2
Data Management Architectures for Analytics

In *Chapter 1*, *An Introduction to Data Engineering*, we looked at the challenges introduced by ever-growing datasets, and how the cloud can help solve these analytic challenges. However, there are many different cloud services, open-source frameworks, file formats, and architectures that can be used in analytic projects, depending on the business requirements and objectives.

In this chapter, we will discuss how analytical technologies have evolved and introduce the key technologies and concepts that are foundational for building modern analytical architectures, irrespective of whether you build them on **Amazon Web Services** (**AWS**) or elsewhere.

The content in this chapter lays an important foundation, as it will provide an introduction to the concepts that we will build on in the rest of the book.

In this chapter, we will cover the following topics:

- The evolution of data management for analytics
- A deeper dive into data warehouse concepts and architecture
- An overview of data lake architecture and concepts
- Bringing together the best of data warehouses and data lakes
- Hands-on – using the AWS **Command Line Interface** (**CLI**) to create **Simple Storage Service** (**S3**) buckets

Technical requirements

To complete the hands-on exercises included in this chapter, you will need access to an AWS account in which you have administrator privileges (as covered in *Chapter 1*, *An Introduction to Data Engineering*).

You can find the code and other content related to this chapter in the GitHub repository at the following link: https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/tree/main/Chapter02.

The evolution of data management for analytics

Innovations in data management and processing over the last several decades have laid the foundations for modern-day analytic systems. When you look at the analytics landscape of a typical mature organization, you will find the footprints of many of these innovations in its data analytics platforms. As a data engineer, you may come across analytic

pipelines that were built using technologies from different generations, and you may be expected to understand them. Therefore, it is important to be familiar with some of the key developments in analytics over time, as well as to understand the foundational concepts of analytical data storage, data management, and data pipelines.

Databases and data warehouses

There are two broad types of database systems, and both of these have been around for many years:

- **Online Transaction Processing (OLTP)** systems are primarily used to store and update transactional data in high volumes. For example, OLTP-type databases are often used to store customer records, including transaction details (such as purchases, returns, and refunds).
- **Online Analytical Processing (OLAP)** systems are primarily used for reporting on large volumes of data. It is common for OLAP systems to take data from multiple OLTP databases, and provide a centralized repository of data that can be used for reporting purposes.

In this book, we are focused primarily on data analytics, and therefore most of our discussions will be around OLAP systems.

Both OLTP and OLAP data processing and analytic systems have evolved over several decades. In the 1980s, the focus was on batch processing, where data would be processed in nightly runs on large mainframe computers.

In the 1990s, the use of databases exploded, and organizations found themselves with tens, or even hundreds, of databases supporting different business processes. Generally, these databases were for transactional processing (OLTP), and the ability to perform analytics across systems was limited.

As a result, in the 1990s, data warehouses (OLAP systems) became a popular tool with which data could be ingested from multiple database systems into a central repository, and the data warehouse could focus on running analytic reports.

The data warehouse was designed to store **integrated**, **highly curated**, and **trusted** data, that was also very structured (formatted with a well-defined schema). Data would be ingested on a regular basis from other highly structured sources, but before entering the data warehouse, the data would go through a significant amount of pre-processing, validation, and transformations. Any changes to the data warehouse schema (how the data was organized, or structured), or the need to ingest new data sources, would require a significant effort to plan the schema and related processes.

Over the last few decades, businesses and consumers have rapidly adopted web and mobile technologies, and this has resulted in rapid growth in data sources, data volumes, and the options for analyzing an increasing amount of data. In parallel, organizations have realized the

business value of the insights they can gain by combining data from their internal systems with external data from their partners, the web, social media, and third-party data providers. To process consistently larger data volumes and increased demands to support new consumers, data warehouses have evolved through multiple generations of technology and architectural innovations.

Early data warehouses were custom-built using common relational databases on powerful servers, but they required IT teams to manage host servers, storage, software, and integrations with data sources. These were difficult to manage, and so in the mid-2000s, there was an emergence of purpose-built hardware appliances designed as **modular data warehouse appliances, built for terabyte- and petabyte-scale big data processing**. These appliances contained new hardware and software innovations and were delivered as easy-to-install and easy-to-manage units from popular vendors such as **Oracle**, **Teradata**, **IBM Netezza**, **Pivotal Greenplum**, and others.

Dealing with big, unstructured data

While data warehouses have steadily evolved over the last 25+ years to support increasing volumes of highly structured data, there has been exponential growth in the amount of semi-structured and unstructured data produced by modern digital platforms (such as mobile and web applications, sensors, IoT devices, social media, and audio and video media platforms).

These platforms produce data at a high velocity, and in much larger volumes than data produced by traditional structured sources. To gain a competitive advantage by transforming customer experience and business operations, it has become essential for organizations to gain deeper insights from these new data sources. Traditional data warehouses are good at storing and managing flat, structured data from sources such as a set of tables, organized as a relational schema. However, they are not well suited to handling the huge volumes of high-velocity semi-structured and unstructured data that are becoming increasingly popular.

As a result, in the early 2010s, new technologies for big data processing became popular. **Hadoop**, an open-source framework for processing large datasets on clusters of computers, became the leading way to work with big data. These clusters contained tens of hundreds of machines with attached disk volumes that could hold tens of thousands of terabytes of data managed under a single distributed filesystem known as the **Hadoop Distributed File System (HDFS)**.

Many organizations deployed **Hadoop distributions** from providers such as **Cloudera**, **Hortonworks**, **MapR**, and **IBM** to large clusters of computers in their data centers. These Hadoop packages included cluster management utilities, as well as pre-configured and pre-integrated open-source distributed data processing frameworks such as **MapReduce**, **Hive**, **Spark**, and **Presto**.

However, building and scaling on-premises Hadoop clusters typically required a large upfront capital investment in machines and storage. And, the ongoing management of the cluster and big data processing pipelines required a team of specialists that included the following:

- Hadoop administrators specialized in cluster hardware and software
- Data engineers specialized in processing frameworks such as Spark, Hive, and Presto

As the volume of data grew, new machines and storage continually needed to be added to the cluster.

Big data teams managing on-premises clusters typically spent a significant percentage of their time managing and upgrading the cluster's hardware and software, as well as optimizing workloads.

Cloud-based solutions for big data analytics

Over the last decade or so, organizations have increasingly adopted public cloud solutions to handle the challenge of increasing data volumes and diversity. Making use of cloud solutions for big data processing has a number of benefits, including:

- On-demand capacity
- Limitless and elastic scaling
- Global footprint
- Usage-based cost models
- Freedom from managing hardware

After AWS launched Amazon EMR in 2009 (a managed platform for running Hadoop frameworks), and Amazon Redshift in 2013 (a cloud-native data warehouse), the number of other companies developing cloud-based solutions for big data analytics rapidly increased.

Today, companies like **Google Cloud Platform** (**GCP**), **Microsoft Azure**, **Snowflake**, and **Databricks** provide a number of solutions for ingesting, storing, and analyzing large datasets in the cloud.

Over time, these cloud data warehouses and other cloud-based big data systems have rapidly expanded their feature sets to exceed those of high-performance, on-premise data warehousing appliances, and Hadoop clusters. Besides no upfront investment, a petabyte scale, and high performance, these cloud-based services provide elastic capacity scaling, a usage-based cost model, and freedom from infrastructure management.

