NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE | NTU Academy for Professional and Continuing Education

(SCTP) Advanced Professional Certificate

Data Science and AI

2.2 Data Architecture

Module Overview

- 2.1 Introduction to Big Data and Data Engineering
- 2.2 Data Architecture
- 2.3 Data Encoding and Data Flow
- 2.4 Data Extraction and Web Scraping
- 2.5 Data Warehouse
- 2.6 Data Pipelines
- 2.7 Testing and Data Orchestration
- 2.8 Out of Core/Memory Processing
- 2.9 Big Data Ecosystem and Batch Processing
- 2.10 Event Streaming and Stream Processing

Learning Objectives

Learners will be able to:

- Consider trade-offs between different data architecture types.
- Perform create and read operations on Redis.
- Fetch data from Google Cloud Storage.

AGENDA | Data Architecture

MODERN PRINCIPLES

Data as Product | Data Mesh | Cloud-Native & IaC | Event-Driven Architecture | Interoperability | Metadata-Driven Architecture | Resilience | Observability & Monitoring

ARCHITECTURE PRINCIPLES

Shared assets | Accessible | Secure | Universal understanding | Curated | Optimal Movement

ARCHITECTURE CONCEPTS

Domain | Services | Scalability | Elasticity | Availability | Reliability | Maintainability | Scaling | Distributed Systems | CAP Theorem

ARCHITECTURE TYPES

Data Warehouse | Data Lake | Data Lakehouse | Modern Data Stack | Lambda Architecture | Kappa Architecture | Dataflow Model | Data Mesh

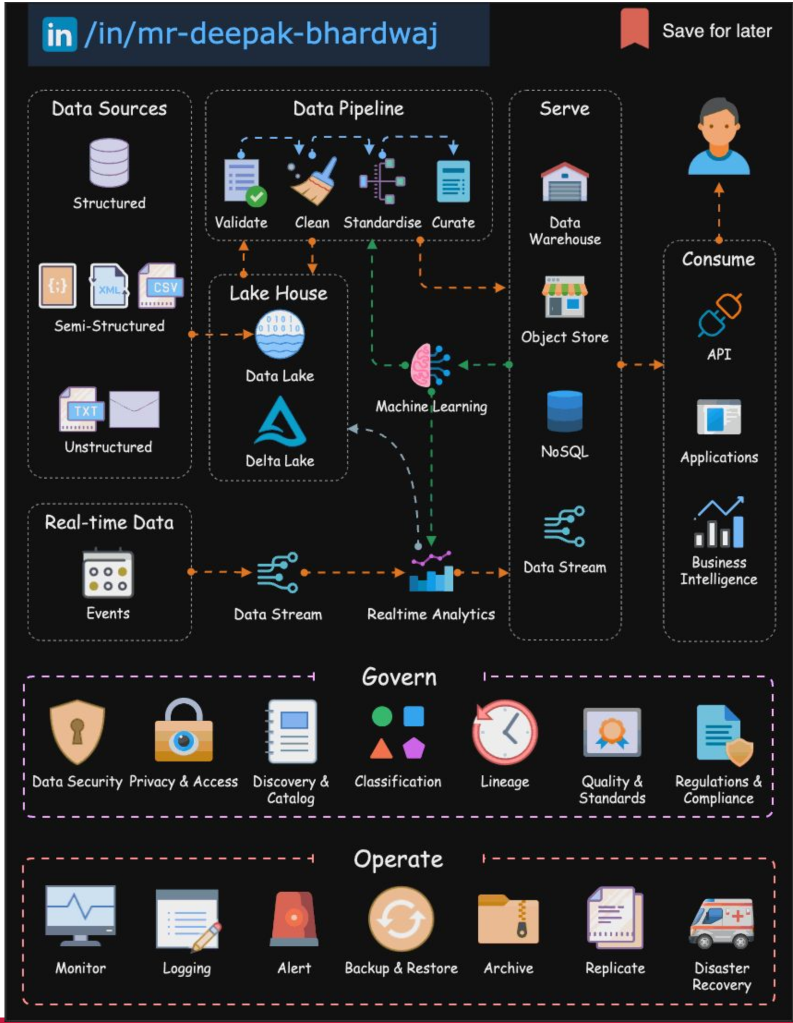
Introduction to Data Architecture

<u>Definition 1</u>

Design of systems to support the evolving **data needs** of an enterprise, achieved by *flexible* and *reversible decisions* reached through a careful evaluation of trade-offs.

<u>Definition 2</u>

Identifying the **data needs** of the enterprise (regardless of structure) and designing and maintaining the *master blueprints* to meet those needs. Using *master blueprints* to guide data integration, control data assets, and align data investments with business strategy.



