**13**

**Data Engineering for Solution Architecture**

In the internet and digitization era, data is being generated everywhere with high velocity and volume. Getting insight from these huge amounts of data at a fast pace is challenging. We need to innovate continuously to ingest, store, and process this data to derive business outcomes.

With the convergence of cloud, mobile, and social technologies, advancements in many fields such as genomics and life sciences are growing at an ever-increasing rate. Tremendous value is found in mining this data for more insight. Modern stream processing systems need to produce continual results based on data with high input rates at low latency.

The concept of *big data* refers to more than just the collection and analysis of data. The actual value for organizations in their data can be used to gain insight and create competitive advantages. Not all big data solutions must end in visualization. Many solutions such as **Machine Learning** (**ML**) and other predictive analytics feed these answers programmatically into other software or applications, extracting the information and responding as designed.

As with most things, getting faster results costs more, and big data is no exception. Some answers might not be needed immediately, so the solution's latency and throughput can be flexible enough to take hours to be completed. Other responses, such as in predictive analytics, may be needed as soon as the data is available.

In this chapter, you will learn about the following topics to handle and manage your big data needs:

* What is big data architecture?
* Designing for a big data processing pipeline
* Data ingestion, storage, processing, and analytics
* Data visualization
* Designing big data architecture
* Big data architecture best practices

By the end of this chapter, you will know how to design big data and analytics architecture. You will learn about the big data pipeline steps, including data ingestion, storage, processing, and visualization, along with various architecture patterns.

**What is big data architecture?**

The sheer volume of collected data can cause problems. With the accumulation of more and more data, managing and moving data along with its underlying big data infrastructure becomes increasingly difficult. The rise of cloud providers has facilitated the ability to move applications to the cloud. Multiple sources of data result in increased volumes, velocity, and variety. The following are some common computer-generated data sources:

* **Application server logs**: Application logs and games
* **Clickstream logs**: From website clicks and browsing
* **Sensor data**: Weather, water, wind energy, and smart grids
* **Images and videos**: Traffic and security cameras

Computer-generated data can vary from semi-structured logs to unstructured binaries. Computer-generated data sources can produce pattern-matching or correlations in data that generate recommendations for social networking and online gaming. You can also use computer-generated data to track applications or service behavior such as blogs, reviews, emails, pictures, and brand perceptions.

Human-generated data includes email searches, natural language processing, sentiment analysis on products or companies, and product recommendations. Social graph analysis can produce product recommendations based on your circle of friends, jobs you may find interesting, or even reminders based on your circle of friends' birthdays, anniversaries, and so on.

Typical barriers you hear from analytics teams that prevent them from delivering the most value to their organization are:

* **Limited insight into customer experiences and operations**: To create new customer experiences, organizations need better visibility into their business. Complex and costly data collection and processing systems and added scale costs require organizations to limit the types and amounts of data they collect and analyze.
* **Need to make quicker decisions**: This is a two-part problem:
  1. Traditional data systems are being overwhelmed, resulting in existing workloads taking a long time to complete.
  2. More decisions need to be made in seconds or minutes, requiring systems to collect and process data in real time.
* **Enabling innovation with machine learning**: Organizations are adding and growing their data science teams to help optimize and grow their business. These users need easier access to data with their choice of tools without the traditional red tape and process that will slow them down.
* **Technical staff and cost to scale self-managed infrastructures**: Customers who manage infrastructure on-premises face difficulties in quickly scaling to meet business demand. Managing infrastructure, high availability, scaling, and operational monitoring is difficult to get right, especially at scale. AWS Managed Services allows customers to focus on building their data applications, not on managing the tools.

In **big data architecture**, the general flow of a significant data pipeline starts with data and ends with insight. How you get from start to finish depends on a lot of factors. The following diagram illustrates a data workflow pipeline that will help you design your data architecture:

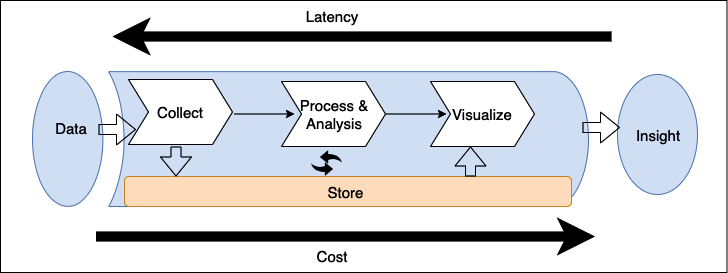


Figure 13.1: Big data pipeline for data architecture design

As shown in the preceding diagram, the standard workflow of the big data pipeline includes the following steps:

1. Data is collected (ingested) by an appropriate tool.
2. The data is stored persistently.
3. The data is processed or analyzed. The data processing/analysis solution takes the data from storage, performs operations, and then stores the processed data again.
4. The data is then used by other processing/analysis tools or by the same tool again to get further answers from the data.
5. To make answers useful to business users, they are visualized using a **business intelligence** (**BI**) tool or fed into an ML algorithm to make future predictions. Once the appropriate answers have been presented to the user, this gives them insight into the data they can then use to make further business decisions.

The tools you deploy in your pipeline determine your *time-to-answer* which is the latency between when your data was created and when you can get insight from it. The best way to architect data solutions while considering latency is to determine how to balance throughput with cost because a higher performance and subsequently reduced latency usually results in a higher price.

**Designing big data processing pipelines**

One of the critical mistakes many big data architectures make is handling multiple stages of the data pipeline with one tool. A fleet of servers managing the end-to-end data pipeline, from data storage and transformation to visualization, may be the most straightforward architecture, but it is also the most vulnerable to breakdowns in the pipeline. Such tightly coupled big data architecture typically does not provide the best possible balance of throughput and cost for your needs. When you are designing a data architecture, use FLAIR data principles as explained below:

* **F**: Findability. The ability to view which data assets are available, access metadata including ownership and data classification, and other mandatory attributes for data governance and compliance
* **L**: Lineage. The ability to find the data origin, trace data back, and understand and visualize data as it flows from data sources to consumption
* **A**: Accessibility. The ability to request a security credential granting entitlement to access the data asset. It also requires a networking infrastructure to facilitate efficient access
* **I**: Interoperability. Data is stored in a format that will be accessible to most, if not all, internal processing systems
* **R**: Reusability. Data is registered with a known schema, and attribution of the data source is clear. May encompass **MDM** (**Master Data Management**) concepts

Big data architects recommend decoupling the pipeline between ingestion, storage, processing, and getting insight. There are several advantages to decoupling storage and processing in multiple stages, including increased *fault tolerance*. For example, if something goes wrong in the second round of processing and the hardware dedicated to that task fails, you won't have to start again from the beginning of the pipeline; your system can resume from the second storage stage. Decoupling your storage from various processing tiers gives you the ability to read and write to multiple data stores.

The following diagram illustrates various tools and processes to consider when designing a big data architecture pipeline:

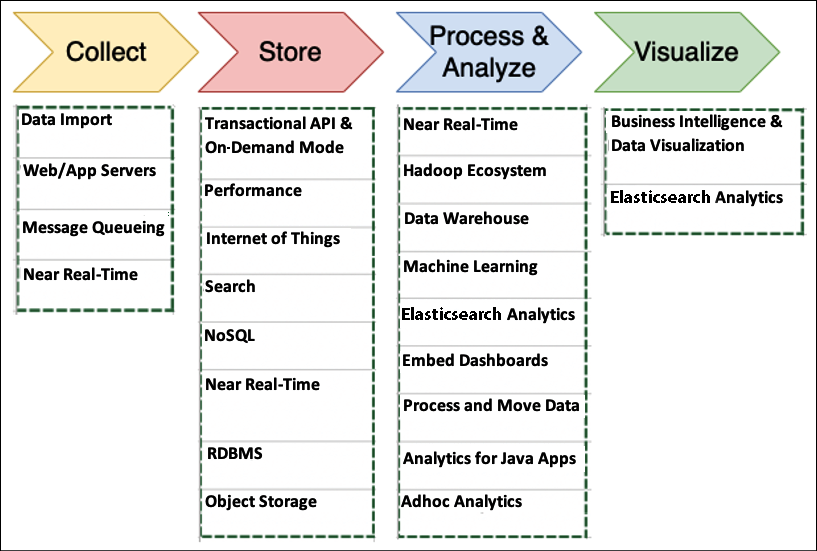


Figure 13.2: Tools and processes for big data architecture design

The things you should consider when determining the right tools for your big data architectures include the following:

* The structures of your data
* The maximum acceptable latency
* The minimum acceptable throughput
* The typical access patterns of your system's end-users

Your data structure impacts both the tools you use to process it and where you store it. The ordering of your data and the size of each object you're storing and retrieving are also essential considerations. The time-to-answer is determined by how your solution weighs latency/throughput and cost.

User access patterns are another essential component to consider. Some jobs require the regular joining of many related tables, and others require daily or less-frequent storage data. Some jobs require a comparison of data from a wide range of data sources, and other jobs pull data from only one unstructured table. Knowing how your end-users will most often use the data will help you determine the breadth and depth of your big data architecture. Let's dive deep into each process and the tools involved in big data architecture.

**Data ingestion**

Data ingestion is the act of collecting data for transfer and storage. There are lots of places that data can be onboarded. Predominantly, data ingestion falls into one of the categories from databases, streams, logs, and files. Among these, databases are the most popular. These typically consist of your main upstream transactional systems that are the primary data storage for your applications. They take on both relational and non-relational flavors, and there are several techniques for extracting data out of them.