Over the last decade, the number of organizations building their big data processing applications in the cloud has accelerated. In the last 5 years alone, thousands of organizations have migrated their existing data warehouses and Hadoop applications from on-premise vendor products and appliances to cloud-based services.

Another trend brought on by the move to the cloud has been the adoption of highly durable, inexpensive, and virtually limitless cloud object stores.

Cloud object stores, such as Amazon S3, can store hundreds of petabytes of data at a fraction of the cost of on-prem storage, and support storing data regardless of its source, format, or structure. They also provide native integrations with hundreds of cloud-native and third-party data processing and analytics tools.

These new cloud object stores have enabled organizations to build a new, more integrated analytics data management approach with decoupled compute and storage, called a **data lake** architecture. A data lake architecture makes it possible to create a single source of truth by bringing together a variety of data of all sizes and types (structured, semi-structured, or unstructured) in one place: a central, highly scalable repository built using inexpensive cloud storage. A wide variety of analytic tools have been created or modified to integrate with these cloud object stores, providing organizations with many options for building data lake-based big data platforms.

Instead of lifting and shifting existing data warehouses and Hadoop clusters to the cloud as they are, many organizations are instead refactoring their previously on-premises workloads to build an integrated cloud data lake. In this approach, all data is first ingested, processed, and stored in the data lake to build a single source of truth, and then a subset of the "hot" data is loaded into the dimensional schemas of a cloud data warehouse to support lower-latency access.

In recent years, another term has been coined to refer to a variety of new technologies that enable integrating the best of data lakes and data warehousing capabilities, called the data lake house approach. There have been solutions from commercial companies (such as AWS, GCP, Snowflake, and Databricks), as well as open-source community-led initiatives (such as Apache Hudi, and Apache Iceberg) that have referred to a data lake house (also sometimes called a *Lakehouse*).

While we will cover these new approaches in more detail in this chapter in the *Bringing together the best of data warehouses and data lakes* section, some of the new capabilities include:

- The ability to quickly ingest any type of data
- Storing and processing petabytes of unstructured, semi-structured, and structured data
- Support for **ACID** transactions (which references 4 key properties of a transaction - namely its atomicity, consistency, isolation, and durability – enabling multiple concurrent users to read, insert, update, and delete records in a dataset managed by the data lakehouse)
- Low-latency data access
- The ability to consume data with a variety of tools, including SQL, Spark, machine learning frameworks, and business intelligence tools

Before we dive deeper into the *data lake house* architecture, let's first review some of the fundamentals of data warehouses.

A deeper dive into data warehouse concepts and architecture

An **Enterprise Data Warehouse (EDW)** is the central data repository that contains structured, curated, consistent, and trusted **data assets** that are organized into a well-modeled schema. The **data assets** in an EDW are made up of all the relevant information about key business domains and are built by integrating data sourced from the following places:

- Run-the-business transactional applications (ERPs, CRMs, and line-of-business applications) that support all the key business domains across an enterprise.
- External data sources such as data from partners and third parties.

An EDW provides business users and decision-makers with an easy-to-use, central platform that helps them find and analyze a well-modeled, well-integrated, single version of truth for various business subject areas such as customers, products, sales, marketing, the supply chain, and more. Business users analyze data in the warehouse to measure business performance, find current and historical business trends, find business opportunities, and understand customer behavior.

In the remainder of this section, we will review the foundational concepts of a data warehouse by discussing a typical data management architecture with an EDW at the center, as depicted in *Figure 2.1*. Typically, a data-warehouse-centric architecture includes the following:

- Data sources from across the business that provide raw data to the data warehouse via **Extract, Transform, Load (ETL)** or **Extract, Load, Transform (ELT)** processes (more on this later in this chapter)
- One or more data warehouses (and, optionally, multiple subject-focused data marts)
- End user analytic tools for consuming data from the warehouse (such as SQL-based analytic tools, and business intelligence visualization systems)

The following diagram shows how an enterprise data warehouse fits into an analytics architecture:

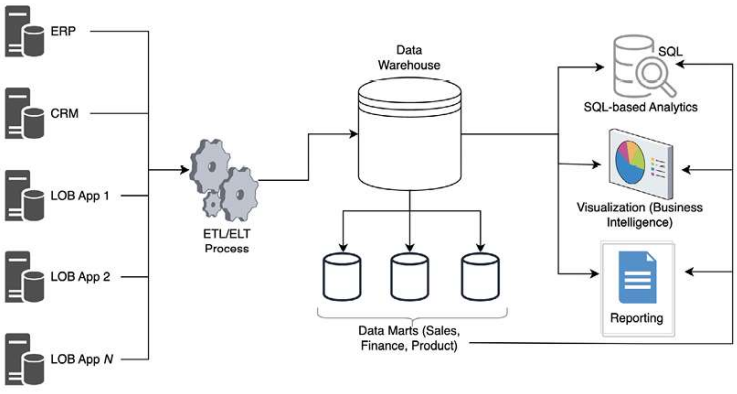

Figure 2.1: Enterprise data warehousing architecture

At the center of our architecture is the EDW, which hosts a set of *data assets* that contain current and historical data about key business subject areas. We also have optional data marts that contain a subset of the data from the warehouse, focused on and optimized for queries in a specific business domain (sales, finance, product, etc.).

On the left-hand side, we have our source systems and an ETL pipeline to load the data into the warehouse and transform the data. On the right-hand side, we can see several systems/applications that consume data from the data warehouse and data marts.

Before we dive deeper into the technical architecture and optimization techniques used in modern data warehouses, let's review some of the foundational concepts around data modeling in data warehouses.

Dimensional modeling in data warehouses

Data assets in the warehouse are typically stored as relational tables that are organized into widely used dimensional models, such as a **star schema** or **snowflake schema**. Storing data in a warehouse using a dimensional model makes it easier to retrieve and filter relevant data, and it also makes analytic query processing flexible, simple, and performant.

Let's dive deeper into two widely used data warehouse modeling techniques, and see how we can organize sales domain entities as an example. Note that this example is just a subsection of a much larger data warehouse schema.

Figure 2.2 illustrates how data in a sales subject area can be organized using a star schema:

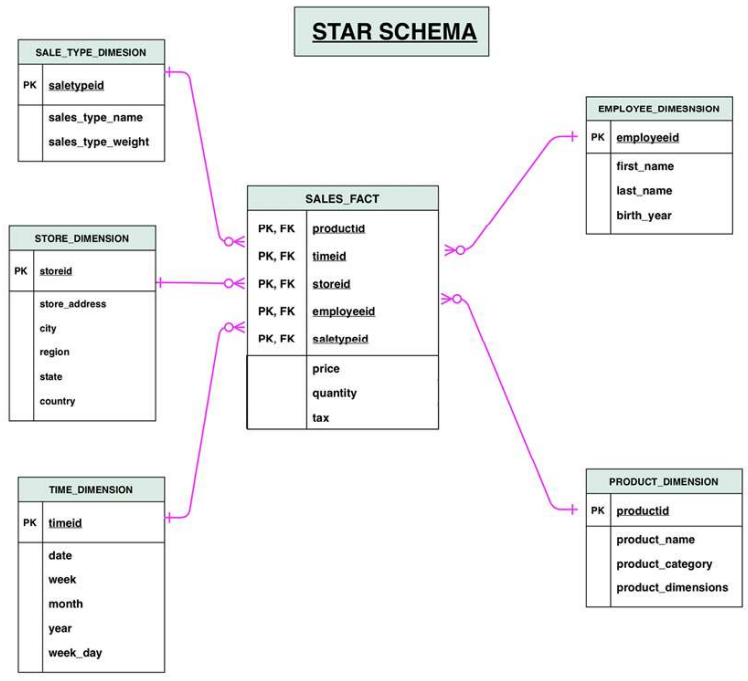

Figure 2.2: Sales data entities organized into a star schema

In data warehouses, tables are generally separated into **fact tables** and **dimension tables**. In *Figure 2.2*, the data entities are organized like a star, with the sales fact table forming the middle of the star, and the dimension tables forming the corners.