Architecture Concepts

Domains

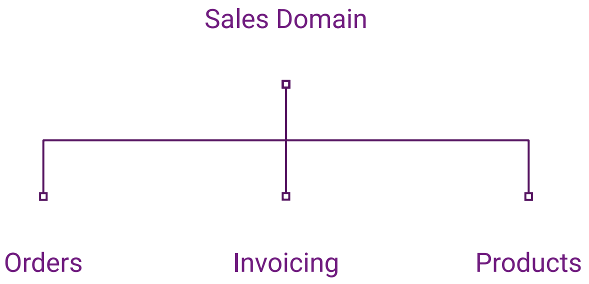
A domain is the real-world subject area for which you're architecting.

Services

A service is a set of functionality whose goal is to accomplish a task.

A domain can contain multiple services.

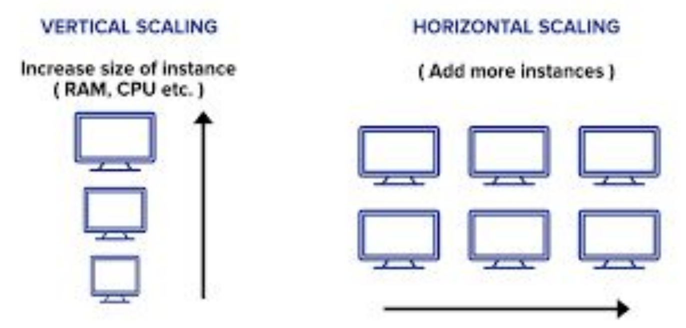
- When thinking about what constitutes a domain, focus on what the domain represents in the real world and work backward.



Data Architecture Concepts

Scalability

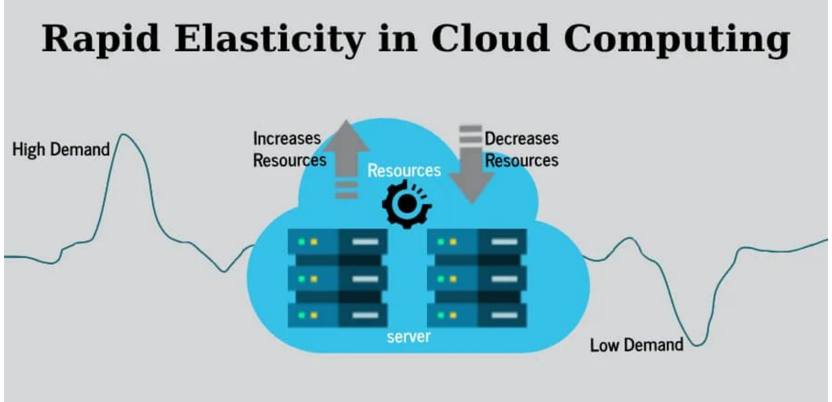
Allows us to *increase the capacity* of a system to improve performance and handle the demand. For example, we might want to scale a system to handle a high rate of queries or process a huge data set.



Elasticity

The ability of a scalable system to *scale dynamically*; a highly elastic system can automatically scale up and down based on the current workload.

Scaling up is critical as demand increases, while scaling down saves money in a cloud environment. Modern systems sometimes scale to zero, meaning they can automatically shut down when idle.



https://www.educba.com/rapid-elasticity-in-cloud-computing/

Data Architecture Concepts

Availability

The percentage of time an IT service or component is in an operable state.

Reliability

The system should continue to work correctly (performing the correct function at the desired level of performance, also known as *fault-tolerant* or *resilient*) even in the face of adversity (hardware or software *faults*, and even human error).

Maintainability

Over time, many different people will work on the system (maintaining current behavior and adapting the system to new use cases), and they should all be able to work on it productively.

Distributed Systems

Distributed systems enhance scalability, availability, and reliability via horizontal scaling, relying on data replication and partitioning (sharding). Node structures include:

- **Leader-follower (master-slave):** Single leader distributes reads/writes to followers. Clients read from leader/followers, write to leader.
- **Multi-leader:** Multiple leaders accept writes, forwarding changes to other nodes.
- **Leaderless:** Any node can accept writes from clients.

Prevalent in modern data technologies like NoSQL databases, cloud data warehouses, and object storage systems.

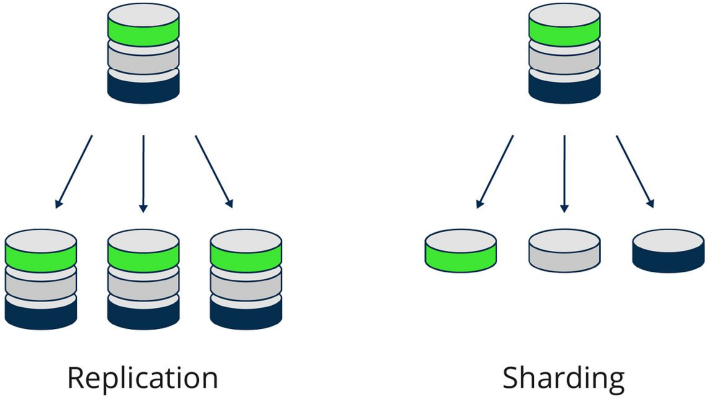
Distributed Systems

Replication

Creates multiple *copies* of the same data and distributing those copies across different servers or nodes. Each copy is referred to as a replica. The primary goal of replication is to enhance data *availability*, fault tolerance (*reliability*), and read *scalability*.

