









Fundamentals of Data Engineering Chapter 3

<input checked="" type="checkbox"/> Favorite	<input type="checkbox"/>
<input checked="" type="checkbox"/> Archived	<input type="checkbox"/>
<input checked="" type="checkbox"/> Fleeting	<input type="checkbox"/>
↗ Area/Resource	
↗ Project	
▼ Type	
📅 Review Date	
📎 Image	
🔗 URL	
🕒 Created	@February 27, 2023 11:11 AM
🕒 Updated	@February 28, 2023 2:48 PM
🔍 Root Area	
🔍 Project Area	
Σ Updated (short)	02/28/2023
↗ Pulls	
🔍 Resource Pulls	
🔍 Project Archived	
Σ URL Base	
Σ 🔍 Recipe Divider	🥗🥗🥗 RECIPE BOOK PROPERTIES 🥗🥗🥗
☰ 🔍 Recipe Tags	
Σ 📖 Book Divider	📖📖📖 BOOK TRACKER PROPERTIES 📖📖📖
☰ 📖 Author	

  Date Started	
  Date Finished	
  Book Status	
  Rating	



This chapter looks into Data Architecture and the best practices relating to it. It claims that good data architecture provides seamless capabilities across every step of the data lifecycle and undercurrent. The final half of the chapter looks at examples of data architectures and how they apply the principles and concepts.

What is Data Architecture?



To understand data architecture one must understand enterprise architecture as Data Architecture is a subset of it.

What is enterprise architecture?



EA is the practice of analyzing, designing, planning and implementing a comprehensive and holistic framework of an organization's information technology (IT) and business strategies. It is a systematic and structured approach that helps align an organization's people, processes, information and technology components to achieve its goals and objectives.

- The primary objective is to ensure an organizations IT and business strategies are aligned and support each other. This is achieved by providing a framework for organizing and integrating different technology components and business processes, and by defining the relationships between them

Enterprise architecture is the design of systems to support change in the enterprise, achieved by flexible and reversible decisions reached through careful evaluation of trade offs.

Flexible and reversible decisions



Reversible decisions allow you to adjust course as the world changes and you gather new information. As organizations grow there is a natural tendency towards enterprise ossification. Adopting a culture of reversible decisions overcomes this tendency and makes organizations more adaptable.

- Jeff Bezos, one way door and two way door analogies. Also prioritize low stakes reversible decisions

Change management



Closely related to reversible decisions. Enterprises often need to undertake large initiatives. These are ideally broken down into smaller changes, each one a reversible decision in itself.

Trade offs



Trade-offs are inevitable and ubiquitous in the engineering space. Digital systems are ultimately constrained by physical limits such as latency, reliability, density and energy consumption. There are also non-physical limits such as characteristics of programming languages and frameworks, as well as practical constraints in budgeting, and complexity. Data Engineers must account for trade-offs at every step of the design stage while minimizing high-interest technical debt.



Technical debt is used to describe the accumulated cost of maintaining and updating software systems that were built quickly and without proper design, planning or testing. This happens when teams prioritize speed over quality and can occur at different levels of a software system, including the code, architecture, infrastructure and testing.

☐ Look into Fundamentals of Software Architecture

Technical solutions exist not for their own sake but in support of business goals.

Data Architecture



Data Architecture is the 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.

Operational architecture



This encompasses the functional requirements of what needs to happen related to people, processes and technology. Aka what needs to be done

- What processes does the data serve?
- How does the organization manage data quality?
- What is the latency requirement from when the data is produced to when it becomes available to query?

Technical architecture



Outlines how data is ingested, stored, transformed, and served along the data engineering lifecycle. Aka how it will happen

- How will you move 10TB of data every hour from a source database to your data lake?

Good data architecture



Good data architecture serves business requirements with a common, widely reusable set of business blocks while maintaining flexibility and making appropriate trade-offs. Bad architecture is authoritarian and tries to cram a bunch of one size fits all decisions into a big ball of mud.