Streams are open-ended sequences of time-series data such as clickstream data from websites or IoT devices, usually published into an API we host. Logs get generated by applications, services, and operating systems. A data lake is a great place to store all of the data for centralized analysis. Data lakes provide a single source of truth to store all data in one place and break data silos across various business units in the organization. In a later section of this chapter, *Designing big data architectures*, you will learn more about data lakes. Files come from self-hosted filesystems or via third-party data feeds via FTP or APIs. As shown in the following diagram, use the type of data your environment collects and how it is collected to determine what kind of ingestion solution is ideal for your needs:

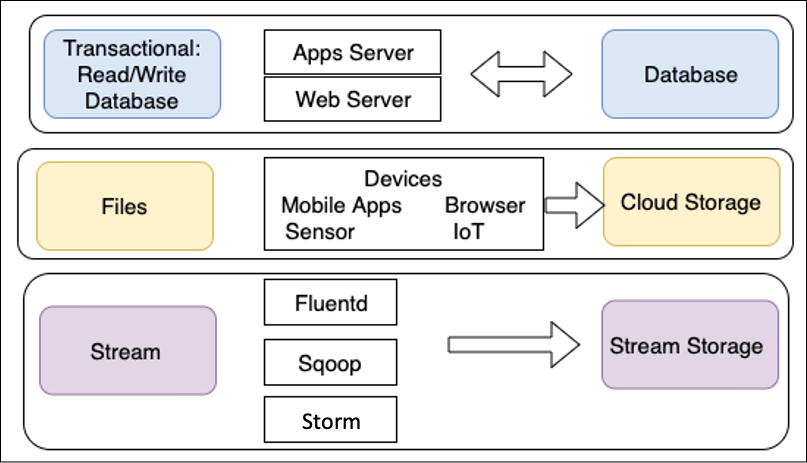


Figure 13.3: Type of data ingestion

As shown, transactional data storage must be able to store and retrieve data quickly. End-users need quick and straightforward access to the data, which makes app and web servers the ideal ingestion methods. For the same reasons, NoSQL and **Relational Database Management System** (**RDBMS**) databases are usually the best solutions for these kinds of processes.

Data transmitted through individual files is typically ingested from connected devices. A large amount of file data does not require fast storage and retrieval compared to transactional data. For file data, often a transfer is one-way, where data is produced by multiple resources and ingested into a single object or file storage for later use.

Stream data such as clickstream logs should be ingested through an appropriate solution such as **Apache Kafka** or **Fluentd**. Initially, these logs are stored in stream storage solutions such as Kafka, so they're available for real-time processing and analysis. Long-term storage of these logs is best in a low-cost solution such as object storage.

Streaming storage decouples your collection system (producers) from the processing system (consumers). It provides a persistent buffer for your incoming data. The data can be processed, and you can pump the data at a rate dependent on your needs. Let's learn about some popular data ingestion technologies.

**Technology choices for data ingestion**

Let's look at some popular open source tools for data ingestion and transfer:

* **Apache DistCp**: DistCp stands for *distributed copy* and is part of the Hadoop ecosystem. The DistCp tool is used to copy large data within a cluster or between clusters. DistCp achieves the efficient and fast copying of data by utilizing the parallel processing distribution capability with MapReduce. It distributes directories and files into map tasks to copy file partitions from source to target. DistCp also does error handling, recovery, and reporting across clusters.
* **Apache Sqoop**: Sqoop is also part of the Hadoop ecosystem project and helps to transfer data between Hadoop and relational data stores such as RDBMS. Sqoop allows you to import data from a structured data store into **Hadoop Distributed File System** (**HDFS**) and to export data from HDFS into a structured data store. Sqoop uses plugin connectors to connect to relational databases. You can use the Sqoop extension API to build a new connector or use one of the included connectors that support data exchange between Hadoop and common relational database systems.
* **Apache Flume**: Flume is open-source software and is mainly used to ingest a large amount of log data. Apache Flume collects and aggregates data to Hadoop reliably and in a distributed manner. Flume facilitates streaming data ingestion and allows analytics.

More open-source projects are available for streaming, such as Apache Storm and Apache Samza, to provide a means of reliably processing unbounded data streams.

**Ingesting data to the cloud**

Public cloud providers such as AWS provide an array of big data services to store and process data on a large scale. The following are some options to move your data to the AWS cloud and utilize the scalability offered by the cloud provider:

* **AWS Direct Connect**: AWS Direct Connect provides up to 100 Gbps of private connectivity between the AWS cloud and your data center. A dedicated network connection reduces network latency and increases bandwidth throughput. It provides a more reliable network speed compared to internet connections where data has to hop through multiple routers. Direct Connect creates a cross-connect between the router managed either by you or a Direct Connect partner, depending on whether you are co-located in one of the AWS Direct Connect locations and the router in that location that AWS owns. The circuit itself provides both a public and a private **Virtual Interface** (**VIF**).

You can use the private VIF to directly access the resources running within your **Virtual Private Cloud** (**VPC**) on AWS and the public VIF to access the public endpoints for AWS services such as **Amazon Simple Storage Service** (**S3**).

* **AWS Snowball**: If you want to transfer a large amount of data, such as hundreds of **terabytes** (**TB**) or **petabytes** (**PB**), to the cloud, it could take years over the internet. AWS Snowball provides a tamper-proof 80 TB storage appliance that can transfer a large amount of data. It works like a large hard disk that you can plug into your on-premises data storage server, load all data, and ship it to AWS. AWS will place your data in a designated location in the cloud storage. AWS Snowball has other flavors, such as Snowball Edge, which comes with compute power along with 100 TB of storage and fulfills the use case of handling data in a remote location, such as on a cruise ship or an oil rig. It is like a small data center where you can load data and perform some analytics using the built-in compute functionality. Data can be loaded to the cloud as soon as the appliance comes online. If you have PBs of data, you can use Snowmobile, a physical 45-foot shipping container with which you can transfer 100 PB of data in one go from your data center to the AWS cloud.
* **AWS Data Migration Service** (**DMS**): AWS DMS makes it easy to securely migrate or replicate your databases and data warehouses to AWS. In DMS, you can create a data migration task, which will be connected to on-premises data via a source endpoint and uses AWS-provided storage such as RDS and Amazon S3 as the target endpoint. DMS supports full data dumps and ongoing **change data capture** (**CDC**). DMS also supports homogeneous (MySQL-to-MySQL) and heterogeneous (MySQL-to-Amazon Aurora) database migrations.

AWS provides more tools, such as **AWS DataSync** for continuous file transfer to AWS from on-premises and **AWS Transfer for SFTP** to securely ingest data from the SFTP server. As you ingest the data, it needs to be put in suitable storage to fulfill the business needs. Similarly, other public cloud providers such as Azure and GCP provide various options to ingest data in their cloud. Streaming data is also becoming very important to ingest and analyze. You will learn more about streaming data in the *Streaming data store* section. Let's learn more about techniques to choose the right storage and the available storage choices.

**Storing data**

One of the most common mistakes when setting up storage for a big data environment is using one solution, frequently an RDBMS, to handle all of your data storage requirements.

You will have many tools available, but none of them are optimized for the task they need to complete. One solution is not necessarily the best for all of your needs; the best solution for your environment might be a combination of storage solutions that carefully balance latency with cost. An ideal storage solution uses the right tool for the right job. The following diagram combines multiple factors related to your data and the storage choice associated with it:

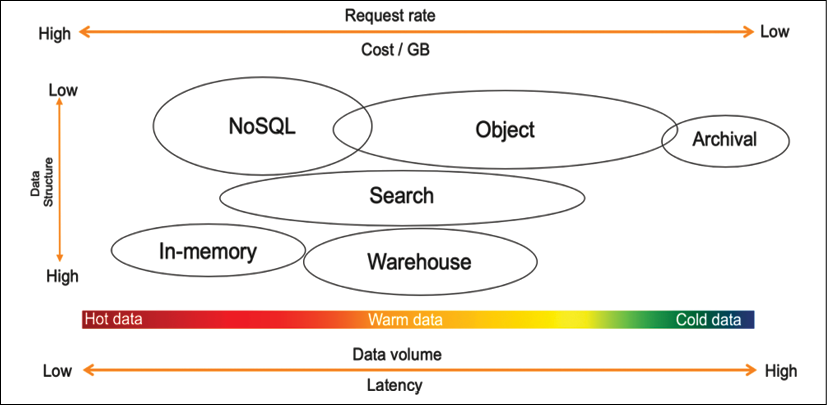


Figure 13.4: Understanding data storage

As shown in the proceeding diagram, choosing a data store depends upon the following factors:

* **How structured is your data?** Does it adhere to a specific, well-formed schema, as with Apache weblogs (logs are generally not well structured and are unsuitable for relational databases), standardized data protocols, and contractual interfaces? Is it completely arbitrary binary data, as in images, audio, video, and PDF documents? Or, is it semi-structured with a general structure but with potentially high variability across the records, as in JSON or CSV?
* **How quickly does new data need to be available for querying?** Is it a real-time scenario where decisions are made as new records stream in, such as campaign managers making adjustments based on conversion rates or a website making product recommendations based on user behavior similarity? Is it a daily, weekly, or monthly batch scenario, such as model training, financial statement preparation, or product performance reporting? Or is it somewhere in between, such as with user engagement emails, where it doesn't require real-time action, but you can have a buffer of a few minutes or even a few hours between the user action and the touchpoint?
* **What is the size of the data ingest?** Is the data ingested record by record as data comes in, such as with JSON payloads from REST APIs that measure at just a few KBs at best? Is it a large batch of records arriving all at once, such as system integrations and third-party data feeds? Or is it somewhere in between, such as with a few micro-batches of clickstream data aggregated together for more efficient processing?
* **What is the total volume of data and its growth rate?** Are you in the realm of GBs and TBs, or do you intend to store PBs or even **exabytes** (**EBs**)? How much of this data is required for your specific analytics use cases? Do the majority of your queries only require a specific rolling window of time? Or, do you need a mechanism to query the entirety of your historical dataset?
* **What the cost will be to store and query the data in any particular location**: When it comes to any computing environment, we generally see a *triangle of constraints* between performance, resilience, and low cost. The better the performance and the higher the resilience you want your storage to have, the more expensive it will be. You may wish to have quick queries over petabytes of data but decide to settle on querying TBs of data in a compressed format to meet your cost requirements.