Partitioning (Sharding)

Partitions a database into smaller, more manageable subsets called *shards*. Each shard contains a portion of the data. The goal of sharding is to distribute the data and workload across multiple servers, improving write *scalability* and large datasets management.

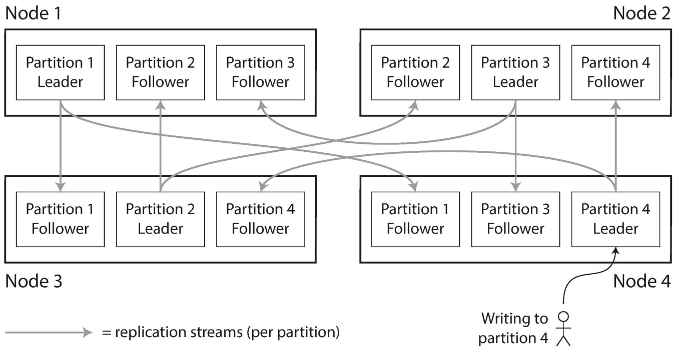


Distributed Systems

Partitioning is usually combined with replication

- Copies of each partition are stored on multiple nodes.
  - Each record belongs to exactly one partition but may be stored on several nodes for fault tolerance.
- A node may store more than one partition.

If a **leader–follower** model is used, each partition's leader is assigned to one node, and its followers are assigned to other nodes. Each node may be the leader for some partitions and a follower for other partitions.



CAP Theorem

The CAP theorem states that a distributed database can only guarantee two out of the three properties:

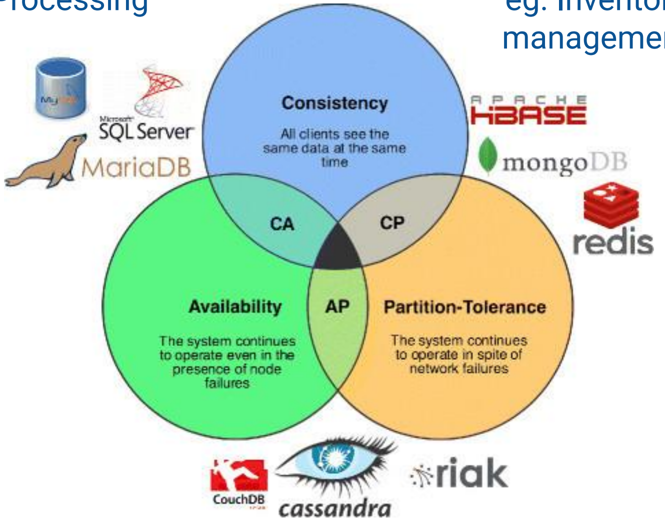
Consistency - All nodes see the same data at the same time, ensuring coherent and accurate reads/writes across the system.

Availability - Every request receives a non-error response, with the system remaining operational despite failures or network partitions.

Partition tolerance - The system continues functioning when network partitions occur, with some nodes unable to communicate with others due to failures or delays.

