Logo

Description automatically generated

**Data Architecture Playbook**

**Document Purpose**

The purpose of this document is to describe Modern data architecture, components, and principles of data architecture. It further explains various data layers of data architecture, recommendations, best practises to be followed in each layer and frequently asked questions raised during the design and development phase.

It also contains the regulations which must be followed in various technical platforms like RDBMS, No SQL, Rest API, Distributed Data system etc.

Along with it also provides solutions and clear architecture for the problem/challenges faced during project phase.

This document helps data engineers understanding how the Data architects’ approach to find a solution for the problems they face while designing the architecture.

**Table of contents**

**Contents**

[1. Data Architecture 6](#_Toc111618062)

[1.1 Introduction to Data Architecture 6](#_Toc111618063)

[1.2 Modern Solution Data Architecture 6](#_Toc111618064)

[1.3 Components of Data Architecture 6](#_Toc111618065)

[1.4 Characteristics of Data Architecture 7](#_Toc111618066)

[1.5 Principles of Data Architecture 8](#_Toc111618067)

[1.6 Roles of Data Architecture 10](#_Toc111618068)

[2. Modern Data Architecture 12](#_Toc111618069)

[2.1. Elements of Modern Data Architecture 12](#_Toc111618070)

[2.3 Assumptions handled in Real time project – Coty 22](#_Toc111618071)

[3 API Documentation 27](#_Toc111618072)

[3.1 API- PacificEast 30](#_Toc111618073)

[3.2 Infrastructure – PacificEast 30](#_Toc111618074)

[3.3 Implementation 31](#_Toc111618075)

[3.4 API Setup 31](#_Toc111618076)

[4 Design 31](#_Toc111618077)

[4.1 Current Scope - PacificEast 32](#_Toc111618078)

[4.2 Current Scope - Melissa 33](#_Toc111618079)

[4.2.1 Implementation 33](#_Toc111618080)

[5 Data pipelines 38](#_Toc111618081)

[5.1 Differences between Data Pipeline and ETL 38](#_Toc111618082)

[5.2 Benefits of Data Pipelines 38](#_Toc111618083)

[5.3 Guidelines for Data Pipeline Design 39](#_Toc111618084)

[6 Storage Resources 42](#_Toc111618085)

[6.1 Defining database, warehouse, lake, lakehouse and Mesh 42](#_Toc111618086)

[6.2 Comparing data storages 44](#_Toc111618087)

[6.3 Caution on data lakes 46](#_Toc111618088)

[7 Distributed Systems 48](#_Toc111618089)

[7.1 Problem with the Centralized Systems 48](#_Toc111618090)

[7.2 How to solve this problem 48](#_Toc111618091)

[7.3 Few of the advantages of Distributed Systems 49](#_Toc111618092)

[7.4 Are Distributed systems false proof? Can we use it for all types of big data? 49](#_Toc111618093)

[7.4.1 When consistency is the priority? 50](#_Toc111618094)

[7.4.2 When Availability is the priority? 50](#_Toc111618095)

[7.5 Demerits of Distributed Systems 50](#_Toc111618096)

[7.6 Conclusion 51](#_Toc111618097)

[8 High Availability 52](#_Toc111618098)

[8.1 Is it worth the money? 52](#_Toc111618099)

[8.2 How Is High Availability Generally Achieved 52](#_Toc111618100)

[8.3 High Availability vs Redundancy 53](#_Toc111618101)

[8.4 High Availability vs Disaster recovery 53](#_Toc111618102)

[8.5 Best practices 53](#_Toc111618103)

[9 Data Security 55](#_Toc111618104)

[9.1 Importance 55](#_Toc111618105)

[9.2 Business Challenges 55](#_Toc111618106)

[9.3 Types of Data Security 56](#_Toc111618107)

[9.4 Data security capabilities and solutions 56](#_Toc111618108)

[9.5 Data security strategies 57](#_Toc111618109)

[10 Fault Tolerance 61](#_Toc111618110)

[10.1 High Availability vs Fault Tolerance 61](#_Toc111618111)

[10.2 Fault Tolerance Requirements 62](#_Toc111618112)

[10.3 Fault Tolerance Architecture 63](#_Toc111618113)

[10.4 Relationship Between Security and Fault Tolerance 63](#_Toc111618114)

[10.5 Others 63](#_Toc111618115)

[11 Scalability 65](#_Toc111618116)

[11.1 Vertical Scaling 65](#_Toc111618117)

[11.2 Horizontal Scaling 66](#_Toc111618118)

[11.3 Challenges faced by developers due to Scalability 66](#_Toc111618119)

[12 Data Lineage 68](#_Toc111618120)

[12.1 Questions that Data Lineage Answer 68](#_Toc111618121)

[12.2 Importance of Data Lineage 68](#_Toc111618122)

[12.3 Implementation 69](#_Toc111618123)

[12.4 Best Practices 70](#_Toc111618124)

[13 Relational Database Management System 72](#_Toc111618125)

[13.1 Relational Database Architecture 72](#_Toc111618126)

[13.2 Database Design Elements 73](#_Toc111618127)

[13.3 Principles of Relational Databases 74](#_Toc111618128)

[13.4 Points to be taken care of, while designing a Database 74](#_Toc111618129)

[13.5 Database Schema Designs and How to Use Them 75](#_Toc111618130)

[13.6 Top performance tuning tips for relational databases 78](#_Toc111618131)

[13.7 Data abstraction levels 78](#_Toc111618132)

[13.8 Database Cache 79](#_Toc111618133)

[13.8.1 Cache tables 79](#_Toc111618134)

[13.9 Materialized view 79](#_Toc111618135)

[14 NoSQL Databases 81](#_Toc111618136)

[14.1 Introduction to NoSQL 81](#_Toc111618137)

[14.2 Why NoSQL databases? 82](#_Toc111618138)

[14.3 Why not SQL databases? 82](#_Toc111618139)

[14.4 Relational vs. NoSQL: The fundamental differences. 83](#_Toc111618140)

[14.5 Compare Various features of RDBMS and NoSQL databases. 84](#_Toc111618141)

[14.6 The importance of cap theorem in the context of NoSQL. 84](#_Toc111618142)

[14.7 How to choose between SQL and NoSQL databases. 85](#_Toc111618143)

[14.8 What are the different NoSQL Databases? 87](#_Toc111618144)

[14.9 When to use the which type of NoSQL databases? 88](#_Toc111618145)

[15 Non-Functional Scenarios 91](#_Toc111618146)

[15.1 ETL and ELT Processes 91](#_Toc111618147)

[15.1.1 ETL vs ELT: Side-by-Side Comparison 92](#_Toc111618148)

[15.2 Data repository (Data Warehouse Vs Data Lake Vs Lake house) 93](#_Toc111618149)

[ Key Technology Enabling the Data Lakehouse 93](#_Toc111618150)

[14.1 Data repository (Data Warehouse Vs Data Lake Vs Lake house) 94](#_Toc111618151)

[14.1.1 How to Find the right data warehouse for your business 94](#_Toc111618152)

[14.1.2 Use cases of data lake house 95](#_Toc111618153)

# Data Architecture

## 1.1 Introduction to Data Architecture

Data architecture is the process of standardizing how organizations collect, store, transform, distribute, and use data.

## 1.2 Modern Solution Data Architecture

Increasingly, every part of the business is turning to data and data analytics to make the essential decisions that drive the business forward and help it to be successful and competitive. Without common standards for data integration and data management, organizations will struggle to derive meaningful results from their data.

## 1.3 Components of Data Architecture

Chart, bubble chart

Description automatically generated

**1.3.1 Data pipelines**

A data pipeline is the process in which data is collected, moved, and refined. It includes data collection, refinement, storage, analysis, and delivery.

**1.3.2 Cloud storage**

Not all data architectures leverage cloud storage, but many modern data architectures use public, private, or hybrid clouds to provide agility.

**1.3.3 Cloud computing**

In addition to using cloud for storage, many modern data architectures make use of cloud computing to analyse and manage data.

**1.3.4 AI and ML models**

AI and ML are used to automate systems for tasks such as data collection, labelling, etc. At the same time, modern data architectures can help organizations unlock the ability to leverage AI and ML at scale.

**1.3.5 Data streaming**

Data streaming is flowing data continuously from a source to a destination for processing and analysis in real-time or near real-time.

**1.3.6 Container orchestration**

A container orchestration system such as open-source Kubernetes is often used to automate software deployment, scaling, and management.

**1.3.7 Real-time analytics**

The goal of many modern data architectures is to deliver real-time analytics, the ability to perform analytics on new data as it arrives in the environment.

## 1.4 Characteristics of Data Architecture

Modern data architectures must be designed to take advantage of emerging technologies such as artificial intelligence (AI), automation, internet of things (IoT), and blockchain.

**1.4.1 Cloud-native**

Data architectures should be designed to support elastic scaling, high availability, end-to-end security for data in motion and data at rest, and cost and performance scalability.

**1.4.2 Scalable data pipelines**

To take advantage of emerging technologies, data architectures should support real-time data streaming and micro-batch data bursts.

**1.4.3 Seamless data integration**

Data architectures should integrate with legacy applications using standard API interfaces. They should also be optimized for sharing data across systems, geographies, and organizations.

**1.4.4 Real-time data enablement**

Modern data architectures should support the ability to deploy automated and active data validation, classification, management, and governance.

**1.4.5 Decoupled and extensible**

Modern data architectures should be designed to be loosely coupled, enabling services to perform minimal tasks independent of other services.

## 1.5 Principles of Data Architecture

**1.5.1 View data as a shared asset**

Enterprises that start with a vision of data as a shared asset ultimately outperform their competition, as CIO explains. Instead of allowing departmental data silos to persist, these enterprises ensure that all stakeholders have a complete view of the company. And by “complete,” I mean a 360-degree view of customer insights along with the ability to correlate valuable data signals from all business functions, including manufacturing and logistics. The result is improved corporate efficiency.

**1.5.2 Provide the right Interfaces for users to consume the data**

Putting data in one place isn’t enough to achieve the vision of a data-driven organization. In order for people (and systems) to benefit from a shared data asset, you need to provide the interfaces that make it easy for users to consume that data. This might be in the form of an OLAP interface for business intelligence, an SQL interface for data analysts, a real-time API for targeting systems, or the R language for data scientists. In the end, it’s about letting your people work in the tools they know and are right for the job they need to perform.

**1.5.3 Ensure security and access controls**

The emergence of unified data platforms like Snowflake, Google BigQuery, Amazon Redshift, and Hadoop has necessitated the enforcement of data policies and access controls directly on the raw data, instead of in a web of downstream data stores and applications. The emergence of data security projects like Apache Sentry makes this approach to unified data security a reality. Look to technologies that allow you to architect for security, and deliver broad self-service access, without compromising control.

**1.5.4 Establish a common vocabulary**

By investing in an enterprise data hub, enterprises can now create a shared data asset for multiple consumers across the business. However, it’s critical to ensure that users of this data analyse and understand it using a common vocabulary. Product catalogs, fiscal calendar dimensions, provider hierarchies and KPI definitions all need to be common, regardless of how users consume or analyse the data. Without this shared vocabulary, you’ll spend more time disputing or reconciling results than driving improved performance.

**1.5.5 Curate the data**

Time and time again, I’ve seen enterprises that have invested in Hadoop or a cloud-based data lake like Amazon S3 or Google Cloud Platform start to suffer when they allow self-serve data access to the raw data stored in these clusters. Without proper data curation (which includes modelling important relationships, cleansing raw data and curating key dimensions and measures), ­end users can have a frustrating experience which will vastly reduce the perceived and realized value of the underlying data. By investing in core functions that perform data curation, you have a better chance of realizing the value of the shared data asset.

* + 1. **Eliminate Data Copies and Movement**

Every time data is moved there is an impact, cost, accuracy and time. Talk to any IT group, or business user for that matter, and they all agree; the fewer times data has to be moved, the better. Part of the promise of cloud data platforms and distributed file systems like Hadoop is a multi-structure, multi-workload environment for parallel processing of massive data sets. These data platforms scale linearly as workloads and data volumes grow. By eliminating the need for additional data movement, modern enterprise data architectures can reduce cost (time, effort, accuracy), increase “data freshness” and optimize overall enterprise data agility.

**1.5.7 Data Privacy and Compliance**

Data privacy deals with handling personal data in compliance with data protection laws, regulations, and general privacy best practices.

Ensuring data privacy is part of the larger topic of [data governance](https://www.talend.com/resources/what-is-data-governance/). Data governance requires organizations to know what data they have, where it’s stored, how it flows through their IT systems, and how it’s used.

**Personal data protection**

Among the most sensitive data is information about people, personal data about any identified or identifiable individual. Personally identifiable information (PII) can be almost anything

**GDPR and other data regulations**

Data privacy laws specify how data should be collected, stored, and shared with third parties. The most widely discussed data privacy laws include:

GDPR: The European Union’s General Data Protection Regulation ([GDPR](https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act)) is the most comprehensive data privacy law in effect. GDPR gives individuals the right to determine what data organizations store, request that organizations delete their data, and receive notifications of data breaches.

CCPA: The California Consumer Privacy Act ([CCPA](https://www.caprivacy.org/)) is a state-level regulation in the United States. It enables California residents to ask organizations what personal data exists about them, delete it on request, and find out what data has been given to third parties.

## 1.6 Roles of Data Architecture

**1.6.1 Data Architect**

The lead role in data architecture initiatives typically goes to data architects. They need a variety of technical skills, as well as the ability to interact and communicate with business users

**1.6.1.1 Roles and Responsibilities**

* Translating business requirements into technical specifications, including data streams, integrations, transformations, databases, and data warehouses
* Defining the data architecture framework, standards, and principles, including modelling, metadata, security, reference data such as product codes and client categories, and master data such as clients, vendors, materials, and employees
* Defining data flows, i.e., which parts of the organization generate data, which require data to function, how data flows are managed, and how data changes in transition
* Collaborating and coordinating with multiple departments, stakeholders, partners, and external vendors

**1.6.2 Data modelers**

They also work with business users to assess data needs and review business processes. Then, they use the information they've gathered to create data models.

**1.6.2.1 Roles and Responsibilities**

* Data modelers are responsible for designing and implementing data modelling solutions using relational, dimensional, and NoSQL databases.
* Data modelers analyze and translate business needs into long-term solution data models by evaluating existing systems and working with a development team to create conceptual data models and data flows that address their needs.
* Maintain logical and physical data models along with accurate metadata.
* Present and communicate modelling results and recommendations to internal stakeholders.
* Perform data profiling/analysis activities that helps to establish, modify and maintain data model.

**1.6.3 Data integration developers**

Once the architecture is implemented, they're tasked with creating ETL and ELT jobs to integrate data sets.

**1.6.3.1 Roles and Responsibilities**

* Data Integration Developer is responsible for the complete life cycle of information systems: from requirements and analysis to design and development. They also conduct testing, implementation, and subsequent maintenance and evolution, always with the aim of achieving greater efficiency and productivity.
* Conception, design, and development of interfaces and integration architectures
* Managing data analysis and integration of disparate systems
* Documentation of technical requirements in the interface context
* Ensuring high quality and optimum performance of data integration systems
* Optimizing data integration platform for increasing data volumes

**1.6.4 Data engineers**

They build pipelines to funnel data to data scientists and other analysts. They also help data science teams with the data preparation process.

**1.6.4.1 Roles and Responsibilities**

* Data engineers helps to build systems that collect, manage, and convert raw data into usable information for data scientists and business analysts to interpret.
* Acquire datasets that align with business needs
* Develop algorithms to transform data into useful, actionable information
* Build, test, and maintain database pipeline architectures
* Collaborate with management to understand company objectives
* Create new data validation methods and data analysis tools
* Ensure compliance with data governance and security policies

Include cost, duration and effort estimate to get to next estimation level approval gate.

