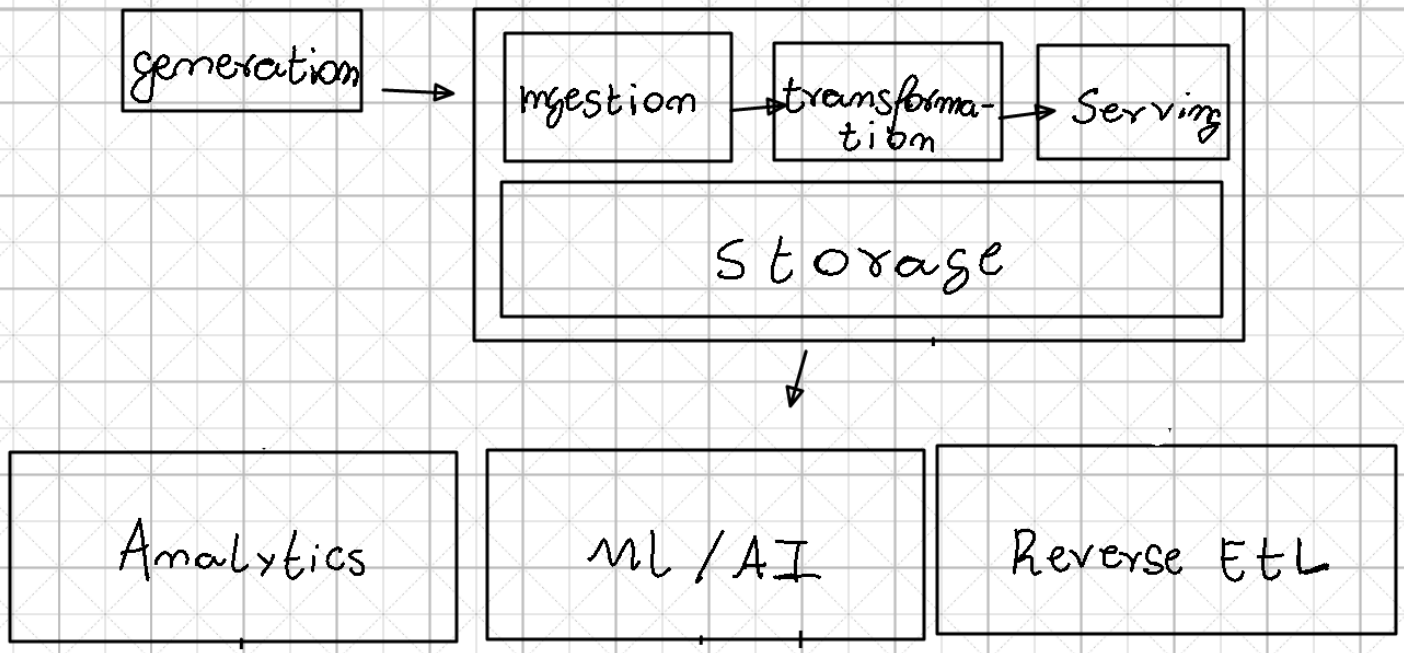
What is Data engineering?

Data Pipeline →



Generation → source systems

Application databases
- Postgre SQL
- My SQL
- Mongo DB

Third-party Systems
- Stripe
- Salesforce
- Google analytics

Storage -

- Types of storage systems
  Ex) file storage vs object storage
- Row-based storage vs columnar storage
- Data wharehouse vs data lakehouse
- Right type of storage cam have a performance difference of up to 100x!

Ingestion -

- Batch vs Streaming ingestion
- ETL vs ELT
- Different ways to ingest data from different source systems

Transformation -

- SQL queries
- Data modeling
- Normalization

Serving

- Data analysis
- Data science
- Reverse ETL

Shared components

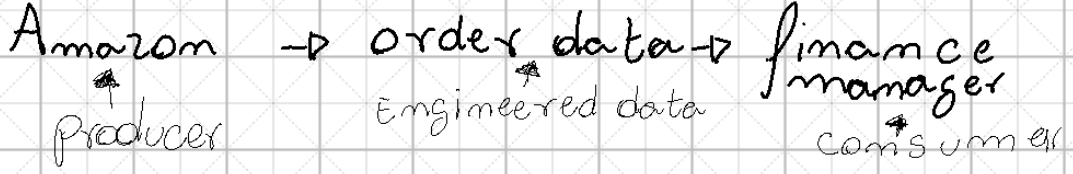
- Data Ops
- Orchestration
- Security
- Data quality
- Data privacy

What is data engineering?