Choose when network partitions are rare, eg. Financial Transaction Processing

Prioritize data consistency over availability, eg. Inventory management



Favor availability and partition tolerance, eg. Recommendation system

Discussion - Architecture Concepts

Instructions

Go to

www.menti.com

Enter the code

1244 9057

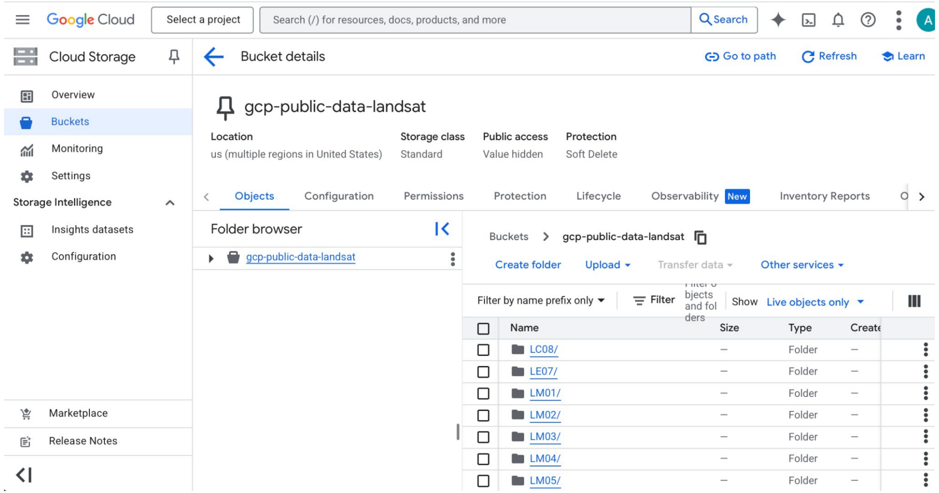
Or use QR code

Types of Data Architecture

Category	Architecture	Tool Examples
Data at Rest	Object Storage	Amazon S3 (AWS), Google Cloud Storage
	Data Lake	AWS Lake Formation (built on Amazon S3), Cloudera
	Data Warehouse	Amazon Redshift, Google BigQuery
	Data Lakehouse	Databricks Lakehouse Platform, Apache Iceberg
Data in Motion	Data Pipelines & Orchestration	Apache Airflow, AWS Glue
	Dataflow Model	Apache Beam, Google Cloud Dataflow
	Lambda Architecture	Apache Spark Streaming, Apache Kafka
	Kappa Architecture	Apache Kafka, Apache Flink
Data Mesh	Data Mesh Platform	Amazon DataZone, Google Dataplex, Azure Purview

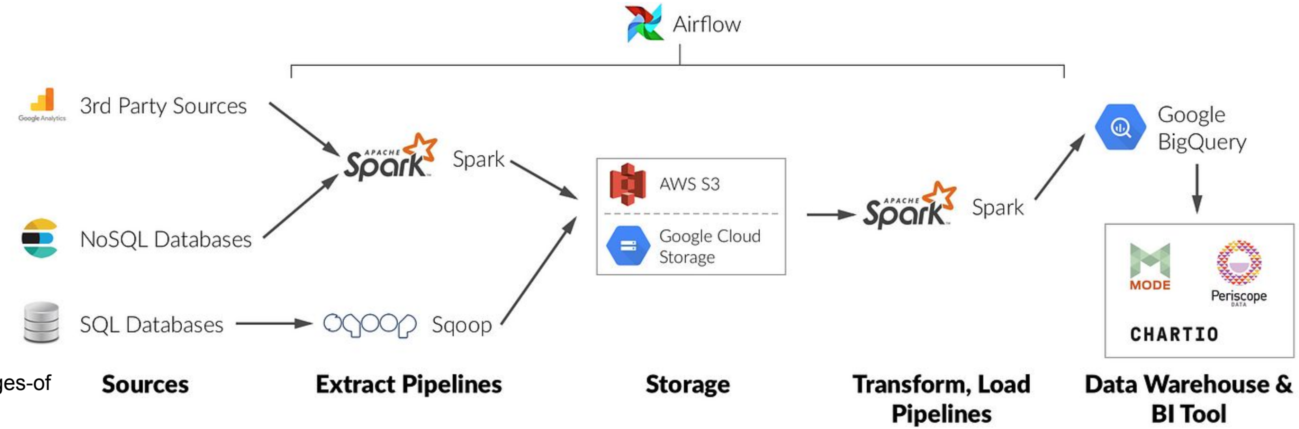
Object Storage

- Data architecture for large unstructured data, stored as distinct units with metadata and unique identifiers.
- Well-suited for vast unstructured data like images, videos, documents, backups. Designed for horizontal scalability by adding nodes.
- Shortcoming: Limited read/write performance for smaller files or high-throughput workloads. Limited advanced search, indexing capabilities compared to file systems/databases. Not ideal for in-place processing or complex analytics.
- Examples: AWS S3, Azure Blob Storage and Google Cloud Storage (GCS).
- [Demo]