A fact table stores granular numeric measurements/metrics for a specific domain (such as sales). In *Figure 2.2*, we have a `SALES_FACT` table that stores facts about an individual sales transaction, such as the sales **price** and **quantity** sold. The fact table also has a large number of **foreign key** columns that reference the primary keys of associated dimension tables.

The dimension tables store the context under which fact measurements were captured. In *Figure 2.2*, for example, we have dimension tables with information on **stores**, **products**, and other dimensions related to each sale transaction. Each individual dimension table essentially provides granular attributes related to one of the dimensions of the fact (such as the store where the sale took place).

For example, for a specific sale, we record the price, quantity, and tax for the sale in the fact table, and then we reference dimensions such as the `store_id` for the store in which the sale took place and the `product_id` for the product that was sold. Instead of storing all the details for the store in the fact table (such as street address, region, country, and phone number), we just store the `store_id` related to the specific sale as a foreign key. When we want to query sales data, we can do a **join** between the `SALES_FACT` table and the `STORE_DIMENSION` table to report on the sale details (price and quantity) along with the region and country of the store (or any other details captured in the dimension table). We can also do a join between other dimension tables to retrieve details about the product sold, the employee that did the sale, the day of the week of the sale, etc.

Dimensional attributes are key to finding and aggregating measurements stored in the fact tables in a data warehouse. Business analysts typically slice, dice, and aggregate facts from different dimensional perspectives to generate business insights about the subject area represented by the star schema. They can find answers to questions such as:

- What is the total volume of a given product sold over a given period?
- What is the total revenue in a given product category?
- Which store sells the greatest number of products in a given category?

In a **star schema**, while data for a subject area is normalized by splitting measurements and context information into separate **fact** and **dimension** tables, individual dimension tables are typically kept denormalized so that all related attributes of a dimensional topic can be found in a single table.

This makes it easier to find all related attributes of a dimensional topic in a single table (fewer joins, and a simpler-to-understand model), but for larger dimension tables, a denormalized approach can lead to data duplication and inconsistencies within the dimension table. Large denormalized dimension tables can also be slow to update.

One approach to work around these issues is a slightly modified type of schema, the **snowflake schema**, which is shown in *Figure 2.3*:

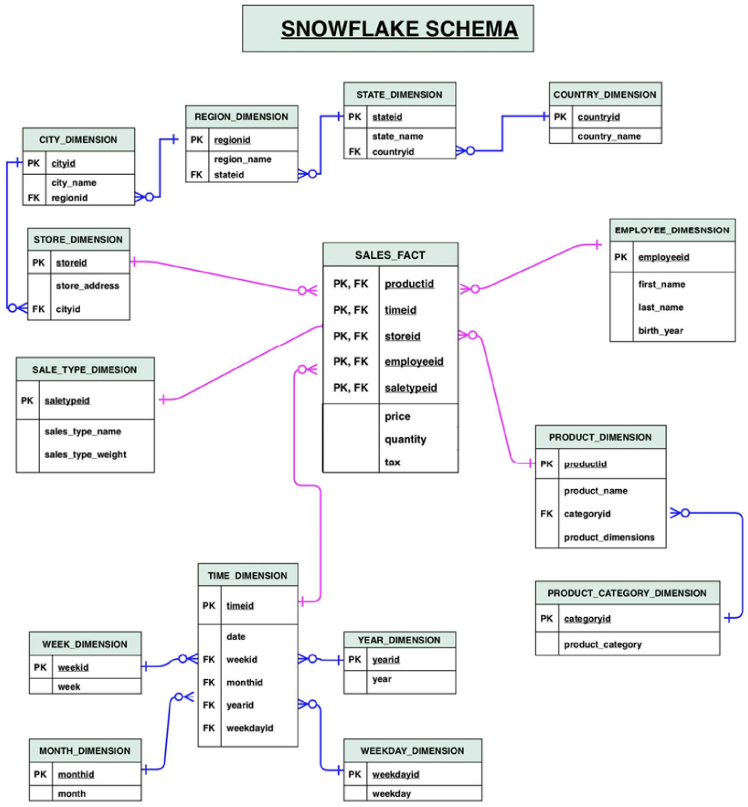

Figure 2.3: Sales data entities organized into a snowflake schema

The challenges of inconsistencies and duplication in a star schema can be addressed by **snowflaking** (basically normalizing) each dimension table into multiple related dimension tables (normalizing the original *product* dimension into *product* and *product category* dimensions, for example). This normalization of tables continues until each individual dimension table contains only attributes with a direct correlation with the table's primary key. The highly normalized model resulting from this snowflaking is called a **snowflake schema**.

The snowflake schema can be designed by extending the star schema or can be built from the ground up by ensuring that each dimension is highly normalized and connected to related dimension tables forming a hierarchy. A snowflake schema can reduce redundancy and minimize disk space, compared to a star schema, which often contains duplicate records. However, on the other hand, the snowflake schema may necessitate complex joins to answer business queries and may slow down query performance.

To decide on whether to use a snowflake or star schema, you need to consider the types of queries that are likely to be run against the dataset and balance the pros and cons of potentially slower and more complex queries with a snowflake schema versus less performant updates and more complexity in managing changes to dimension tables with a star schema.

Let's now take a closer look at the role of data marts, which can be used to provide a data repository that is easier to work with, with a schema focused on a specific aspect of a business.

Understanding the role of data marts

Data warehouses contain data from all relevant business domains and have a comprehensive yet complex schema. Data warehouses are designed for the cross-domain analysis that's required to inform strategic business decisions. However, organizations often also have a narrower set of users who want to focus on a particular line of business, department, or business subject area. These users prefer to work with a repository that has a simple-to-learn schema, and only the subset of data that focuses on the area they are interested in. Organizations typically build **data marts** to serve these users.

A data mart is focused on a single business subject repository (for example, marketing, sales, or finance) and is typically created to serve a narrower group of business users, such as a single department. A data mart often has a set of denormalized fact tables organized into a much simpler schema compared to that of an EDW. Simpler schemas and a reduced data volume make data marts faster to build, simpler to understand, and easier to use for end users. A data mart can be created either as:

- **Top-down**: Data is taken from an existing data warehouse, focused on a slice of business subject data
- **Bottom-up**: Data is sourced directly from the transactional databases that are used to run a specific business domain of interest

Both data warehouses and data marts provide an integrated view of data from multiple sources, but they differ in the scope of data they store. Data warehouses provide a central store of data for the entire business, or division, and cover all business domains. Data marts serve a specific business function by providing an integrated view of a subject area relevant to that business function.