# Modern Data Architecture

The data generated by several application silos are combined and greatly enhanced to provide a better customer experience. Deriving value from the data includes building a unified data architecture and a collaborative effort of data engineering and data science teams. Data engineering involves building and maintaining the data infrastructure and data pipelines, and Data science involves transforming crude data into something useful and deriving insights through analytical and ML workloads.

Diagram

Description automatically generated

## Elements of Modern Data Architecture

* + 1. **Data Sources**

Data sources generate data in a variety of ways in structured, unstructured, or semi-structured format. Data sources can be transactional applications and operational systems that generate relational data, or they can be social media mobile apps, IoT devices, clickstreams or log files that generate non-relational data.

Data generated from relational databases typically have static schemas whereas distributed non-relational data stores have dynamic schemas. Data generated by dissimilar systems arrive in different formats such as json, csv, parquet, Avro etc.

* + 1. **Data Ingestion**

Provide connectors to extract data from a variety of data sources and load it into the lake. A well-architected ingestion layer should:

* Support multiple data sources: Databases, Emails, Webservers, social media, IoT, and FTP.
* Support multiple ingestion modes: Batch, Real-Time, One-time load
* Support any data: Structured, Semi-Structured, and Unstructured.
* Flexible enough to support new data sources.
* Provide data curation facilities.
  + - 1. **Layers involved in Data Ingestion**
         1. **Landing layer**
* Also called the Ingestion Layer/Landing Area because it is literally the sink of our Data Lake. The main objective is to ingest data into Raw as quickly and as efficiently as possible. To do so, data should remain in its native format. We don’t allow any transformations at this stage.
* No overriding is allowed, which means handling duplicates and different versions of the same data.
* data should be landed to the data lake environment as Raw original format.
  + - 1. **Methods of Integrations**
* Use JDBC connection and build custom framework to pull data from many databases like RDBMS, NO SQL. For more details, please refer Section 13 and 14.
* Build custom code to connect with SFTP if your source data present in any of the client servers.
* Use streaming based ingestion tool to pull data from source generators.
* Build APIs to establish secure connection and pull data from various data sources. For more details, please refer Section 3.
  + - 1. **Guidelines/Best Practises**
* Data validation should be done before landing to the Raw layer.
* From our experience we advise customers to start with generic division: subject area/data source/object/year/month/day of ingestion/raw data.
* It is important to mention that end users shouldn’t be granted access to this layer.
* The data here is not ready to be used, it requires a lot of knowledge in terms of appropriate and relevant consumption. Raw is quite like the well-known DWH staging.
  + - 1. **Questions to be raised**
* What type of job required for data ingestion whether it is batch Job or Real time job.?
* What is the volume of data to be fetched daily.?
* What frequency of data to be retrieved from source.?
* What are sources/Applications, data should be fetched.?
* What are file formats to be considered for data ingestion.?
* What type of data to be fetched whether structured, semi-structured or un-structured.?
* what is a data source whether it will be RDBMS database, data warehouse, NO SQL, SFTP server, Any API, Any IoT device. Etc.
  + - 1. **Pitfalls of Data Ingestion**
* Source data should be archived after process the data. If we missed it, then there is no track on the source data received.
  + 1. **Data Storage**
* Data storage is one of the key components of a Data Lake architecture.  
  A well-architected storage layer should:
* Be highly scalable and available.
* Provide low-cost storage.
* Be able to store raw, in-process and curated data.
* Support any data format and allow compression and encryption techniques.
* Provide fast access for data exploration workloads.
  + - 1. **Layers involved in Data Storage**
         1. **Raw layer**
* may be considered as optional in most implementations. If we anticipate that our Data Lake Architecture will grow fast, this is the right direction.
* The main objective of this layer is to improve performance in data transfer from Raw to Curated.
  + - 1. **Guidelines/Best Practises**
* While in Raw, data is stored in its native format, when it is required to Standardize the format that fits best for cleansing. The structure is the same as in the previous layer, but it may be partitioned to lower grain if needed.
  + - 1. **Questions to be raised in Data Storage**
* Is File level validation is required for source data.?
* Is Archival required after file validation.?
* What type of cleansing would be required.?
* What are data quality checks (DQ) to be performed for each dataset which will be configured in config file.?
* Is Schema level validation is required for source data.?
* Any table to be created on top of the stagging data.?
* Will data be partitioned or refreshing daily data under data path in data lake.?
* Do you want to perform any Schema enforcement.?
  + - 1. **Pitfalls of Data Storage**
* File format should be chosen properly else it will impact the performance and storage.
  + 1. **Data Processing**
* Data processing involves various methods such as cleansing, profiling, validating, enriching, and aggregating datasets. It involves data modelling and mapping source-destination schemas. The data architecture should support both schema enforcement to avoid inadvertent changes (schema-on-write) and at the same time offer flexibility to modify schemas (schema-on-read) as the requirements evolve.
* For slow-moving datasets, batch processing techniques are employed to churn large datasets, perform complex transformations, and generate deep insights. Previously, batch processing used to be long-running jobs, but lower latencies are possible using distributed massive parallel processing engines such as Spark.
* For fast-moving datasets, real-time streaming techniques such as aggregating and filtering on rolling time-windows are employed to generate immediate insights using Spark streaming or Flink. Languages such as python, java, Scala and SQL are predominantly used for data processing.
  + - 1. **Layers involved in Data Processing**
         1. **Curated layer**
* also called Curated Layer/Conformed Layer. Data is transformed into consumable data sets, and it may be stored in files or tables.
* The purpose of the data, as well as its structure at this stage is already known. You should expect cleansing and transformations before this layer. Also, denormalization and consolidation of different objects is common.
  + - 1. **Guidelines/Best Practises**
* Data Quality checks should be done through framework before insert data into Curated layer.
* Transformed data should be stored in this layer with optimized file format and stored as Partitioned tables.
* End users are granted access only to this layer.
  + - 1. **Questions to be raised in Data processing**
* Is there any transformation should be applied on top of the refined data.? If yes, what are transformations required.?
* Is there any SCD (Slowly changing dimension) process to be followed on stagging data.?
* Do you want to perform any ACID transactions, Upsert/Delete operation on transformed data.?
* Do you want to enable Time travel on your data.?
  + - 1. **Pitfalls in Data Processing**
* Partitions would be decided based on usage, types of queries used and volume of the table. Should not be applied to all table by default.
  + 1. **Data Consumption**

The first step for data exploration is dataset discovery. Identification of the right dataset to work with is essential before one start exploring it. For the data lake to be successful, the data exploration facility needs to provide the following key features:

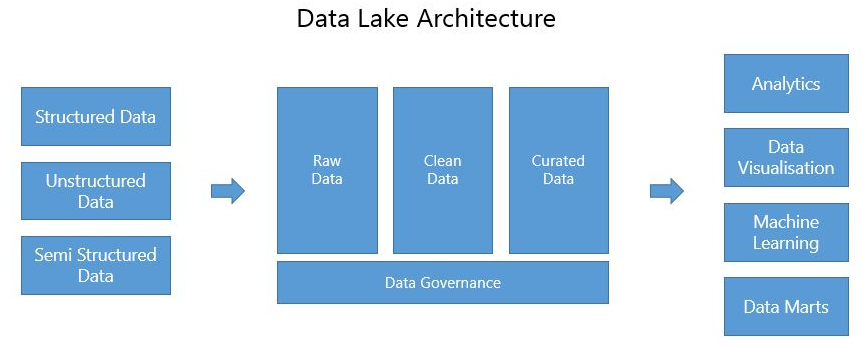
* Flexible Access
  + Support different mechanisms and/or tools to access data with friendly GUI/Dashboards. Support different type of workloads.
* Self-service
  + The real consumers of data should have the ability to explore the data with minimal dependency on the IT.
* Teamwork
  + Data lakes work best in a collaborative environment where analysis and findings of one group of users can be shared with other users (or groups), avoiding the need for duplicate effort and improve the overall business outcome.
    - 1. **Layers involved in Data Consumption**
         1. **Consumption Layer**
* also called the Trusted Layer/Secure Layer/Production Layer, sourced from Cleansed and enforced with any needed business logic. These might be surrogate keys shared among the application, row level security or anything else that is specific to the application consuming this layer.
* The structure of the data will remain the same, as in Cleansed.
  + - 1. **Guidelines/Best Practises**
* If any of your applications use machine learning models that are calculated on your Data Lake, you will also get them from here.
* if any additional security level access to be provided for your data, extend your data to this layer, and apply security policies.
  + - 1. **Questions to be raised**
* Is there any enrichment required on top of curated data.?   
  If yes, what are enrichments required.
* Who are all will access which data marts/entities from provision layer.?
* Do you have any data science scenarios available as down streams.?
* Which tool are going to use for implementing machine learning purpose.?
* Which Visualization tool will be going to connect consumption data.?
* Is JDBC connection available to connect.?
  + - 1. **Pitfalls in Data Consumption**
* New table should be maintained in the same database after applied business logic. Source table should not be affected.
* Business logics should not be hard coded in the framework. Should be derived from config files/database.
* Access should be provided properly based on the need. Do not provide whole data access to anyone.
  + 1. **Data Governance**

It contains Infrastructure and Operations Management Provisioning, Managing, Monitoring, and scheduling your Hadoop clusters.

* **Data Security**
  + - Security should be implemented at all layers of the lake starting from Ingestion, Storage, Discovery, and Consumption through analytics. The basic requirement is to restrict the access to data to trusted users and services only. The following are the key features available for data lake security.
      * Authentication
      * Authorization
      * Accounting
      * Data Protection
* **Data Quality**
  + - Data quality is a necessary condition for consumers to get business value out of the lake. A successful data lake implementation must support for Data Discovery, Data Profiling, Data Quality Rules, Data Quality Monitoring, Data Quality Reporting and Data Remediation  
      Metadata Management & Lineage
    - Provide data Auditing, Lineage, Metadata Management, Data Lifecycle Management and Policy enforcement and Data Discovery/Search mechanism in a data lake.
* **Data Lineage**
  + - Data lineage deals with data’s origins, what happens to it and where it moves over time. It simplifies tracing errors back to its source in a data analytics process from its journey from origin to destination visualized through appropriate tools.
* **Data Catalog**
  + - A [Data Catalog](https://www.alation.com/product/data-catalog/) is a collection of metadata, combined with data management and search tools, that helps analysts and other data users to find the data that they need
* **Master Data Management**
  + - An essential part of serving ready-to-use data. You need to either find a way to store your MD on the Data Lake or reference it while executing ELT processes.
    1. **Data Auditing**
* It is the process of recording access and transformation of data from risk and compliance perspective.
* It involves tracking changes to key dataset elements and capture “who / when / how” information about changes to these elements
  + 1. **Data Discovery/Search**
* After the ingestion of large data collections, data understanding stage is of paramount importance before one can start preparing data or analysis. Tagging (i.e., metadata tagging) is used to express the data understanding, through organizing, identifying, and interpreting the raw data ingested in the lake.
  1. **Building Data Lake from Modern Data Architecture**

Build Data Lake using below layers from Modern data architecture,

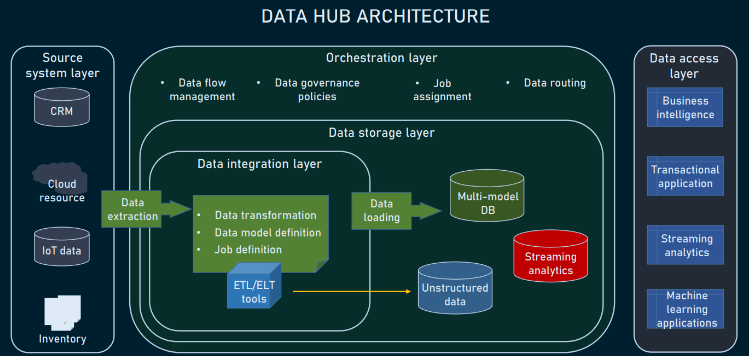
* Data Ingestion - The data ingestion is responsible for the extraction of data from its original source and integration of sources with the data lake.
* Data Stagging - The data stagging layer hold lot of data quality checks passes, cleansed data and any data type conversion required for source data.
* Data Processing Layer - The data integration layer holds any transformations required to make the data digestible for end users. This often involves such operations as data harmonization, mastering, and enrichment with metadata.
* Data Storage layer - The storage layer corresponds to the needs of database management and data modelling.
* Data Consumer Layer - The data access layer unites all the access points connected to the data hub (transactional application, BI systems, machine learning training software, etc).



* + 1. **Best practices for Data Lake Implementation**
* Architectural components, their interaction and identified products should support native data types
* Design of Data Lake should be driven by what is available instead of what is required. The schema and data requirement are not defined until it is queried
* Design should be guided by disposable components integrated with service API.
* Data discovery, ingestion, storage, administration, quality, transformation, and visualization should be managed independently.
* The Data Lake architecture should be tailored to a specific industry. It should ensure that capabilities necessary for that domain are an inherent part of the design
* Faster on-boarding of newly discovered data sources is important
* Data Lake helps customized management to extract maximum value
* The Data Lake should support existing enterprise data management techniques and methods
  1. **Building Data hub from Modern Data Architecture**

Build Data hub using below layers from Modern data architecture,

* Data Ingestion - The data ingestion is responsible for the extraction of data from its original source and integration of sources with the data hub.
* Data Processing Layer - The data integration layer holds any transformations required to make the data digestible for end users. This often involves such operations as data harmonization, mastering, and enrichment with metadata.
* Data Storage layer - The storage layer corresponds to the needs of database management and data modelling.
* Data Consumer Layer - The data access layer unites all the access points connected to the data hub (transactional application, BI systems, machine learning training software, etc).
* The orchestration layer provides control over data integration, flows, transformations, and data governance.



* 1. **Characteristics Comparison between Data warehouse, Data Lake and Data Hub**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Data Warehouse | Data lake | Data Hub |
| Purpose | Business Intelligence and Data Analytics, Single source of storage | Data science and Machine learning purposes | Any operational, Analytical, data science or Machine learning purposes |
| Data format | Structured only | Semi structured or Unstructured | Mostly structured, semi structured or Unstructured |
| Data Quality | High Quality | Medium to Low | High Quality |
| Performance | Medium to Low | Medium to Low | High to Medium |
| Data Integration Type | Mostly Traditional ETL | Mostly ELT | Both ELT and ELT |
| Data Governance | High Data Governance | Low to No data governance | High Data Governance |

* 1. **Data life cycle in Data Architecture**

overall data lifecycle in a data lake. In theory, a well-managed data lake retains raw data permanently, while constantly improving and evolving process data to meet your business needs.

Graphical user interface

Description automatically generated

* 1. **Reference Architecture of Modern Data Architecture components**

It provides a unified area for the storage of the internal data of an enterprise or organization. A data lake uses a scalable distributed file system for storage. Most data lake practices recommend using distributed systems, such as Amazon S3, Alibaba Cloud OSS, OBS, and HDFS, as the data lake's unified storage.

Diagram

Description automatically generated

* 1. **Commercially Available Data Lake Solutions**

AWS-recommended data lake solution. This solution is based on AWS Lake Formation, which is a management component and works with AWS's other services to form an enterprise data lake.