Finally, what type of analytic queries will run against the data? Will it be powering a dashboard with a fixed set of metrics and drill-down? Will it participate in large numerical aggregations rolled up by various business dimensions? Or, will it be used for diagnostics, leveraging string tokenization for full-text searching and pattern analysis?

When you determine all characteristics of your data and understand the data structure, you can then assess which solution you need to use for your data storage. Let's learn about the various solutions for storing data.

**Technology choices for data storage**

As we discussed, a single tool can't do everything. You need to use the right tool for the right job, and a data lake enables you to build a highly configurable big data architecture to meet your specific needs. Business problems are far too broad, deep, and complex for one tool to solve everything, especially big data and analytics.

For example, hot data will need to be stored and processed in memory, so caches or in-memory databases like Redis or SAP Hana are appropriate. AWS offers the ElastiCache service, providing a managed Redis or Memcached environment. NoSQL databases are ideal when facing high velocity but small-sized records, for example, user-session information or IoT data. NoSQL databases are also useful for content management to store data catalogs. Let's learn about the most popular and commonly used storage for structured data.

**Structured data stores**

Structured data stores have been around for decades and are the most familiar technology choice for storing data. Most transactional databases such as Oracle, MySQL, SQL Server, and PostgreSQL are row-based due to dealing with frequent data writes from software applications. Organizations often repurpose transactional databases for reporting purposes, requiring frequent data reads but much fewer data writes. Looking at high data-read requirements, more innovation is coming into querying on structured data stores, such as the columnar file format, which helps to enhance data read performance for analytics requirements.

Row-based formats store the data in rows in a file. Row-based writing is the fastest way to write the data to the disk, but it is not necessarily the quickest read option because you need to skip over a lot of irrelevant data. Column-based formats store all the column values together in the file. This leads to better compression because the same data types are now grouped. It also typically provides better read performance because you can skip columns that are not required.

Let's look at common choices for the structured data store. Take an example where you need to query the total number of sales in a given month from the order table, which has fifty columns. In a row-based architecture, the query will scan the entire table with all fifty columns, but in columnar architecture, the query will just scan the order sales column, thus improving data query performance. Let's look into more details about relational databases, focusing on transaction data and data warehousing to handle data analytics needs.

**Relational databases**

RDBMS is more suitable for **Online Transaction Processing** (**OLTP**) applications. Some popular relational databases are Oracle, MSSQL, MariaDB, PostgreSQL, and so on. Some of these traditional databases have been around for decades. Many applications, including e-commerce, banking, and hotel booking, are backed by relational databases. Relational databases are very good at handling transaction data where complex joint queries between tables are required. Looking at transaction data needs, the relational database should adhere to the **Atomicity, Consistency, Isolation, Durability** (**ACID**) principles, as follows:

* **Atomicity**: Atomicity means the transaction will be executed fully from end to end, and, in the case of any error, the entire transaction will roll back.
* **Consistency**: Consistency means that when transactions are completed, all data should be committed to the database.
* **Isolation**: Isolation requires that multiple transactions can run concurrently in isolation without interfering with each other.
* **Durability**: In case of any interruption, such as a network or power failure, the transaction should be able to resume to the last known state.

Often, data from relational databases is offloaded to data warehousing solutions for reporting and aggregation purposes. Let's learn more about data warehousing.

**Data warehousing**

Data warehouse databases are more suitable for **Online Analytical Processing** (**OLAP**) applications. Data warehouses provide fast aggregation capabilities over vast volumes of structured data. While these technologies, such as Amazon Redshift, Netezza, and Teradata, are designed to execute complex aggregate queries quickly, they are not optimized for high volumes of concurrent writes. So, data needs to be loaded in batches, preventing warehouses from serving real-time insights over hot data.

Modern data warehouses use a columnar base to enhance query performance. Examples of this include Amazon Redshift, Snowflake, and Google BigQuery. These data warehouses provide very fast query performance due to columnar storage and improved I/O efficiency. In addition to that, data warehouse systems such as Amazon Redshift increase query performance by parallelizing queries across multiple nodes and taking advantage of **massively parallel processing** (**MPP**).

Data warehouses are central repositories that store accumulations of data from one or multiple sources. They store current and historical data used to help create analytical reports for business data analytics. However, data warehouses store data centrally from various systems, but they cannot be treated as a data lake. Data warehouses handle only structured relational data, while data lakes work with structured and unstructured data such as JSON logs, and CSV data.

Data warehouse solutions such as Amazon Redshift can process petabytes of data and provide decoupled compute and storage capabilities to save costs. In addition to columnar storage, Redshift uses data encoding, distribution, and zone maps to increase query performance. More traditional row-based data warehousing solutions include Netezza, Teradata, and Greenplum.

However, data warehouses cause various applications' data to be placed in separate physical locations. Data architects then have to build entirely new infrastructure around the data warehouse. The limitations of data warehouses became evident with the increasing variety of enterprise data such as text, IoT, images, audio, and videos. In addition, the rise of ML and AI introduced iterative algorithms that required direct data access and were not based on SQL. You will learn more about overcoming these challenges in a later section of this chapter, *Designing big data architectures*.

**NoSQL databases**

NoSQL databases such as DynamoDB, Cassandra, and MongoDB address the scaling and performance challenges you often experience with a relational database. As the name suggests, NoSQL means a non-relational database. NoSQL databases store data without an explicit and structured mechanism to link data from different tables (no joins, foreign keys, and normalization enforced).

NoSQL utilizes several data models, including columnar, key-value, search, document, and graph. NoSQL databases provide scalable performance, high availability, and resilience. NoSQL typically does not enforce a strict schema, and every item can have an arbitrary number of columns (attributes), which means one row can have four columns, while another row can have ten columns in the same table. The partition key is used to retrieve values or documents containing related attributes. NoSQL databases are highly distributed and can be replicated. They are durable and don't experience performance issues when highly available.

**SQL versus NoSQL databases**

SQL databases have been around for decades, and most of us are probably already very familiar with relational databases. Let's learn some significant differences between SQL and NoSQL databases:

|  |  |  |
| --- | --- | --- |
| Properties | SQL Databases | NoSQL Databases |
| Data model | In SQL databases, the relational model normalizes data into tables containing rows and columns. A schema includes tables, columns, relationships between tables, indexes, and other database elements. | NoSQL databases do not enforce a schema. A partition key is commonly used to retrieve values from column sets. It stores semi-structured data such as JSON, XML, or other documents such as data catalogs and file indexes. |
| Transaction | SQL-based traditional RDBMSes support and are compliant with the transactional data properties of ACID. | To achieve horizontal scaling and data model flexibility, NoSQL databases may trade some ACID properties of traditional RDBMSes. |
| Performance | SQL-based RDBMSes were used to optimize storage when storage was expensive and minimize the disk footprint. For traditional RDBMSes, performance has mostly relied on the disk. To achieve performance query optimizations, index creation and modifications to the table structure are required. | For NoSQL, performance depends upon the underlying hardware cluster size, network latency, and how the application is calling the database. |
| Scale | SQL-based RDBMS databases are easiest to scale vertically with high configuration hardware. The additional effort requires relational tables to span across distributed systems, such as performing data sharding. | NoSQL databases are designed to scale horizontally using distributed clusters of low-cost hardware to increase throughput without impacting latency. |

Depending on your data, various categories of NoSQL data stores exist to solve a specific problem. Let's understand the types of NoSQL databases.

**Types of NoSQL data store**

The following are the major NoSQL database types:

* **Columnar databases**: Apache Cassandra and Apache HBase are the popular columnar databases. The columnar data store helps you scan a particular column when querying the data rather than scanning the entire row. Suppose an item table has ten columns with one million rows, and you want to query the number of a given item available in inventory. In that case, the columnar database will apply the query to the item quantity column rather than scanning the entire table.
* **Document databases**: Some of the most popular document databases are MongoDB, Couchbase, MarkLogic, DynamoDB, DocumentDB, and Cassandra. You can use a document database to store semi-structured data in JSON and XML formats.
* **Graph databases**: Popular graph database choices include Amazon Neptune, JanusGraph, TinkerPop, Neo4j, OrientDB, GraphDB, and GraphX on Spark. A graph database stores vertices and links between vertices called **edges**. Graphs can be built on both relational and non-relational databases.
* **In-memory key-value stores**: Some of the most popular in-memory key-value stores are Redis and Memcached. They store data in memory for read-heavy applications. Any query from an application first goes to an in-memory database and, if the data is available in the cache, it doesn't hit the master database. The in-memory database is suitable for storing user-session information, which results in complex queries and frequently requests data such as user profiles.