So far, we have discussed various aspects of data warehouses and data marts, the central data repositories of our *EDW architecture* from *Figure 2.1*. Now, let's do a deeper dive into the technical architecture and optimizations that make modern data warehouses effective at querying large volumes of data.

Distributed storage and massively parallel processing

In *Figure 2.4*, we see the underlying architecture of an **Amazon Redshift** cluster. There are various different types of Redshift nodes, and *Figure 2.4* shows a Redshift cluster based on **RA3 nodes** (which we will take a closer look at in *Chapter 9*, *A Deeper Dive into Data Marts and Amazon Redshift*):

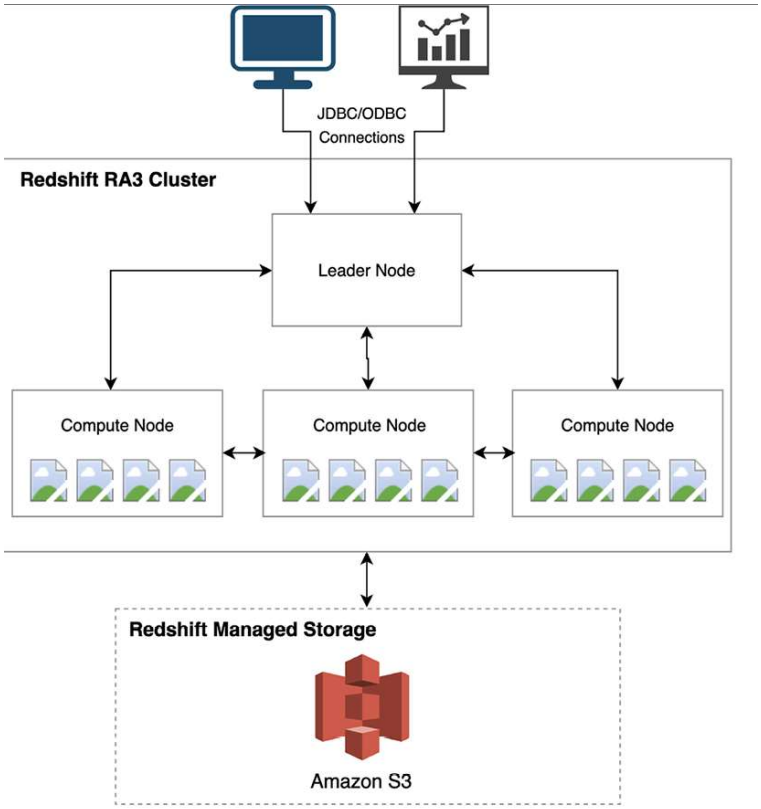

Figure 2.4: MPP architecture of an Amazon Redshift RA3 cluster

As seen in *Figure 2.4*, an Amazon Redshift cluster contains a leader node and one or more compute nodes:

- The **leader node** interfaces with client applications, receives and parses queries, and coordinates query execution on compute nodes.
- Multiple **compute nodes** have high-performance storage for storing a subset of the warehouse data and run query execution steps in parallel on the data that they store.
- For RA3 node types (as illustrated in this diagram), Amazon S3 is used as **Redshift Managed Storage** (**RMS**) for warehouse data, and the compute node high-performance local storage is used as a cache for hot data.

Each compute node has its own independent processors, memory, and high-performance storage volumes that are isolated from other compute nodes in the cluster (this is called a shared-nothing architecture).

Cloud data warehouses implement a distributed query processing architecture called **Massively Parallel Processing** (**MPP**) to accelerate queries on massive volumes of data. In this approach, the cluster leader node first compiles the incoming client query into a distributed execution plan. It then coordinates the execution of segments of compiled query code on multiple compute nodes of the data warehouse cluster, in parallel. Each compute node executes assigned query segments on a portion of the distributed dataset.

Columnar data storage and efficient data compression

In addition to providing massive storage and cluster computing, modern data warehouses also boost query performance through **column-oriented** storage and **data compression**. In this section, we'll examine how this works, but first, let's understand how OLTP databases store their data.

OLTP applications typically work with entire rows that include all the columns of the table (for example, reading/writing a sales record, or looking up a catalog record). To serve OLTP applications, backend databases need to efficiently read and write full rows to the disk. To speed up full-row lookups and updates, OLTP databases use a row-oriented layout to store table rows on the disk. In a **row-oriented** physical data layout, all the column values of a given row are co-located, as depicted in *Figure 2.5*:

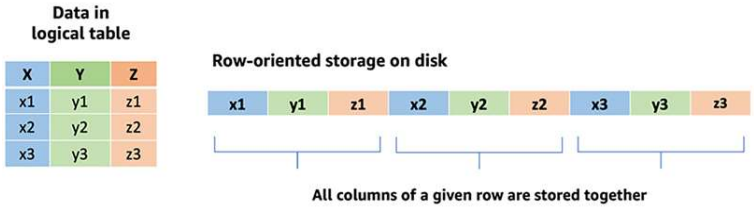

Figure 2.5: Row-oriented storage layout

Most analytics queries that business users run against a data warehouse are written to answer a specific question and typically include grouping and aggregations (such as sum, average, or mean) on a large number of rows, but of a narrow set of columns from the fact and dimension tables (these typically contain many more columns than the narrow set of columns included in the query). Analytics queries typically need to scan through a large number of rows but need data from only a narrow set of columns that relate to the specific query. A **row-oriented** physical data layout forces analytics queries to scan a large number of full rows (all columns), even though they need only a subset of the columns from these rows. Analytics queries on a row-oriented database can thus require a much higher number of disk I/O operations than necessary.

Modern data warehouses store data on disks using a **column-oriented** physical layout. This is more suitable for analytical query processing, which only requires a subset of columns per query. While storing a table's data in a column-oriented physical layout, a data warehouse breaks a table into groups of rows, called row chunks/groups. It then takes a row chunk at a time and lays out data from that row chunk, one column at a time, so that all the values for a column (that is, for that row chunk) are physically co-located on the disk, as depicted in *Figure 2.6*:

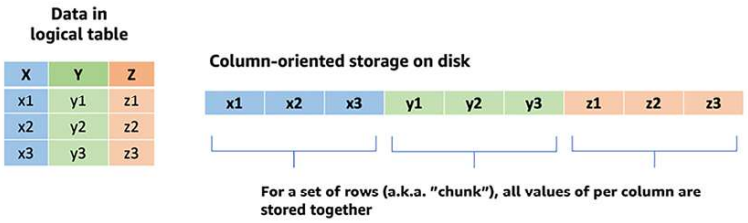

Figure 2.6: Column-oriented storage layout

Data warehouses repeat this for all the row chunks of the table. In addition to storing tables as row chunks and using a column-oriented physical layout on disks, data warehouses also maintain in-memory maps of the locations of these chunks. Modern data warehouses use these in-memory maps to pinpoint column locations on the disk and read the physically co-located values of the column. This enables the query engine to retrieve data for only the narrow set of columns needed for a given analytics query. By doing this, the disk I/O is significantly reduced compared to what would be required to run the same query on a row-oriented database.