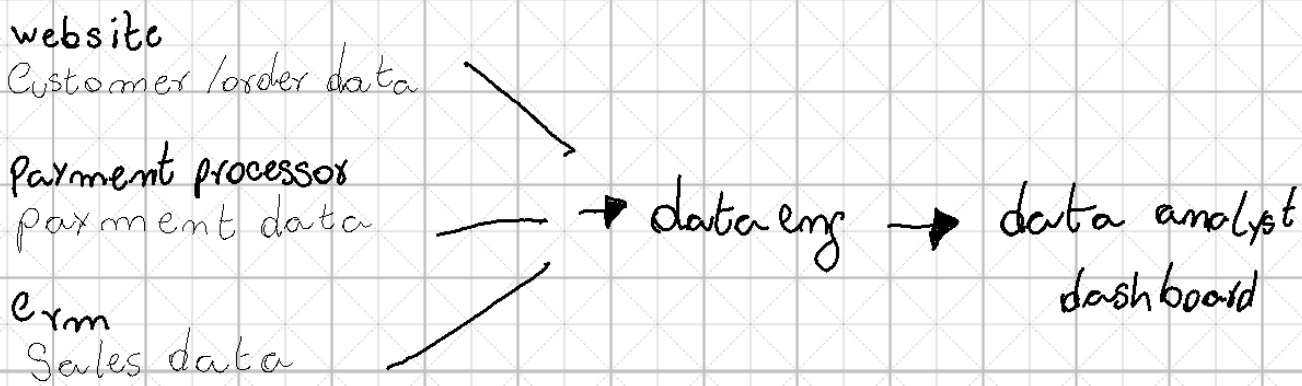
and why is it important

- Data engineering serves as a bridge between the data "producer" and the data "Consumer"

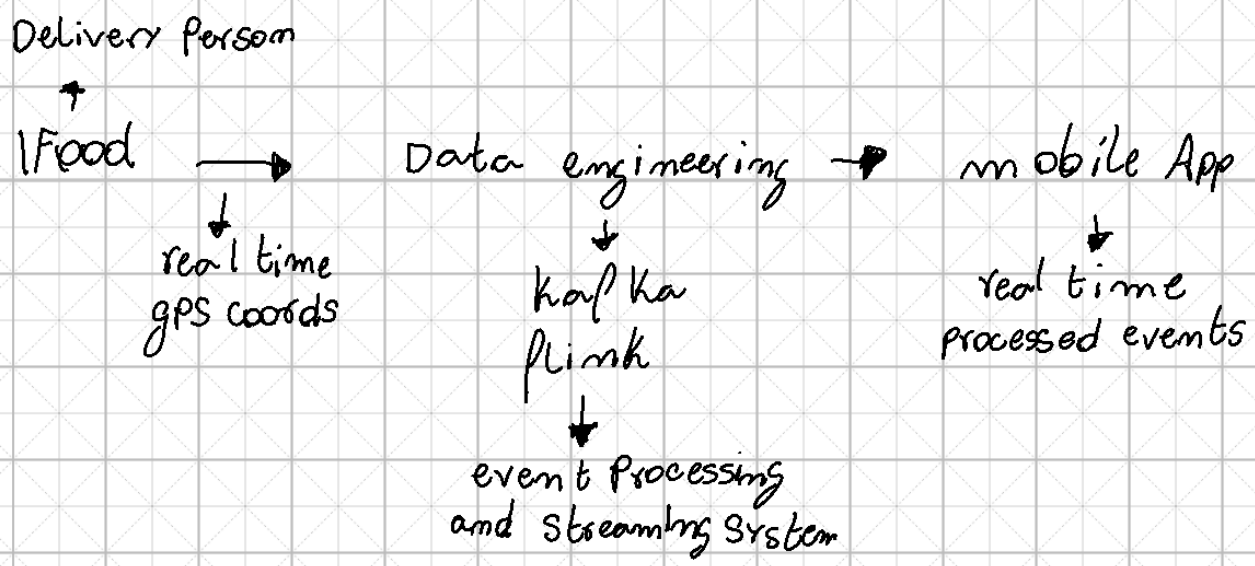
Example -



Example - Business Analytics



Example - real time



Historical context-

1980 - 2000 : Data Warehouses

Data warehouses
+
Scalable analytics (BI)