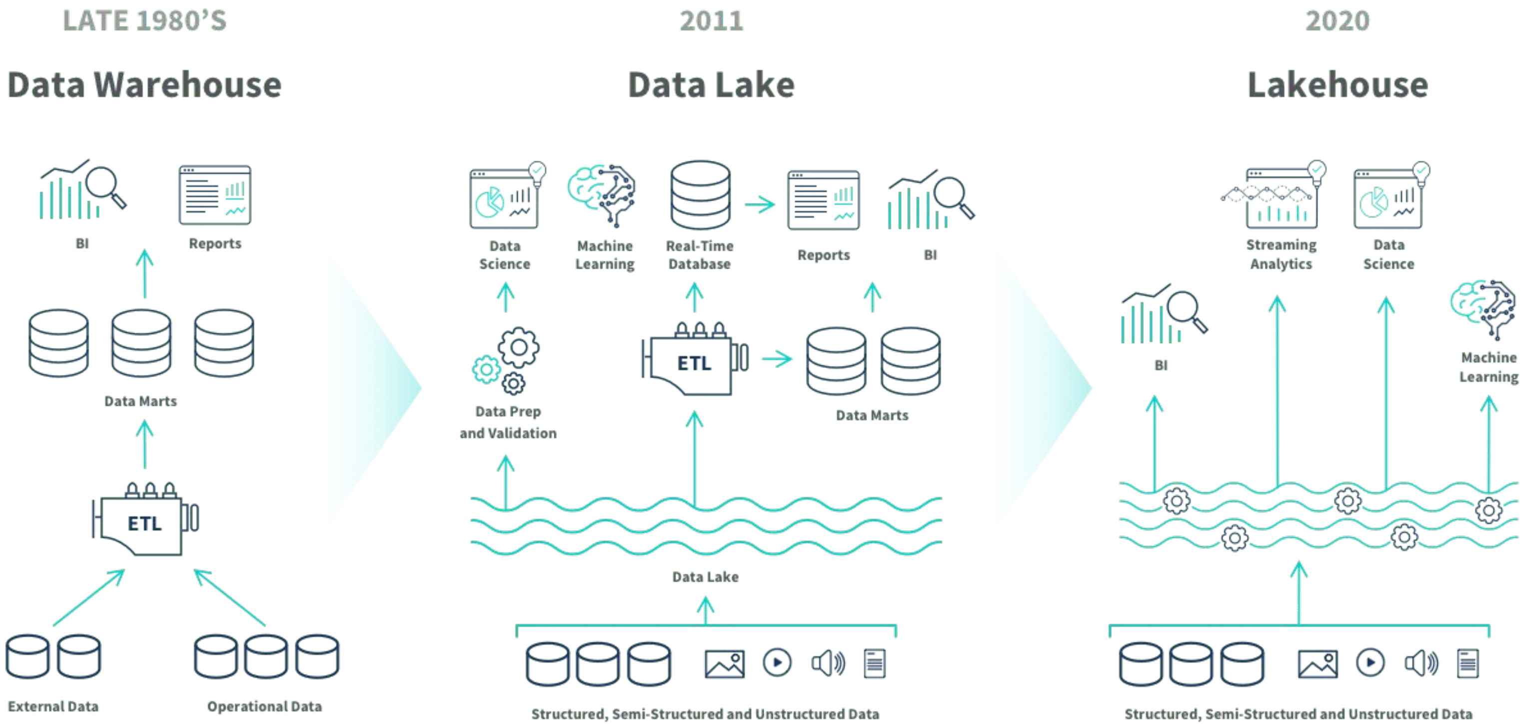
Data Pipelines

- Data pipelines are the paths data takes through the architecture once it's been created.
- The pipeline includes data collection, storage, access, cleaning, analysis, and presentation.
- It is often thought of as plumbing, where computational and storage resources (pipes) are sized appropriately for efficient data transmission.
- Building a pipeline requires continued maintenance and optimization as part of data infrastructure management.



https://medium.com/@natekupp/getting-started-the-3-stages-of-data-infrastructure-556dac82e825

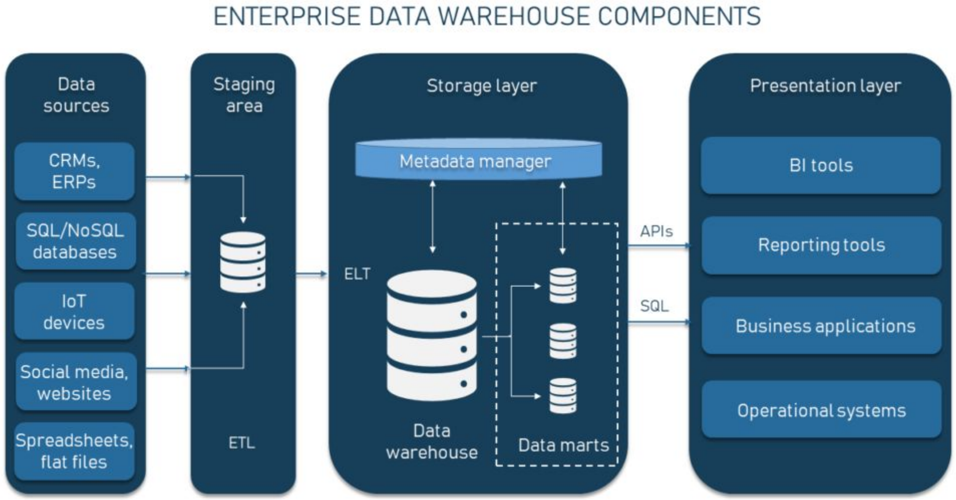
From Data Warehouse To Lakehouse



https://www.qlik.com/us/data-lake/data-lakehouse

Data Warehouse

- *Centralized* repository for integrated, historical data from multiple sources.
- Designed for complex queries, reporting, and analysis for business intelligence.
- Data is *extracted, transformed, and loaded* (ETL) to create a unified view.
- Optimized for read-heavy workloads using indexing, partitioning, etc.
- Data is often *denormalized* to reduce complex joins.
- Shortcoming: Complex, time-consuming, and costly to build and maintain.
- Examples: Snowflake, Amazon Redshift, Google BigQuery.



https://www.altexsoft.com/blog/enterprise-data-warehouse-concepts/

Data Lake

- Centralized repository for vast raw, unprocessed data in native formats - structured, semi-structured, unstructured (text, images, videos, sensor data, etc.).
- Unlike data warehouses, no predefined schema imposed. Data stored in various raw formats, well-suited for data exploration and experimentation.
- Shortcoming: Became a "data swamp" due to lack of schema management, data cataloging, and discovery tools leading to unmanageable data growth.
- One of the first and most prominent implementation is the Hadoop Distributed File System (HDFS). Data Lake can also be implemented on top of Object Storage such as AWS S3 and GCS.