In addition to using a column-orientated storage layout, modern data warehouses also employ multiple **compression algorithms** for a table. The warehouse is able to match individual columns with the compression algorithm that is most optimal for the given column's type and profile of its data content.

In addition to saving storage space, compressed data requires much lower disk I/O to read and write data to the disk. Compression algorithms provide much better compression ratios when all values being compressed have the same data type and have a larger percentage of duplicates. Since column-oriented databases lay out values of the same column (hence, the same data type, such as strings or integers) together, data warehouses achieve good compression ratios, resulting in faster read/writes, and smaller on-disk footprints.

Now that we have a better understanding of the architecture of modern data warehouses, let's look at the processes (often referred to as pipelines), that are used to move data into a data warehouse, and to transform the data to optimize it for analytics.

Feeding data into the warehouse – ETL and ELT pipelines

To bring data into the warehouse (and optionally, data marts), organizations typically build data pipelines that do the following:

- Extract data from source systems.
- Transform source data by validating, cleaning, standardizing, and curating it.
- Load the transformed source data into the enterprise data warehouse schema, and optionally a data mart as well.

In these pipelines, the first step is to **extract** data from source systems, but the next two steps can either take on a **Transform-Load** or **Load-Transform** sequence (so either ETL or ELT).

The data warehouses of a modern organization typically ingest data from a diverse set of sources, such as **ERP** and **CRM** application databases, files stored on **Network-Attached Storage** (**NAS**) arrays, **SaaS** applications, and external partner applications. The components that are used to implement the **Extract** step of both ETL and ELT pipelines typically need to connect to these sources and handle diverse data formats (including relational tables, flat files, and continuous streams of records).

The decision as to whether to build an ETL or ELT data pipeline is based on the following:

- The complexity of the required data transformations
- The skills and tools the organization has available to build data transformation steps
- The speed at which source data needs to be made available for analysis in the data warehouse after it's produced in the source system.

Figure 2.7 shows a typical ETL pipeline for loading data into a data warehouse:

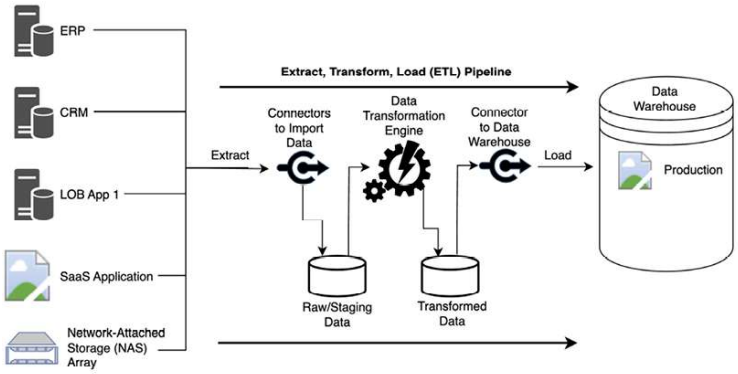

Figure 2.7: ETL pipeline

With an **ETL** pipeline, transformations are performed outside of the data warehouse using custom scripts, a cloud-native ETL service such as AWS Glue, or a specialized ETL tool from a commercial vendor such as Informatica, Talend, DataStage, Microsoft, or Pentaho.

An ETL pipeline may be a single system that has connectors to extract and load the data in addition to doing the transformations, or there may be multiple systems in the pipeline. For example, an ETL pipeline could consist of the following:

- One or more systems that extract data from various sources (databases, SaaS solutions, file storage, etc.) and write the data to a raw/staging storage area
- One or more transformation jobs that read data from the raw/staging storage area, transform the data, and then write it to a transformed

storage area
- Another system that reads data from the transformed storage area and loads the data into the data warehouse

A transformation engine may run multiple transformation jobs to perform tasks such as validating data, cleaning data, and transforming data for the target data warehouse dimensional schema.

An ETL approach to building a data pipeline is typically used when the following are true:

- Source database technologies and formats are different from those of the data warehouse
- The engineering team wants to perform transformations using a programming language (such as PySpark) rather than pure SQL
- Data transformations are complex and compute-intensive

On the other hand, an ELT pipeline extracts data (typically highly structured data) from various sources and loads it as is into the staging area of the data warehouse. The database engine powering the data warehouse is then used to perform transformation operations on the staged data and writes the transformed data to a production table (ready for consumption).

Figure 2.8 shows a typical ELT pipeline:

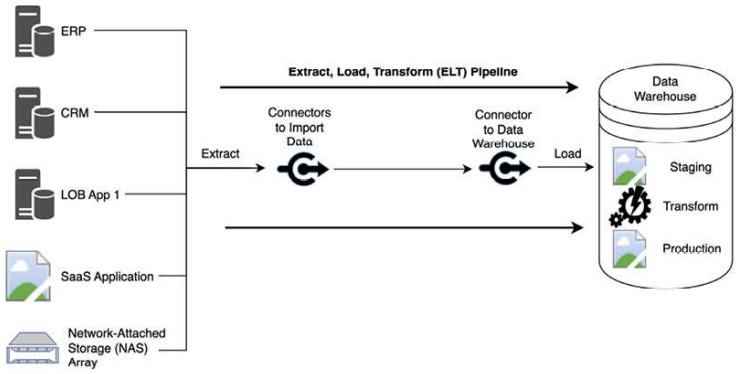

Figure 2.8: ELT pipeline

The ELT approach allows for rapidly loading large amounts of source data into the warehouse. Furthermore, the MPP architecture of modern data warehouses can significantly accelerate the transformation steps in ELT pipelines. The ELT approach is typically leveraged when the following are true:

- Data sources and the warehouse have similar database technologies, making it easier to directly load source data into the staging tables in the warehouse.
- A large volume of data needs to be quickly loaded into the warehouse.
- All the required transformation steps can be executed using the native SQL capabilities of the warehouse's database engine.

With an ELT approach, the data transformation tasks are generally performed using SQL code. While there is a large amount of SQL knowledge and skills available on the market, there are other options for performing transformations (such as by using Apache Spark with PySpark or Scala code). Using an ELT approach limits your transformations to using SQL, which may or may not be best based on the skill sets and requirements of your organization.

The primary difference between ETL and ELT is about where the data transformation takes place. With ELT, the data is loaded directly into the data warehouse, and the data warehouse engine is used for the transformation (typically using SQL to create a new, transformed version of the data). With ETL, an engine outside of the data warehouse first transforms the data before writing it to the data warehouse.

In this section, we learned how data warehouses can store and process petabytes of structured data. Modern data warehouses provide high-performance processing using a dimensional data model (such as star or snowflake), compute parallelism, and a columnar physical data layout with compression. Data management architectures at modern organizations, however, also need to store and analyze exploding volumes of semi-structured and unstructured data. In the next section, we'll learn about a newer architecture, called data lakes, that today's leading organizations typically implement to store, process, and analyze structured, semi-structured, and unstructured data.

An overview of data lake architecture and concepts

As we saw in the previous section, EDWs have been the go-to repositories for storing highly structured tabular data sourced from the transactional databases used by business applications. However, the lack of a well-defined tabular structure makes typical data warehouses less suitable for storing unstructured and semi-structured data. Also, while they are good for use cases that need SQL-based processing, data warehouses are limited to processing data primarily using only SQL, and SQL is not the right tool for all data processing requirements. For example, extracting metadata from unstructured data, such as audio files or images, is best suited for specialized machine learning tools.