MPP:
Massively parallel Processing

BI engineer

ETL developer

Data warehouse engineer

on-prem (vs the cloud)
Monolithic data stores

2000's: The Beginning of "Big Data"

Internet + google, Amazon, etc → the birth of Cloud (built on cheap commodity hardware)

AWS
Starting with S3 and EC2

2013 google publishes a paper on Map Reduce - Hadoop

2000's - 2010'S: Big Data engineering

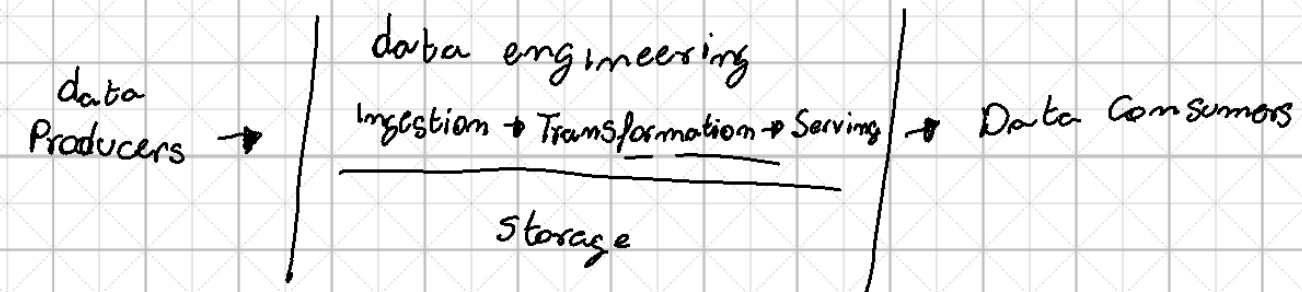
Big Data = Hadoop

Hadoop Ecosystem

Hadoop, Yarn, HDFS

- A lot of work to manage
- requires a specialized engineering team

2020's: Modern data stack



* Compose together Lego-like modular components
* No single tech stack makes up the entire pipeline

Data Maturity

: Determines the complexity of your data pipeline

Data engineer's place within the data team -

Data producers
Software engineers
DevOps engineers
→ "Bridge" → Data engineers → Data Consumers
Data Scientists
Data analysts

* Data engineers act as a bridge between data producers and data consumers

Responsibilities - business & technical

business

- Communicate with both technical and non-technical people
- Understand how to scope and manage a data project
- Minimize cost and work within budget

technical

- Create a good data architecture
- Build and manage a reliable and performant data pipeline
- Security, data governance, automation, observability, etc..

Required technical skills -

SQL | Python / R | Basic DevOps (eg. Docker) | Bash

Data gen- source systems

Structured vs unstructured data

↓ ↓

tables (sql) Deep learning

↓ ↓

BI, Classical ML Neural networks

↓ ↓

small # of features large # of features

types of databases

1- Relational databases (RDBMS) 2- Non-Relational databases (NO-SQL)

3- Key value Store

1- often used to store transactional data

2- JSON structure, use NOSQL if data is less structured, No-sql databases use distributed architecture, this allows it to scale horizontally.

3- Can be -

memory based:

Catching application data

Very fast Lookup/ High concurrency

Data not persisted

} Persisted -

fast performance

Simple/Single-table

Easy to use (store anything in value)

highly scalable

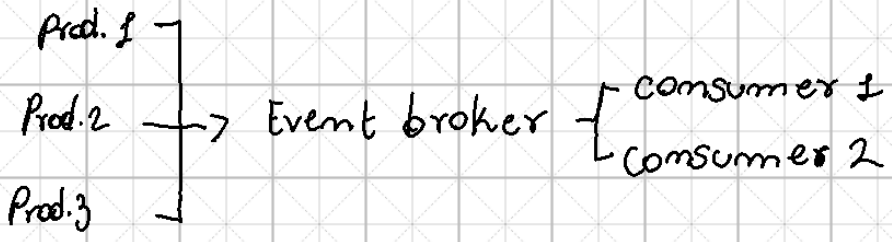
Third party systems via API.

Common api types - Rest API { GraphQL { gRPC

Event streams -

Event producers

Event consumers



Exactly-once vs at-least-once delivery

Exactly once -

1 6 4 5 3 2 → Event broker → 6 5 4 3 2 1

At-Least-once

1 6 4 5 3 2 → Event broker → 6 5 4 3 2 1 1

↳ Event 1 is duped

Idempotency

An operation is "idempotent when the same result comes out, no matter how many times you run it.

why is idempotency important - allows for detection of duplicates

Popular event streaming platforms

AWS SQS	Amazon Kinesis	RabbitMQ	Kafka	Pulsar	Spark

Caveats

Do NOT consider the source system as someone else's problem

upstream source system → Downstream data pipeline

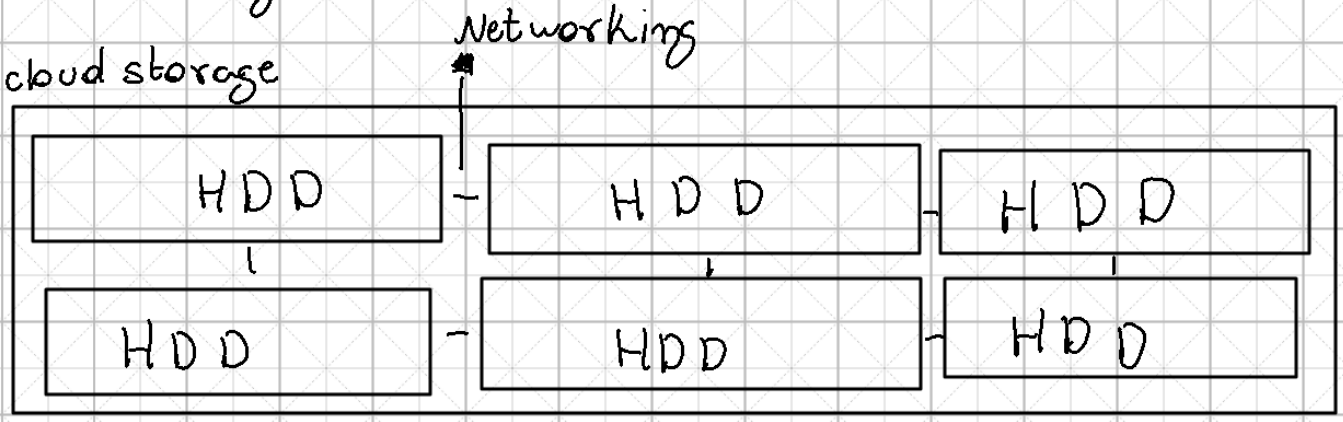
validation and testing

communicate with your data producers

Storage -

HDD'S / SSD's / RAM

Networking :



Serialization

Data → serializer → file / Data store / memory

Pandas → serializer → parquet

Row based serialization file formats

- XML
- Json
- Csv

→ fast lookup of individual rows

Column-based serialization file formats

- Parquet
- orc
- Arrow

→ aggregation by column

Row based databases

Columnar databases

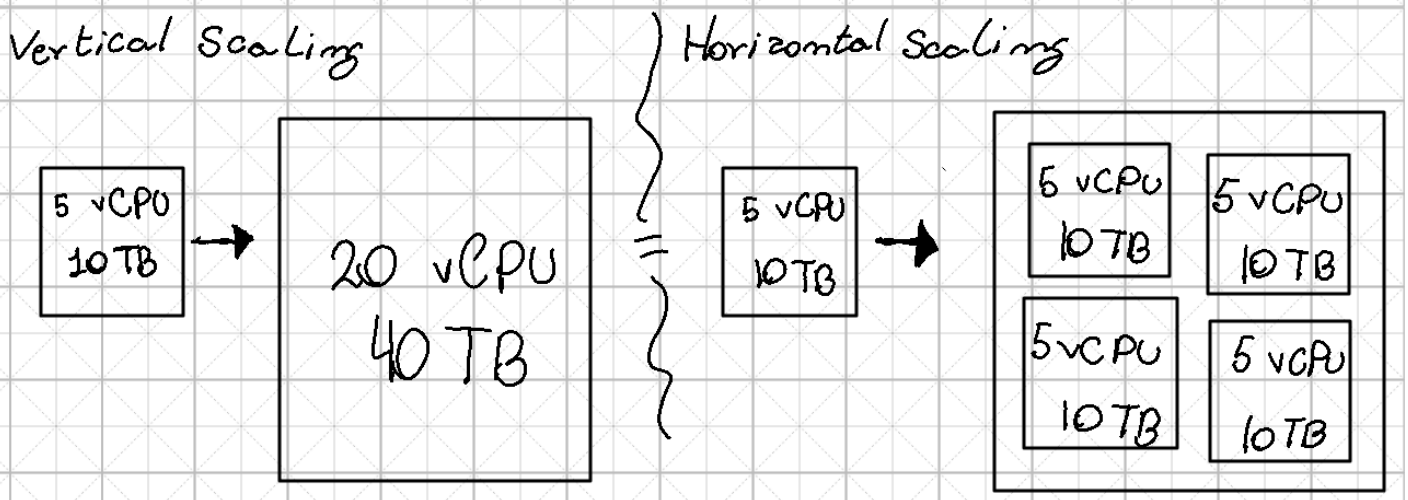
Compression

- Reduce the ammount of data stored
- faster query (less to search
- faster transport (less data to move)

Caching

- Some storages are faster to access
- Memory is 1000x faster than SSD's and CPU caches are orders of magnitude faster than memory
- faster storage is more limited and expensive

Single Machine vs. Distributed Storage !



Distributed Storage = Horizontally scaling storage

+

Distributed Compute = Retrieve and process data in a distributed way by using multiple machines

=

Store, retrieve and process data FASTER, because you can do them in parallel, at a much Larger scale

Strong vs eventual consistency

Strong - any client asking for the same data gets the same result

e.g. → Order database, only one table

eventual - data changes based on differences on multiple databases

e.g. → order database, multiple tables, difference between them

ACID vs. BASE

Single Machine transactional database	Distributed Database

Acid

- ATomicity
- Consistency (strong)
- Isolation
- Durability

Base

- Basically Available
- Soft-state
- Eventual consistency

Types of storage systems

file storage
- Local fs
- NAS
- Cloud

block storage
- HDD / SSD
- AWS EBS

Object storage
- AWS S3

Cache storage
- RAM
- Memcached
- Redis

Streaming storage
- Buffering
- Often ephemeral, but can be persisted

File storage vs. Object storage

Scale
↓
File < millions
Object = ∞

structure
↓
file = tree
Object = flat

Latency
↓
file: Fast
Object = slow

Updates
↓
file = Mutable
object = Immutable

OLTP (row-based database) in memory

- ✓ - Specific row - single transaction
- ✓ - Online transaction processing
- ✓ - e.g. PostgresSQL, Mongo DB

} Speed

OLAP (column-based database) in memory

- ✓ - Analytics use case (aggregating total orders / customers)
- ✓ - Online analytical processing

} Volume

Data warehouse, Lake, Lakehouse

Data warehouse

↳ OLTP → OLAP

- Well organized

* Amazon Redshift
* Google big query
* Snowflake

Clean, organized data

typical use case: Business analytics

Central storage system where:
: all data is aggregated
: downstream users Pull the data from

Data Lake

↳
- Inexpensive object storage
- Structured and unstructured
- Easily becomes "data swamps"

* AWS S3

Data Lakehouse

↳ · OLAP

- Advantage of the warehouse + Lake
- Good integration with external query engines

* Iceberg
* Apache hudi
* Delta Lake

object storage + Open table format

AWS S3 + Iceberg

Data storage Lifecycle - hot, warm, cold

high data access frequency → low data access frequency

hot

AWS S3

- Accessed often
- Expensive
- fast

warm

Aws S3 Infrequence access tier

- Accessed infrequently
- Inexpensive

cold

Aws Glacier

- Rarely accessed
- very cheap

Ingestion - source systems → Storage system

frequency: batch, micro-batch, streaming

batch

- Daily or hourly
- Ex: tax reports

Micro-batch

- minutes to seconds
- Ex: Near real time analytics

Streaming

- Seconds to Subsec
- Ex: Uber

ETL vs ELT (Batch ingestion)

ETL:

Data Sources → Extract → Transform → Load → storage

* traditonally, the most common pattern
* Fits with traditonal data warehouse system

ELT:

Data Sources → Extract → Load → Storage → Transform

* New trend: more common now
* Fits with data Lakehouse

Streaming Ingestion Challenges:

Strict latency requirements

Late-arriving data

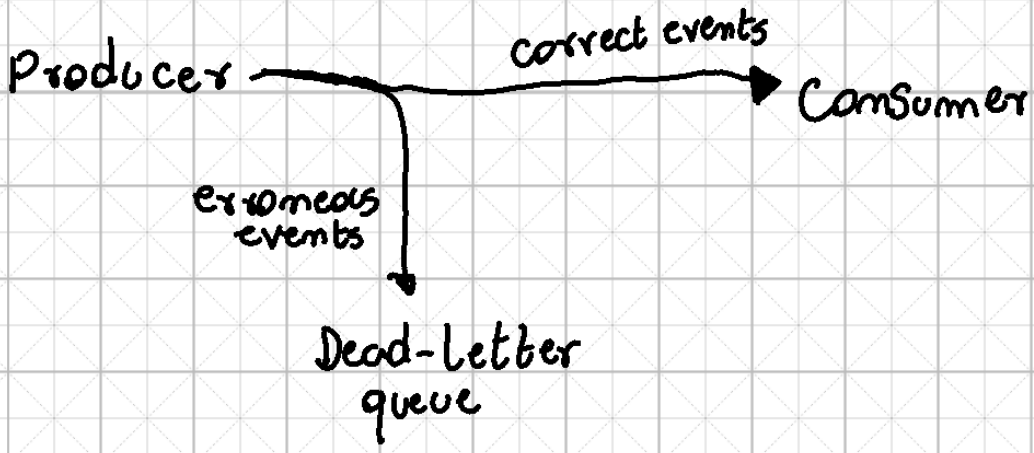
Ordering

Multiple delivery

Retention period

Message size

Error handling for streaming -



Scalability and schema changes

Scalability

- Make sure your system can scale up and down automatically based on workload
- for streaming data, add a sufficient buffer for fluctuating event volumes

Schema

- Upstream changes to a fixed schema will break the ingestion pipeline
- Schema migration features help, but aren't perfect - frequently communicate with upstream teams

Transformation -

Queries:

DDL
data definition Language → Create table Customers

DML
data manipulation Language → Select * From Orders where order_id = 1

DCL
data control Language → Grant Select ON main_db TO user_name
Dave

Data Modeling -

· A data model is a simplified version of real world business activities.

Conceptual → Logical → Physical

* Ingredients of successfull data modeling

· Communicate with the data consumers from the beginning

· Model the data at the lowest granularity possible

Different modeling techniques

Inmon
↳ Highly normalized data

Kimball
↳ Flexibility and faster iteration
· star schema

Data vault
↳ Hubs: core business concepts
Links: relationships between hubs
Satellite: store info about the hubs

Wide tables
↳ Join many related tables into a single highly de-normalized table

Serving -

Machine Learning

CSV → Object Storage (S3)

Data Lakehouse → Data science team

Jupyter Notebook → Data science team

Feature store → Feast / Tecton

Deep Learning -

ML

- Structured (tabular) data
- Limited # of features (100-1000s)
- Smaller data sets
- Relatively small compute (CPU's)

Deep Learning

- Unstructured data - e.g. images
- Large # of features (millions)
- Much larger data sets
- Requires lots of compute (GPU's)

Reverse ETL

extract data from storage system and load it back into the external system

Data OPS

DevOPS

- Build/manage cloud Infra
- Observability of cloud Infra
- Build automated CI/CD Pipeline

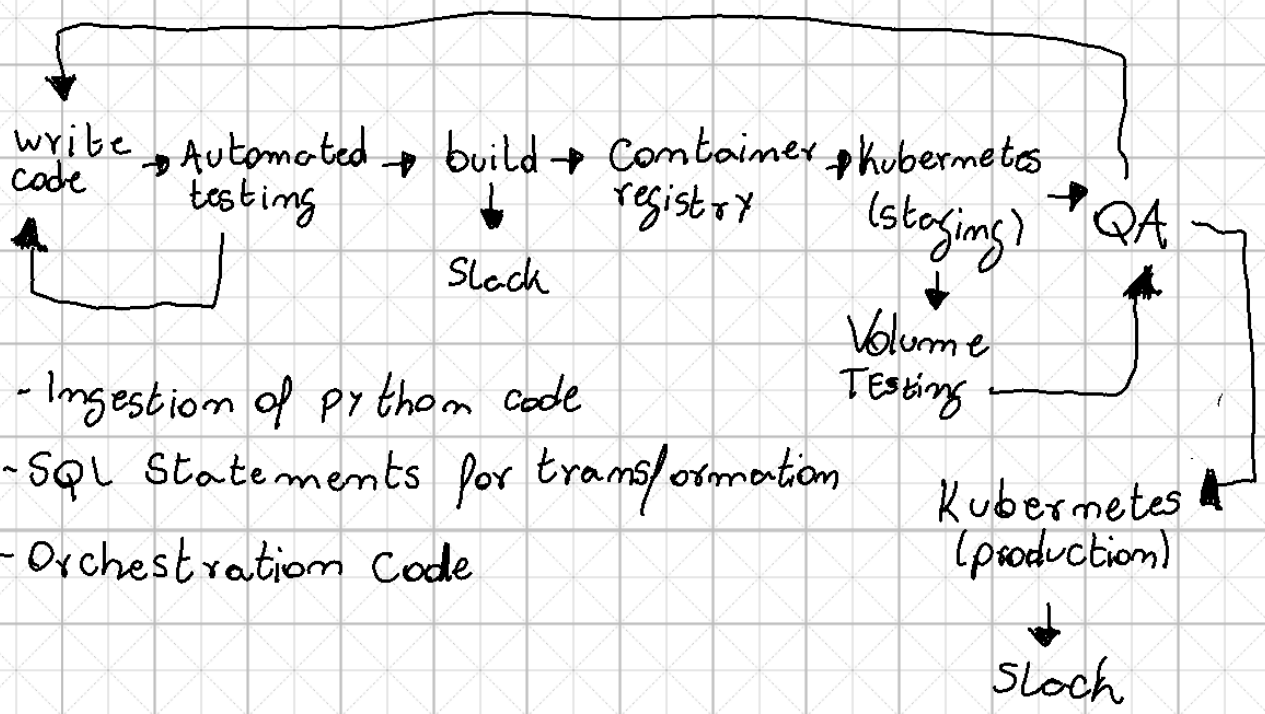
DataOPS

- Build/manage cloud infra for data tools
- Observability of data systems
- Automation (including CI/CD)

Automation - CI/CD pipeline

most small teams overlook dataops / Learn from software engineering

Example:



- Ingestion of python code
- SQL Statements for transformation
- Orchestration Code

Orchestration:

- Central place from which most data engineering jobs can be managed organizing all jobs that need to be completed
- its optional for small data teams only

Why is orchestration important?

↳ improve efficiency (Parallel executions) } Complex dependency management } improved reliability Retries and time outs

Automation

end-to-end visibility
Data Lineage

} tools - Apache Airflow Prefect dagster flyte

Other undercurrents:

Security
- Authentication
- Authorization or "Access Control"
- Principle of Least Privilege

} Privacy
- Coming into focus
- High profile breaches
- personally identifiable Information (PII)

Development Workflows

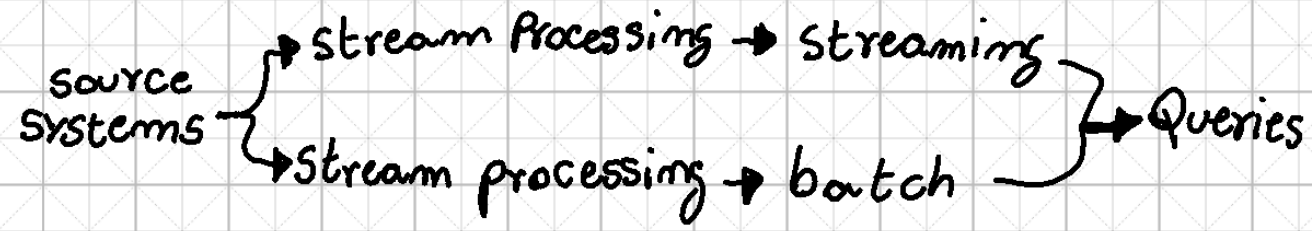
Development (local laptop) → Production (AWS)

Benefits -
- fast iteration
- isolation (safe to tinker with)
- cost reduction (local pc for dev)

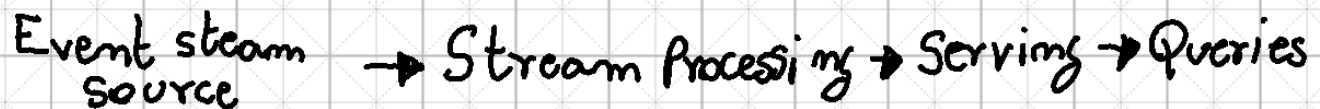
Challenges -
- sampling a large dataset is not allways easy
- Dev / Prod environment parity is not allways possible
- Cloud tools not allways available locally

Lambda and Kappa architectures

Lambda -

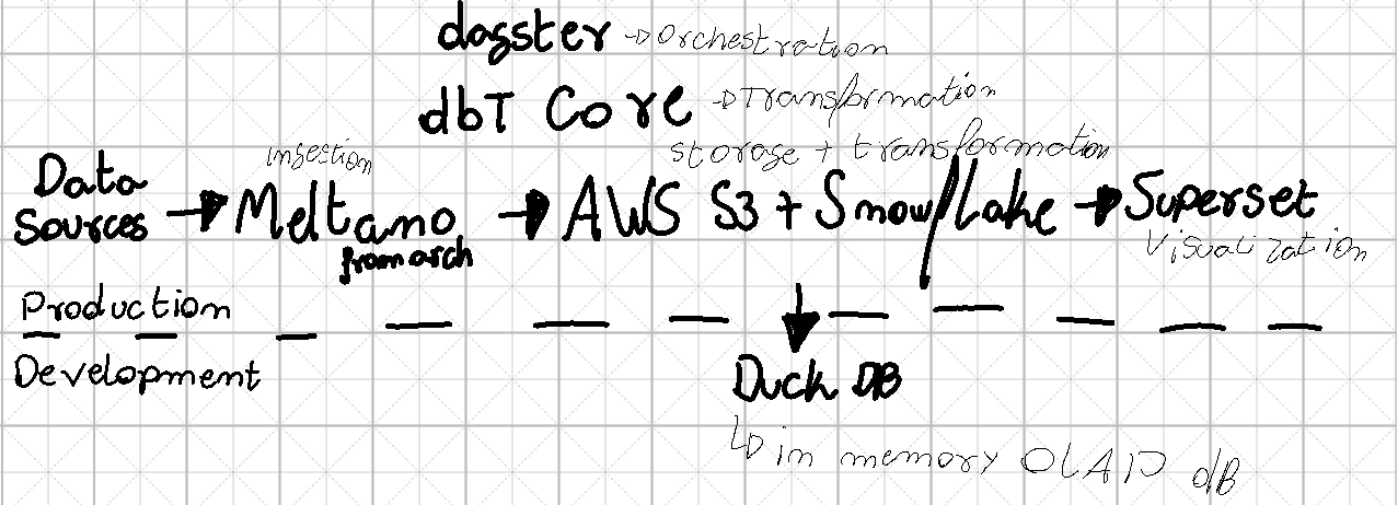


Kappa -



BI stack (data warehouse)

BI stack v1 - Data warehouse



Orchestrators - Airflow vs Dagster

Apache Airflow	Dagster
• No local env setup	• Local dev env
• