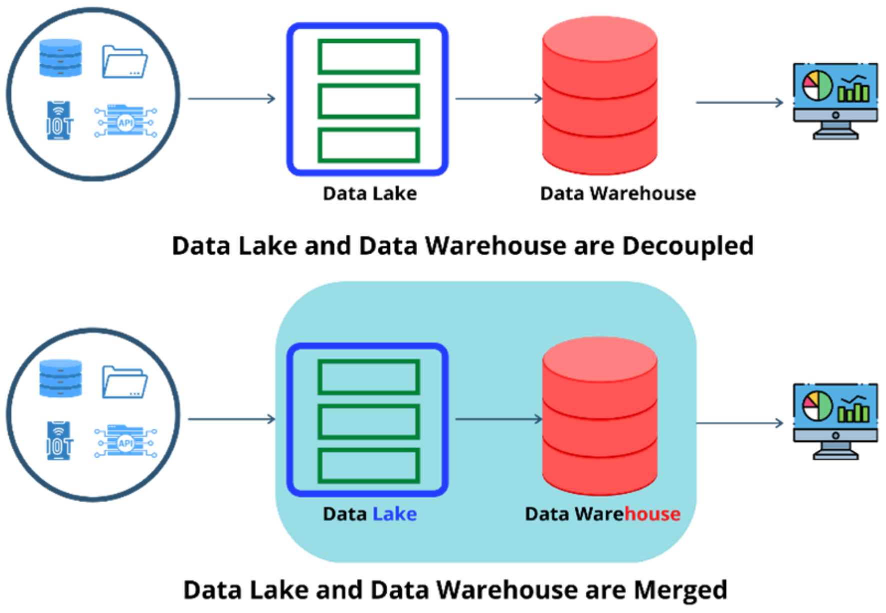

IOT
API
Data Lake
Data Warehouse
Data Lake and Data Warehouse are Decoupled
IOT
API
Data Lake
Data Warehouse
Data Lake and Data Warehouse are Merged

Figure 2.21 – Rise of the Lakehouse architecture

In this section, we learned about the Kappa and Lambda architecture, and the rise of the Lakehouse architecture.

Lakehouse architecture

Before we discuss the **Lakehouse** architecture, it is important to understand what I refer to as the power struggle between traditional and modern data processing systems.

The give-and-take struggle

There was a card game that I used to play when I was younger. In this game, to advance to the next level, you were required to sacrifice your best card.

Through many years in the data processing world, I have seen the same give-and-take struggle between the traditional method of data processing (database/data warehouses) versus the modern one (data lakes).

You can see this power struggle depicted in the following diagram:

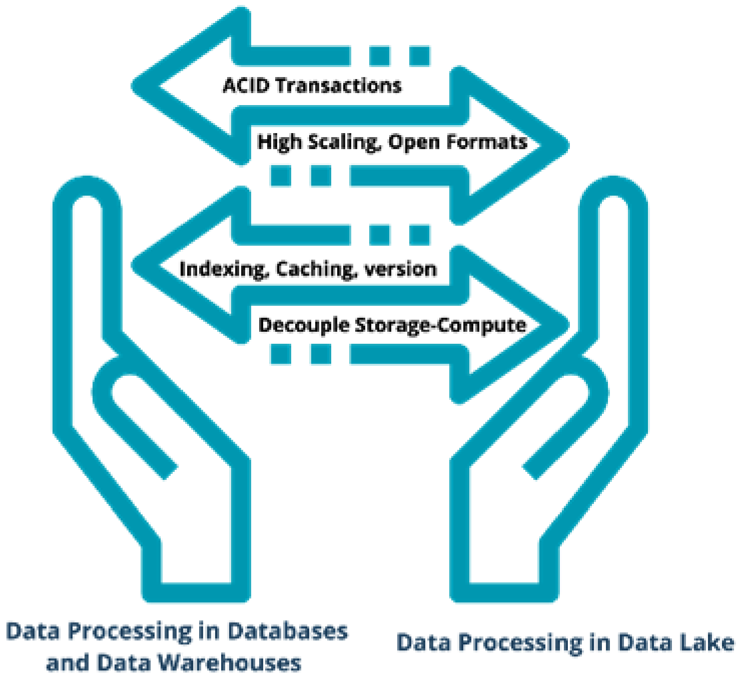

ACID Transactions
High Scaling, Open Formats
Indexing, Caching, version
Decouple Storage-Compute
Data Processing in Databases and Data Warehouses
Data Processing in Data Lake

Figure 2.22 – The power struggle

The decision to move toward modern data processing platforms has forced us to make some sacrifices; you gain some and you lose some. Let me highlight some of these give-and-take situations in which you must move from traditional to modern methods, as follows:

- Sacrifice durability and consistency features such as **atomicity, consistency, isolation, and durability** (**ACID**) transactions but gain the ability to process on a highly scalable platform.
- Sacrifice performance features such as indexing and caching but gain the ability to process data in multiple formats.
- Sacrifice security features such as versioning and auditing but gain the ability to decouple storage and compute.

The Lakehouse architecture, as depicted in the following diagram, merges the low-cost storage of a data lake with the **massively parallel processing** (**MPP**) power of a warehouse to serve the diverse and ever-evolving requirements of modern-era analytics:

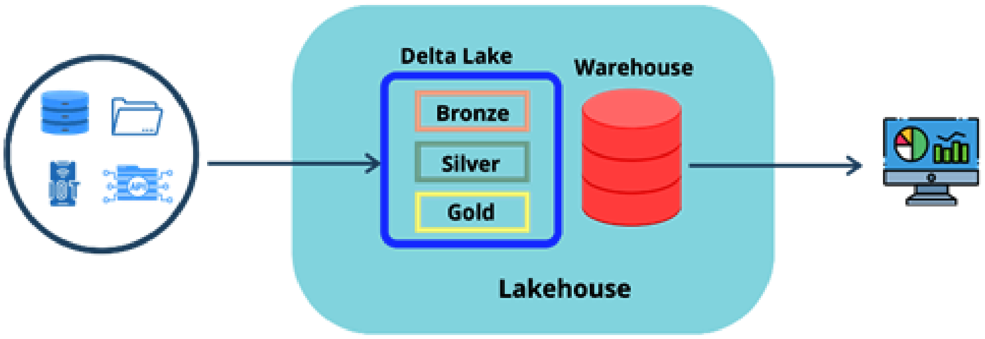


Figure 2.23 – Lakehouse architecture

Here are a few important features of the Lakehouse architecture:

- **Supports low-cost storage**—The Lakehouse architecture carries forward the support for using low-cost storage. Data can be structured, semi-structured, or unstructured over open formats such as Parquet.
- **Supports ACID transactions**—Now, this one is big. An absence of transaction support hugely impacts the data integrity and quality in a data lake. Over the years, dealing with duplicated and deleted data has been a huge challenge. In the early days, I remember going through a multi-step process in Hive to deal with this. A few years later, Spark took over, but the challenge stayed the same. But the struggle is finally over…. long live **Databricks Delta Lake**!

> **Important note**
> Since its introduction, Delta Lake has proved to be a lifesaver for data engineers, not only to resolve the CDC challenge but many other ones, such as schema evolution and time travel. The Lakehouse architecture assumes the use of an open source storage engine such as Delta Lake for ACID transactions, metadata management, governance, and many other features.

- **Metadata management**—The Lakehouse architecture offers strong metadata management features such as schema enforcement and evolution. Once again, the absence of such features in the past has forced us to perform similar checks using code.

- **Segregation of storage and compute**—The Lakehouse architecture preserves the idea of decoupling storage and compute for massive scalability and cost-saving.
- **Support for BI tools**—Lakehouse architecture supports connectivity to all major BI tools using standard **Java Database Connectivity** (**JDBC**)/**Open Database Connectivity** (**ODBC**) connectivity.
- **Native support for Structured Query Language (SQL)**—Users can easily utilize the power of the SQL language for performing analytics.

> **Important note**
>
> In Azure, the Lakehouse architecture is implemented using Synapse Analytics. In addition to using Synapse SQL for performing analysis, it provides support for Apache Spark.

- **Supports streaming workloads**—The Lakehouse architecture supports real-time analytics and reporting using streaming data.