A cloud data lake is a central, highly scalable repository in the cloud where an organization can manage exabytes of various types of data, including:

- Structured data (row-column based tables)
- Semi-structured data (such as JSON and XML files, log records, and sensor data streams)
- Unstructured data (such as audio, video streams, Word/PDF documents, and emails)

Data from any of these sources can be quickly loaded into the data lake as is (keeping the original source format and structure). Unlike with data

warehouses, data does not need to first be converted into a standard structure before it is consumed.

A cloud data lake also natively integrates with cloud analytic services that are decoupled from data lake storage and enables diverse analytic tools, including **SQL**, code-based tools (such as **Apache Spark**), specialized **machine learning tools**, and **business intelligence** visualization tools.

In the next section, we dive deeper into the architecture of a typical data lake.

Data lake logical architecture

Let's take a closer look at the architecture of a cloud-native data lake by looking at its logical architecture, as shown in *Figure 2.9*:

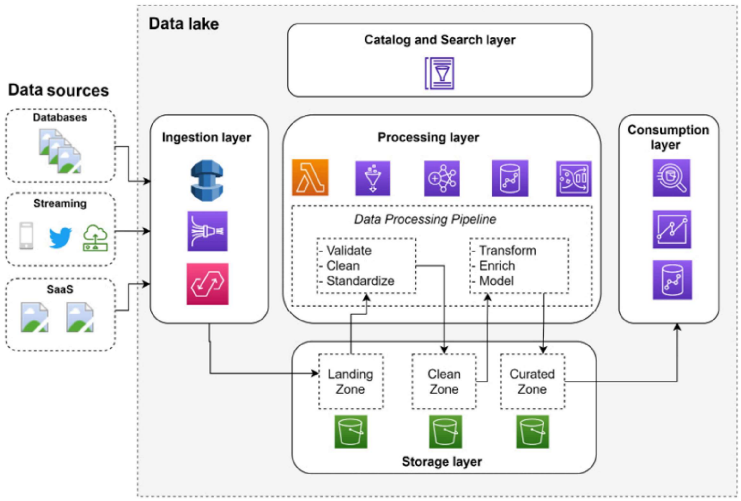

Figure 2.9: Logical layers of a data lake architecture

We can visualize a data lake architecture as a set of independent components organized into five logical layers. A layered, component-oriented data lake architecture can evolve over time to incorporate new innovations in data management and analytics methods, as well as to make use of new tools. This keeps the data lake responsive to new data sources and changing requirements. In the next section, we will dive deeper into these layers.

The storage layer and storage zones

At the bottom of the data lake architecture illustrated in *Figure 2.9* is the storage layer, built on a cloud object store such as Amazon S3. This provides virtually unlimited, low-cost storage that can store a variety of data, irrespective of the structure or format.

The storage layer is organized into different **zones**, with each zone having a specific purpose. Data moves through the various zones of the data lake, with new, modified copies of the data in each zone as the data goes through various transformations. There are no hard rules about how

many zones there should be, or the names of zones, but the following zones are commonly found in a typical data lake:

- **Landing/raw zone**. This is the zone where the ingestion layer writes data, as is, from the source systems. The landing/raw zone permanently stores the raw data from the source.
- **Clean/transform zone**. The initial data processing of data in the landing/raw zone, such as validating, cleaning, and optimizing datasets, writes data into the clean/transform zone. The data here is often stored in optimized formats such as **Parquet** and is often partitioned to accelerate query execution and downstream processing. Data in this zone may also have had PII information removed, masked, or replaced with tokens.
- **Curated/enriched zone**. The data in the clean/transformed zone may be further refined and enriched with business-specific logic and transformations, and this data is written to the curated/enriched zone. This data is in its most consumable state and meets all organizational standards (in terms of cleanliness, file formats, and schema). Data here is typically partitioned, cataloged, and optimized for the consumption layer.

Depending on the business requirements, some data lakes may include more or fewer zones than the 3 zones highlighted above. For example, a very simple data lake may just have two zones (the raw and curated zones), while some data lakes may have 5 or more zones to handle intermediate stages or specific requirements.

Catalog and search layers

A data lake typically hosts a large number of datasets (potentially thousands), from a variety of internal and external sources. These datasets are often useful to multiple teams across the organization, and these teams need the ability to search for available datasets and review the schema and other metadata of those datasets.

A **technical catalog** is used to map the many files stored in the storage layer into a logical representation of databases and tables, with each table having columns of a specific data type (the table schema). A technical catalog, such as the AWS Glue Data Catalog, stores the metadata that defines the relationship between physical files on storage and a table definition in the catalog. Consumption tools (such as Amazon Athena) can use the technical catalog to understand which files to read from the storage layer when a user queries a specific database and table.

A **business catalog** focuses on the metadata that is important to the business. This may include attributes such as the data owner, the date the dataset was last updated, a description of the table purpose, column definitions, and more. The business catalog may also integrate with the technical catalog in order to provide information on the table schema in the business catalog interface. The business catalog should also support the ability to do advanced searches, enabling teams to find data that is relevant to their use cases. We do a deep dive into data cataloging in *Chapter 4, Data Governance, Security, and Cataloging*.

Ingestion layer

The **ingestion layer** is responsible for connecting to diverse types of data sources and bringing their data into the landing/raw zone of the storage layer. This layer may contain a variety of independent tools, each purpose-built to connect to a data source with a distinct profile in terms of:

- Data structure (structured, semi-structured, or unstructured)
- Data delivery type (table rows, data stream, data file)
- Data production cadence (batch or streaming)

This approach provides the flexibility to easily add new tools to match a new data source's distinct profile.

A typical ingestion layer may include tools such as **AWS Database Migration Service** (**DMS**) for ingesting from various databases, **Amazon Kinesis Firehose** for ingesting streaming data, and **Amazon AppFlow** for ingesting data from SaaS applications. We do a deep dive into the ingestion layer in *Chapter 6, Ingesting Batch and Streaming Data*.

The processing layer

Once the ingestion layer brings data from a source system into the landing zone, it is the **processing layer** that makes it ready for consumption by data consumers. The processing layer transforms the data in the lake through various stages of data cleanup, standardization, and enrichment. Along the way, the processing layer stores transformed data in the different zones – writing it into the clean zone and then the curated zone, and then ensuring that the technical data catalog gets updated. Tools commonly used in this layer include **AWS Glue** and **Amazon EMR**.

Components in the ingestion and processing layers are used to create ELT pipelines. In these pipelines, the ingestion layer components extract data from the source systems and load the data into the data lake, and then the processing layer components transform it to make it suitable for consumption by components in the consumption layer. We will do a deep dive into the processing layer in *Chapter 7*, *Transforming Data to Optimize It for Analytics*.

The consumption layer

Once data is ingested and processed to make it consumption-ready, it can be analyzed using several techniques, such as interactive query processing, business intelligence dashboarding, and machine learning. To perform analytics on data in the lake, the consumption layer provides purpose-built tools that are able to access data from the storage layer, and the schema from the catalog layer (to apply schema-on-read to the lake-hosted data). We will do a deeper dive into data consumption in *Chapter 8, Identifying and Enabling Data Consumers*.

Data lake architecture summary

In this section, we learned about data lake architectures and how they can enable organizations to manage and analyze vast amounts of structured, unstructured, and semi-structured data.

Analytics platforms at a typical organization need to serve warehousing-style structured data analytics use cases (such as for complex queries and BI dashboarding), as well as use cases that require managing and analyzing vast amounts of unstructured and semi-structured data. As a result, organizations typically end up building both a data warehouse and a data lake, often with little interaction between the warehouse and the lake.

In the next section, we will look at modern data management and analytic architectures that integrate the best of both data warehouses and data lakes.

Bringing together the best of data warehouses and data lakes

In today's highly digitized world, data about customers, products, operations, and the supply chain can come from many sources and can have a diverse set of structures. To gain deeper and more complete data-driven insights into a business topic (such as a customer journey, customer retention, product performance, etc.), organizations need to analyze all topic-relevant data, of all structures, from all sources, together.

A data lake is well suited to storing all these different types of data inexpensively and provides a wide variety of tools to work with and consume the data. This includes the ability to transform data with frameworks such as **Apache Spark**, to train machine learning models on the data using tools such as **Amazon SageMaker**, and to query the data using **SQL** with tools such as **Amazon Athena**, **Presto**, or **Trino**.

However, there are some limitations to traditional data lakes. For example, traditional implementations of data lakes do not support the **ACID** (atomicity, consistency, isolation, and durability) properties common in most databases. Also, due to the use of inexpensive object storage as the storage layer, the query performance does not match what is possible with data warehouses that use high-performance, SSD-based local storage.

These limitations cause complexity when you have multiple teams working on the same dataset, as one team updating data in the data lake while another team attempts to query the data lake can lead to inconsistencies in the queries. Also, when you have heavily used dashboards and reporting as is common with **Business Intelligence** applications, the performance of queries on data lake data may not meet requirements.

These challenges are often worked around by loading a subset of the data from the data lake into a data warehouse, such as **Amazon Redshift** or **Snowflake**. This offers the performance needed for demanding business intelligence applications and also provides consistency when multiple teams are working with the same dataset. However, data warehouse storage is expensive, and some use cases require joining data across a diverse set of data and it is not economical to load all this data into the data warehouse.

To work around these challenges, new table formats have been created that simplify the process of updating data lake tables in a transactionally safe way, and new functionality is available to enable federated queries (joins of data across different storage engines) in an approach often referred to as a **data lake house**.

The data lake house approach

During the year 2020, a number of vendors started using a new term to talk about an approach that brought together the best of data warehouses and data lakes. Some vendors referred to this as a **Lakehouse**, while others called it a **data lake house**, or **data lakehouse**. In addition to these differences in the name of the approach, the different vendors had slightly different definitions of what a lake house was.

Even today you can read blogs and articles about different lake house approaches from companies such as AWS, Azure, Google, Snowflake, Databricks, Dremio, and Starburst. Because of this, I would consider the lake house terminology more of a marketing term than a technical term. Ultimately, there is no standard definition of a lake house, beyond the intention of different vendors to provide the best of both data warehouses and data lakes with their own technology stacks.

However, there are a number of widely adopted technologies and approaches that enable these companies to blur the lines between a data warehouse and a data lake.

New data lake table formats

Over the past few years, a number of new table formats have been proposed that are effectively a new generation of the original Hive table format, a format developed at Facebook more than a decade ago. While Hive has been fundamental in enabling the creation and growth of data lakes, there are a number of challenges with the Hive format that put significant limits on Hive-based data lakes. As a result, today there are three primary competing new table formats that can be used to develop modern data lakes.

While each of these table formats has its own strengths and weaknesses, they are all intended to enable simplified and more consistent updates and reads of data lake data (especially when you have multiple teams working with the same dataset), as well as to offer performance improvements, and the ability to query a table as it was at a certain point in time (often referred to as time travel). Many big data solution providers are adding support for one or more of these table formats, as they build out their “data lake house” offerings. These new table formats provide functionality for working with data in a data lake that is similar to what traditionally was only available in databases and data warehouses.

The three main new-generation table formats are:

- **Delta Lake.** This is a table format created by the Databricks company and is offered both as a commercial paid version with enhanced func-

tionality and as an open-source version, which is a Linux Foundation project. A number of commercial and open-source tools are able to work with Delta Lake files, including Apache Spark, Presto, Snowflake, Redshift, and others.

- **Apache Hudi.** This is a table format that was created by Uber, and later donated to the Apache Software Foundation, and is now available as an open-source solution. Hudi has been fairly widely adopted, and blog posts/case studies that mention the use of Apache Hudi include those from companies such as Amazon Transportation Service, Walmart, Robinhood, and GE Aviation.
- **Apache Iceberg.** This table format was created by two engineers at Netflix and later donated to the Apache Software Foundation, making it available as an open-source solution. Companies that have publicly referenced making use of Apache Iceberg, or who have contributed to the open-source project, include Airbnb, Expedia, Adobe, Apple, and Lyft.

It is impossible to predict with certainty which one of these table formats (or potentially others yet to be popularized) will become dominant over the next few years, but recently it has seemed like Apache Iceberg is generating increasing interest and gaining more and more support from big data product vendors. We will cover these new table formats in more detail in *Chapter 14*, *Transactional Data Lakes*. For now, though, let's look at how federated queries enable querying across different database engines.

Federated queries across database engines

Another approach that has become common in the quest to combine the best of data lakes and data warehouses is functionality that enables queries across different database engines or storage platforms. For example, cloud data warehouses such as Amazon Redshift and Snowflake are able to query data loaded into the data warehouse, as well as data in an Amazon S3-based data lake.

With this approach, the most recent 12 months of data could be loaded into a data warehouse, while the previous 4 years of data could be stored in the data lake. Most queries would only query the most recent 12 months of data, and that would be stored within the highly performant storage of the data warehouse. However, for those queries that needed to access the historical information, the data warehouse could join tables in the S3 data lake with the recent data in the data warehouse.

This query federation can extend beyond just joining tables in the data warehouse and tables in the S3-based data lake to querying data in other storage engines as well. For example, with Amazon Redshift, you can define external tables that point to data in a PostgreSQL or MySQL database.

With federated queries, the requirement to copy data between different data systems through ETL pipelines is reduced. However, querying across different systems does not perform as well as querying on local-only data, so there are use cases where you would still want to copy data between systems. Nevertheless, having the ability to query data across systems is

useful in many situations, and helps create a more integrated big data ecosystem. AWS messaging around the lake house concept has often included highlighting this ability to have access to data in different systems, with a shared data catalog, and without needing to always create a copy of the source data in each system.

In later chapters, the hands-on exercises will cover various tasks related to ingesting, transforming, and querying data in the data lake, but in this chapter, we are still setting up some of the foundational tasks. In the next section, we will use the AWS **Command Line Interface** (**CLI**) to create a number of Amazon S3 buckets.

Hands-on – using the AWS Command Line Interface (CLI) to create Simple Storage Service (S3) buckets

In *Chapter 1*, *An Introduction to Data Engineering*, you created an AWS account and an AWS administrative user and then ensured you could access your account. Console access allows you to access AWS services and perform most functions; however, it can also be useful to interact with AWS services via the **CLI** at times.

In this hands-on section, you will learn how to access the AWS CLI, and then use the CLI to create Amazon S3 buckets (a storage container in the Amazon S3 service).