Graphical user interface, application

Description automatically generated

* 1. **Questions asked in Real time project (Coty)**

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Questions | Response | Stakeholder |
| 1 | Please share a sample data for creation of forecasting models. (10 records of data / junk can be helpful on column names etc.) |  | Coty |
| 2 | On predictive use-case, we can use Big Query ML or Vertex AI as ML preference for both ML Ops requirements and Looker integrations? Are there any suggestions between either of these two choices? |  | Google |
| 3 | There can be a latency issue while creating an intermediate layer in between Google BigQuery data model layer and ML layer to prepare the data in the required granularity for ML build? Are there any recommendations or possible work arounds? |  | Google |
| 4 | For the Google public datasets do we need to do perform cleansing/validation on the data from Source files or is it fair assume that we can directly dump into GCS.? |  | Google |
| 5 | How many files would be there for each data source? |  | COTY & Google |
| 6 | What is a max data size would be there in each data source? |  | COTY & Google |
| 7 | Do we need to do any filter on the source data into GCS? |  | Google |
| 8 | Assuming that the data sources loaded in the landing layer may have varying file formats and we might have to perform a cleansing/validation and convert into the accepting file format for GBQ. Are there any other suggestions to this step? |  | Google |
|  | What are the different file formats present in each of the data source? |  | COTY & Google |
| 9 | As per our understanding, periodically there may be addition/deletion to the columns when loading to Big query staging layer? (For reference - Addition is Time stamp and Deletion is PII Data Masking) |  | Google |
| 10 | For this POC should we cover SCD process, If so, can we handle the incremental process through overwrite/ SCD for the POC? |  | Google |
| 11 | From BigQuery staging layer, are there any expected transformation types to be executed with respect to each data source? |  | Google |
| 12 | We are assuming that we will connect to SFTP to access source data. Please confirm |  | COTY |
| 13 | Do we have any existing data model or need to create new data model of the consumption layer on the Big query? |  | COTY |

## Assumptions handled in Real time project – Coty

|  |  |
| --- | --- |
| Activities | Assumptions |
| Discovery Phase |  |
| Discovery | Brillio will require approximately 2 weeks at the beginning of the engagement to complete a discovery workshop of existing landscape. |
| Discovery Deliverable Review and Sign Off |  |
| Design Phase |  |
| Design Ingestion | Assumed that only one data file is to be considered for each data sources in this POC Coty On-prem data:- Considered the source type as Files and No of files would be 1 for each data source and File size would be max. of 2 GB.  Google data:- Google data would be considered that directly loaded to GBQ consumption layer with data model.  Azure data:- Considered as Azure data would be in Blob storage and source type would be Files. file size would be max. 2 GB |
| Design Data Model | Coty On-prem data:- Considered that does not required any modification from the source schema and add created timestamp to the staging table along with source scheme if required. Google Data:- No data model is required as Google data directly loaded to GBQ Consumption.  Azure Data:- Considered that does not required any modification from the source schema and add created timestamp to the staging table along with source scheme if required. |
| Design Feature Engineering | Assuming the data should be available to the Data Science team in the 3rd week to create the feature engineering |
| Design GCS bucket & folder creation | Considered that single GCS bucket would be created and will create multiple folders to handle the multiple datasets accordingly. |
| Data Ingestion to GCS | Coty On-Prem Data - SFTP credentials and access are already provisioned. Can connect to SFTP server, download the file once and upload the file to GCS manually.  Google Data:- No data ingestion is required as Google data directly loaded to GBQ Consumption.  Azure data - Cloud VPN setup is already provisioned and connect Azure Blob storage through Cloud VPN, download the source files once and upload to the GCS. |
| Design Review and Sign Off |  |
| Build Phase |  |
| Implement Physical Data Model | Coty On-prem data and Azure Data:- Google Big query access are provisioned already and create single table for each data source and No Normalisation would be considered for staging and Consumption tables. Google Data:- No physical data model implementation is required as Google data directly loaded to GBQ Consumption. |
| Review of the Physical Data Model |  |
| Creating Data prep Jobs & perform DQ | Basic Data quality checks would be considered for each source file for this POC. |
| Migration of Data to Staging | Direct loading from GCS to Staging GBQ layer excluding the partition and clustering features due to file size is Small for this POC |
| Migration of Data to Consumption | Resultant data of Transformations will be loaded to Consumption excluding the partition and clustering features due to file size is Small for this POC |
| Build Transformation SQL Scripts in GBQ | Basic transformations features which are available in GBQ by default would be considered for this POC. |
| Execute Data prep & Data Migration Jobs | Executing the Data Migration jobs on demand basis, No Scheduling and Orchestration would be considered. |
| Data Testing | Validating the DQ checks and Transformations applied to the data set through GBQ queries. |
| Data Exploration | Define the Forecasting Window which is based on model performance. If the FW is i.e., 60 days and the pattern is changing every 15 days, then for retraining we need to change the FW |
| Data Sanity Check | Assuming COTY to provide a SPOC for this POC for resolving any blockers, finalizing the features once its created. |
| Pattern Discovery | During the data exploration and pre-processing if we find any issues in the data i.e., imbalanced dataset or majority of any of the data sources are missing then we will flag it to the COTY spoc before creating any base model |
| Pre-Processing |  |
| Model Development & Validation | Assuming for Forecasting model we have only ARIMA/ARIMA PLUS models in BigQuery ML assuming there is no additional hyper parameter tuning in BigQuery ML In BigQuery ML's ARIMA, Trend modelling is an automated process thereby leaving very limited scope for the DS team to model. |
| Model Testing |  |
| Review Forecasting Models | The current scope is assumed at developing the forecasting model based on different scenarios of macro-economic factors |
| Data Exploration for Looker Dashboards |  |
| Mockup Creation | Assumed Initial version of mock-ups are to be given by COTY. |
| Review Mock-ups |  |
| Looker ML Modelling | Assuming Mock-ups are ready and signed off from COTY Team |
| Review ML Modelling |  |
| Brand Health Dashboard | Assuming Looker ML Modelling is reviewed & signed off from COTY |
| Review Dashboard 1 - Brand Health |  |
| Insight Exploration Dashboard | LookML Modelling should be signed from COTY |
| Review Dashboard 2 - Insight Exploration |  |
| Mobility-based Forecasts Dashboard | Google BigQuery ML Code is implemented and stored this code's output in the table for creating this dashboard. |
| Review Dashboard 3 - Mobility Based Forecasting Dashboard |  |
| Dashboard Testing | Assuming All the Dashboards are ready to be tested. |
| User Acceptance Test & Support |  |
| GO Live Phase |  |
| Review Final Deliverables | COTY to review & approve the final deliverables in a timely manner to avoid any change in project timelines |
| Knowledge Transfer & Handover | All the Dashboards are ready to be used and signed off from COTY. |

# API Documentation

**What is API:**

API stands for Application Programming Interface. A Web API is an application programming interface for the Web. A Browser API can extend the functionality of a web browser. A Server API can extend the functionality of a web server.

**What is REST:**

**Representational state transfer** (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

**Anatomy of REST:**

Alright, so now we know that data can be requested by the client and the server will respond appropriately.

Endpoint: It is the URL where the REST Server is listening.

**Method:** REST implements multiple 'methods' for different types of requests, the following are most popular:

* + GET: Get resource from the server
  + POST: Create resource to the server
  + PATCH or PUT: Update existing resource on the server
  + DELETE: Delete existing source from the server

**Headers:** The additional details provided for communication between client and server (REST is stateless). Some of the common headers are:  
 Request:

* + - host: the IP of client (or from where request originated)
    - accept-language: language understandable by the client
    - user-agent: data about client, operating system, and vendor

Response:

* + status: the status of request or HTTP code
  + content-type: type of resource sent by server.
  + set-cookie: sets cookies by server

**Data:** (also called body or message) contains info send to the server.

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **CRUD** | **Entire Collection (e.g., /customers)** | **Specific Item (e.g., /customers/{id})** |
| POST | Create | 201 (Created), 'Location' header with link to /customers/{id} containing new ID. | 404 (Not Found), 409 (Conflict) if resource already exists. |
| GET | Read | 200 (OK), list of customers. Use pagination, sorting, and filtering to navigate big lists. | 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. |
| PUT | Update/ replace | 405 (Method Not Allowed) unless you want to update/replace every resource in the entire collection. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| PATCH | Update/ Modify | 405 (Method Not Allowed) unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 405 (Method Not Allowed) unless you want to delete the whole collection—not often desirable. | 200 (OK). 404 (Not Found), if ID not found or invalid. |

**POST:**

The POST verb is most-often utilized to create new resources. It is used to create subordinate resources. That is, subordinate to some other (e.g., parent) resource. In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI), etc.

**Examples:**

*POST* [*http://www.example.com/customers*](http://www.example.com/customers)

**GET:**

According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption—calling it once has the same effect as calling it 10 times, or none. Additionally, GET (and HEAD) is idempotent, which means that making multiple identical requests ends up having the same result as a single request.

**Examples:**

*GET* [*http://www.example.com/customers/12345*](http://www.example.com/customers/12345)

**DELETE:**

DELETE is easy to understand. It is used to \*\*delete\*\* a resource identified by a URI.

**Examples:**

*DELETE* [*http://www.example.com/customers/12345*](http://www.example.com/customers/12345)

**How this implement in RNL Program:**

Data Enrichment for EM purpose is currently happening through PacificEast and Melissa. The data enrichment with Melissa is API based solution while PacificEast is currently based on a file sharing process. Which lead us to some manual processing. The objective is to develop an automated solution using an API-based approach with PacificEast and Melissa in our new environment.

## API- PacificEast

PacificEast does provide the data enrichment of contacts and Email related data. It can provide the basic level of address validation as well. Below are the validation and match can be performed using PacificEast

|  |  |  |
| --- | --- | --- |
| **Data Enrichment Product** | **Input parameters** | **Response** |
| **Name and Address** | Name and Address | Additional details w.r.t the name and address |
| **Email Append** | Name and Address | Email Address |
| **Email Validation** | Email Address | Validates the input email address and provides the information about the deliverability of emails sent to the address |
| **Forward Phone Append** | Name and Address | Phone, name and address related data |
| **Phone Line Information** | Phone number | Line Information like phone Service Type, Location, service provider etc. |
| **Phone Risk Score** | Phone number | Generate the risk score and risk level |
| **Telified-3D** | Phone number, Name and Address | Verify that the name and address are associated with the given phone number |
| **Telified-AI** | Phone number without Name and Address | Verify whether the phone number is active or not |

## 3.2 Infrastructure – PacificEast

* **Architecture –** Hosted in several data canters throughout North America and are configured to provide high availability. It uses the combination of load balancing, failover, and external monitoring to maintain high service uptimes.
* **Security –** PacificEast use the Transport Layer Security (TLS) for all communication between client applications. And each request must include the account Key
* **Production vs Development Server –** The development servers hold the same access to the same data sources as the production servers. But Development server should not exceed 100 queries per day or 1,000 queries in total. Queries exceeding these totals may be billable
* **Support –** Email support for all the development servers but email and call support for production server

## 3.3 Implementation

PacificEast and Melissa web services can be configured with REST API interface, to handle the request and response of the queries submitted to the services. These queries are formatted in a REST style URL with input criteria and key-value pairs.

The REST API (Representational State Transfer) is used to communicate between the client and the server over HTTP. It takes advantage of the HTTP methodologies to establish communication and enables the servers to cache the response, which improves the application’s performance.

## API Setup

* API URL for PacificEast and Melissa
* Account Key for authorization
* Account and configuration of REST API Interface with PacificEast and Melissa
* Configuration of GET and POST REST API

1. **Design**

The high-level design with the data flow from landing to CRM alignment storage. Following are the few points to know as part of Data Enrichment.

* On-demand enrichment from DIA interface.
* Power platform set the Enrichment flag and ADF pipeline to Enrich the data
* Data, which is already enriched, will be referred first in historical data. If match is not found, then it will be sent to PacificEast / Melissa
* ADF pipeline will call the Rest API to connect with PacificEast web services
* GET & POST methods to request and response
* Updating back the enriched data to CRM Alignment storage.

**Graphical user interface, application

Description automatically generated**

## Current Scope - PacificEast

The following type of data is getting enriched in the existing environment.

* **5-PacificEast Started –** This is for EM purposes and enriching Phone data
* **57-PacificEast Cell Append Started –** For Appending Cell details and it is for FM Bio Purpose
* **61-PacificEast Line Status Started –** This is also applicable for FM Bio and updating Line Status
* **54-PacificEast NCOA Started –** For updating Change in address and is applicable for FM Bio
* **51-PacificEast Phone Append Started –** For Appending phone data for FM Bio records
* **64-PacificEast WirelessID Started –** For updating Wireless ID details for FM Bio file

## Current Scope - Melissa

Melissa API is used for two purposes.

* + - * + EM scoring
        + EM Historic Data

### 4.2.1 Implementation

1. **Data Enrichment Flag-**

* Enrichment flag will set by DQE process with respect to the Purposes.
* Melissa and PacificEast flag separately two columns to be maintained at each Purpose table (MD\_Flag, PE\_Flag)
* Same Two flags must be maintained at CRM alignment as well and should be part of data model while processing the data from Staging to CRM alignment
* Flag values –
  + - Staging – (DIA Review / No Enrichment) for each Melissa and PacificEast column
    - CRM Alignment – DIA Review DIA Approved / DIA Corrected / No Enrichment Enriched / Not Enriched

1. **Azure Data Factory** – Create two pipelines for each DE platform to execute the following activity:
   * EM\_Historical Pull the columns to be passed as parameter and records with DIA Approved flag from CRM alignment DB
   * Code to format the request along with API URL, Account Key and Parameter for each row
   * Parameterize the request to send the formatted code and pass it to Rest API
   * Data masking on Account key passed for the Melissa and PacificEast subscription
   * Store the output response from REST API
   * Update back the data and flag with Enriched at CRM alignment
   * Synch the enriched data with \_All table
   * ADF Pipelines –
     + **EM\_PacificEast\_FPAEnrichment**
     + **EM\_Melissa\_AddressEnrichment**
2. **Audit logs** – Integrate with the audit framework and log the audit w.r.t Data Enrichment. Also add the individual audit of this process into **audit.data\_enrichment** table. Where the count of all the flags corrected from various operations will be placed  
   * Data % and count marked for Enrichment in DQE
   * Data % and count corrected by DIA
   * Data % and count sent for final Enrichment
   * Data % and count not corrected by 3rd party
3. **Rest API –** configuration of REST API with ADF. Record level Request and response using GET & POST method.
4. **Storage –** All the operations will be on top of CRM alignment DB. Below are the tables on which Data Enrichment should be done.
   * **Alignment.ContactBase** – ContactId is the unique identifier and flow till ADS
   * **Alignment.Customeraddressbase** – ContactId is the unique identifier and flow till ADS
5. **Column mapping and parameters to pass–**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Enrichment** | **Operation** | **Columns** | **Alignment Column name** | **Alignment Table Name** | **EM Historical All data columns** |
| Melissa | Input / Output | a1 | Line1 | alignment.customer\_address\_base | ADDRESS1 |
| Melissa | Input / Output | a2 | Line2 | alignment. customer\_address\_base | ADDRESS2 |
| Melissa | Input / Output | city | City | alignment.customer\_address\_base | CITY |
| Melissa | Input / Output | state | StateOrProvince | alignment.customer\_address\_base | STATE |
| Melissa | Input | postal | PostalCode | alignment.customer\_address\_base |  |
| Melissa | Input | ctry | Country | alignment.customer\_address\_base |  |
| Melissa | Output | AddressKey | address\_key | alignment.customer\_address\_base | NR |
| Melissa | Output | Plus4 | PostalCode | alignment.customer\_address\_base | ZIP |
| Melissa | Output | PostalCode |
| Melissa | Output | CountyFIPS | country\_fips | alignment.customer\_address\_base | COUNTY\_CODE |
| Melissa | Output | Latitude | Latitude | alignment.customer\_address\_base | LATITUDE |
| Melissa | Output | Longitude | Longitude | alignment.customer\_address\_base | LONGITUDE |
| Melissa | Status Update | MD\_Flag | MD\_Flag | alignment.customer\_address\_base | MD\_FLAG |
| Melissa | Status Update | getutc() | md\_received\_date | alignment.customer\_address\_base | NR |
| PacificEast | Input | firstName | FirstName | alignment.contact\_base |  |
| PacificEast | Input | lastName | LastName | alignment.contact\_base |  |
| PacificEast | Input | address | Line1 | alignment.customer\_address\_base |  |
| PacificEast | Input | city | City | alignment.customer\_address\_base |  |
| PacificEast | Input | state | StateOrProvince | alignment.customer\_address\_base |  |
| PacificEast | Input | postal | ZIP | alignment.customer\_address\_base |  |
| PacificEast | Output | phoneNumber | Telephone1 | alignment.customer\_address\_base, alignment.contact\_base | HOME\_PHONE |
| PacificEast | Status Update | PE\_Flag | PE\_Flag | alignment.customer\_address\_base, alignment.contact\_base | PE\_FLAG |
| PacificEast | Status Update | getutc() | pe\_received\_date | alignment.customer\_address\_base, alignment.contact\_base | NR |

1. **Data Enrichment Flags –**

Graphical user interface, application

Description automatically generated

1. **ADF Pipelines** 
   * **EM\_Melissa\_AddressEnrichment** – Pipeline with the ETL audit integration and collecting the data from alignment table to send for enrichment

Graphical user interface

Description automatically generated with medium confidence

For each loop to send the request (POST) through rest API for each record once at a time and GET the response to update back the data in alignment table

Graphical user interface

Description automatically generated

* + **EM\_PacificEast\_FPAEnrichment** Pipeline with the ETL audit integration and collecting the data from alignment table to send for enrichment

Graphical user interface, application

Description automatically generated

For each loop to send the request (POST) through rest API for each record once at a time and GET the response to update back the data in alignment table

Graphical user interface, application

Description automatically generated

# Data pipelines

A data pipeline is a series of data processing steps. Data pipelines consist of three key elements: a source, a processing step or steps, and a destination.

Graphical user interface, application

Description automatically generated

## **Differences between Data Pipeline and ETL**

* The major dissimilarity of ETL is that it focuses entirely on one system to extract, transform, and load data to a particular data warehouse. Alternatively, ETL is just one of the components that fall under the data pipeline.
* ETL pipelines move the data in batches to a specified system with regulated intervals. Comparatively, data pipelines have broader applicability to transform and process data through streaming or real-time.
* Data pipelines do not necessarily have to load data to a data warehouse but can choose to load to a selective target such as Amazon’s S3 (Simple Storage Service) bucket or even hook it up to a completely different system.

## **5.2 Benefits of Data Pipelines**

* **Simple and Effective**: Although data pipelines may have the complex infrastructure and functioning process, its use and navigation are quite straightforward. Similarly, the learning process of building a data pipeline is achievable through the common practice of (JVM) Java Virtual Machine language to read and write the files.
* **Compatibility with Apps**: The embedded nature of the data pipelines makes it easier to use for customers and digital marketing strategists alike. Its fitting compatibility prevents the need to install, have config files, or rely on a server. You can have complete data access by just embedding the small size of the data pipeline into an app.
* **Flexibility of Metadata**: The separation of custom fields and records is one of the efficient traits of the data pipeline. The metadata allows you to track down the source of the data, creator, tags, instructions, new changes, and visibility options.
* **Built-In Components:** Although the customizable option is accessible to you, data pipelines have built-in components that allow you to get your data in or out of the pipeline. After built-in activation, you can start working with the data through stream operators.
* **Quick Real-Time Data Segmentation**: Whether your data is stored in the form of excel file, at an online social media platform, or on a remote database – data pipelines can break down the data small chunks that are fundamentally part of the bigger streaming workflow. And real-time functioning does not need an extraneous amount of time to process your data. Consequently, this leaves a wiggle room for you to process and infer data at hand more easily.
* **In-memory Processing**: With the availability of data pipelines, you don’t need to store or save new changes in the data in a file, disk, or random database. Pipelines exert in-memory function that makes the accessibility of data quicker than storing it in a disk.

## 5.3 Guidelines for Data Pipeline Design

There are many factors to consider when designing data pipelines, and early decisions have tremendous implications for future success. The following section is meant to be a reference point for asking the right questions from the start of the data pipeline design process.

**Step 1: Determine the goal**

When designing a data pipeline, the priority is to identify the outcome or value the data pipeline will bring to your company or product. At this stage, we ask relevant questions such as:

* What are our objectives for this data pipeline?
* How do we measure the success of the data pipeline?
* What use cases will the data pipeline serve (reporting, analytics, machine learning)?

**Step 2: Choose the data sources**

We then consider the possible data sources that’ll enter the data pipeline. At this stage, it’s critical to ask questions such as:

* What are all the potential sources of data?
* In what format will the data come in (flat files, JSON, XML)?
* How will we connect to the data sources?

**Step 3: Determine the data ingestion strategy**

With the pipeline goal and data sources understood. We need to ask questions about how the pipeline will collect the data. At this point, we ask questions such as:

* What communication layer will we be using to collect data (HTTP, MQTT, gRPC)?
* Would we be utilizing third-party integration tools to ingest the data?
* Are we going to be using intermediate data stores to store data as it flows to the destination?
* Are we collecting data from the origin in predefined batches or in real time?

**Step 4: Design the data processing plan**

Once data has been ingested, it has to be processed and transformed for it to be valuable to downstream systems. At this stage, it’s necessary to ask questions such as:

* What data processing strategies are we utilizing on the data (ETL, ELT, cleaning, formatting)?
* Are we going to be enriching the data with specific attributes?
* Are we using all the data or just a subset?
* How do we remove redundant data?

**Step 5: Set up storage for the output of the pipeline**

Once the data has been processed, we must determine the final storage destination for our data to serve various business use cases. At this step, we ask questions such as:

* Are we going to be using big data stores like data warehouses or data lakes?
* Would the data be stored on cloud or on-premises?’
* Which of the data stores will serve our top use cases?
* In what format will the final data be stored?

**Step 6: Plan the data workflow**

We then need to design the sequencing of processes in the data pipeline. At this stage, we ask questions such as:

* What downstream jobs are dependent on the completion of an upstream job?
* Are there jobs that can run in parallel?
* How do we handle failed jobs?

**Step 7: Implement a data monitoring and governance framework**

In this step, we establish a data monitoring and governance framework, which helps us observe the data pipeline to ensure a healthy and efficient channel that’s reliable, secure, and performs as required. In this step, we determine:

* What needs to be monitored?
* How do we ensure data security?
* How do we mitigate data attacks?
* Is the data intake meeting the estimated thresholds?
* Who is in charge of data monitoring?

**Step 8: Plan the data consumption layer**

This step determines the various services that’ll consume the processed data from our data pipeline. At the data consumption layer, we ask questions such as:

* What is the best way to harness and utilize our data?
* Do we have all the data we need for our intended use case?
* How do our consumption tools connect to our data stores?

**Step 9: Detect Failures in the pipeline**

This step helps in finding errors/failures in the pipeline. Good [logging practices](https://www.bmc.com/blogs/monitoring-logging-tracing/) help surface errors, and well-written logs allow the error to be identified and located quickly. The logs should answer the following questions.

* Why did the pipeline task fail?
* When was the pipeline scheduled?
* Whether third-party processor produce logs and you have access to them?

**Step 9: Recover from the failures**

Resilience is adapting in the event of failure. By designing observability into the pipeline, failures are announced and seen and allow action to be taken. Following questions will help while ensuring recovery from the failures.

* Whether data validators on the pipeline that ensure the appropriate data is being returned to the consumer?
* Whether proper logs in place throughout the pipeline, failures will be exposed and, whether through machine or human correction, the system is free to adapt?

# Storage Resources

Data storage is a big deal. Data companies are in the news a lot lately, especially as companies attempt to maximize value from big data’s potential. For the lay person, data storage is usually handled in a traditional database. But for big data, companies use data warehouses and data lakes.

Data lakes are often compared to data warehouses—but they should not be. Data lakes and data warehouses are very different, from the structure and processing all the way to who uses them and why. In this article, we’ll:

## Defining database, warehouse, lake, lakehouse and Mesh

**Database**

A database is a storage location that houses [structured data](https://www.bmc.com/blogs/structured-vs-unstructured-data/). We usually think of a database on a computer—holding data, easily accessible in a number of ways. Arguably, you could consider your smartphone a database on its own, thanks to all the data it stores about you.

For all organizations, the use cases for databases include:

* Creating reports for financial and other data
* Analyzing relatively small datasets
* Automating business processes
* Auditing data entry

Popular databases are:

* Oracle
* [PostgreSQL](https://www.bmc.com/blogs/mongodb-vs-postgresql/)
* [MongoDB](https://www.bmc.com/blogs/mongodb-overview-getting-started-with-mongodb/)
* [Redis](https://www.bmc.com/blogs/redis-data-types/)
* [Elasticsearch](https://www.bmc.com/blogs/elasticsearch-introduction/)
* [Apache Cassandra](https://www.bmc.com/blogs/apache-cassandra-introduction/)

**Data warehouse**

Data warehouses are large storage locations for data that you accumulate from a wide range of sources. For decades, the foundation for business intelligence and data discovery/storage rested on data warehouses. Their specific, static structures dictate what data analysis you could perform.

Data warehouses are popular with mid- and large-size businesses as a way of sharing data and content across the team- or department-siloed databases. Data warehouses help organizations become more efficient. Organizations that use data warehouses often do so to guide management decisions—all those “data-driven” decisions you always hear about.

Popular companies that offer data warehouses include:

* [Snowflake](https://www.bmc.com/blogs/import-data-s3-snowflake/)
* Yellowbrick
* Teradata

**Data lake**

A data lake is a large storage repository that holds a huge amount of raw data in its original format until you need it. Data lakes exploit the biggest limitation of data warehouses: their ability to be more flexible.

As we’ll see below, the use cases for data lakes are generally limited to data science research and testing—so the primary users of data lakes are data scientists and engineers. For a company that actually builds data warehouses, for instance, the data lake is a place to dump and temporarily store all the data until the data warehouse is up and running. Small and medium sized organizations likely have little to no reason to use a data lake.

Popular data lake companies are:

* [Hadoop](https://www.bmc.com/blogs/hadoop-introduction/)
* Azure
* Amazon S3

**Data Lakehouse:**

New systems are beginning to emerge that address the limitations of data lakes. A lakehouse is a new, open architecture that combines the best elements of data lakes and data warehouses. Lakehouses are enabled by a new system design: implementing similar data structures and data management features to those in a data warehouse directly on top of low-cost cloud storage in open formats. They are what you would get if you had to redesign data warehouses in the modern world, now that cheap and highly reliable storage (in the form of object stores) are available.

**Data Mesh:**

Data mesh is a new approach to thinking about data based on a distributed architecture for data management. The idea is to make data more accessible and available to business users by directly connecting data owners, data producers, and data consumers. Data mesh aims to improve business outcomes of data-centric solutions as well as drive adoption of modern data architectures.

From the business point of view, data mesh introduces new ideas around “data product thinking.” In other words, thinking about data as a product that fulfils a “job to be done”, for example, to improve decision-making, help detect fraud or alert the business to changes in supply chain conditions. To create high-value data products, companies must address culture and mindset shifts and commit to a more cross-functional approach to business domain modelling.

## Comparing data storages

Now that we have got the concepts down, let’s look at the differences across databases, warehouses, and data lakes in six key areas.

**Data**

Database and data warehouses can only store data that has been structured. A data lake, on the other hand, does not respect data like a data warehouse and a database. It stores all types of data: structured, semi-structured, or unstructured.

All three data storage locations can handle [hot and cold data](https://www.bmc.com/blogs/cold-vs-hot-data-storage/), but cold data is usually best suited in data lakes, where the latency isn’t an issue. (More on latency below.)

**Processing**

Before data can be loaded into a data warehouse, it must have some shape and structure—in other words, a model. The process of giving data some shape and structure is called [schema-on-write](https://www.elastic.co/blog/schema-on-write-vs-schema-on-read). A database also uses the schema-on-write approach. A data lake, on the other hand, accepts data in its raw form. When you do need to use data, you have to give it shape and structure. This is called schema-on-read, a very different way of processing data.

**Cost**

One of most attractive features of big data technologies is the cost of storing data. Storing data with big data technologies is relatively cheaper than storing data in a data warehouse. This is because data technologies are often open source, so the licensing and community support is free. The data technologies are designed to be installed on low-cost commodity hardware.

Storing a data warehouse can be costly, especially if the volume of data is large. A data lake, on the other hand, is designed for low-cost storage. A database has flexible storage costs which can either be high or low depending on the needs.

**Agility**

A data warehouse is a highly structured data bank, with a fixed configuration and little agility. [Changing the structure](https://www.hindawi.com/journals/sp/2017/7392349/) is not too difficult, at least technically, but doing so is time consuming when you account for all the business processes that are already tied to the warehouse.

Likewise, databases are less agile to configure because of their structured nature.

Conversely, a data lake lacks structure. This agility makes it easy for data developers and data scientists to easily configure and reconfigure data models, queries, and applications. (That explains why data experts primarily—not lay employees—are working in data lakes: for research and testing. The lack of structure keeps non-experts away.)

**Security**

Data warehouse technologies, unlike big data technologies, have been around and in use for decades. Data warehouses are much more mature and secure than data lakes.

Big data technologies, which incorporate data lakes, are relatively new. Because of this, the ability to secure data in a data lake is immature. Surprisingly, databases are often less secure than warehouses. That’s likely due to how databases developed for small sets of data—not the big data use cases we see today. Luckily, data security is maturing rapidly.  
**Users**

Data warehouses, data lakes, and databases are suited for different users:

* Databases are very flexible and thus suited for any user.
* Data warehouses are used mostly in the business industry by business professionals.
* Data lakes are mostly used in scientific fields by data scientists.

Table

Description automatically generated

## Caution on data lakes

Companies are adopting data lakes, sometimes instead of data warehouses. But data lakes are not free of drawbacks and shortcomings. New technology often comes with challenges—some predictable, others not. Data lakes are no different. It isn’t that data lakes are prone to errors. Instead, companies venturing into data lakes should do so with caution.

Data lakes won’t solve all your data problems. In fact, they may add fuel to the fire, creating more problems than they were meant to solve. That’s because data lakes tend to overlook data best practices.

**Data lakes allow you to store anything without questioning whether you need all the data.** This approach is faulty because it makes it difficult for a data lake user to get value from the data.

**Data lakes do not prioritize which data is going into a supply chain and how that data is beneficial.**This lack of data prioritization increases the cost of data lakes (versus data warehouses and databases) and muddies any clarity around what data is required. This slows, perhaps halts, your entire analytical process. Avoid this issue by summarizing and acting upon data beforestoring it in data lakes.

**Data latency is higher in data lakes.**Data lakes are often used for reporting and analytics; any lag in obtaining data will affect your analysis. Latency in data slows interactive responses, and by extension, the clock speed of your organization. Your reason for that data, and the speed to access it, should determine whether data is better stored in a data warehouse or database.

**Data lakes do not have rules overseeing what they can take in, increasing your organizational risk.** The fact that you can store all your data, regardless of the data’s origins, exposes you to a host of regulatory risks. Multiply this across all users of the data lake within your organization. The lack of data prioritization further compounds your compliance risk.

**Data lakes foster data overindulgence.** Too much unprioritized data creates complexity, which means more costs and confusion for your company—and likely little value. Organizations should not strive for data lakes on their own; instead, data lakes should be used only within an encompassing data strategy that aligns with actionable solutions.

Data is only valuable if it can be utilized to help make decisions in a timely manner. A user or a company planning to analyse data stored in a data lake will spend a lot of time finding it and preparing it for analytics—the exact opposite of data efficiency for data-driven operations.  
  
Instead, you should always view data from a supply chain perspective: beginning, middle, and end. No matter the data, you should always plan a strategy for how you will:

* Find the data
* Bring data into organizational data storage
* Explore and transform the data

Such an approach allows optimization of value to be extracted from data.