A big ball of mud is used to describe a system or application that is complex, disorganized, and difficult to maintain or change. It is typically the result of years of ad-hoc development, where decisions are made on the fly and without a clear architecture plan. As a result the system is hard to understand and maintain.



Agility is the foundation for good data architecture, it acknowledges that the world is fluid.

Bad data architecture



Tightly coupled, rigid, overly centralized, or uses the wrong tools for the job, hampering development, and change management.

Principles of Good Data Architecture

- ☐ AWS well architected framework

☐ GCP Five principles for cloud native architecture

1. Choose common components wisely
2. Plan for failure
3. Architect for scalability
4. Architecture is leadership
5. Always be architecting
6. Build loosely coupled systems
7. Make reversible decisions
8. Prioritize security
9. Embrace FinOps

Principle 1: Choose common components wisely



Common components are anything that has broad applicability within an organization. Including, object storage, version control, observability, monitoring, orchestration systems and processing engines.



Good Data engineers choose common components that can facilitate team collaboration, and breaking down silos. They enable agility within and across teams in conjunction with shared knowledge and skills. They must support robust permissions and security to enable sharing of assets among teams while preventing unauthorized access.

Principle 2: Plan for failure

| Everything fails, all the time



Modern hardware is highly robust and durable. Even so hardware components will fail, given enough time. To build highly robust systems you must consider failures in your design.

- Reliability
- Availability
- Recovery time objective
 - The maximum acceptable time for a service or system outage.
- Recovery point objective
 - The acceptable state after recovery.

Principle 3: Architect for Scalability



Scalability is key in Data Systems, and your system must have the ability to scale up and down based on requirements. This is known as an elastic system that can scale dynamically in response to load in an automated fashion. Some cloud services can scale to zero.

Principle 4: Architecture is leadership



Data architects should be highly technically competent but delegate most individual contributor work to others. Strong leadership skills combined with high technical competence is rare and extremely valuable. Data architects should mentor current data engineers, making careful decisions in relation to the business context, and train engineers in best practices.

Principle 5: Always be architecting



Data architects don't serve in their role simply to maintain the existing states, instead they constantly design new and exciting things in response to changes in business and technology. An architect's job is to develop deep knowledge of the baseline architecture, develop a target architecture and map out a sequencing plan to determine priorities and the order of architecture changes.

Principle 6: Build loosely coupled systems

When the architecture of the system is designed to enable teams to test, deploy and change systems without dependencies on other teams, teams require little communication to get work done. In other words, both the architecture and the teams are loosely coupled. - Google devops tech architecture guide.

Bezos API mandate



The Bezos API mandate is viewed as a watershed moment for Amazon as it put data and services behind APIs which enabled loose coupling of business functions.

Loose coupling in software architecture

1. Systems are broken into many small components
2. These systems interface with other services through abstraction layers, such as a messaging bus or an API. These abstractions hide and protect internal details of the service, such as the database backed or internal classes and method calls.
3. Due to 2 internal changes within a service does not require changes within another service. Each piece can evolve and improve separately.
4. As a consequence of property 3. there is no waterfall, global release cycle for the whole system. Instead each component is updated separately as changes and

improvements are made.



Loose coupling of both technology and human systems will allow your data engineering teams to more efficiently collaborate with one another and with other parts of the company. This principle facilitates principle 7.

Principle 7: Make reversible decisions



With a rapidly shifting data landscape it is important to aim for reversible decisions as they simplify your architecture and keep it agile.

Principle 8: Prioritize Security



Every data engineer must assume responsibility for the security of the systems they build and maintain. Two main ideas in this principle are zero-trust security and the shared responsibility security model.