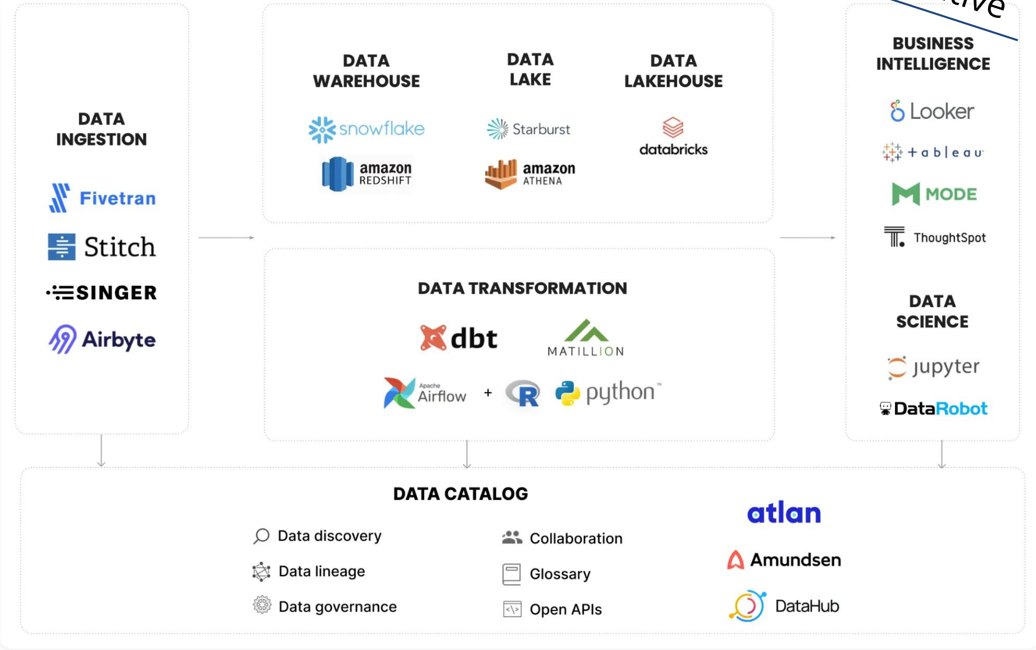
Data Lakehouse

- Conceived in response to shortcomings of data lakes.
- Incorporates data warehouse controls, management, and structures while housing data like a lake, supporting various query/transformation engines.
- Supports ACID transactions.
- Convergence with cloud data warehouses, which incorporate separate compute/storage, scalability, diverse data type support, and advanced processing integration.
- Examples: Databricks’ data lakehouse.

Modern Data Stack

- A popular analytics architecture.
- Uses modular, cloud-based, plug-and-play components for cost-effectiveness.
- Components: data pipelines, storage, transformation, data management/governance, monitoring, visualization, exploration.
- Reduces complexity through modularization.
- Key outcomes: self-service analytics/pipelines, agile data management, open source or clear pricing.

Illustrative

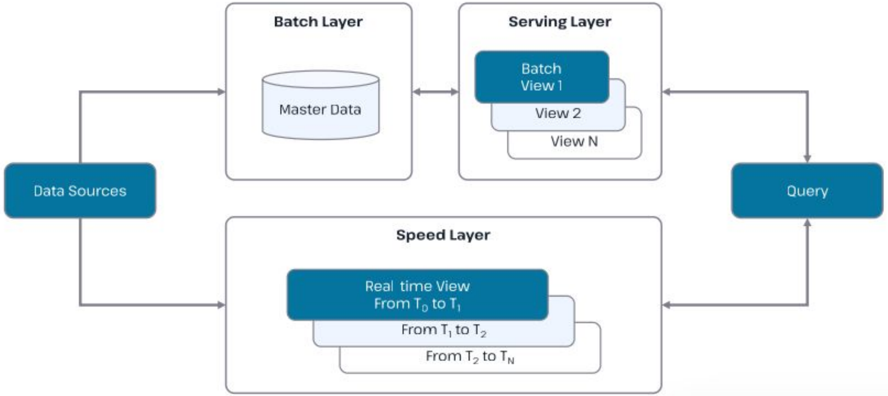


https://atlan.com/modern-data-stack-101/

Lambda Architecture

- Emerged to handle reconciling batch and streaming data.
- Separate systems for batch, streaming, and serving data.
- **Stream layer** focuses on low-latency processing, often using NoSQL.
- **Batch layer** processes data in systems like data warehouses, creating precomputed/aggregated views.
- **Serving layer** aggregates query results from both layers for combined view.

- **Shortcoming**: Managing multiple systems, codebases and reconciling code/data.

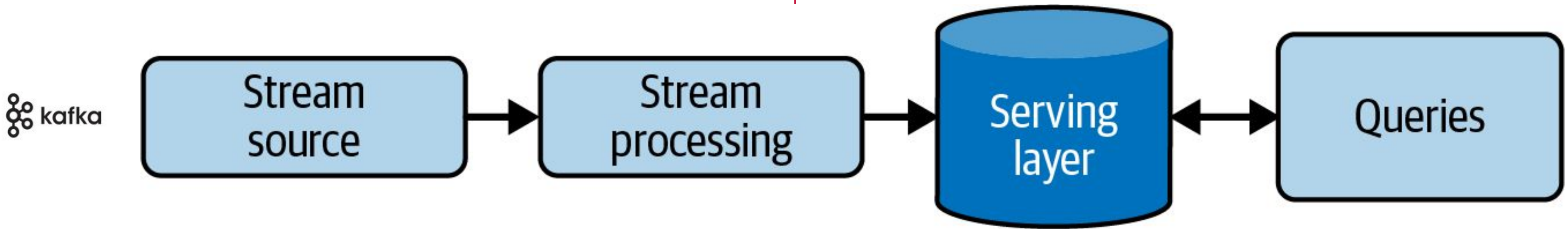


https://hazelcast.com/foundations/software-architecture/lambda-architecture/

Kappa Architecture

- Proposed as an alternative to Lambda architecture.
- Suggests using a stream-processing platform for all data tasks - ingestion, storage, serving.
- Enables true event-based architecture.

- Real-time and batch processing applied to same data by reading live event stream and replaying chunks.
- **Shortcoming**: Complicated and expensive in practice. Batch storage/processing remains more efficient and cost-effective for enormous historical datasets.

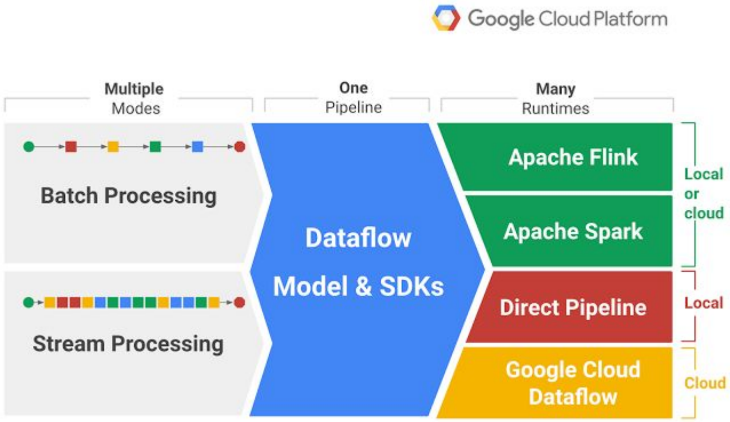


Dataflow Model

- Unifying batch and stream processing code paths is a central challenge.
- Kappa architecture uses unified queuing/storage, but requires different tools for real-time stats and batch aggregation.
- Google's Dataflow model and Apache Beam framework address this.
- In Dataflow, all data is events aggregated over various windows, with real-time streams unbounded and batches bounded.
- Different window types used for real-time aggregation.
- Real-time and batch processing occur in same system with similar code, following "batch as streaming" philosophy.
- Frameworks like Flink and Spark embraced similar approach.

POPULAR DATAFLOW PRODUCTS

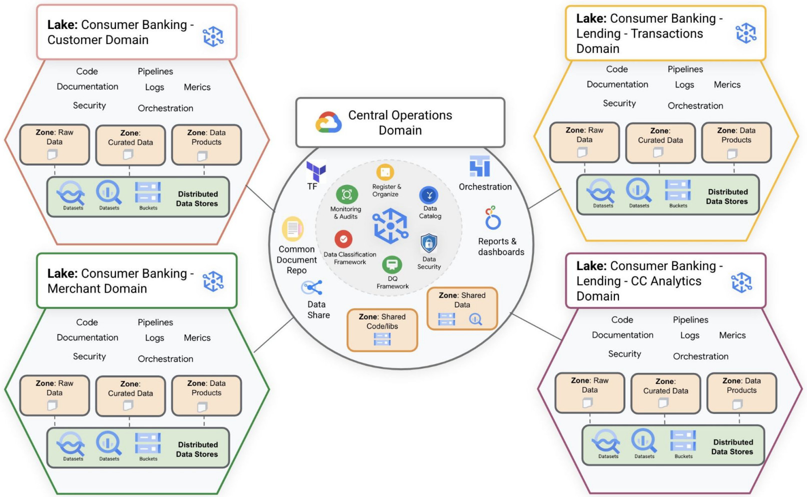
Product	Engine
Google Cloud Dataflow	Apache Beam
Apache Spark (Databricks)	Apache Spark
AWS Glue	Apache Spark
Azure Data Factory	Apache Spark



https://www.bigdatawire.com/2016/05/02/dataflow-tops-spark-benchmark-test/

Data Mesh

- A recent response to centralized data platforms like data lakes and warehouses, addressing their limitations.
- Applies *domain-driven design* concepts, tackling division between operational and analytical data.
- *Decentralization* is core - domains host and serve their own datasets, reversing data flow from centralized platforms.
- Four key components of the data mesh:
    - Domain-oriented decentralized data ownership and architecture
    - Data treated as a product
    - Self-serve data infrastructure functioning as a platform
    - Federated computational governance



https://www.infoworld.com/article/2338279/preview-google-cloud-dataplex-wows.html