NoSQL has many use cases, but you need to index all your data to build a data search. Let's learn more about search data stores.

**Search data stores**

The Elasticsearch service is one of the most popular search engines for big data use cases like clickstream and log analysis. Search engines work well for warm data that can be queried ad hoc across any number of attributes, including string tokens.

Amazon OpenSearch Service provides data search capabilities and the support of open source Elasticsearch clusters and includes API access. It also provides Kibana as a visualization mechanism to search for indexed data stores. AWS manages capacity, scaling, and patching of clusters, removing any operational overhead. Log search and analysis is a popular big data use case where OpenSearch helps you analyze log data from websites, server fleets, IoT sensors, and so on. OpenSearch and Elasticsearch are utilized by various applications in industries such as banking, gaming, marketing, application monitoring, advertisement technology, fraud detection, recommendations, and IoT. Now ML-based search services, such as Amazon Kendra, are also available, which provide more advanced search capabilities using natural language processing.

**Unstructured data stores**

When you look at the requirements for an unstructured data store, it seems that Hadoop is a perfect choice because it is scalable, extensible, and very flexible. It can run on consumer hardware, has a vast ecosystem of tools, and appears to be cost-effective to run. Hadoop uses a *master-and-child-node* model, where data is distributed between multiple child nodes, and the master node coordinates jobs for running queries on data. The Hadoop system is based on MPP, making it fast to perform queries on all types of data, whether structured or unstructured.

When a Hadoop cluster is created, each child node created from the server comes with a block of the attached disk storage called a local HDFS disk store. You can run the query against stored data using common processing frameworks like Hive, Pig, and Spark. However, data on the local disk persists only for the life of the associated instance.

If you use Hadoop's storage layer (HDFS) to store your data, you are coupling storage with compute. Increasing storage space means having to add more machines, which increases compute capacity as well. For maximum flexibility and cost-effectiveness, you need to separate compute and storage and scale them both independently. Overall, object storage is more suited to data lakes to store all kinds of data in a cost-effective and performant manner. Cloud-based data lakes backed by object storage provide flexibility to decouple compute and storage. Let's learn more about object storage.

**Object storage**

Object storage refers to data stored and accessed with units often referred to as objects stored in buckets. In object storage, files or objects are not split into data blocks, but data and metadata are kept together. There is no limit on the number of objects stored in a bucket, and they are accessed using API calls (usually through HTTP GET and PUT) to read and write from and to buckets. Typically, object storage is not filesystem mounted on operating systems because the latency of API-based file requests and lack of file-level locking provide poor performance as a filesystem. Object storage offers scale and has a flat namespace reducing management overhead and metadata management. Object storage has become more popular with the public cloud and go-to storage to build a scalable data lake in the cloud. The most popular object storage is Amazon S3, Azure Blob storage, and Google storage in GCP.

**Blockchain data store**

With the rise in cryptocurrencies, you must have heard about blockchain a lot. Blockchain technology enables the building of decentralized applications that can be verified by multiple parties rather than depending upon a single authority. Blockchain achieves decentralized verification by facilitating a blockchain network (peer-to-peer network) where participants have access to a shared database to record transactions. These transactions are immutable and independently verifiable by design.

Blockchain is not just about crypto; blockchain technologies help to solve two types of customer needs. In the first case, multiple parties work with a centralized authority to maintain verifiable transaction records. For example, manufacturers can store data from multiple systems in a centralized ledger. In the event of issues, manufacturers can quickly trace the root cause of defects and take preventive actions. Similarly, government vital record offices can implement a centralized ledger that maintains a trusted and complete record of the digital history of their citizens in a single place for vital records such as birth certificates and marriage certificates.

In other use cases, multiple parties work together in a decentralized setting where a centralized trusted authority is not required. For example, financial consortiums can reduce the time and complexity of cross-boundary payments and asset transfers by directly working with multiple parties such as insurance, trading vendors, and banks in a decentralized way. Similarly, retailers can partner with third-party loyalty programs to build seamless rewards programs for their customers without needing a central bank or vendor to process rewards.

To maintain record sanity, customers need a centralized ledger that records all application data changes and maintains an immutable record of these changes to a ledger database.

This database should be highly performant, immutable, and cryptographically verifiable, eliminating the need to build complex audit tables or set up blockchain networks. One such ledger database is Amazon **Quantum Ledger Database** (**QLDB**), which maintains a complete and verifiable history of data changes in an application that they own and manage in a centralized way.

Customers need the immutable and verifiable capability provided by a ledger and want to allow multiple parties to transact without a trusted central authority. In that case, they can use a scalable blockchain service. If you are looking for managed blockchain, some of the most popular blockchain networks include **Amazon Managed Blockchain** (**AMB**), **R3 Corda**, **Ethereum**, and **Hyperledger**.

Streaming data processing used to be a niche technology, but now it's becoming common as every organization wants to get fast insight from real-time data processing. Let's learn more about streaming data stores.

**Streaming data stores**

Streaming data has a continuous data flow with no start and end. Now lots of data coming from various real-time resources needs to be stored and processed quickly, such as stock trading, autonomous cars, smart spaces, social media, e-commerce, gaming, and ride apps, and so on. Netflix provides real-time recommendations based on the content you are watching, and Lyft rideshare uses streaming to connect passengers to a driver in real time.

Storing and processing streaming data is a challenging task as there is a continuous stream of data coming, and you cannot predict the storage capacity. Along with high volume, streaming data comes with very high velocity, which requires a scalable storage system that can store the data and provide the ability to replay it. Data streams can become very expensive to maintain and complex to manage over time. Popular streaming data storage is Apache Kafka, Apache Flink, Apache Spark Streaming, Apache Samza, and Amazon Kinesis. Now, AWS also provides managed Kafka, known as Amazon Managed Streaming for Kafka. Let's learn more details about streaming data ingestion and storage technology:

* **Amazon Kinesis**: Amazon Kinesis offers three capabilities. The first, **Kinesis Data Streams**(**KDS**), is a place to store a raw data stream to perform any downstream processing of the desired records. The second is **Amazon Kinesis Data Firehose**(**KDF**) to facilitate transferring these records into common analytic environments like Amazon S3, Elasticsearch, Redshift, and Splunk. Firehose will automatically buffer up all the records in the stream and flush out to the target as a single file or set of records based on either a time or data-size threshold that you can configure or whichever is reached first.

The third is **Kinesis Data Analytics**(**KDA**) to perform analytics on the records of the stream by performing SQL operations. The output can subsequently flow into further streams you create to build an entire serverless streaming pipeline.

* **Amazon Managed Streaming for Kafka**(**MSK**): MSK is a fully managed, highly available, and secure service. Amazon MSK runs applications on Apache Kafka in the AWS cloud without needing Apache Kafka infrastructure management expertise. Amazon MSK provides a managed Apache Kafka cluster with a ZooKeeper cluster to maintain configuration and build a producer/consumer for data ingestion and processing.
* **Apache Flink**: Flink is another open-source platform for streaming data and batch data processing. Flink consists of a streaming dataflow engine that can process bounded and unbounded data streams. A bounded data stream has a defined start and end, while an unbounded data stream has a start but no end. Flink can perform batch processing as well on its streaming engine and supports batch optimizations.
* **Apache Spark Streaming**: Spark Streaming helps ingest live data streams with high throughput and a fault-tolerant, scalable manner. Spark Streaming divides the incoming data streams into batches before sending them to the Spark engine for processing. Spark Streaming uses DStreams, which are sequences of **resilient distributed datasets** (**RDDs**).
* **Apache Kafka**: Kafka is one of the most popular open-source streaming platforms that helps you publish and subscribe to a data stream. A Kafka cluster stores a recorded stream in a Kafka topic. A producer can publish data in a Kafka topic, and consumers can take the output data stream by subscribing to the Kafka topic.

Streaming storage needs to persist a continuous stream of data and provide the ability to maintain the order if required. You will learn more about streaming architecture in the upcoming section, *Streaming data architecture*. Once you ingest and store data, it's important to process the data in the desired structure to visualize and analyze for business insights. Let's learn more details about data processing and transformation.

**Processing data and performing analytics**

Data analytics is the process of ingesting, transforming, and visualizing data to discover valuable insights for business decision-making. Over the previous decade, more data has been collected, and customers are looking for greater insights into their data.

These customers also want these insights in the least amount of time, sometimes even in real time. They want more ad hoc queries to answer more business questions. To answer these questions, customers need more powerful and efficient systems.

Batch processing typically involves querying large amounts of cold data. In batch processing, it may take hours to get answers to business questions. For example, you may use batch processing to generate a billing report at the end of the month. Stream processing in real time typically involves querying small amounts of hot data, and it takes only a short amount of time to get answers. MapReduce-based systems such as Hadoop are examples of platforms that support the batch jobs category. Data warehouses are examples of platforms that support the query engine category.

Streaming data processing activities ingest a sequence of data and incrementally update functions in response to each data record. Typically, they ingest continuously produced streams of data records, such as metering data, monitoring data, audit logs, debugging logs, website clickstreams, and location-tracking events for devices, people, and physical goods.