Zero trust security models



Traditional architectures place a lot of faith in perimeter security, crudely a hardened network perimeter with “trusted things” inside and “untrusted things” outside. Unfortunately, this approach has always been vulnerable to insider attacks, as well as external threats such as spear phishing. Therefore this model assumes that any user attempting to access a service or network is a potential threat unless proven otherwise, regardless of whether they are inside or outside the network.

The shared responsibility model



Amazon pushes this model which divides security into the security of the cloud and security in the cloud. AWS is responsible for the security of the cloud, and users are responsible for security in the cloud.

Data engineers as security engineers



All data engineers should consider themselves security engineers. Those who handle data must assume responsibility for its security.

Principle 9: Embrace FinOps



FinOps is an evolving cloud financial management discipline and cultural practice that enables organizations to get maximum business value by helping engineering, finance, technology, and business teams to collaborate on data-driven spending decisions.

- Professional movement that advocates a collaborative working relationship between DevOps and Finance.



Pay as you go approaches in cloud environments makes spending far more dynamic, and therefore engineers need to learn to think about the cost structure of cloud systems.

Major architecture concepts



With the release of new technologies and tools, one can lose sight of the main goals of all these architectures: to take data and transform it into something useful for downstream consumption.

Domain and services



Domain: The real world subject area for which you're architecting.

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

- Domains can contain multiple services



To identify domains focus on what it represent in the real world and work backward. The best advice is to go and talk with users and stakeholders, listen to what they are saying and build the services that will help them do their job. Do not architect in a vacuum.

Distributed systems, scalability, and designing for failure

Data engineers are interested in four closely related characteristics of data systems:

Scalability

- Allows us to increase the capacity of a system to improve performance and handle the demand

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

Availability

- Percentage of time an IT service or component is in an operable state

Reliability

- The system's probability of meeting defined standards in performing its intended function during a specified interval



These characteristics are related as follows: If a system fails to meet performance requirements during a specified interval, it may become unresponsive. Thus low reliability can lead to low availability. On the other hand dynamic scaling helps ensure adequate performance without manual intervention from engineers-elasticity improves reliability.



Distributed systems are widespread in data architecture. Almost every cloud data warehouse object system you use has some notion of distribution under the hood.



Designing Data-Intensive Applications

Tight versus Loose coupling: Tiers, monoliths and microservices



Designing data architecture allows you to decide how much interdependence you want to include within your various domains, services and resources. Extremely centralized approaches are known as tightly couple. Decentralized domains and services that do not have strict dependence on each other, is known as loose coupling. Designing good data architecture relies on trade-offs between the tight and loose coupling of domain and services.

Architecture tiers



Your architecture has layers- data, application, business logic, presentations and so forth. You need to know how to decouple these layers.



Single tier architecture: Your database and application are tightly coupled, residing on a single server. This is fine for testing systems but not advised for production use, as it is not redundant.



Multitier: Architecture is composed of several layers. These layers are bottom up and hierarchical, meaning the lower layer isn't necessarily dependent on the upper layers, the upper layers depend on the lower layers. The notion is to separate data from the application and application from the presentation.

- Three tier architecture consists of data, application logic and presentation tiers. Each tier is isolated from the other, allowing for separation of concerns.



In single tier architecture, the data and logic layers share and compete for resources (disk, CPU and memory) in ways that are avoided in multitier architecture.

Monolithes



The general notion of a monolith includes as much as possible under one roof; in its most extreme version, a monolith consists of a single codebase running on a single machine that provides both application logic and user interface. Ignoring the growing complexity of their monolith will devolve into a big ball of mud.

Microservices



Microservice architecture comprises separate, decentralized and loosely coupled services. Each service has a specific function and is decoupled from other services operating within its domain. If one service goes down it won't impact other services.

☐ Software architecture: The Hard Parts

Considerations for data architecture



In the data space the monolith system is very common. A move towards a microservice equivalent with a data warehouse is to decouple the workflow with domain-specific data pipelines connecting to corresponding domain-specific data warehouses.

☐ Data mesh

User access: Single versus multitenant



Multitenancy refers to the ability of a software system or application to serve multiple tenants, or customers, on a shared infrastructure. Multiple users or organizations can access the same software system and their data is kept separate and secure from other tenants. All cloud services are multitenants although it occurs at various grains.



With multitenancy there are two factors to consider: performance and security. With multiple large tenants within a cloud system, with the system support consistent performance for all tenants, or will there be a noisy neighbor problem? (High usage from one tenant degrade performance for other tenants). Regarding security, data must be properly isolated from different tenants.

Event-Driven architecture



Events are broadly defined as something that happened, typically a change in the state of something. An event driven workflow encompasses the ability to create, update, and asynchronously move events across various parts of the data engineering lifecycle. This workflow boils down to three main areas: event-production, routing and consumption. An event must be produced and routed to something that consumes it without tightly coupled dependencies among the producer, event router and consumer.

Brownfield versus Greenfield projects



When designing a data architecture you need to know whether you're starting with a clean slate or redesigning an existing architecture. Projects fall broadly into two buckets.

Brownfield projects



Involve refactoring and reorganizing an existing architecture and are constrained by the choices of the present and past. Because a key part of architecture is change management you must figure out a way around these limitations and design a path forward to achieve your new business and technical objectives.



A popular approach is the strangler pattern: New systems slowly and incrementally replace a legacy architecture's components. Eventually the legacy system is completely replaced. This allows for flexible and reversible decisions while assessing the impact of the deprecation on dependent systems.

Greenfield projects



Fresh start project unconstrained by legacy systems.



A common pitfall is teams feeling compelled to reach for the latest and greatest technology fad without understanding how it will impact the project. Always prioritize requirements over building something cool.

Examples and Types of Data Architecture



Due to the abstract nature of data architecture reasoning by example is helpful as it allows us to develop a deeper understanding of the concepts by relating to real life examples.

Data Warehouse



A data warehouse is a central hub used for reporting and analysis. Data is highly structured and formatted for analytics use cases.

A subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of managements decisions.

Subject oriented: Data warehouses are designed to be subject-oriented as the data is served to provide a comprehensive view around a businesses processes.

Integrated: Data in a data warehouse consolidates data from multiple sources. It is designed to provide a view of the data from multiple sources that are ingested into it.

Non-volatile: Data warehouses are designed to be non volatile, meaning that once data is entered into the warehouse it cannot be updated or altered. It designed to ensure data consistency and reliability as it shows a view of a business over time.

Time variant: Stores historical data to analyze a business's processes over time. Allowing for the performance of trend analysis etc...