Accessing the AWS CLI

The AWS CLI can be installed on your personal computer/laptop or can be accessed from the AWS Console. To access the CLI on your personal computer, you need to generate a set of access keys.

Your access keys consist of an **access key ID** (which is comparable to a username), and a **secret access key** (which is comparable to a password). With these two pieces of information, you can authenticate as your user and perform all actions that your user is authorized to perform.

However, if anyone else were able to get your access key ID and secret access key, then they would have access to the same permissions. In the past, there have been instances where users have accidentally exposed their access key ID and secret access key, enabling malicious third parties to fraudulently use their account.

As a result, the easiest and most secure way to access the CLI is via the AWS CloudShell service, which is accessible via the console.

Using AWS CloudShell to access the CLI

CloudShell provides a terminal interface in your web browser, as part of the AWS Console, that can be used to explore and manage AWS resources, using the permissions of the user with which you log in to the console. With this approach, there is no need to generate access keys for your user.

Use the following steps to access the CLI via AWS CloudShell:

1. Log in to the AWS Console (`https://console.aws.amazon.com`) using the credentials you created in *Chapter 1*.
2. Click on the CloudShell icon, or use the search bar to search for the **CloudShell** service.

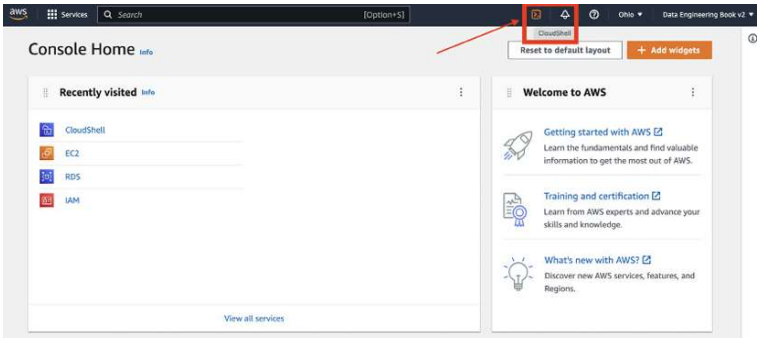

Figure 2.10: Accessing the AWS CloudShell service in the console

3. To interact with AWS services, you run the command `aws`. To learn how to interact with a specific service, you can run the `aws` command, followed by the name of the AWS service you want to interact with (such as S3), followed by `help`. For example, to learn how to use the Amazon S3 service, run the following command, which will display the help page for the S3 service:

```
aws s3 help
```

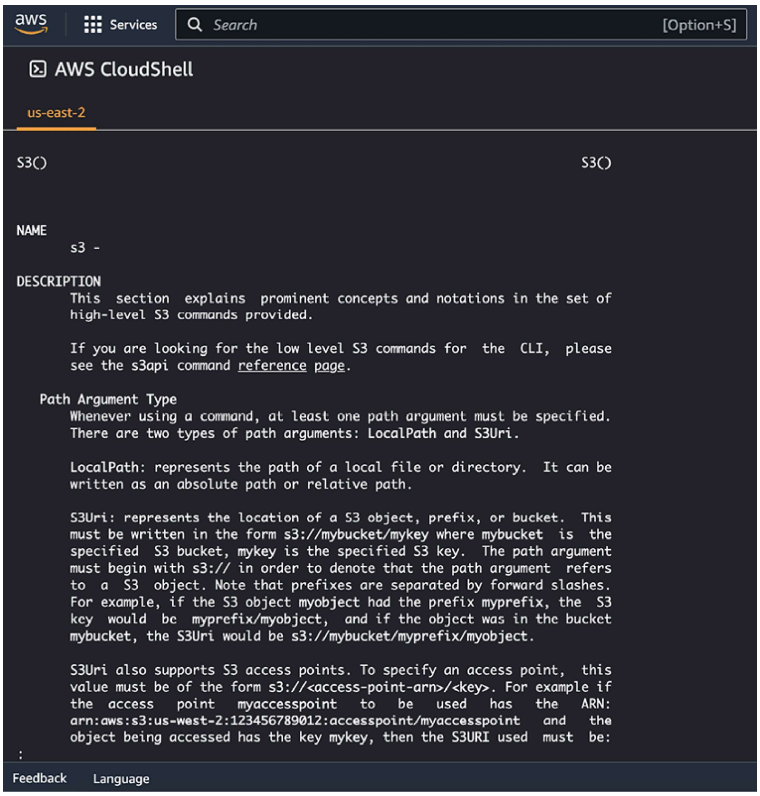

Figure 2.11: Displaying the AWS CLI help page for Amazon S3

4. Press the *SPACE* bar to display subsequent pages of the help, or press the letter *q* to quit the help pages.

Creating new Amazon S3 buckets

Amazon S3 is an object storage service that offers near-unlimited capacity with high levels of durability and availability. To store data in S3, you need to create a bucket. Once created, the bucket can store any number of objects.

Each S3 bucket needs to have a globally unique name, and it is recommended that the name be DNS-compliant. For more information on rules for bucket names, see **https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html**:

1. To create an S3 bucket using the AWS CLI, run the following command at the Command Prompt in CloudShell. In the following command, replace `<bucket-name>` with a unique name for your bucket:

```
$ aws s3 mb s3://<bucket-name>
```

   Remember that the bucket name you specify here must be globally unique. If you attempt to create a bucket using a name that another AWS account has used, you will see an error similar to the following:

```
$ aws s3 mb s3://test-bucket
make_bucket failed: s3://test-bucket An error occurred (BucketAlreadyExists) when calling the C
```

   If your `aws s3 mb` command returned a message similar to the following, then congratulations! You have created your first bucket.

```
make_bucket: <bucket-name>
```

2. We can now create additional buckets that we are going to use in some of the hands-on exercises in later chapters. As discussed earlier in this chapter, data lakes generally have multiple zones for data at different stages. To set up our data lake, we want to create S3 buckets for the **landing zone**, **clean zone**, and **curated zone**. Refer back to the section titled *The storage layer and storage zones* earlier in this chapter for a description of each of the zones.

In order to ensure that each bucket name created is globally unique, you can append some unique characters to each name, such as your initials, and if necessary, some numbers. In the examples below, I am going to append `gse23` to each bucket name to ensure it is unique.

Run the following three commands in the CloudShell terminal to create your buckets (replacing `gse23` with your own identifier).

```
$ aws s3 mb s3://dataeng-landing-zone-gse23
$ aws s3 mb s3://dataeng-clean-zone-gse23
$ aws s3 mb s3://dataeng-curated-zone-gse23
```

We have now created the storage buckets that will form the foundation of the three zones of our data lake (landing zone, clean zone, and curated zone). In later chapters, we will ingest data into the landing zone, and then create transformed copies of that data in the other zones.

Summary

In this chapter, we learned about the foundational architectural concepts that are typically applied when designing real-life analytics data management and processing solutions. We also discussed three analytics data management architectures that you would commonly find in organizations today: data warehouse, data lake, and data lakehouse.

In the next chapter, we will provide an overview of several AWS services that are used in the creation of these architectures, including services for ingesting data, services that help perform data transformation, and services that are designed for querying and analyzing data.

Learn more on Discord

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:

https://discord.gg/9s5mHNyECd