Data Mesh - Google DataPlex

Dataplex overview

Intelligent data management and governance across distributed data

Unified metadata across distributed data

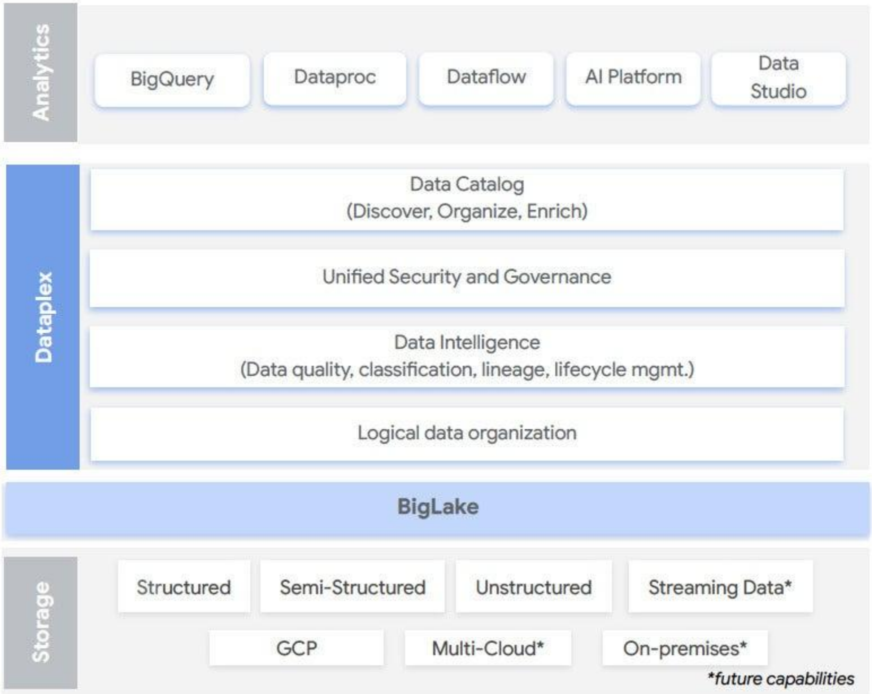
Automatic data discovery & metadata harvesting, enriched with business context. Logically unify and organize your data without any data movement.

Centralized Security & Governance

Central policy management, monitoring and auditing for data authorization, retention, and classification.

Intelligent Data Management

Built-in AI-driven intelligence with data classification, data quality, data lineage and lifecycle management.



https://www.infoworld.com/article/2338279/preview-google-cloud-dataplex-wows.html

Principles of Good Data Architecture

01. Data as a Shared Asset

Focus: Breaking down silos, enabling holistic company views
Benefits: Competitive advantage, 360-degree customer insights, cross-functional data correlation

02. User Interfaces for Data Consumption

Focus: Accessible interfaces (OLAP, SQL, APIs), data warehouses/lakes/marts
Benefits: Empowered users, scalable architecture, effective use of familiar tools

03. Security and Access Controls

Focus: Enforcing policies, secure self-service, modern cloud platforms
Benefits: Data protection, compliance, safe real-time data access

04. Common Vocabulary

Focus: Standardizing definitions (catalogs, calendars, KPIs), semantic layers
Benefits: Faster analysis, reduced disputes, consistent understanding

05. Data Curation

Focus: Modeling, cleansing, curating dimensions/measures
Benefits: Improved data quality, better user experience, maximized data value

06. Minimizing Data Copies and Movement

Focus: Reducing unnecessary movement, leveraging cloud scalability
Benefits: Lower costs, increased data freshness, greater agility

Modern Principles

01. Data as Product

Focus: Quality, discoverability, documentation & usability
Benefits: Higher data quality, better collaboration

02. Domain-Oriented Decentralized Ownership

Focus: Align data ownership with domain expertise
Benefits: Scalability, domain-specific speed

03. Cloud-Native & Infrastructure as Code

Focus: Elasticity, automation, repeatable setups
Benefits: Fast provisioning, automation, traceability

04. Event-Driven Architecture

Focus: Real-time data flows, system responsiveness
Benefits: Real-time insights, decoupled systems

05. Interoperability & Open Standards

Focus: System integration, vendor independence
Benefits: Vendor flexibility, smooth integration

06. Metadata-Driven Architecture

Focus: Automation, data lineage, discovery
Benefits: Enhanced observability, automation

07. Resilience & Fault Tolerance

Focus: Service availability, data protection
Benefits: High availability, user satisfaction

08. Observability & Monitoring

Focus: Bottleneck identification, operational optimization
Benefits: Proactive issue resolution, performance optimization

End of Lesson - Exit Ticket

Survey Link

https://www.menti.com