The tools of the trade for the Lakehouse architecture include **Azure Synapse Analytics**, **Amazon Redshift Spectrum**, **Google BigQuery**, and **Snowflake**.

These sections should explain how a Lakehouse architecture provides a nice blend of traditional, Lambda, and Kappa architectures. The way I see it, the Lakehouse is the right recipe for creating a data lake—one that not only works but also provides the correct balance between **user experience** (**UX**), scalability, and cost.

Summary

In this chapter, we learned about a data lake and its various characteristics, such as multiple zones, the ability to deal with multiple formats, governance, and—most importantly—its ability to decouple storage and compute.

We also learned about four key data lake architectures: traditional, Lambda, Kappa, and Lakehouse. We analyzed each one from their applicability to certain case scenarios, costing, and overall management.

In the next chapter, I will highlight various tools and services available in Microsoft Azure required to build a data lake using the Lakehouse architecture.

3
Data Engineering on Microsoft Azure

In the previous chapter, we discussed how cloud adoption offers greater flexibility and faster deployments for data engineering and analytical workloads. In this chapter, we'll discuss the major tools and services in Microsoft Azure that may help us implement such a solution.

In this chapter, we will cover the following topics:

- Introduction to data engineering in Azure
- Performing data engineering in Azure
- How to open a free account with Azure

Introducing data engineering in Azure

In recent years, Microsoft Azure has added several powerful services to its arsenal that seamlessly collect, store, process, and publish data for both batch and streaming workloads. Gone are the days where choices for storage and compute were severely limited among cloud vendors. As a user, you simply needed to conform with the supplied tools and services: now, your options are more extensive.

Today, the cloud ecosystem looks very different from what it did previously. The growth of cloud services allows users to choose from a variety of storage, compute, and deployment options. As an example, if I want to run a Spark program, I can choose from at least four different options in **Microsoft Azure**. The real question is, if all four options are running **Apache Spark**, then why are these options even required?

Important Note

The array of options available on the cloud are not limited to compute only: the same variety exists for data collection, infrastructure deployments, and pipeline orchestration services.

Here are some of the reasons why several options for performing similar kinds of operations in data engineering are available on Microsoft Azure:

- **Ease of use**: The ease of use of a service is a key factor in deciding which option to use. As a data engineer, you may code the entire **ETL** program yourself or opt for a service that can auto-generate and invoke the code for you after creating a visual workflow.
- **Data engineering team skills**: The skill level of the data engineering team plays a very important role in deciding which option to choose. Teams with skilled data engineers typically tend to choose services where data workflows need to be coded from scratch. Doing so offers engineers complete control over their programs, as well as any other dependencies.
- **The desired level of administration and monitoring**: Having a skilled data engineering team does not automatically mean that the same team will be involved in administering and monitoring the operations. The choice of service may need to be adjusted, depending on who will be managing it in production.
- **Self-serve analytics**: These days, the concept of self-serve analytics is on the rise. This means that end users are getting more and more involved in creating ETL/ELT workflows. For this specific set of users, the ability to create a drag-and-drop workflow in a few clicks is extremely desirable.

Important Note

The idea of self-serve analytics is gaining a lot of popularity these days. Self-serve analytics enables fast-paced delivery of end user analytical goals.

- **Level of control that's desired over the infrastructure**: The level of control that's desired over the infrastructure is an important factor that impacts the choice of service. In some cases, the cloud-provided service may lack some desired functionality and does not permit changing the underlying infrastructure. In this case, the data engineers may want to choose a service that offers complete control of the infrastructure.
- **Complete data isolation**: Some stringent compliance frameworks do not permit the use of multi-tenant services. In such cases, complete data isolation is mandatory, thus impacting the choices that may be suitable.

Important Note

There is an ongoing debate around whether cloud services are truly compliant with well-known frameworks such as HIPAA. The multi-tenancy nature of cloud storage is a huge roadblock in making that claim. It is fair to say that cloud storages support compliance frameworks but are not truly compliant.

- **Prior familiarity with certain frameworks**: This is an extremely important factor for organizations migrating their analytics from on-premises to the cloud. Since they are already familiar with a certain tool or framework, they tend to choose the closest option available in the cloud.
- **Cost**: Finally, the cost of the service is a major factor that impacts what choice should be made. Typically, the easier the usage and management of a particular service, the higher its cost. Therefore, before choosing a service, careful cost analysis is deemed necessary.

Important Note

As a data engineer, it is understandable to have a bias toward specific tools, languages, and frameworks. But once you leave, it becomes the customer's responsibility to manage this. Try to keep your bias aside and provide the customer with a solution that uses technologies that the customer feels comfortable with.

Performing data engineering in Microsoft Azure

Data engineering in Microsoft Azure can be performed using the following three options:

- Self-managed data engineering services (**IaaS**)
- Azure-managed data engineering services (**PaaS**)
- Data engineering as a service (**SaaS**):

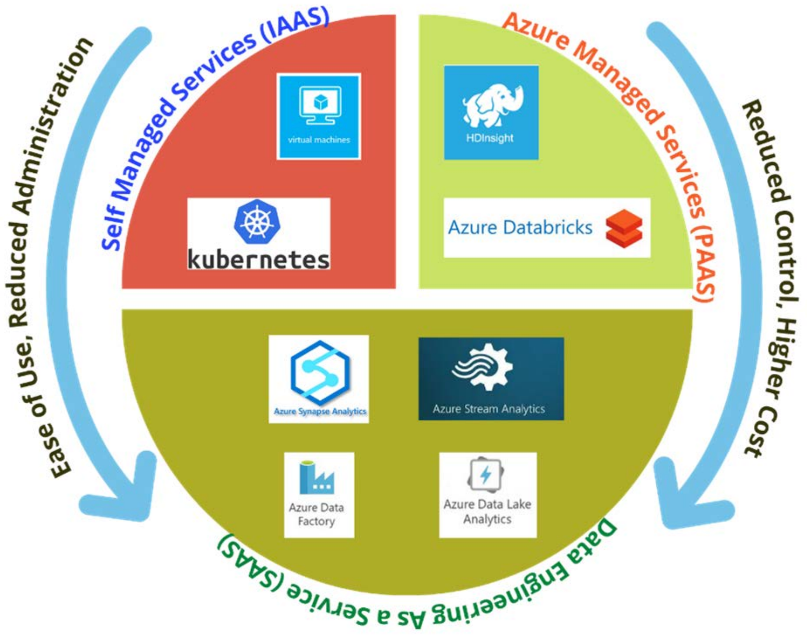

Self Managed Services (IAAS)
virtual machines
kubernetes
Azure Managed Services (PAAS)
HDInsight
Azure Databricks
Azure Synapse Analytics
Azure Stream Analytics
Azure Data Factory
Azure Data Lake Analytics
Data Engineering As a Service (SAAS)
Ease of Use, Reduced Administration
Reduced Control, Higher Cost

Figure 3.1 – Data engineering option in Microsoft Azure

Self-managed data engineering services (IaaS)

In the early phases of data engineering, using well-known distributed frameworks such as **Hadoop**, **Spark**, and **Kafka** rose sharply. As a result, many organizations were deploying Hadoop/Spark/Kafka using on-premises infrastructures. Since Hadoop/Spark/Kafka are multi-node frameworks, this meant the installations were performed using physical and virtual machines hosted on either the organization's owned or co-located data centers.

Then came the period when the cloud started to become a reality and organizations started to move their Hadoop/Spark/Kafka clusters to the cloud. In the very early stages of the cloud evolution, using a self-managed service was perhaps the only option available for data engineering.

IaaS use case

Here are a few use cases where using an IaaS layer is warranted:

- An organization that has heavily **customized** Hadoop/Spark/Kafka clusters. This means they are tied to specific versions that are unavailable on the cloud PaaS equivalent service.
- The organization has developed its own libraries for data engineering and data science that can't easily be deployed on PaaS or SaaS offerings on the cloud.
- An organization where security regulations do not permit the use of multi-tenant cloud services. This forces them to use single-tenant dedicated cloud resources.
- An organization that cannot tolerate variability in the compute power of the use of shared services.

Let's take a look at how a typical Hadoop/Spark/Kafka deployment is performed using self-managed services that use the **Infrastructure as a Service** (**IaaS**) layer on Microsoft Azure.