The following diagram illustrates a data lake pipeline for processing, transforming, and visualizing data using the AWS cloud tech stack:

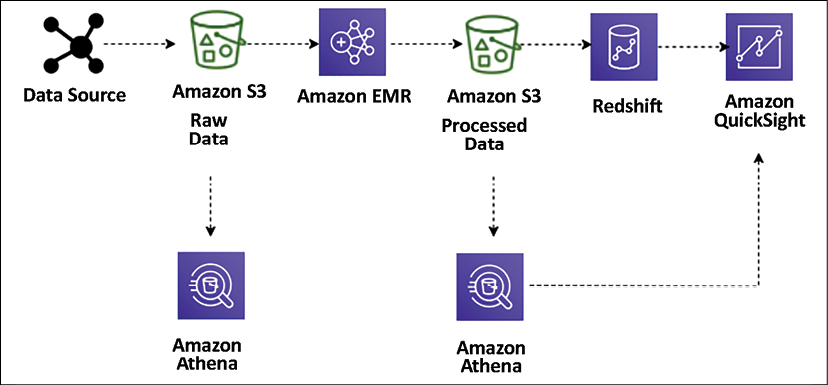


Figure 13.5: Data lake ETL pipeline for big data processing

Here, the **Extract, Transform, Load** (**ETL**) pipeline uses Amazon Athena for ad hoc querying of data stored in Amazon S3. The data ingested from various data sources (for example, web application servers) generates log files that persist into S3. These files are then transformed and cleansed into a set form required for meaningful insights using Amazon **Elastic MapReduce** (**EMR**) and loaded into Amazon S3. Amazon EMR provides a managed Hadoop server in the cloud to perform data processing using various open source technologies such as Hive, Pig, Spark, and so on.

These transformed files are loaded into Amazon Redshift using the COPY command and visualized using Amazon QuickSight. Using Amazon Athena, you can query the data directly from Amazon S3 when the data is stored and after transformation (with aggregated datasets). You can visualize the data from Athena in Amazon QuickSight. You can easily query these files without changing your existing data flow.

Let's look at some popular tools for data processing.

**Technology choices for data processing and analysis**

The following are some of the most popular data processing technologies that help you to perform transformation and processing for a large amount of data:

* **Apache Hadoop** uses a distributed processing architecture in which a task is mapped to a cluster of commodity servers for processing. Each piece of work distributed to the cluster servers can be run or re-run on any server. The cluster servers frequently use HDFS to store data locally for processing. The Hadoop framework takes a big job, splits it into discrete tasks, and processes them in parallel. It allows for massive scalability across an enormous number of Hadoop clusters. It's also designed for fault tolerance, where each of the worker nodes periodically reports its status to a master node, and the master node can redistribute work from a cluster that doesn't respond positively. Some of the most popular frameworks used with Hadoop are Hive, Presto, Pig, and Spark.
* **Apache Spark** is an in-memory processing framework. Apache Spark is a massively parallel processing system with different executors that can take apart a Spark job and run tasks in parallel. To increase the parallelism of a job, add nodes to the cluster. Spark supports batch, interactive, and streaming data sources. Spark uses **directed acyclic graphs** (**DAGs**) for all the stages during the execution of a job. The DAGs can keep track of your data or lineage transformations during the jobs and efficiently minimize the I/O by storing the DataFrames in memory. Spark is also partition-aware to avoid network-intensive shuffles.
* **Hadoop User Experience** (**HUE**) enables you to run queries and scripts on your cluster through a browser-based user interface instead of the command line. HUE provides the most common Hadoop components in a user interface. It enables browser-based viewing and tracking of Hadoop operations. Multiple users can access the cluster via HUE's login portal, and administrators can manage access manually or with LDAP, PAM, SPNEGO, OpenID, OAuth, and SAML2 authentication. HUE allows you to view logs in real time and provides a metastore manager to manipulate Hive metastore contents.
* **Pig** is typically used to process large amounts of raw data before storing it in a structured format (SQL tables). Pig is well suited to ETL operations such as data validation, data loading, data transformation, and combining data from multiple sources in multiple formats. In addition to ETL, Pig also supports relational operations such as nested data, joins, and grouping. Pig scripts can use unstructured and semi-structured data (such as web server logs or clickstream logs) as input. In contrast, Hive consistently enforces a schema on input data. Pig Latin scripts contain instructions on filtering, group, and joining data, but Pig is not intended to be a query language. Hive is better suited to querying data. The Pig script compiles and runs to transform the data based on the instructions in the Pig Latin script.
* **Hive**is an open-source data warehouse and query package that runs on top of a Hadoop cluster. SQL is a widespread skill to have that helps the team make an easy transition into the big data world. Hive uses a SQL-like language called **Hive Query Language** (**HQL**), making it easy to query and process data in a Hadoop system. Hive abstracts the complexity of writing programs in a coding language such as Java to perform analytics jobs.
* **Presto**is a Hive-like query engine, but it is much faster. It supports the ANSI SQL standard, which is easy to learn and the most popular skill set. Presto supports complex queries, joins, and aggregation functions. Unlike Hive or MapReduce, Presto executes queries in memory, which reduces latency and improves query performance. You need to be careful while selecting the server capacity for Presto, as it needs to have high memory. A Presto job will restart in the event of memory spillover.
* **HBase** is a NoSQL database developed as a part of the open-source Hadoop project. HBase runs on HDFS to provide non-relational database capabilities for the Hadoop ecosystem. HBase helps to store large quantities of data in columnar format with compression. Also, it provides a fast lookup because large portions of the data cache are kept in memory while cluster instance storage is still used.
* **Apache Zeppelin** is a web-based editor for data analytics built on top of the Hadoop system, also known as a Zeppelin notebook. It uses the concept of an interpreter for its backend language and allows any language to be plugged into Zeppelin. Apache Zeppelin includes some basic charts and pivot charts. It's very flexible in terms of any output from any language backend that can be recognized and visualized.
* **Ganglia** is a Hadoop cluster monitoring tool. However, you need to install Ganglia on the cluster during launch. The Ganglia UI runs on the master node, which you can see using an SSH tunnel. Ganglia is an open-source project designed to monitor clusters without impact on their performance. Ganglia can help to inspect the performance of the individual servers in your cluster and the performance of clusters as a whole.
* **JupyterHub** is a multi-user Jupyter notebook. Jupyter Notebook is one of the most popular tools among data scientists to perform data engineering and ML. The JupyterHub notebook server provides each user with a Jupyter notebook web-based IDE. Multiple users can use their Jupyter notebooks simultaneously to write and execute code for exploratory data analytics.
* **Amazon Athena** is an interactive query service for running queries on Amazon S3 object storage using standard ANSI SQL syntaxes. Amazon Athena is built on top of Presto and extends ad hoc query capabilities as a managed service. The Amazon Athena metadata store works like the Hive metadata store to use the same DDL statements from the Hive metadata store in Amazon Athena. Athena is a serverless and managed service, which means all infrastructure and software handling and maintenance is taken care of by AWS, and you can directly start running your query in the Athena web-based editor.
* **Amazon Elastic MapReduce** (**EMR**) is essentially Hadoop in the cloud. You can utilize the Hadoop framework with the power of the AWS cloud using EMR. EMR supports all the most popular open-source frameworks, including Apache Spark, Hive, Pig, Presto, Impala, HBase, and so on. EMR provides decoupled compute and storage, which means you don't always have to keep running a large Hadoop cluster; you can perform data transformation and load results into persistent Amazon S3 storage and shut down the server. EMR provides autoscaling and saves you from the administrative overhead of installing and updating servers with various software.
* **AWS Glue** is a managed ETL service, which helps in data processing, data cataloging, and ML transformations to find duplicate records. AWS Glue Data Catalog is compatible with the Hive data catalog and provides a centralized metadata repository across various data sources, including relational databases, NoSQL, and files. AWS Glue is built on top of a warm Spark cluster and provides ETL as a managed service. AWS Glue generates code in PySpark and Scala for common use cases so that you are not starting from scratch to write ETL code. Glue job authoring functionality handles any errors in the job and provides logs to understand underlying permission or data formatting issues. Glue provides workflows that help you build an automated data pipeline with simple drag-and-drop functionality.

Data analysis and processing are huge topics that warrant a book on their own. This section gave a very high-level overview of popular and common tools used for data processing. There are many more proprietary and open-source tools available. As a solution architect, you need to be aware of various tools available on the market to make the right choice for your organization's use case.

Business analysts need to create reports and dashboards and perform ad hoc queries and analyses to identify data insights. Let's learn about data visualization in the next section.

**Visualizing data**

Data insights are used to answer important business questions such as revenue by customer, profit by region, or advertising referrals by site, among many others. In the big data pipeline, enormous amounts of data are collected from a variety of sources. However, it is difficult for companies to find information about inventory per region, profitability, and increases in fraudulent account expenses. Some of the data you continuously collect for compliance purposes can also be leveraged for generating business.

The two significant challenges of BI tools are the cost of implementation and the time it takes to implement a solution. Let's look at some technology choices for data visualization.

**Technology choices for data visualization**

The following are some of the most popular data visualization platforms, which help you to prepare reports with data visualization as per your business requirements:

* **Amazon QuickSight** is a cloud-based BI tool for enterprise-grade data visualizations. It comes with a variety of visualization graph presets such as a line graph, pie charts, treemaps, heat maps, histograms, and so on. Amazon QuickSight has a data-caching engine known as **Super-fast, Parallel, In-memory Calculation Engine** (**SPICE**), which helps render visualizations quickly. You can also perform data preparation tasks such as renaming and removing fields, changing data types, and creating new calculated fields. QuickSight also provides ML-based visualization insights and other ML-based features such as auto forecast predictions.
* **Kibana** is an open-source data visualization tool used for stream data visualization and log exploration. Kibana offers close integration with Elasticsearch and uses it as a default option to search for data on top of the Elasticsearch service. Like other BI tools, Kibana also provides popular visualization charts such as histograms, pie charts, and heat maps and offers built-in geospatial support.
* **Tableau** is one of the most popular BI tools for data visualization. It uses a visual query engine, which is a purpose-built engine used to analyze big data faster than traditional queries. Tableau offers a drag-and-drop interface and the ability to blend data from multiple resources.
* **Spotfire** uses in-memory processing for faster response times, enabling extensive datasets from various resources. It provides the ability to plot your data on a geographical map and share it on Twitter. With Spotfire recommendations, it inspects your data automatically and makes suggestions about how to best visualize it.
* **Jaspersoft** enables self-service reporting and analysis. It also offers drag-and-drop designer capability.
* **Power BI** is a popular BI tool provided by Microsoft. It provides self-service analytics with a variety of visualization choices.

Data visualization is an essential and massive topic for solution architects. As a solution architect, you need to be aware of the available tools and make the right choice as per your business requirements for data visualization.

Now you have learned about various data pipeline components, from ingestion, storage, and processing to visualization. Let's put them together and learn how to orchestrate a big data architecture in the next section.

**Designing big data architectures**

Big data solutions are comprised of data ingestion, storage transformation, and visualization in a repeated manner to run daily business operations. You can build these workflows using the open source or cloud technologies you learned about in previous sections.

First, you need to learn which architecture style is right for you by working backward from the business use case. You need to understand the end-user of your big data architecture and create a user persona to understand the requirement better. To identify key personas you are targeting with big data architecture, you need to understand some of the following points:

* Which teams, units, or departments inside your organization are they a part of?
* What is their level of data analysis and data engineering proficiency?
* What tools do they typically use?
* Do you need to cater to employees, customers, or partners of the organization?

For your reference, taking an example of a retail store chain analysis, you may identify the following personas:

* **Product manager** persona who owns a product line/code but only sees turnover for their product.
* **Store manager** persona who wants to know the sales turnover and product mix for a single store (only able to see their store).
* **Admin** persona to have access to all data.
* **Data analyst** to access all data with PII data redacted.
* **Customer retention managers** want to understand repeated customer traffic.
* **Data scientists** need access to raw and processed data to build recommendations and forecast.

Once you understand your user persona, next identify business use cases that these personas are looking to solve, for example:

* How many customers are spending more over time? Less over time? Describe these customers.
* Of those customers who are spending more over time, which categories are growing at a faster rate?
* Of those customers who are spending less over time, in which categories are they becoming less engaged?
* Which demographic factors (e.g. household size, presence of children, income) appear to affect customer spending? Which demographic factors appear to affect engagement with certain categories?
* Is there evidence to suggest that direct marketing improves overall engagement?
* Does direct marketing for one category improve engagement in other categories?

While you get details on the use case, the essential aspect of building your data architecture is to understand access patterns and data retention, which can be analyzed by using the following queries:

* How often do key users and personas run their reports, queries, or models?
* What is their expectation for data freshness?
* What is their expectation of data granularity?
* What portion of data is most frequently accessed for analysis?
* How long do you intend to retain data for analysis?
* At what point can data age out of the data lake environment?

There is always some kind of sensitivity attached when you deal with data. Each country and area has its local regulatory compliance requirements, which solutions architects need to understand, such as:

* What compliance requirements does your business have?
* Are you subject to data locality, data privacy, or data redaction requirements?
* Who is authorized to see which records and which attributes in the dataset?
* How will you enforce the deletion of records on request?
* Where can you store data, for example, local to geolocation, county, or global?

As a data architect, you also need to consider the return on investment and how it will help overall business decisions. To understand, you may want to go through the following points:

* What primary business processes and decisions does your data lake support?
* What level of granularity is required for these decisions?
* What is the impact of data latency on business decisions?
* How do you plan to measure success?
* What is the expected return on the time and material invested?

Ultimately, you want to build a data architecture where you can provide flexibility to make technology choices. For example, use the best of cloud-based managed services and open-source technologies to capitalize on existing skills and investments. You want to build big data solutions to take advantage of parallelism to achieve high performance and scalability. It would be best if you make sure any components of your big data pipeline can scale in or scale out independently so that you can adjust it according to different business workloads.

To utilize the full potential of your solution, you want to provide interoperability with existing applications so that components of the big data architecture are also used for machine learning processing and enterprise BI solutions. It will enable you to create an integrated solution across data workloads. Let's learn about some big data architecture patterns.

**Data lake architecture**

A data lake is a centralized repository for both structured and unstructured data. The data lake is a combination of the different kinds of data found in the corporation. It has become the place where you can offload all enterprise data to a low-cost storage system such as Amazon S3. You have access to data using a generic API and open file formats, such as Apache Parquet and ORC. The lake stores data as is, using open-source file formats to enable direct analytics and machine learning uses.

The data lake is becoming a popular way to store and analyze large volumes of data in a centralized repository. Data can be stored as is in its current format, and you don't need to convert data into a predefined schema, which increases the data ingestion speed. As illustrated in the following diagram, the data lake is a single source of truth for all data in your organization:

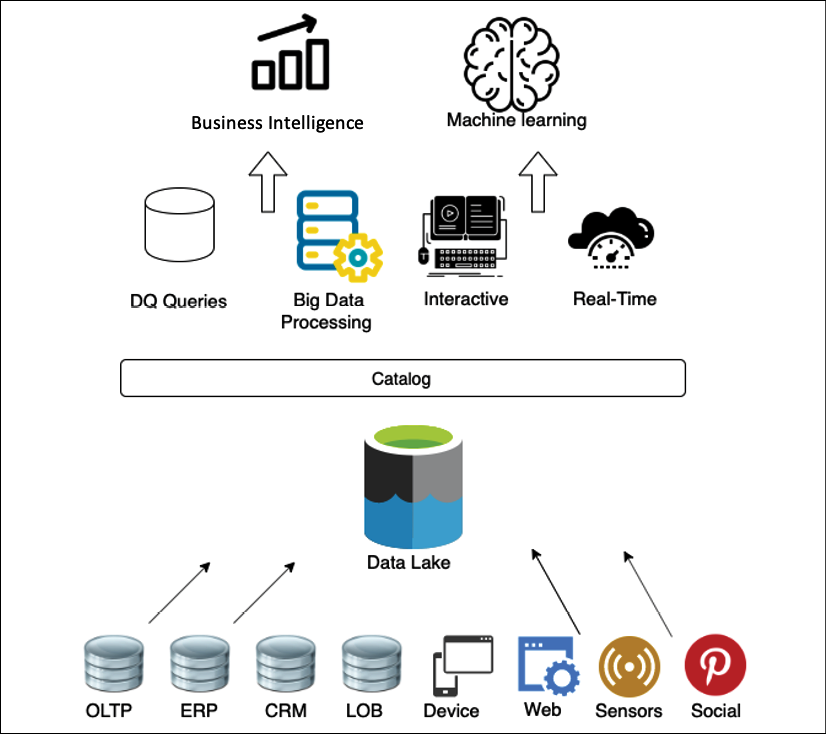


Figure 13.6: Object store for data lake

The following are the benefits of a data lake:

* **Data ingestion from various sources**: Data lakes let you store and analyze data from multiple sources such as relational and non-relational databases and streams in one centralized location for a single source of truth. This answers questions such as *why is the data distributed in many*places? And *where is the single source of truth?*
* **Collecting and efficiently storing data**: A data lake can ingest any kind of data structure, including semi-structured and unstructured data, without the need for any schema. This answers questions such as *how can I ingest data quickly from various sources and in multiple formats and store it efficiently at scale?*
* **Scale up with the volume of generated data**: Data lakes allow you to separate the storage and compute layers to scale each component separately. This answers questions such as *how can I scale up with the volume of data generated?*
* **Applying analytics to data from different sources**: With a data lake, you can determine the schema on reading and create a centralized data catalog on data collected from various resources. This enables you to perform quick ad hoc analysis. This answers questions such as *can I apply multiple analytics and processing frameworks to the same data?*

You need an unlimited scalable data storage solution for your data lake. Decoupling your processing and storage provides a significant number of benefits, including the ability to process and analyze the same data with a variety of tools. Although this may require an additional step to load your data into the right tool, Amazon S3 as your central data store provides even more benefits than traditional storage options. The following diagram provides a view of the data lake using AWS services:

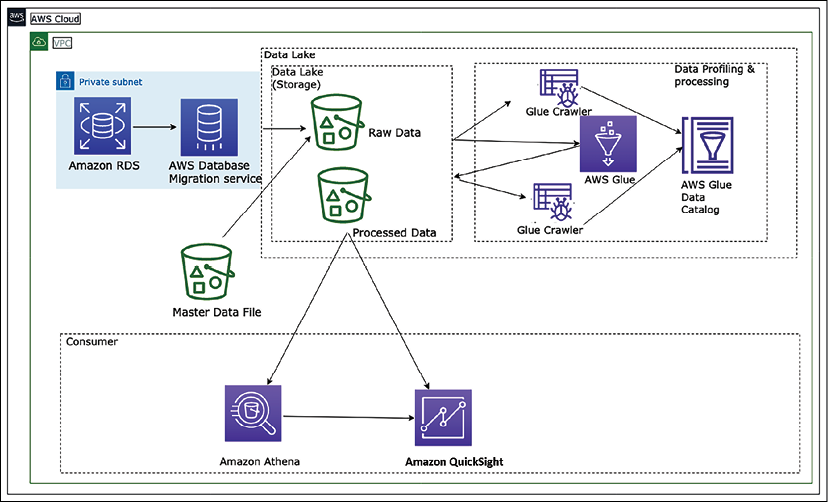


Figure 13.7: Data lake architecture in AWS platform

The preceding diagram depicts a data lake using Amazon S3 storage. Data is ingested to centralized storage from various resources such as relational databases and master data files. All of the data is stored in the raw layer of the data lake in its original format. This data is cataloged and transformed using the AWS Glue service. AWS Glue is a serverless data cataloging and ETL service based on the Spark framework in the AWS cloud platform. Transformed data is stored in the data lake's process layer, which can be consumed for different purposes. Data engineers can run ad hoc queries using Amazon Athena, a serverless query service built on top of managed Presto instances, and use SQL to query the data directly from Amazon S3. Business analysts can use Amazon QuickSight, Tableau, or Power BI to build visualizations for business users or load selective data in Amazon Redshift to create a data warehouse mart. Finally, data scientists can consume this data using Amazon SageMaker to perform machine learning.

The beauty of the data lake is that you are future-proofing your architecture. Twelve months from now, there may be new technology you want to use. With your data in the data lake, you can insert this new technology into your workflow with minimal overhead. By building modular systems in your big data processing pipeline, with shared object storage such as Amazon S3 as the backbone, you can replace specific modules when they become obsolete or when a better tool becomes available.

One tool cannot do everything. You need to use the right tool for the right job, and data lakes enable you to build a highly configurable big data architecture to meet your specific needs. Business problems are far too broad, deep, and complex for one tool to solve everything, especially big data and analytics.

However, with time organizations realized that data lakes have their limitations. As data lakes use cheap storage, organizations store as much of their data as they can in data lakes, providing the flexibility of open, direct access to files. Quickly data lakes started becoming **data swamps** due to issues of data quality and granular data security. However, to address the data lake's performance and quality issues, the organization processes a small subset of data in the data lake to a downstream data warehouse to use in BI applications for important decisions.

The dual system architecture between a data lake and data warehouse requires continuous data engineering to maintain and process data between these two systems. Each data processing step risks incurring failures that reduce data quality, while keeping the data lake and warehouse consistent is difficult and costly. Apart from paying for continuous data-processing costs, users pay double the storage cost for data copied to a warehouse. To address the dual-system problem, a new type of architecture is emerging called the data lakehouse. Let's learn more details about lakehouse architecture.

**Lakehouse architecture**

A new architecture paradigm has emerged called **lakehouse architecture** to address the limitations of data lakes and data warehouses. Lakehouse architecture aims to leverage the benefits of both, leveraging the scale of a data lake to ingest and store an ever-increasing amount of data in open formats that customers want to analyze, and enabling the user-friendliness of SQL queries and guarantees of a data warehouse. The main aspects of lakehouse architecture are:

* Data storage in open-data formats
* Decoupled storage and compute
* Transactional guarantees
* Support diverse consumption needs
* Secure and governed

Let's look at what the stages in a data lakehouse pattern are.

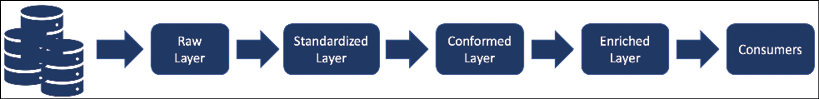


Figure 13.8: Lakehouse architecture layers

These layers can be described as follows:

* **Raw Layer**: This layer acts as the landing zone for all the source data in the format delivered by the source. The data here can be stored for a more extended period and archived for audit and reproducibility purposes.
* **Standardized Layer**: As the data that arrives in the raw layer can be in specific formats as delivered by the source, the standardized layer is used to store the data in a standard format (typically Parquet) after performing schema validations, schema evolution control, data quality rules, tokenization rules, and cleansing rules for the data. A typical example of a cleansing rule is standardizing the DateTime format to a standard format (e.g. ISO 8601).The data stored here is also optimized for analytical queries as it is partitioned and stored in columnar format. This data is typically also cataloged in a central data catalog for discovery.This layer acts as the consumption layer for standardized raw data in the organization.
* **Conformed Layer**: Typically, in any organization, some common entities/subject areas are well defined and commonly understood and used across the organization. Such entities can be treated as confirmed entities and end up in the conformed layer.

The definition of these common entities needs to be governed centrally as they are usually formed based on the master data of an organization. All these entities are also logged in the central data catalog with clear ownership and metadata for PII/PCI, retention, purpose, and so on. One of the benefits of managing the conformed entities centrally is clear enterprise ownership. As several parties use this data within the organization, if the ownership is distributed, the definitions can become ambiguous, and maintenance and retention of history, governance, and data management of these conformed entities can become a challenge.

* **Enriched Layer**: This is more of a logical layer, as it is aimed at data engineering teams, who would create their data products combining conformed entities and standardized raw data. Primarily, these business domain-focused teams would have many end products useful for a particular business domain; however, in some cases, these could also be products useful for other business domains. These could be called "golden datasets" with a proper business definition and can be offloaded to the data lake for sharing. All the end product datasets of this layer should also be added to the central data catalog with proper labels, metadata, and the purpose of the datasets.

The following diagram shows a sample lakehouse architecture using Redshift Spectrum for data sharing. Amazon Redshift Spectrum provides the ability to query data from the data lake without storing data in the data warehouse. Suppose you were already using Amazon Redshift for data warehousing. In that case, you don't need to load the entire data into the Amazon Redshift cluster. Still, you can simply use the spectrum to query data directly from the Amazon S3 data lake and combine it with data warehouse data.

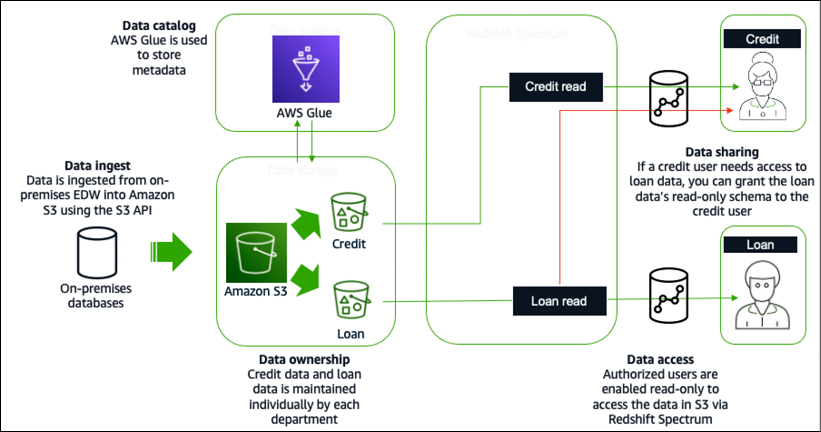


Figure 13.9: Lakehouse architecture in AWS cloud platform using Redshift Spectrum

Data is ingested from an on-prem **enterprise data warehouse** (**EDW**) into S3 using the S3 API in the preceding diagram. AWS Glue is used to store the metadata and the credit and loan data individually. Data analysts in the loan department would be granted read-only access to the loan data for data access. Similarly, credit analysts would be granted read-only access to the credit data. For data sharing, if a credit analyst needs access to the loan data, the credit analyst can be given the loan data's read-only schema.

There are benefits of lakehouse architecture; however, it doesn't solve the problem for large organizations having a complex application landscape driven by geographically separated business units. These business units have built data lakes and data warehouses as their analytical sources. Each business unit may merge multiple internal application data lakes to support their business. Centralized enterprise data lakes or data lakehouses are difficult to achieve as the pace of change is generally low and it's difficult to meet all requirements across different business units. To handle this problem, you need domain-oriented decentralized data ownership and architecture. That's where data mesh comes into the picture. Let's learn more about data mesh architecture.

**Data mesh architecture**

The major difference between data mesh and data lake architecture is that rather than trying to combine multiple domains into a centrally managed data lake, data is intentionally left distributed. Data mesh provides a pattern that allows a large organization to connect multiple data lakes/lakehouses within large enterprises, and to facilitate sharing with partners, academia, and even competitors. Data mesh marks a welcome architectural and organizational paradigm shift in how we manage large analytical datasets. The paradigm is founded on four principles:

1. Domain-oriented decentralization of ownership and architecture
2. Data served as a product
3. Federated data governance with centralized audit controls
4. Common access that makes data consumable

Data mesh is an organizational architecture that addresses these dimensions: domain-oriented decentralized data ownership and architecture, data as a product, self-serve data infrastructure as a platform, and federated computational governance. It encourages data-driven agility and supports domain-local governance through a lightweight centralized policy. Data mesh provides better ownership by isolating data resources with clear accountability. The core concept of data mesh is to feature data domains as nodes, which exist in data lake accounts.

A data producer contributes one or more data products to a central catalog in a data mesh account where federated data governance is applied to how data products are shared, delivering discoverable metadata and audibility. A data consumer searches for a catalog and gains access to a data product by accepting a resource share via the data mesh pattern. Below is a data mesh architecture in the AWS cloud:

Timeline

Description automatically generated

Figure 13.10: Data mesh architecture in AWS cloud platform

The following are the components implemented to build a data mesh as shown in the preceding diagram:

* Central AWS account where data products are registered, which is comprised of databases, tables, columns, and rows.
* Access control tags and tag access policies managed centrally.
* Stored data permissions that implement sharing with a consumer. Permissions can be direct or based on tags.
* Applies security and governance policies to producer and consumer accounts and the data products they publish.

With a data mesh architecture, you can aim to accelerate the independent delivery of the business domain lakehouses. Data mesh increases data security and compliance within domains and enables self-service data product creation, discovery, and subscription, allowing consumers to access data products transparently. There is a growing need to provide fast insight and act quickly based on customer needs, which makes streaming data analytics an essential aspect of any business. Let's learn more details about streaming data analytics architecture.

**Streaming data architecture**

Streaming data is one of the fastest-growing data segments. You need to ingest real-time data from various resources such as video, audio, application logs, website clickstreams, and IoT telemetry data and quickly process to provide fast business insights. Streaming data use cases follow a similar pattern: data flows from data producers through streaming storage and data consumers to storage destinations. Sources continuously generate data delivered via the ingest stage to the stream storage layer, where it's durably captured and made available for streaming processing. The stream processing layer processes the data in the storage layer and sends the processed information to a specified destination.

Streaming data architecture is different as it needs to process a continuous stream of massive data with very high velocity. Often this data is semi-structured and needs a good amount of processing to get actionable insights. While designing streaming data architecture, you need to easily scale data storage while getting real-time pattern identification from time-series data.

You need to think about the producer who generated a stream of data, such as IoT sensors, how to store the data and process it using a real-time data processing tool, and finally, how to query the data in real time. The following diagram shows a streaming data analytics pipeline using a managed service in the AWS platform:

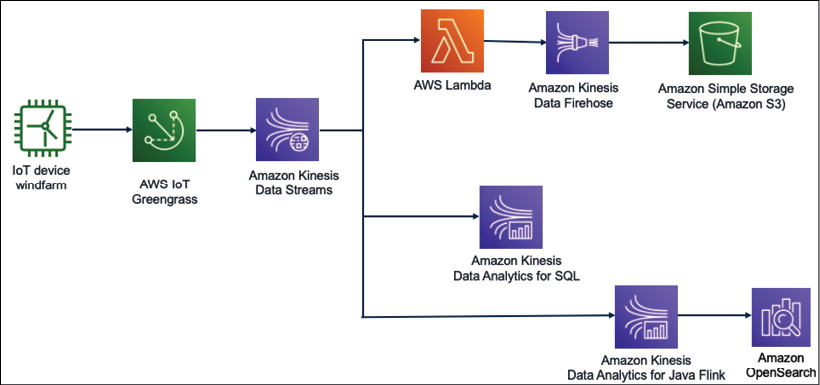


Figure 13.11: Streaming data analytics for IoT data

In the preceding diagram, data is ingested from the windfarm to understand wind turbine health and speed. It's important to control wind turbines in real time to avoid costly repairs in the case of high wind speeds beyond the limit that the wind turbine can handle.

The wind turbine data is ingested to Kinesis Data Streams using AWS IoT. Kinesis Data Streams can retain the streaming data for up to a year and provide replay capability. These are subjected to the fan-out technique to deliver the data to multiple resources, where you can massage data using Lambda and store it to Amazon S3 for further analytics using Amazon Kinesis Firehose.

You can perform real-time queries on streaming data using simple SQL queries with Kinesis Data Analytics for SQL. You can automate a data pipeline to transform streaming data in real time using Kinesis Data Analytics for Java Flink and store the processed data in Amazon OpenSearch to get data insights. You can add Kibana on top of OpenSearch to visualize the wind turbine data in real time.

The challenge with these use cases is the set-up time and effort that developers require to create the resources and establish the best practices needed by the streaming data services (such as access control, logging capabilities, and data integrations). The above solution is data agnostic and easily customizable, enabling customers to quickly modify pre-configured defaults and start writing code to include their specific business logic.

**Big data architecture best practices**

In previous sections, you learned about various big data technology and architecture patterns. Let's look at the following reference architecture diagram with different layers of a data lake architecture to learn best practices.

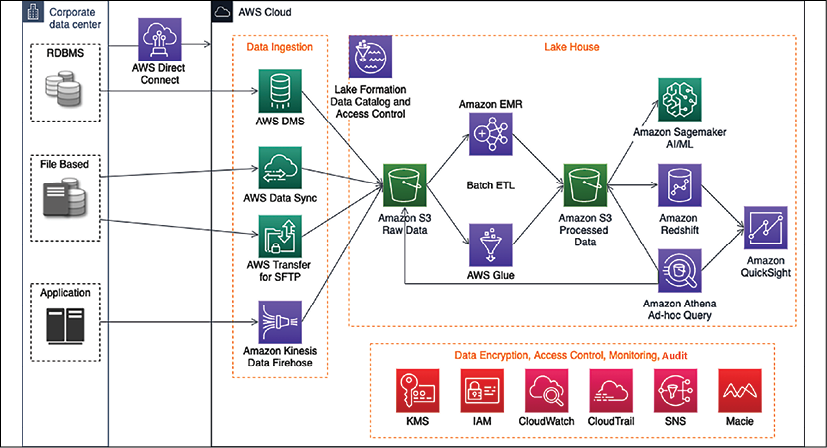


Figure 13.12: Data lake reference architecture

The preceding diagram depicts an end-to-end data pipeline in a data lake architecture using the AWS cloud platform with the following components:

* AWS Direct Connect to set up a high-speed network connection between the on-premises data center and AWS to migrate data. If you have large volumes of archive data, it's better to use the AWS Snow family to move it offline.
* A data ingestion layer with various components to ingest streaming data using Amazon Kinesis, relational data using AWS **Data Migration Service** (**DMS**), secure file transfer using AWS Transfer for SFTP, and AWS DataSync to update data files between cloud and on-prem systems.
* A centralized data storage for all data using Amazon S3, where data storage has multiple layers to store raw data, processed data, and archive data.
* A cloud native data warehouse solution, Amazon Redshift, with Redshift Spectrum to support lakehouse architecture.
* An ad hoc query functionality using Amazon Athena.
* A quick ETL pipeline based on Spark using AWS Glue.
* Amazon EMR to re-utilize existing Hadoop scripts and other Apache Hadoop frameworks.
* Amazon Lake Formation to build comprehensive data cataloging and granular access control at the data lake level.
* The AI/ML extension with Amazon SageMaker.

Other components include Amazon **KMS** (**Key Management Service**) for data encryption, Amazon **IAM** (**Identity and Access Management**) for access control, Amazon Macie for PII data detection to adhere to data compliance such as PCI-DSS, CloudWatch to monitor the operation, and CloudTrail to audit the data lake activities.

You need to validate your big data architecture using the following criteria:

* Security
  + Classify data and define corresponding data protection policies using resource-based access control.
  + Implement a strong identity foundation using user permission and **Single Sign-On** (**SSO**).
  + Enable environment and data traceability for audit purposes.
  + Apply security at all layers and protect data in transit and at rest using SSL and encryption at rest at all layers.
  + Keep people away from data such as locking down write access to production datasets.
* Reliability
  + Enforce data hygiene using automated data profiling using data cataloging.
  + Manage the lifecycle of data assets, transitioning, and expiration using data tiering between the data warehouse and data lake.
  + Preserve data lineage by maintaining the history of data movement through the data catalog.
  + Design resiliency for analytics pipelines and monitor system SLAs with automated recovery of ETL job failures.
* Performance efficiency
  + Use data profiling to improve performance with data validation and to build a sanitization layer.
  + Continuously optimize data storage, such as using data compression with Parquet format, data partition, file size optimization, and so on.
* Cost optimization
  + Adopt a consumption model and determine if you need an ad hoc or fast query pattern.
  + Delete out-of-use data. Define data retention rules and delete or archive data that is out of the retention period.
  + Decouple compute and storage with a data lake-based solution.
  + Implement migration efficiency using different migration strategies for various data sources and volumes.
  + Use managed and application-level services to reduce the cost of ownership.
* Operational excellence
  + Perform operations as code using tools such as CloudFormation, Terraform, and Ansible.
  + Automate operations such as building an orchestration layer with Step Functions or Apache Airflow.
  + Anticipate failure in advance by continuously monitoring and automating the recovery of ETL job failures.
  + Measure the health of your workload.

You can use the above checklist as a guide to validate your big data architecture. Data engineering is a very vast topic that warrants multiple books to cover each topic in depth.

In this chapter, you learned about various components of data engineering with a popular architecture pattern, which will help you get started and explore the topic in more depth.

**Summary**

In this chapter, you learned about the big data architecture and components for a big data pipeline design. You learned about data ingestion and various technology choices available to collect batch and stream data for processing. As the cloud is central to storing the vast amounts of data being produced today, you learned about the various services available to ingest data in the AWS cloud ecosystem.

Data storage is one of the central points when it comes to handling big data. You learned about various kinds of data stores, including structured and unstructured data, NoSQL, and data warehousing, with the appropriate technology choices associated with each. You learned about data lake architecture and its benefits.

Once you collect and store data, you need to transform it to get insights into that data and visualize your business requirements. You learned about data processing architecture and technology choices to choose open source and cloud-based data processing tools as per your data requirements. These tools help you to get data insights and visualizations as per the nature of your data and organizational needs.

You learned about various big data architecture patterns, including data lake, lakehouse, data mesh, and streaming data architecture, along with reference architecture. Finally, you learned big data architecture best practices by putting all your learning together in the reference architecture.

As you collect more data, it's always beneficial to get future insights, which can be exceptionally beneficial for business. To predict future outcomes based on historical data, you often need machine learning. Let's learn more about machine learning and how to make your data architecture future-proof in the next chapter.