Organizational data warehouse architecture



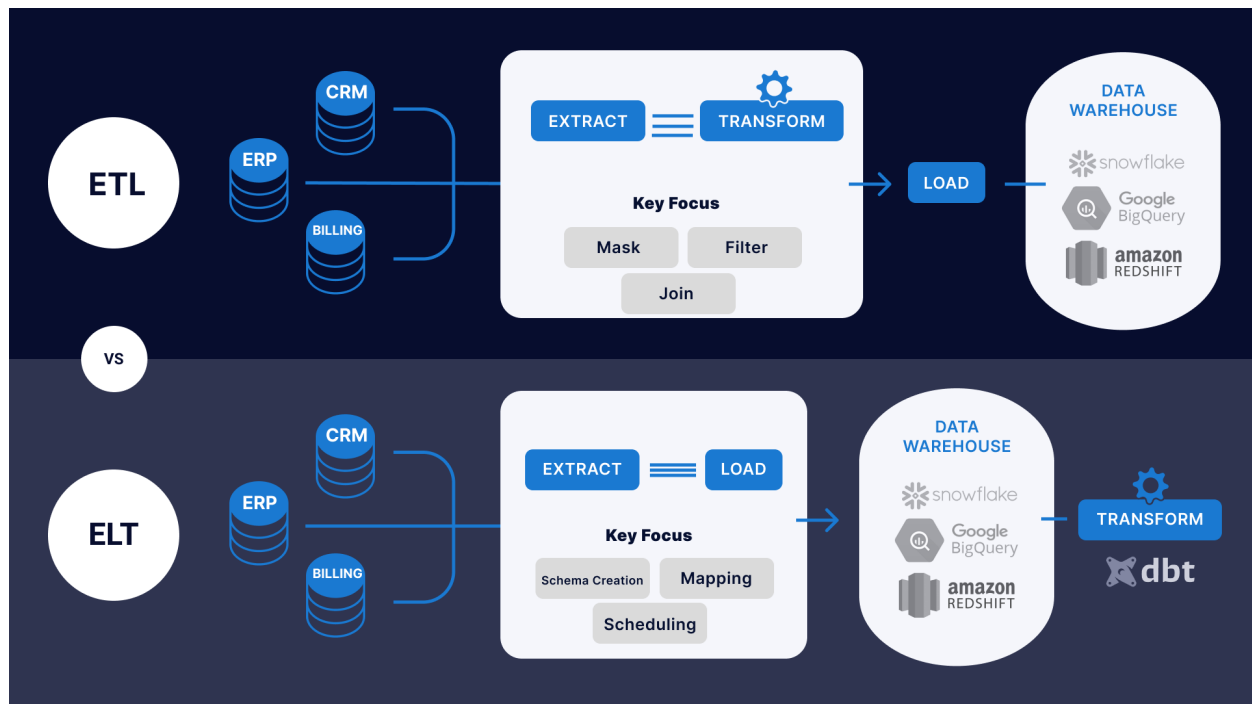
Organizes data associated with certain business team structures and processes.

1. Separates online analytical processing from production databases
2. Centralizes and organizes data

Technical data warehouse architecture



Reflects the technical nature of the data warehouse. It is designed to support an organizations technical and operational needs.



ETL vs ELT for loading data into a warehouse



There are two primary ways to load data into a warehouse. ETL transforms the data prior to arriving into the warehouse. Whereas ELT takes advantages of the separation between storage and compute, and the computing powers of most cloud data warehouses. DBT is commonly used in transforming data in a data warehouse. Complex structures such as JSON data can be stored and transformed in data warehouses now.

Separation of compute and storage



Previously compute and storage in data warehouses were tightly coupled, leading to competition of resources. This also meant that to run data warehouses, very large servers were required. Separating compute and storage means that companies can scale their storage and compute independently of each other. It also reduces the I/O bottle necks when processing and storing data on the same servers.

☐ Look into BigQuery architecture

Data marts



A data mart is a more refined subset of a warehouse designed to serve analytics and reporting focused on a sub organization. This makes data more accessible to analysts and report developers. And, provides an additional stage of transformation beyond that provided by the initial ETL or ELT pipelines.

Data Lake



A centralized repository that can store raw, unprocessed data in it's native format, enabling organizations to store and process data in it's unaltered format from multiple sources.