# Distributed Systems

A distributed system is a computing environment in which various components are spread across multiple computers (or other computing devices) on a network. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task.

Graphical user interface, application, Teams

Description automatically generated

## **Problem with the Centralized Systems**

Centralized systems follow a client-server architecture that is built around a single server with large computational capabilities. Centralized systems are easy to set up, easy to manage and monitor, easy to physically secure and maintain the server, can be developed quickly and cost less to set up for a small system.

However, when the data is huge, comes with different varieties (Structured, semi structured and unstructured) and growing exponentially, centralized system will not be a feasible solution. Though we can solve this problem by vertical scaling to some extent, there are some physical limits beyond which Vertical scaling is not possible.

## How to solve this problem

The trick is to distribute the workload among multiple computers and collect useful result. Distributed systems provide scalability and improved performance in ways that monolithic systems can't and because they can draw on the capabilities of other computing devices and processes, distributed systems can offer features that would be difficult or impossible to develop on a single system.

## **Few of the advantages of Distributed Systems**

1. **Reliability, high fault tolerance**: A system crash on one server does not affect other servers.
2. **Scalability**: In distributed computing systems you can add more machines as needed.
3. **Flexibility**: It makes it easy to install, implement and debug new services.
4. **Fast calculation speed**: A distributed computer system can have the computing power of multiple computers, making it faster than other systems.
5. **Openness**: Since it is an open system, it can be accessed both locally and remotely.
6. **High performance**: Compared to centralized computer network clusters, it can provide higher performance and better cost performance.

## Are Distributed systems false proof? Can we use it for all types of big data?

To answer this question, one must understand the CAP theorem.

CAP Theorem talks about possibility of achieving Consistency, Availability & Partition Tolerance in a distributed system. It states that, it is possible to provide either consistency or availability—but not both.

A screenshot of a computer

Description automatically generated with low confidence

In other terms,

* High consistency comes at the cost of lower availability.
* High availability comes at the cost of lower consistency.

### When consistency is the priority?

Consistent databases should be used when the value of the information returned needs to be accurate. Financial data is a good example. When a user logs in to their banking institution, they do not want to see an error that no data is returned, or that the value is higher or lower than it is. Banking apps should return the exact value of a user’s account information. In this case, banks would rely on consistent databases.

Options for consistency: MongoDB, Redis and HBase.

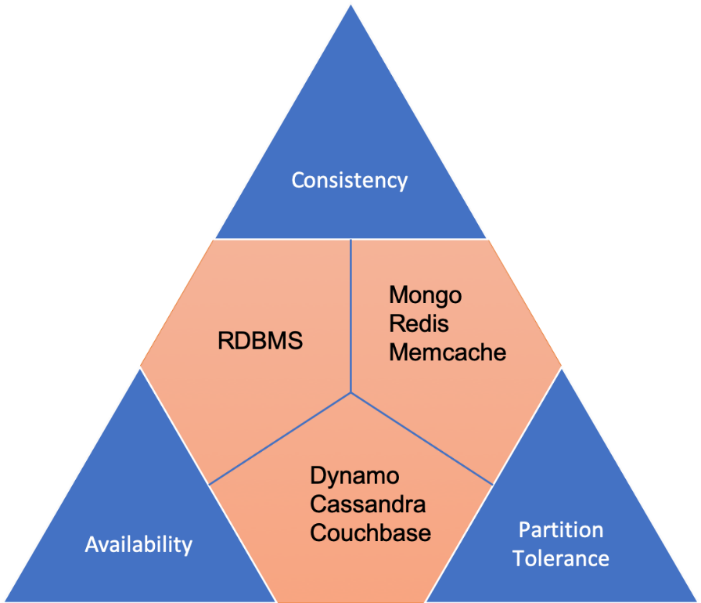
### When Availability is the priority?

Availability databases should be used when the service is more important than the information.

An example of having a highly available database can be seen in e-commerce businesses. Online stores want to make their store

and the functions of the shopping cart available 24/7 so shoppers can make purchases exactly when they need.

Database options for availability: Cassandra, DynamoDB and Cosmos DB



So, one should be careful enough while switching over to Distributed systems and according to the priority should choose the appropriate Distributed systems tool.

## Demerits of Distributed Systems

**Difficult troubleshooting**: Troubleshooting and diagnostics are more difficult due to distribution across multiple servers.

**Less software support**: Less software support is a major drawback of distributed computer systems.

**High network infrastructure costs**: Network basic setup issues, including transmission, high load, and loss of information.

**Security issues**: The characteristics of open systems make data security and sharing risks in distributed computer systems.

## Conclusion

Though Distributed systems solves the limitations of centralized systems they come with their own limitations and demerits. Based on our requirement, budget and other aspects we need to take the decision.

# High Availability

High availability is a label applied to systems that can operate continuously and dependably without failing. These systems are extensively tested and have redundant components to ensure high quality operational performance. In short, high availability systems will be available no matter what occurs.

**Importance**  
Customer satisfaction often relies on whether or not customers can access your product or service when they need to and whether or not they can depend on it to work. High availability architecture ensures that your website, application, or server continues to function through different demand loads and failure types.

## ****8.1** **Is it worth the money?****

The fact that going for high availability architecture gives you higher performance is all right, but it comes at a big cost too. You must ask yourself if you think the decision is justified from the point of view of finance.

A decision must be made on whether the extra uptime is truly worth the amount of money that must go into it. You must ask yourself how damaging potential downtimes can be for your company and how important your services are in running your business.

## ****How Is High Availability Generally Achieved****

**Deploy multiple application servers**

Overburdened servers tend to slow down or eventually crash. You must implement applications over multiple different servers to ensure your applications keep running efficiently and downtime is reduced.

**Scale up and down**

Another way to achieve high availability is by scaling your servers up or down depending on the load and availability of the application. You can achieve vertical and horizontal scaling outside the application at the server level.

**Maintain an automated recurring online backup system**

Automating backup ensures the safety of your critical business data in the event you forget to manually save multiple versions of your files. It is a good practice that pays dividends under multiple different circumstances, including internal sabotage, natural disasters and file corruption.

## High Availability vs Redundancy

Redundancy is often a component of high availability, but they have different meanings. High availability means that a system will be available regardless of circumstances, while redundancy in a system means that multiple components can replace one another to keep things running in case something happens.

## High Availability vs Disaster recovery

High availability is a concept wherein we eliminate single points of failure to ensure minimal service interruption. On the other hand, disaster recovery is the process of getting a disrupted system back to an operational state after a service outage. As such, we can say that when high availability fails, disaster recovery kicks in.

## Best practices

Diagram

Description automatically generated with low confidence

From an enterprise IT perspective, downtime maps directly to lost revenue streams and dissatisfied users. While the accurate calculations vary vastly between organizations, the average may cost $9,000 per minute according to recent research by Ponemon Institute. For organizations as large as Amazon, the cost of downtime is as high as $13.22 million per hour.

So how can you maximize availability of your IT services for lowest financial and business risk? Follow these industry-proven best practices:

* **Assess your business requirements.** Evaluate IT from a business and user perspective.
* **Know the true cost of downtime.** Account for dissatisfied customers and lost user base.
* **Understand your SLAs.** Does the availability correspond to desired metrics?
* **Set Recovery Point Objective (RPO) and Recovery Time Objective (RTO) based on your expected availability percentage.**The system should be capable of recovering to a state with prior to the longest expected duration of downtime. For example, for five-nines, the RTO should be less than 30 seconds.
* **Prepare a thorough disaster recovery program.**Even with the SLAs in place, vendors only reimburse the IT service cost during the period of downtime. The lost business opportunity and revenue are not calculated.
* **Introduce redundancy strategically.** Mission-critical IT workloads are more in need of redundancy than other operational IT workloads that may not be frequently accessed. The cost of making every workload redundancy might not provide ROI.
* **Understand the metrics.**Make sure you and relevant stakeholders understand the differences between and purposes of reliability, availability, and uptime.

# Data Security

## Importance

Data security is the practice of protecting digital information from unauthorized access, corruption, or theft throughout its entire lifecycle. It’s a concept that encompasses every aspect of information security from the physical security of hardware and storage devices to administrative and access controls, as well as the logical security of software applications. It also includes organizational policies and procedures.

When properly implemented, robust data security strategies will protect an organization’s information assets against cybercriminal activities, but they also guard against insider threats and human error, which remains among the leading causes of data breaches today. Data security involves deploying tools and technologies that enhance the organization’s visibility into where its critical data resides and how it is used. Ideally, these tools should be able to apply protections like encryption, data masking, and redaction of sensitive files, and should automate reporting to streamline audits and adhering to regulatory requirements.

## Business Challenges

Consumer awareness of the importance of data privacy is on the rise. Fuelled by increasing public demand for data protection initiatives, multiple new privacy regulations have recently been enacted, including Europe’s General Data Protection Regulation (GDPR) and the California Consumer Protection Act (CCPA). These rules join longstanding data security provisions like the Health Insurance Portability and Accountability Act (HIPAA), protecting electronic health records, and the Sarbanes-Oxley Act (SOX), protecting shareholders in public companies from accounting errors and financial fraud. With maximum fines in the millions of dollars, every enterprise has a strong financial incentive to ensure it maintains compliance.  
 The business value of data has never been greater than it is today. The loss of trade secrets or intellectual property (IP) can impact future innovations and profitability. So, trustworthiness is increasingly important to consumers, with a full 75% reporting that they will not purchase from companies they don’t trust to protect their data.

## Types of Data Security

**Encryption**

Using an algorithm to transform normal text characters into an unreadable format, encryption keys scramble data so that only authorized users can read it.  File and database encryption solutions serve as a final line of defence for sensitive volumes by obscuring their contents through encryption or tokenization. Most solutions also include security key management capabilities.  
  
**Data Erasure**  
More secure than standard data wiping, data erasure uses software to completely overwrite data on any storage device. It verifies that the data is unrecoverable.  
   
**Data Masking**  
By masking data, organizations can allow teams to develop applications or train people using real data. It masks personally identifiable information (PII) where necessary so that development can occur in environments that are compliant.

**Data Resiliency**  
Resiliency is determined by how well an organization endures or recovers from any type of failure – from hardware problems to power shortages and other events that affect data availability (PDF, 256 KB). Speed of recovery is critical to minimize impact.

## Data security capabilities and solutions

**Data discovery and classification tools**  
Sensitive information can reside in structured and unstructured data repositories including databases, data warehouses, big data platforms, and cloud environments. Data discovery and classification solutions automate the process of identifying sensitive information, as well as assessing and remediating vulnerabilities.  
**Data and file activity monitoring**  
File activity monitoring tools analyse data usage patterns, enabling security teams to see who is accessing data, spot anomalies, and identify risks. Dynamic blocking and alerting can also be implemented for abnormal activity patterns.  
**Vulnerability assessment and risk analysis tools**  
These solutions ease the process of detecting and mitigating vulnerabilities such as out-of-date software, misconfigurations, or weak passwords, and can also identify data sources at greatest risk of exposure.  
  
**Automated compliance reporting**  
Comprehensive data protection solutions with automated reporting capabilities can provide a centralized repository for enterprise-wide compliance audit trails.

## Data security strategies

A comprehensive data security strategy incorporates people, processes, and technologies. Establishing appropriate controls and policies is as much a question of organizational culture as it is of deploying the right tool set. This means making information security a priority across all areas of the enterprise.  
  
**Physical security of servers and user devices**  
Regardless of whether your data is stored on-premises, in a corporate data center, or in the public cloud, you need to ensure that facilities are secured against intruders and have adequate fire suppression measures and climate controls in place. A cloud provider will assume responsibility for these protective measures on your behalf.  
 **Access management and controls**The principle of “least-privilege access” should be followed throughout your entire IT environment. This means granting database, network, and administrative account access to as few people as possible, and only those who absolutely need it to get their jobs done.  
  
**Application security and patching**  
All software should be updated to the latest version as soon as possible after patches or new versions are released.  
  
**Backups**  
Maintaining usable, thoroughly tested backup copies of all critical data is a core component of any robust data security strategy. In addition, all backups should be subject to the same physical and logical security controls that govern access to the primary databases and core systems.  
  
**Employee education**  
Training employees in the importance of good security practices and password hygiene and teaching them to recognize social engineering attacks transforms them into a “human firewall” that can play a critical role in safeguarding your data.  
  
**Network and endpoint security monitoring and controls**Implementing a comprehensive suite of threat management, detection, and response tools and platforms across your on-premises environment and cloud platforms can mitigate risks and reduce the probability of a breach.

**Data Privacy and Compliance**

Data privacy generally means the ability of a person to determine for themselves when, how, and to what extent [personal information](https://www.cloudflare.com/learning/privacy/what-is-personal-information/) about them is shared with or communicated to others. This personal information can be one's name, location, contact information, or online or real-world behaviour.

**Why is data privacy important?**

In many jurisdictions, privacy is considered a fundamental human right, and data protection laws exist to guard that right. Organizations use data protection practices to demonstrate to their customers and users that they can be trusted with their personal data.

Personal data can be misused in several ways if it is not kept private or if people don’t have the ability to control how their information is used:

* Criminals can use personal data to defraud or harass users.
* Entities may sell personal data to advertisers or other outside parties without user consent, which can result in users receiving unwanted marketing or advertising.
* When a person's activities are tracked and monitored, this may restrict their ability to express themselves freely, especially under repressive governments.

**What are some of the challenges users face when protecting their online privacy?**

**Online tracking**: User behavior is regularly tracked online. [Cookies](https://www.cloudflare.com/learning/privacy/what-are-cookies/) often record a user's activities, and while most countries require websites to alert users of cookie usage, users may not be aware of to what degree cookies are recording their activities.

**Losing control of data:** With so many online services in common use, individuals may not be aware of how their data is being shared beyond the websites with which they interact online, and they may not have a say over what happens to their data.

**Lack of transparency:** To use web applications, users often have to provide personal data like their name, email, phone number, or location; meanwhile, the privacy policies associated with those applications may be dense and difficult to understand.

**Social media:** It is easier than ever to find someone online using social media platforms, and social media posts may reveal more personal information than users realize. In addition, social media platforms often collect more data than users are aware of.

**Cyber crime:** Many attackers try to steal user data in order to commit fraud, compromise secure systems, or sell it on underground markets to parties who will use the data for malicious purposes. Some attackers use [phishing attacks](https://www.cloudflare.com/learning/access-management/phishing-attack/) to try to trick users into revealing personal information; others attempt to compromise companies' internal systems that contain personal data.

**What are some of the challenges businesses face when protecting user privacy?**

**Communication:**Organizations sometimes struggle to communicate clearly to their users what personal data they are collecting and how they use it.

**Cyber crime:** Attackers target both individual users and organizations that collect and store data about those users. In addition, as more aspects of a business become Internet-connected, the attack surface increases.

**Data breaches:** A data breach can lead to a massive violation of user privacy if personal details are leaked, and attackers continue to refine the techniques they use to cause these breaches.

**Insider threats:** Internal employees or contractors might inappropriately access data if it is not adequately protected.

**What are the laws that govern data privacy?**

As technological advances have improved data collection and surveillance capabilities, governments around the world have started passing laws regulating what kind of data can be collected about users, how that data can be used, and how data should be stored and protected. Some of the most important regulatory privacy frameworks to know include:

* [**General Data Protection Regulation (GDPR)**](https://www.cloudflare.com/learning/privacy/what-is-the-gdpr/)**:** Regulates how the personal data of European Union (EU) data subjects, meaning individuals, can be collected, stored, and processed, and gives data subjects rights to control their personal data (including a [right to be forgotten](https://www.cloudflare.com/learning/privacy/right-to-be-forgotten/)).
* **National data protection laws**: Many countries, such as Canada, Japan, Australia, Singapore, and others, have comprehensive data protection laws in some form. Some, like Brazil's General Law for the Protection of Personal Data and the UK's Data Protection Act, are quite like the GDPR.
* [**California Consumer Privacy Act (CCPA)**](https://www.cloudflare.com/learning/privacy/what-is-the-ccpa/): Requires that consumers be made aware of what personal data is collected and gives consumers control over their personal data, including a right to tell organizations not to sell their personal data.

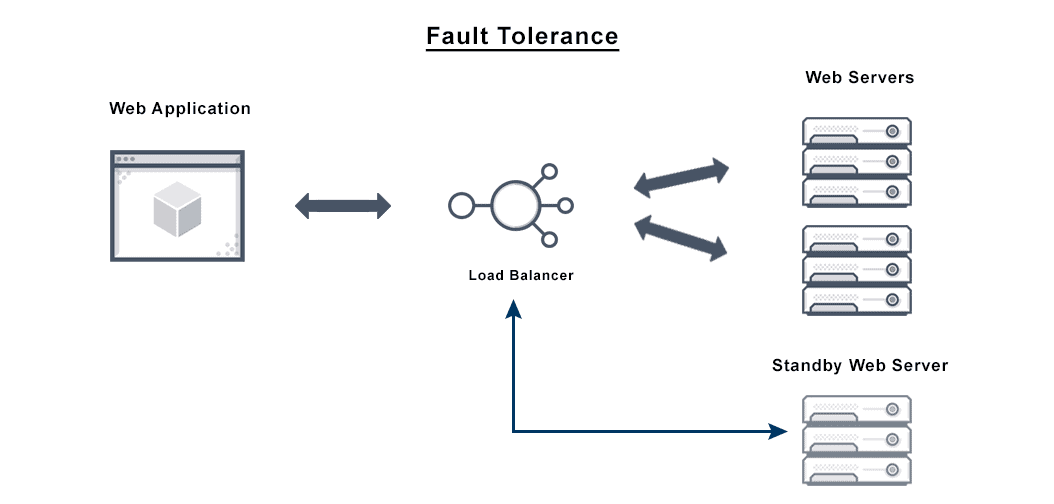
There are also industry-specific privacy guidelines in some countries: for instance, in the United States, the Health Insurance Portability and Accountability Act (HIPAA) governs how personal healthcare data should be handled.

**What are some of the most important technologies for data privacy?**

* [Encryption](https://www.cloudflare.com/learning/ssl/what-is-encryption/) is a way to conceal information by scrambling it so that it appears to be random data. Only parties with the [encryption key](https://www.cloudflare.com/learning/ssl/what-is-a-cryptographic-key/) can unscramble the information.
* [Access control](https://www.cloudflare.com/learning/access-management/what-is-access-control/) ensures that only authorized parties access systems and data. Access control can be combined with [data loss prevention (DLP)](https://www.cloudflare.com/learning/access-management/what-is-dlp/) to stop sensitive data from leaving the network.
* [Two-factor authentication](https://www.cloudflare.com/learning/access-management/what-is-two-factor-authentication/) is one of the most important technologies for regular users, as it makes it far harder for attackers to gain unauthorized access to personal accounts.

# Fault Tolerance

Fault Tolerance simply means a system’s ability to continue operating uninterrupted despite the failure of one or more of its components. This is true whether it is a computer system, a cloud cluster, a network, or something else.



## High Availability vs Fault Tolerance

Highly available systems are designed to minimize downtime to avoid loss of service. Expressed as a percentage of total running time in terms of a system’s uptime, 99.999 percent uptime is the ultimate goal of high availability.

Although both high availability and fault tolerance reference a system’s total uptime and functionality over time, there are important differences and both strategies are often necessary. For example, a totally mirrored system is fault-tolerant; if one mirror fails, the other kicks in and the system keeps working with no downtime at all. However, that’s an expensive and sometimes unwieldy solution.

On the other hand, a highly available system such as one served by a load balancer allows minimal downtime and related interruption in service without total redundancy when a failure occurs. A system with some critical parts mirrored and other, smaller components duplicated has a hybrid strategy.

In an organizational setting, there are several important concerns when creating high availability and fault tolerant systems:

**Cost** Fault tolerant strategies can be expensive because they demand the continuous maintenance and operation of redundant components. High availability is usually part of a larger system, one of the benefits of a load balancing solution, for example.

**Downtime** The greatest difference between a fault-tolerant system and a highly available system is downtime, in that a highly available system has some minimal permitted level of service interruption. In contrast, a fault-tolerant system should work continuously with no downtime even when a component fails. Even a system with the five nines standard for high availability will experience approximately 5 minutes of downtime annually.

**Scope** High availability systems tend to share resources designed to minimize downtime and co-manage failures. Fault tolerant systems require more, including software or hardware that can detect failures and change to redundant components instantly, and reliable power supply backups.

Certain systems may require a fault-tolerant design, which is why fault tolerance is important as a basic matter. On the other hand, high availability is enough for others. The right business continuity strategy may include both fault tolerance and high availability, intended to maintain critical functions throughout both minor failures and major disasters.

## Fault Tolerance Requirements

Depending on the fault tolerance issues that your organization copes with, there may be different fault tolerance requirements for your system. That is because fault-tolerant software and fault-tolerant hardware solutions both offer very high levels of availability, but in different ways.

Fault-tolerant servers use a minimal amount of system overhead to achieve high availability with an optimal level of performance. Fault-tolerant software may be able to run on servers you already have in place that meet industry standards.

## Fault Tolerance Architecture

There is more than one way to create a fault-tolerant server platform and thus prevent data loss and eliminate unplanned downtime. Fault tolerance in computer architecture simply reflects the decisions administrators and engineers use to ensure a system persists even after a failure. This is why there are various types of fault tolerance tools to consider.

At the drive controller level, a redundant array of inexpensive disks (RAID) is a common fault tolerance strategy that can be implemented. Other facility level forms of fault tolerance exist, including cold, hot, warm, and mirror sites.

Fault tolerance computing also deals with outages and disasters. For this reason a fault tolerance strategy may include some uninterruptible power supply (UPS) such as a generator—some way to run independently from the grid should it fail.

Byzantine fault tolerance (BFT) is another issue for modern fault tolerant architecture. BFT systems are important to the aviation, blockchain, nuclear power, and space industries because these systems prevent downtime even if certain nodes in a system fail or are driven by malicious actors.

## Relationship Between Security and Fault Tolerance

Fault tolerant design prevents security breaches by keeping your systems online and by ensuring they are well-designed. A naively designed system can be taken offline easily by an attack, causing your organization to lose data, business, and trust. Each firewall, for example, that is not fault tolerant is a security risk for your site and organization.

## Others

**Fault Tolerance in Cloud Computing**

Conceptually, fault tolerance in cloud computing is mostly the same as it is in hosted environments. Cloud fault tolerance simply means your infrastructure is capable of supporting uninterrupted functionality of your applications despite failures of components.

In a cloud computing setting that may be due to autoscaling across geographic zones or in the same data centers. There is likely more than one way to achieve fault tolerant applications in the cloud in most cases. The overall system will still demand monitoring of available resources and potential failures, as with any fault tolerance in distributed systems.

**Characteristics of a Fault Tolerant Data Center**

To be called a fault tolerant data center, a facility must avoid any single point of failure. Therefore, it should have two parallel systems for power and cooling. However, total duplication is costly, gains are not always worth that cost, and infrastructure is not the only answer. Therefore, many data centers practice fault avoidance strategies as a mid-level measure.

**Load Balancing Fault Tolerance Issues**

[Load balancing](https://www-stage.avinetworks.com/glossary/load-balancing/) and failover solutions can work together in the [application delivery](https://www-stage.avinetworks.com/why-avi/application-delivery-platform/) context. These strategies provide quicker recovery from disasters through redundancy, ensuring availability, which is why load balancing is part of many fault tolerant systems.

Load balancing solutions remove single points of failure, enabling applications to run on multiple network nodes. Most load balancers also make various computing resources more resilient to slowdowns and other disruptions by optimizing distribution of workloads across the system components. Load balancing also helps deal with partial network failures, shifting workloads when individual components experience problems.

# Scalability

Scalabilityis the ability to expand or contract the capacity of system resources in order to support the changing usage of your application. This can refer both to increasing and decreasing usage of the application.

Increased usage of your application brings three main challenges to your database server:

1. The CPU and/or memory becomes overloaded, and the database server either cannot respond to all the request throughput or do so in a reasonable amount of time.
2. Your database server runs out of storage, and thus cannot store all the data.
3. Your network interface is overloaded, so it cannot support all the network traffic received.

## Vertical Scaling

Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput. Additionally, there are increased costs with high-performance hardware, as costs do not scale linearly.

**Pros**

* The main benefit of vertical scaling is that nothing changes about your database infrastructure other than the hardware specifications of the machine running the database.
* As such, it’s transparent to the application. The only difference is that you have more CPUs, memory, and/or storage space.
* Vertical scaling is a good option to try first if massive storage and processing are not required.

**Cons**

* The downside of scaling up is that servers with more storage and processing power can be a lot more expensive.
* There is also a physical limit on the amount of CPUs, memory, network interfaces, and hard-drives that can be used on a single machine. For those scaling up using a cloud platform provider, you will eventually reach the highest tier of machine available.
* If scaling vertically requires a migration between hardwares, it could result in downtime or service disruption.

When cost and/or machine limitations become an issue, be sure to consider **horizontal scaling**.

## Horizontal Scaling

**Horizontal scaling**, also known as scale-out, refers to bringing on additional nodes to share the load. This is difficult with relational databases due to the difficulty in spreading out related data across nodes. With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. This allows them to be distributed across nodes more simply, as queries do not have to “join” them together across nodes.

## Challenges faced by developers due to Scalability

1. **In vertical scaling, database growth is constrained by the hardware**

Even though it may seem that vertical scaling is a good option because it does not imply reprogramming the product, in reality, it has its limitations. There is a time when the hardware cannot grow anymore because the resources are limited by the maximum available capabilities at the moment that the need to scale up the database emerges.

For this reason, the best option system developers have to scale up the database iscombining the vertical and horizontal scaling model.

1. **Horizontal scaling is not supported by all applications**

As we have seen in the previous section, horizontal scaling has many challenges, sincethere are certain products that do not support it and require reprogramming. To overcome this challenge, developers can go one of two ways:

**Data replication** for read-intensive workloads, companies can have a primary copy that accepts data modifications and multiple read-only copies of that data. The disadvantage of this is that the primary copy becomes a bottleneck when writing data.

**Federated database** This implies distributing read and write operations across many nodes. This is accomplished by fragmenting data into several database servers. There is a database replication component where certain data is stored in all or several nodes.

1. **Scaling a relational database is hard**

These databases were developed back when the volume of data was low. But nowadays, Big Data is making scaling a relational database difficult.

When scaling up a database, one can usually opt for vertical or for horizontal scaling. However, in some cases, this is not the best solution.In order to overcome these problems and perform successful scaling, the solution is NoSQL databases.

These databases are not expressly limited by a single-server architecture. These types of databases can massively scale on distributed systems. They can work with multiple servers working together, each sharing part of the load.

Using this approach, the database can operate with hundreds of servers and a vast amount of information. Furthermore,it can process tens of thousands of transactions per second with low-cost hardware operating on any kind of environment. In addition, if a node fails, other nodes can take over, thereby overcoming single point failure.

# Data Lineage

Data lineage refers to the data’s “line of descent.” In other words, a record of how data got to a specific location and the intermediate steps and transformations that took place as it travelled through business systems. Data lineage essentially provides a map of the data journey that includes all steps along the way, as illustrated below.

## 12.1 Questions that Data Lineage Answer

Data lineage clarifies how data flows across the organization. This includes the availability, ownership, sensitivity, and quality of data. It helps ensure that you can generate confident answers to questions about your data:

* What data in my enterprise needs to be governed for [compliance with industry regulations](https://www.informatica.com/solutions/data-governance-and-compliance/ensure-regulatory-compliance.html)?
* What data sources have the personal information needed to develop new [customer experience](https://www.informatica.com/solutions/360-engagement/customer-experience.html) initiatives? And how is this data distributed across the organization?
* What data is appropriate to migrate to the cloud and how will this affect users?
* Where do we have data flowing into locations that violate data governance policies?
* How does data quality change across multiple lineage hops?
* How can data scientists improve confidence in the data needed for advanced analytics?
* How can the person in charge (usually the DPO) of data compliance make sure that no data is exploited with the arrival of data compliance laws such as the [BCBS-239](https://zeenea.com/what-is-the-bcbs-239/) or the GDPR etc?

## 12.2 Importance of Data Lineage

* Understanding the provenance and lineage of data sources is valuable for several reasons:
* Evaluating the trustworthiness of data based on its provenance
* Understanding and correcting sources of error
* Identifying incorrect assumptions about data that may skew analysis
* Providing audit trails for data governance and regulatory purposes
* Ensuring data flows are protected and not subject to tampering
* Identifying and avoiding data duplication to simplify operations and reduce cost

## 12.3 Implementation

Implementing data lineage strongly depends on your organization’s data culture. Ensure you have an established data management framework and build a strong collaboration with data management professionals and other stakeholders for successful data lineage implementation.

Steps to successfully implement data lineage in your organization.

* **Identify key business drivers:** Discuss reasons to implement data lineage and find whether they are crucial for meeting business objectives. These reasons can include business changes, data quality initiatives, auditory requirements, or legislation requirements.
* **Onboard senior management on the project:** Implementing data lineage requires many resources (both human and financial) and time. Ensure you have the support of senior management to move the implementation project toward completion. You can convince management by explaining the benefits of data lineage and how it helps in complying with industry regulations.
* **Scope the initiative:** Once senior management approves the project, decide its scope based on the identified business drivers and critical data elements (CDE). Critical data elements have the most significant impact on the organization’s performance and customer experience.
* **Define the scope:** Scope of data lineage starts with data sources and ends at the final usage point. Large organizations can fix a limited length of data lineage since they have many subsidiaries to avoid complications.
* **Prepare business requirements:** Stakeholders may have different expectations for data lineage. Primarily, there are business stakeholders and technical stakeholders who have different interests. Business stakeholders are more interested in value, data lineage on conceptual **data model** levels, and root cause analysis. On the contrary, technical stakeholders have interests in impact analysis, metadata design lineage, and data lineage on a physical level.
* **Fix a method to document data lineage:** You can either go with descriptive or automated data lineage documentation. Assess which way would be more suitable for your organization, considering the time and resources it will consume.
* **Choose a suitable data lineage software:** Select a data lineage software solution that best suits your goals and expectations. You can explore master data management software that offers automated lineage capabilities.

## 12.4 Best Practices

Lineage helps you get trustworthy and accurate data to support your company’s decision-making process. Planning and implementing is a critical element of data governance - you need to be sure where your data is coming from and where it’s taking you.

There are a few practices you can consider while planning and implementing data lineage in your organization:

* **Automate data lineage extraction:** Data and its lineage is a dynamic entity. You need to move beyond manually capturing data lineage in spreadsheets and automate the process to compete in an agile environment.
* **Include metadata source:** Database management systems, big data tools, ETL software, and other custom applications create their own data about the data they process. Include this metadata in your lineage as it helps in understanding data flow and modifications.
* **Verify metadata sources:** Encourage owners of applications and tools to verify the respective metadata sources since they are the ones who clearly understand the accuracy and relevance of metadata.
* **Plan progressive extraction:** Extract metadata and lineage in the same order in which data flows through your system. It simplifies mapping connections, relationships, and dependencies among systems and within data.
* **Validate end-to-end data lineage:** Validate lineage progressively by starting from high-level connections between systems and then delve into connected datasets followed by data elements before validating transformations documentation.
* **Implement data catalog software:** Adopt an intelligent and automated data catalog software to collect lineage data from all sources. This software also enables you to extract and infer lineage from metadata..

# Relational Database Management System

RDBMS stands for Relational Database Management System. It is an information management system that is oriented on a data model. Here all the information is properly stored as tables. RDBMS Example systems are SQL Server, Oracle, MySQL, MariaDB, and SQLite.

Graphical user interface, diagram, application

Description automatically generated

## Relational Database Architecture

Database Architecture is an essential unit of product design\architecture and performance. Standardization and Rules are much needed while planning a database design factor. Database Architecture covers complete schema design, under flow of application, using SQL, NoSQL and In-Memory cache structure. Interesting subject of schema design is – How to achieve everything? Performance, Load, RDBMS (multiple database entity relationship), Loose\Tight coupling with n-databases, degree of parallelism (DOP) and such.

Superlative Architecture Design – a distribution of user requests at various levels of the database such as Cache, NoSQL and SQL Database. Cache and NoSQL plays vital role as virtual database in front of transactional database and serving maximum data to reduce SQL database burden. To manage and maintain a virtual database, transaction chain is the key and must be well formed and horizontally scaled.

## Database Design Elements

The Main Database design elements are

1. **Database Technology Selection:** Database Technology Selection depends on criteria of technical operation, Development and Resource costing. The most obvious goal should be data-centric solutions such as Capabilities, Robustness, Large Data size Pipelines, Reliability, Availability, Scalability and Disaster Solutions. Limited programmatic solution needs to meet the requirement solutions such as procedure, function, Recursive Loops, etc.
2. **Database Modelling:** Database modelling is a logical architecture and a conceptual structure of a database with entity relationship. It represents the flow of application in order to schema-entity level of varying structure including data type and constraint.

As part of good architecture, the table structure should be in normalized form with appropriate data types. That will make the read and write transactions to be distributed in multiple entities.

Whenever you find business logic is fully database dependent with much load, you should look forward to managing product specific, module-wise separate database among multiple database server instance, if required.

1. **Product Development Factor:** Transaction concurrency plays vital role in database load management and performance. A bunch of database operations and requests should be in with well-formed ACID property to avoid database Locking and Blocking problem.
2. **Database Administration Plan:**  In order to have decent administration plan once should have following things well documented:
3. Index Maintenance Plan
4. Backup strategies
5. High Availability (Failover)
6. Disaster Recovery
7. User Management
8. Security measures, policies and standards
9. Installation of platforms
10. Query performance monitoring
11. Functional unit monitoring

There are more but these are the needed things in an administration plan along with the standard resolving steps.

## Principles of Relational Databases

Knowing and understanding these principles will help you develop and design RDBs:

1. Information is represented logically in tables.
2. Data must be logically accessible by table, primary key, and column.
3. Null values must be uniformly treated as “missing information,” not as empty strings, blanks, or zeros.
4. Metadata (data about the database) must be stored in the database just as regular data is.
5. A single language must be able to define data, views, integrity constraints, authorization, transactions, and data manipulation.
6. Views must show the updates of their base tables and vice versa.
7. A single operation must be able to retrieve, insert, update, or delete data.
8. Batch and end-user operations are logically separate from physical storage and access methods.
9. Batch and end-user operations can change the database schema without having to recreate it or the applications built upon it.
10. Integrity constraints must be available and stored in the RDB metadata, not in an application program.
11. The data manipulation language of the relational system should not care where or how the physical data is distributed and should not require alteration if the physical data is centralized or distributed.
12. Any row processing done in the system must obey the same integrity rules and constraints that set-processing operations do.

## Points to be taken care of, while designing a Database

1. Relationship between tables must be with integer value.
2. Use LOB datatype only when required.
3. Each varying columns must be defined with length.
4. Avoid duplicate type of records with n-Level normalization.
5. Audit history tables.
6. No heap table in database.
7. Database indexing.

## Database Schema Designs and How to Use Them

Database schemas are the blueprints that help developers visualize how databases should be built. They provide a reference point that indicates what fields of information the project contains. If there is any issue, the developer can simply refer to the schema.

Data administrators also use schemas to ensure that the database is fully designed before implementation. This saves valuable time and money, because once a database is implemented, making changes can be difficult. The schema ensures that all the stakeholders have fully considered every aspect of the project, which reduces the need for change.

#### Flat Model: The flat model is for small, simple applications.

A flat model schema is a single, two-dimensional array where elements in each column are the same type of data, and elements in the same row relate to each other.

#### undefined

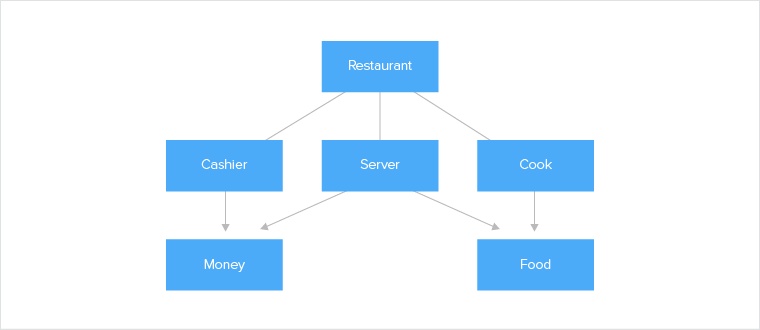
#### Hierarchical Model: The hierarchical model is for nested data, like XML or JSON.

#### Hierarchical models have a tree-like structure, with a "root" node of data and child nodes that branch out from that root. There is a one-to-many relationship between parent and child nodes. This type of data schema is best reflected in XML or JSON files, where an entity can have sub-entities that are not shared with other entities.

#### undefined

#### Network Model: The network model is useful in mapping and spatial data, also for depicting workflows. The network model is like the hierarchical model in that it represents a series of nodes and vertices; however, unlike the hierarchical model, it allows for many-to-many relationships. From a theoretical standpoint, this means that the graph can have cycles. A cycle in the graph indicates that there is a path of vertices in which you can start and end at the same node.

#### 



#### Relational Model: The relational model best reflects Object-Oriented Programming applications. If you are building out a piece of software that is following the Object-Oriented Programming approach, it would be best to store each object's data as its own table with the database. For example, if you're programming a car, you might have an Object for the tires, axles, engine, seats, paint, etc. The tires attach to the axles which spin because of the engine, and so on. Representing each of these objects as their own table, with a link between the appropriate entities (tire to axle, axle to

#### engine, etc) would be an optimal way to neatly store data and understand how the car works.

#### undefined

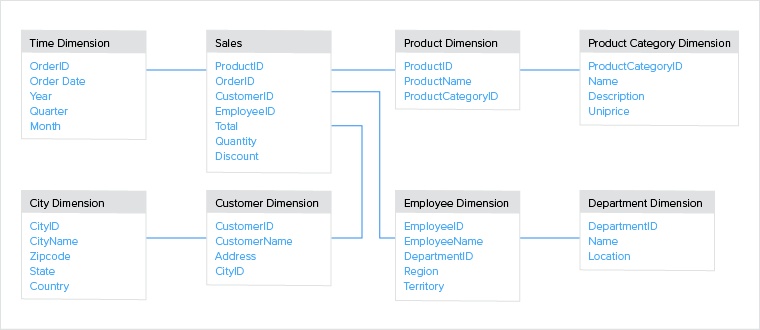
#### Star Schema: The star schema and snowflake schema are for analysing large datasets

#### The star schema is a different way of organizing your data. It is an excellent design approach for storing and analysing massive amounts of data, and it relies on the usage of "facts" and "dimensions." A "fact" is a numerical data point that drives business processes, and a "dimension" is a description of that fact. Using car sales numbers, for example, the "fact" table would contain information about the number of units sold, and a corresponding "dimensional" table would have the colours of those cars.

#### undefined

#### Snowflake Schema: As the star schema is an adaptation of the relational database model, the [snowflake](https://www.integrate.io/blog/top-etl-snowflake-tools/) schema is an adaptation of the star schema. Its name derives from how one would depict an ERD (entity-relation diagram) of a snowflake schema: you guessed it, it starts to look like a snowflake.

#### As with the star schema, the snowflake schema has a central fact table that stores the main data points and references to its dimensional tables. Unlike the star schema, the snowflake schema dimensional tables can have their own dimensional tables, thus expanding how descriptive a dimension can be.



## Top performance tuning tips for relational databases

Although newer relational databases and faster hardware run most SQL queries with a significantly small response time, there is always room for improvement. These tips that every developer/DBA should consider when designing their database or writing SQL scripts. It also talks about some common mistakes developers typically make and how to avoid them.

#### 1. Update Database statistics

#### 2. Create optimized indexes

#### 3. Avoid functions on RHS of the operator

#### 4. Specify optimizer hints in SELECT

#### 5. Use EXPLAIN

#### 6. Avoid foreign key constraints

#### 7. Two heads are better than one

#### 8. Select limited data

#### 9. Drop indexes before loading data

## Data abstraction levels

1. **Physical level:** This is the lowest level of abstraction, and it describes how data is stored.
2. **Logical level:** The next level of abstraction is logical; it describes what type of data is stored in a database and what is the relationship between these data.
3. **View level:** The highest level of abstraction and it describes the only entire database.

## Database Cache

A database cache is a system that caches (saves) results from a database, in order to return the result faster next time.

There are simplified two types of database cache:

**Internal cache:** It’s keeping things ready that it thinks you might need, based on usage patterns. You have no influence on this.

**Query cache:** If you make a query more than once (for instance for a user profile), the result is cached and returned from RAM instead. When the allotted RAM runs out, the least recently used query is deleted to make space for new ones. When the underlying data changes, either on a table or row/document level, depending on the database, the cache is cleared.

Another factor that is very important, and in some ways more important that the actual database caches, is File System cache. It means that the files that make up the database can be kept in RAM.

## Cache tables

You use cache tables to store data that you access frequently but that does not change often. A cache table can improve query performance by storing the data locally instead of accessing the data directly from the data source.

## Materialized view

In computing, a materialized view is a database object that contains the results of a query. For example, it may be a local copy of data located remotely or may be a subset of the rows and/or columns of a table or join result or may be a summary using an aggregate function.

**Why should I use Materialized Views?**

we use Materialized Views when we need speed over large amounts of data, because they often provide a performance boost over regular database tables or views, especially when we start making more complex queries that involve filtered selections, aggregation, or projections. When customers come to us looking for help optimizing their endpoint’s, Materialized Views are a common way we can grab back performance by cutting down on the number of bytes or rows read when the endpoint is called.

Materialized Views give us an advantage over regular tables in 3 specific categories:

* Speed,
* Simplicity, and
* Consistency.

**When should I not use Materialized Views?**

1. They take up additional storage space. Data storage costs money. Materialized Views will often contain orders of magnitude less rows than the original data tables, but this is still something to keep in mind.
2. Not all SQL queries can be materialized. Depending on the database you use, you may find problems materializing queries with UNION or JOIN, for example, and you’ll need to avoid clauses like LIMIT or ORDER BY.
3. They are not (necessarily) kept up to date automatically. Again, depending on your database, you may need to explicitly refresh data in your Materialized View as new data is ingested into your base tables.

# NoSQL Databases

## Introduction to NoSQL

NoSQL is a modern database management system that stores information in JSON documents instead of the columns and rows used by relational databases. It can deliver data for applications in ways that make development easier and more robust. NoSQL databases are built from the ground up to be flexible, scalable, and capable of rapidly responding to the data management demands of modern businesses. NoSQL databases are used in real-time web applications and huge data. NoSQL systems are also sometimes called Not only SQL to emphasize that they may support SQL-like query languages or sit alongside SQL databases.

I. **Supporting SQL and NoSQL developers:**

Traditional relational systems manage tabular data and return it as rows and columns. NoSQL databases can do this without forcing application developers into using a static schema that has to be reworked every time there is a change. Instead, NoSQL databases give developers the flexibility they need to help them excel at their work. NoSQL systems hold hierarchical JSON data but return it to the application as full or partial JSON data structures, full-text search matches, SQL query results, key-based values, or even big data analytics systems. This convergence of the best of relational and the best of modern NoSQL simplifies the information architecture of enterprises and helps developers deliver applications more efficiently with familiar concepts and tooling, without needing to learn a dozen different platforms.

II. **Scaling beyond SQL databases:**

To operate at scale, NoSQL systems approach cluster-based computing with efficient, automatic cluster management to keep data synchronized and flowing at high speeds. For example, teams are now expected to build enterprise data management infrastructure that includes the following characteristics.

• Support large numbers of concurrent users (tens of thousands, perhaps millions)

• Deliver highly responsive experiences to a globally distributed base of users

• Be always available – no downtime

• Handle semi- and unstructured data

## Why NoSQL databases?

Relational databases were born in the era of mainframes and back-office business applications – long before the internet, the cloud, big data, mobile, etc. NoSQL databases emerged as a result of the exponential growth of the internet and the rise of web applications. Google Bigtable research was released in 2006, as well as an Amazon Dynamo research paper in 2007. Efficient distributed key-value (KV) engines were essential to this evolutionary step and have propelled the technology much further. New databases were engineered to meet the next generation of enterprise requirements, which companies like Couchbase have taken even further to meet needs going into the future – the need to develop with agility and to operate at any scale. Agility means providing flexible schemas, APIs, robust SQL-based querying, text search, analytics, and more. Scalability allows data to grow without sacrificing performance.

The popularity of NoSQL databases over the last decade or so has been driven by an explosion of data. Before what’s commonly described as ‘the big data revolution’, relational databases were the norm – these are databases that contain structured data. Structured data can only be structured if it is based on an existing schema that defines the relationships (hence relational) between the data inside the database. However, with the vast quantities of data that are now available to just about every business with an internet connection, relational databases simply aren’t equipped to handle the complexity and scale of large datasets.

## Why not SQL databases?

This is for a couple of reasons. The defined schemas that are a necessary component of every relational database will not only undermine the richness and integrity of the data you’re working with, but relational databases are also hard to scale. Relational databases can only scale vertically, not horizontally. That’s fine to a certain extent, but when you start getting high volumes of data – such as when millions of people use a web application, for example – things get really slow, and you need more processing power. You can do this by upgrading your hardware, but that isn’t sustainable. By scaling out, as you can with NoSQL databases, you can use a distributed network of computers to handle data. That gives you more speed and more flexibility. This isn’t to say that relational and SQL databases have had their day. They still fulfil many use cases. The only difference is that NoSQL can offers a level of far greater power and control for data intensive use cases. Indeed, using a NoSQL database when SQL will do is only going to add more complexity to something that just doesn’t need it.

## Relational vs. NoSQL: The fundamental differences.

1. **Data Model:**RDBMS databases are used for normalized structured (tabular) data strictly adhering to a relational schema. NoSQL datastores are used for non-relational data, e.g. key-value, document tree, graph.
2. **Transaction Guarantees:**All RDBMS databases support ACID transactions, but most NoSQL datastores offer BASE transactions.
3. **CAP Trade-offs:**RDBMS databases prioritize strong consistency over everything else. But NoSQL datastores typically prioritize availability and partition tolerance (horizontal scale) and offer only [eventual consistency](https://en.wikipedia.org/wiki/Eventual_consistency).
4. **Performance**: RDBMS are designed for fast transactions updating multiple rows across tables with complex integrity constraints. SQL queries are expressive and declarative. You can focus on what a transaction should accomplish. RDBMS will figure out how to do it. It will optimize your query using relational algebra and find the best execution plan. NoSQL datastores are designed for efficiently handling a lot more data than RDBMS. There are no relational constraints on the data, and it does not need to be even tabular. NoSQL offers performance at a higher scale by typically giving up strong consistency. Data access is mostly through REST APIs. NoSQL query languages (such as GraphQL) are not yet as mature as SQL in design and optimizations. So you need to take care of both what and how to do it efficiently.
5. **Scalability:** RDBMS scale vertically.You need to upgrade hardware (more powerful CPU, higher storage capacity) to handle the increasing load. NoSQL datastores scale horizontally. NoSQL is better in handling partitioned data, so you can scale by adding more machines.

## Compare Various features of RDBMS and NoSQL databases.

The following table provides a high-level comparison of the characteristics of relational and NoSQL databases.

|  |  |
| --- | --- |
| **Relational databases** | **NoSQL databases** |
| Moderate incoming data velocity | High incoming data velocity |
| Data comes from one or a few locations | Data comes from many locations |
| Primarily structured data | Structured and semi-unstructured data |
| Nested or complex transactions | Simple transactions |
| Protect uptime via failover or log shipping | Protect uptime via architecture |
| High availability | Continuous availability |
| Deploy an app to one server | Deploy an app to many servers |
| Primarily write data to one location | Write data to any location |
| Primary concern is scaling reads | Scale writes and reads |
| Scale up for more users and data | Scale out for more users and data |
| Maintain data volumes with purges | High data volumes retained forever |

## The importance of cap theorem in the context of NoSQL.

It is very important to understand the CAP theorem as it makes the basics of choosing any NoSQL database based on the requirements. It states that in networked shared-data systems or distributed systems, we can only achieve at most two out of three guarantees for a database: Consistency, Availability and Partition Tolerance.

Diagram

Description automatically generated

1. **Consistency:** Means that all clients see the same data at the same time, no matter which node they connect to in a distributed system. To achieve consistency, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed successful.
2. **Availability:** Means that every non-failing node returns a response for all read and write requests in a reasonable amount of time, even if one or more nodes are down. Another way to state this — all working nodes in the distributed system return a valid response for any request, without failing or exception.
3. **Partition Tolerance:** Means that the system continues to operate despite arbitrary message loss or failure of part of the system. In other words, even if there is a network outage in the data centre and some of the computers are unreachable, still the system continues to perform. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.
   * + 1. **NoSQL databases follow the BASE properties:**
4. Basically Available
5. Soft-State
6. Eventual Consistency

## How to choose between SQL and NoSQL databases.

In the past, most organizations opted for SQL databases because of their ability to protect data and ensure its integrity. But the rise of the internet and cloud technologies—and the proliferation of data that went with them—has caused many organizations to turn to NoSQL databases, in large part because they can better handle the abundance of unstructured and semi-structured data. SQL or NoSQL, both offer advantages and disadvantages, but they differ in how they’re built, how they store data, and how applications access them. And it’s only by understanding these differences can an organization make an informed decision about which type will best suit their workloads now and in the foreseeable future.

If you are evaluating whether a NoSQL database is appropriate for your production environment, you should address the following technical, business, and deployment considerations.

* + - 1. **Technical considerations:**

Ask the following technical questions before going for a NoSQL database.

1. Can the database serve as the primary data source for the online application?
2. Does the database have features that prevent the loss of critical data? Are writes durable in nature by default so that the data is safe?
3. Is the database fault-tolerant, and is it capable of providing continuous availability?
4. Can the database easily replicate data located in the same data centre, across multiple data centres, and across different cloud availability zones?
5. Does the database offer read/write anywhere capabilities? (Can the system write to and read from any node in the cluster?)
6. Does the database provide a robust set of security features?
7. Does the database support backup and recovery procedures that are easy to create and manage?
8. Does the database require special caching layers?
9. Is the database capable of managing big data and delivering high performance regardless of data size?
10. Does the database offer linear scalability for adding new nodes?
11. Can new nodes be added and removed online without impacting your business?
12. Does the database support key platforms and developer languages?
13. Does the database provide a query language that is similar to SQL?
14. Can the database run on commodity hardware with no special requirements?
15. Is the database easy to implement and maintain for large deployments?
16. Does the database offer data compression that results in significant storage savings?
17. Is it easy to run analytic operations on the database?
18. Can the database easily interface with and support modern data warehouses or data lakes that use Hadoop?
19. Is it easy to carry out search operations and functions directly on the NoSQL database?
20. Can the database isolate the online, analytic, and search workloads within a single application?
21. Does the database have solid command-line and visual tools for development, administration, and performance management?
    * + 1. **Business Consideration:**

The following list presents business-related questions that you should ask before deciding to use a NoSQL database.

1. Does a commercial entity back the database?
2. If so, does the commercial entity provide 24x7 enterprise-level support and services?
3. Does the database have professional online documentation?
4. Does the database have customers across a wide range of industries that use the product in critical production environments?
5. Does the database have an attractive pricing structure?
6. If the database is open source, does it have a thriving open-source community?
   * + 1. **Augmentation considerations:**

Some organizations choose to augment an existing system by adding a NoSQL component. This often happens with applications that have outgrown a relational database due to scale problems, the need for better availability, or other issues. Parts of the existing system continue to use the existing RDBMS, while the developers modify other application components to use the NoSQL database. If a system is too costly or is breaking due to increased user concurrency, data velocity, or data volume from cloud applications, you can replace it fully with a NoSQL database.

* + - 1. **Deployment considerations:**

You can use a NoSQL database as part of a new application, to augment an existing application, or to fully replace an application. Before choosing a database, consider your deployment needs

## What are the different NoSQL Databases?

**Diagram

Description automatically generated**

1. **Document databases.** Data is stored as documents in a format such as JSON or XML, with each document assigned its own unique key, but without being limited to predefined fields or elements. Example products include MongoDB, CouchDB, and BaseX.
2. **Graph databases.** Data is stored in graph structures that define how the data is interconnected, essentially treating the relationships between the data as important as the data itself. Product examples include Neo4J and VertexDB.
3. **Key-value databases.** Data is stored in a key-value structure that associates unique identifiers with specific data blobs, making it possible to store just about any type of data in whatever form it’s delivered. Example products include Redis, Berkeley DB, Apache Cassandra and Amazon DynamoDB.
4. **Column-oriented databases.** Data is stored as strongly typed columns rather than rows, making it possible to query and aggregate large volumes of data very quickly. These types of databases go by other names as well, such as column-store, wide-column store, and column-family. Example products include HBase, Bigtable, and Cloudera.

Each type of NoSQL database offers advantages and disadvantages, depending on the workloads an organization is trying to support. In addition, not all NoSQL products fit neatly into each of these categories. For example, Azure Cosmos DB provides five APIs that make the service more universal. These include the SQL API, API for MongoDB, Cassandra API, Gremlin API, and Table API.

## When to use the which type of NoSQL databases?

Let’s take a look at these four models, how they’re different from one another, and some examples of the product options in each.

Diagram, venn diagram

Description automatically generated

* + - 1. **When should you use a key/value pair-based NoSQL DBMS?**

Key/value pair-based NoSQL databases are the most basic type of NoSQL database. They’re useful for storing basic information, like details about a customer.

* + - 1. **Which key/value pair-based DBMS should you use?**

There are number of different key/value pair databases. The most popular is Redis. Redis is incredibly fast and very flexible in terms of the languages and tools it can be used with. Other than Redis, other options include Memcached. As well as those, there are several other multi-model options such as Amazon DynamoDB, Microsoft’s Cosmos DB, and OrientDB.

* + - 1. **Column-based NoSQL database management systems**

Column-based databases separate data into discrete columns. Instead of using rows – whereby the row ID is the main key – column-based database systems flip things around to make the data the main key.

By using columns, you can gain much greater speed when querying data. Although it’s true that querying a whole row of data would take longer in a column-based DBMS, the use cases for column-based databases mean you probably won’t be doing this. Instead, you’ll be querying a specific part of the data rather than the whole row.

* + - 1. **When should you use a column-based NoSQL DBMS?**

Column-based systems are most appropriate for big data and instances where data is relatively simple and consistent (they don’t particularly handle volatility that well).

* + - 1. **Which column-based NoSQL DBMS should you use?**

The most popular column-based DBMS is Cassandra. The software prizes itself on its performance, boasting 100% availability thanks to lacking a single point of failure, and offering impressive scalability at a good price. Cassandra’s popularity speaks for itself – Cassandra is [used by 40% of the Fortune 100](https://db-engines.com/en/system/Cassandra).There are other options available, such as HBase and Cosmos DB.

* + - 1. **Document-oriented NoSQL database management systems**

Document-oriented NoSQL systems are very similar to key/value pair database management systems. The only difference is that the value that is paired with a key is stored as a document. Each document is self-contained, which means no schema is required – giving a significant degree of flexibility over the data you have. For software developers, this is essential – it’s for this reason that document-oriented databases such as MongoDB and CouchDB are useful components of the [full-stack](https://hub.packtpub.com/devops-engineering-and-full-stack-development/) development tool chain. Some search platforms such as [Elasticsearch](https://hub.packtpub.com/how-does-elasticsearch-work-tutorial/) use mechanisms similar to standard document-oriented systems – so they could be considered part of the same family of database management systems.

* + - 1. **When should you use a document-oriented DBMS?**

Document-oriented databases can help power many different types of websites and applications – from stores to content systems. However, the flexibility of document-oriented systems means they are not built for complex queries.

* + - 1. **Which document-oriented DBMS should you use?**

The leader in this space is, MongoDB. With an amazing 40 million downloads (and apparently 30,000 more every single day), it’s clear that MongoDB is a cornerstone of the NoSQL database revolution. There are other options as well as MongoDB – these include CouchDB, Couchbase, DynamoDB and Cosmos DB.

* + - 1. **Graph-based NoSQL database management systems**

The final type of NoSQL database is graph-based. The notable distinction about graph-based NoSQL databases is that they contain the relationships between different data. Subsequently, graph databases look quite different to any of the other databases above – they store data as nodes, with the ‘edges’ of the nodes describing their relationship to other nodes.

Graph databases, compared to relational databases, are multidimensional in nature. They display not just basic relationships between tables and data, but more complex and multifaceted ones.

* + - 1. **When should you use a graph database?**

Because graph databases contain the relationships between a set of data (customers, products, price etc.) they can be used to build and model networks. This makes graph databases extremely useful for applications ranging from [fraud detection](https://neo4j.com/use-cases/fraud-detection/) to smart homes to search.

* + - 1. **Which graph database should you use?**

The world’s most popular graph database is Neo4j. Its purpose built for data sets that contain strong relationships and connections. Widely used in the industry in companies such as eBay and Walmart, it has established its reputation as one of the world’s best NoSQL database products.

* + - 1. **SQL / NoSQL Across AWS, Azure & GCP.**

**A picture containing diagram

Description automatically generated**

# Non-Functional Scenarios

## ETL and ELT Processes

Extract, transform, and load (ETL) is a [data integration](https://rivery.io/integrations/) methodology that extracts raw data from sources, transforms the data on a secondary processing server, and then loads the data into a target database.

The extracted data only moves from the processing server to the data warehouse once it has been successfully transformed.

Diagram

Description automatically generated with low confidence

ELT loads raw data directly into a target data warehouse, instead of moving it to a processing server for transformation. With ELT, [data cleansing](https://dataladder.com/data-cleaning-guide/), enrichment, and transformation all occur inside the data warehouse itself. Raw data is stored indefinitely in the data warehouse, allowing for multiple transformations.

Cloud data warehouses such as [Snowflake](https://www.snowflake.com/), [Amazon Redshift](https://aws.amazon.com/redshift/), [Google BigQuery](https://cloud.google.com/bigquery/), and [Microsoft Azure](https://azure.microsoft.com/en-us/) all have the digital infrastructure, in terms of storage and processing power, to facilitate raw data repositories and in-app transformations.

Although ELT is not used universally, the method is becoming more popular as companies adopt cloud infrastructure.

Diagram

Description automatically generated

## ETL vs ELT: Side-by-Side Comparison

|  |  |  |
| --- | --- | --- |
| Category | ETL | ELT |
| Definition | Data is extracted from a source system, transformed on a secondary processing server, and loaded into a destination system. | Data is extracted from a source system, loaded into a destination system, and transformed inside the destination system. |
| Extract | Raw data is extracted using API connectors. | Raw data is extracted using API connectors. |
| Transform | Raw data is transformed on a processing server. | Raw data is transformed inside the target system. |
| Load | Transformed data is loaded into a destination system. | Raw data is loaded directly into the target system. |
| Speed | ETL is a time-intensive process; data is transformed before loading into a destination system. | ELT is faster by comparison; data is loaded directly into a destination system, and transformed in-parallel. |
| Code-Based Transformations | Performed on secondary server. Best for compute-intensive transformations & pre-cleansing. | Transformations performed in-database; simultaneous load & transform; speed & efficiency. |
| Maturity | Modern ETL has existed for 20+ years; its practices & protocols are well-known and documented. | ELT is a newer form of data integration; less documentation & experience. |
| Privacy | Pre-load transformation can eliminate PII (helps for HIPPA). | Direct loading of data requires more privacy safeguards. |
| Maintenance | Secondary processing server adds to the maintenance burden. | With fewer systems, the maintenance burden is reduced. |
| Costs | Separate servers can create cost issues. | Simplified data stack costs less. |
| Requeries | Data is transformed before entering destination system; therefore raw data cannot be requeried. | Raw data is loaded directly into destination system and can be requeried endlessly. |
| Data Lake Compatibility | No, ETL does not have data lake compatibility. | Yes, ELT does have data lake compatibility. |
| Data Output | Structured (typically). | Structured, semi-structured, unstructured. |
| Data Volume | Ideal for small data sets with complicated transformation requirements. | Ideal for large datasets that require speed & efficiency. |

## Data repository (Data Warehouse Vs Data Lake Vs Lake house)

A picture containing timeline

Description automatically generated

### Key Technology Enabling the Data Lakehouse

There are a few key technology advancements that have enabled the data lakehouse:

* metadata layers for data lakes
* new query engine designs providing high-performance SQL execution on data lakes
* optimized access for data science and machine learning tools.
* Architecture of Data Lake House

Graphical user interface, application

Description automatically generated

## Data repository (Data Warehouse Vs Data Lake Vs Lake house)

Table

Description automatically generated

## How to Find the right data warehouse for your business

Before you pick a solution, don’t forget to consider:

* The type and amount of data you want to store
* How dynamically you need it to scale
* How fast you need your queries
* Whether you want manual or automatic maintenance
* The compatibility of the data warehouse with your existing tech stack
* The cost

## Use cases of data lake house

* Reliability 🡪 Keeping the data lake and data warehouse Consistent.
* Data Staleness 🡪 Data in warehouse is older.
* Limited Support on advanced Analytics --> Top ML Systems don't work well on data warehouses.
* Total Cost 🡪 Extra Cost for data copied to warehouse.

16. References

<https://www.bmc.com/blogs/data-architecture/>

<https://www.cio.com/article/190852/what-is-a-data-architect-its-data-framework-visionary.html#:~:text=Data%20architect%20role&text=The%20data%20architect%20is%20responsible,%2C%20control%2C%20and%20purge%20data>

<https://www.freelancermap.com/blog/what-does-integration-developer-do/>

<https://www.coursera.org/articles/what-does-a-data-engineer-do-and-how-do-i-become-one>

<https://www.alibabacloud.com/blog/data-lake-concepts-characteristics-architecture-and-case-studies_596910>

<https://www.yash.com/blog/data-lake-architectural-components/>

<https://www.alation.com/blog/what-is-a-data-catalog/>

<https://www.atscale.com/blog/the-six-modern-principles-of-modern-data-architecture/>

<https://www.alibabacloud.com/blog/data-lake-concepts-characteristics-architecture-and-case-studies_596910>

https://www.altexsoft.com/blog/data-pipeline-components-and-types/

Specify any requirements for keeping the application software or database software up to date:

* How frequently are the application software updates likely to be released and is there a requirement to be current to a certain level (e.g. major release – 1)?
* Is there a requirement for the database version to be current to a certain level (e.g. migrate to the latest version within ‘x’ months of release)?
* How often are application patches likely to be released and will they be any requirement concerning their application?