Azure Virtual Machines (**Azure VMs**) allows you to create on-demand compute resources for data processing. The following list describes some of the features available using Azure VMs:

- Take the inventory of the on-premises Hadoop/Spark/Kafka nodes – the total number of nodes, aggregated CPU, and aggregated memory.
- Shortlist virtual machines on Microsoft Azure that offer similar aggregated CPU and memory as the on-premises inventory. This step requires careful planning and calculations. It is not necessary to procure the same number of nodes on Microsoft Azure, so long as the aggregated CPU and memory meet or exceed the on-premises deployment. In distributed computing, horizontal scalability of nodes is preferable. It is highly recommended to compare different classes of virtual machines on Microsoft Azure; in many cases, you may be able to procure smaller class virtual machines that are cheaper than higher class ones, both offering the same aggregated CPU and memory.
- Procure virtual machines on Microsoft Azure.

- Configure networking among nodes, open required firewall ports, and install operating system prerequisites.
- Install and configure Hadoop/Spark/Kafka and any other ecosystem tools, such as Sqoop, Oozie, Hive, and so on. These steps need to be performed on multiple nodes.
- Install and configure administration and monitoring software.

Here are a few pros and cons of self-managed services:

- Performing deployments using virtual machines provides the data engineer with complete freedom to choose any framework and any version, and have complete control over its dependencies.
- It should be evident from the deployment process outlined previously that choosing self-managed services requires careful planning, design, and execution.
- Due to the vast variety of touchpoints, the data engineering team in charge of the deployment needs to be skilled in several areas, including cloud deployments, networking, administration, and monitoring. Note that all these skills are above and beyond the usual expectations of a data engineering team, which would normally involve things such as pipeline development and data orchestration.
- Scaling a cluster using a self-managed service can become a challenge.

As we mentioned previously, deploying a Hadoop/Spark/Kafka cluster using self-managed services is not optional. However, there are ways you can minimize the effects of the limitations we have just discussed.

Hadoop/Spark/Kafka distributions

The deployment process of Hadoop/Spark/Kafka clusters using VMs involves multiple nodes. A simpler way to deal with installing, upgrading, administrating, and monitoring these clusters is by using distributions such as **Cloudera**, **Hortonworks**, and **MapR**. Each of these distributions include software that enables cluster provisioning using a minimal setup. **Ambari** enables users to easily create, administer, and monitor a Hadoop cluster, whereas **Cloudera Manager** enables users to provision **Hadoop** and **Spark** clusters.

Infrastructure as Code

Infrastructure as Code (**IaC**) is a practice of deploying cloud resources such as VMs using configuration files. Using this practice, you may either create the entire infrastructure or make modifications to it. Using IaC significantly decreases manual effort for reoccurring tasks while maintaining a high level of consistency and accuracy. We will discuss this practice in greater detail in *Chapter 11*, *Infrastructure Provisioning*.

Azure Resource Manager (**ARM**) centralizes IaC deployments using resource templates.

Containerization

Containerization is the practice of abstracting application code from the underlying infrastructure. Using containerization provides several benefits, including segregating the host environment, choosing a software version, and dependency management. This means the cluster nodes can now run in their own isolated environment called containers. These containers can be orchestrated using the well-known orchestration engine known as **Kubernetes**. **Azure Kubernetes Service** (**AKS**) allows you to easily deploy and manage containerized applications.

> **Important Note**
> In the cloud era, a lot of organizations are adopting technologies that keep them cloud-agnostic. Who knows which vendor will drop their prices and when? Containerizing their applications keeps them open to moving between cloud vendors as desired.

Azure-managed data engineering services (PaaS)

Right after the big data wave started to gain momentum, cloud vendors such as Microsoft Azure started to create managed services to provide customers with fast deployments and easy management. Before we go into the specifics of the services, let's focus on what an Azure-managed service that uses the **platform as a service** (**PaaS**) layer on Azure means:

- The resource is deployed with minimal configuration and user input.
- Azure takes care of monitoring the resource for downtime and recovers it automatically in case a failure is detected.
- Globally available.
- Well integrated with other Azure services.
- Automatically provides data redundancy.
- Contains built-in security mechanisms for data protection.