- Started with Hadoop Distributed File System

Weaknesses

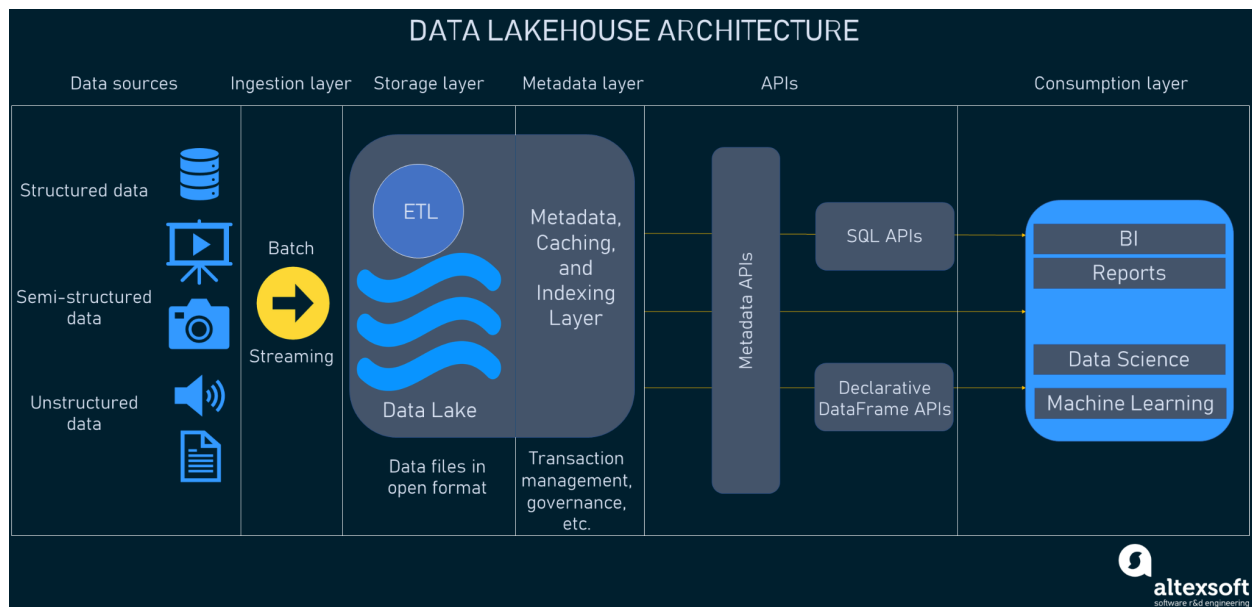


Data lakes became dumping grounds such as a data swamp. Data grew to unmanageable sizes with little in the way of schema management, data cataloging, and discovery tools.

Data Lakehouses



The lakehouse aims to combine the advantages of data warehouses with the advantages of data lakehouses. DataBricks provides a platform that incorporates the controls, data management, and structures found in data warehouses while still data in object storage and supporting a variety of query and transformation engines. Data is not structured at the time of ingestion but it instead uses a schema on read approach, meaning its structure is defined at time of analysis.

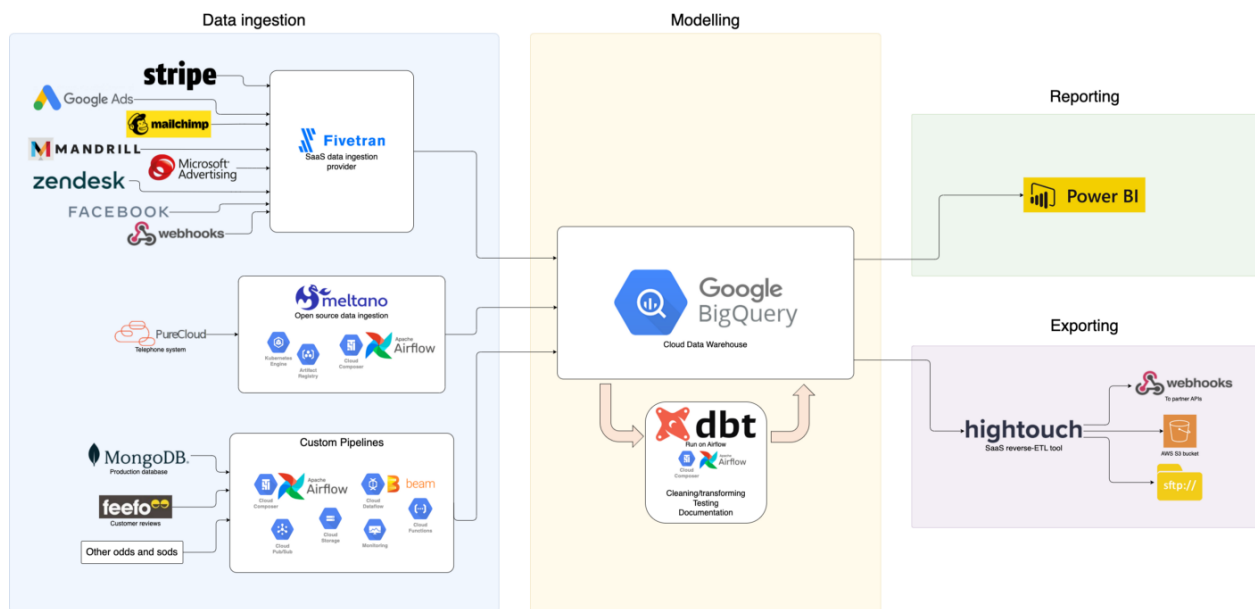


- The trend of converging between data warehouses and data lakehouses will only continue

Modern data stack



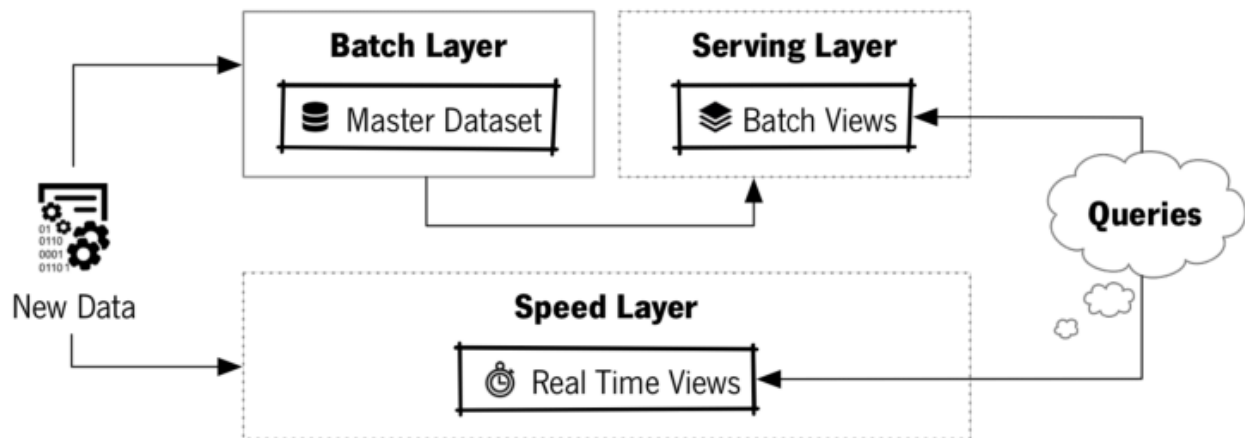
The modern data stack makes use of cloud-based, plug-and-play, easy-to-use, off-the-shelf components to create a modular and cost-effective data architecture. These components include data pipelines, storage, transformation, data management/governance, monitoring, visualization and exploration. This leads to self-service, agile data management and using open source tools with clear pricing structures.



Lambda architecture



Data engineers wanted a way to combine batch processing and stream processing in a single architecture, as there were scenarios where real time analysis was needed as well as long term analysis of the data was relevant.



You have three systems operating independently of each other. Batch, streaming and serving. The source system is immutable, and append-only sending data to two destinations for processing, stream and batch. The stream (speed) layer is the lowest possible latency, usually a noSQL database. In the Batch layer data is processed and transformed in a system such as a data warehouse, creating precomputed and aggregated views of the data. The serving layer provides a combined view by aggregating query results from the two layers. This layer can be DataStudio or Tableau.

Weaknesses



Managing multiple systems with different codebases is difficult.

Kappa architecture



This architecture lies on the thesis of using streaming processing as the backbone for all data handling-ingestion, storage and serving. Real-time and batch processing can be applied seamlessly to the same data by reading the live event stream directly and replaying large chunks of data for batch processing.

Weaknesses



Streaming is complex, it is also expensive which does not make it a good solution for all transformations.

The Dataflow model and unified batch and streaming



The dataflow model and the apache beam framework allows for unifying code paths. The core idea is to view all data as events, as the aggregation is performed over various types of windows. Ongoing real.time event streams are unbounded data. Data batches are simply bounded event streams, and the boundaries provide a natural window. Real time and batch processing happens in the same systems using nearly identical code. This is the philosophy of batch as a special case of streaming.

Architecture for IoT



The Internet of Things is the distributed collection of devices, aka things. These things are connected via a network that allow them to connect data and communicate with eachother. Data ingestion for IoT is complex but rewarding to learn.



Devices are the physical hardware connected to the internet, sensing the environment around them and collecting and transmitting data to a downstream destination. Any device capable of collection data from it's environment is an IoT device.

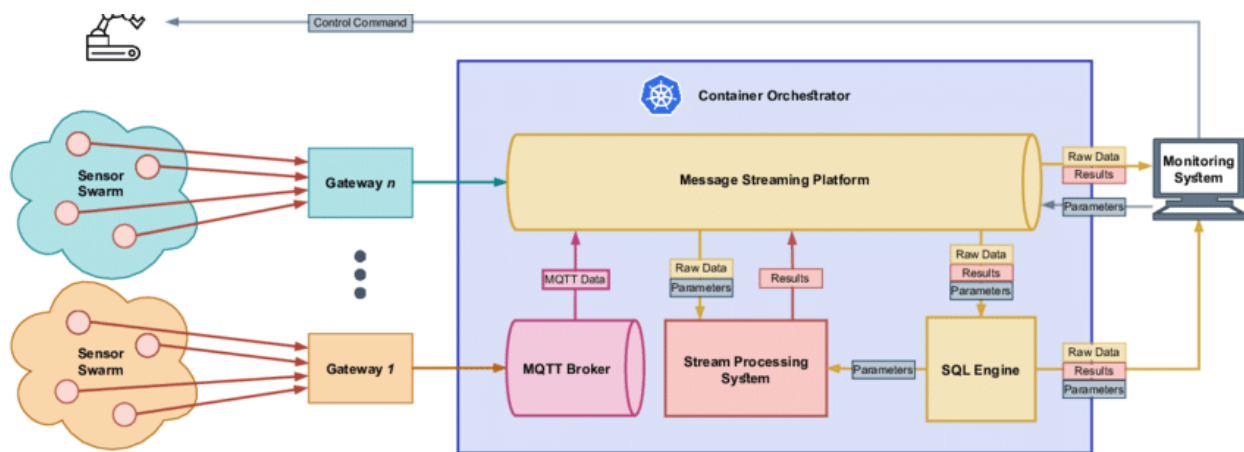


A data engineer does not need to know the inner details of IoT devices but should know what the device does, the data it collects, any edge computations or ML it runs before transmitting the data and how often it sends data.

IoT gateway



An IoT gateway is a hub for connection devices and securely routing devices to the appropriate destinations on the internet. While you can connect a device directly to the internet without an IoT gateway, the gateway allows devices to connect using extremely little power.



Ingestion



Ingestion begins with a gateway, from there events and measurements can flow into an event ingestion architecture.

Storage



Storage requirements will depend a great deal on the latency requirements for the IoT devices in the system. In many cases a message queue or a time based database is appropriate.

Serving



Data can be batch or near time analyzed. Detections for anomalies can also occur.

Data Mesh



Emerged to help with the challenges organizations face managing data at scale. It proposes a new way of organizing data and data teams that emphasizes autonomy, decentralized governance and domain-driven design. Instead of flowing data from from domains into a centrally owned platform, domains need to host and serve their domain datasets in a easily consumable way.

- Domain oriented decentralized data ownership and architecture
- Data as a product
- Self-serve data infrastructure as a platform
- Federated computational governance

☐ Look more into the data mesh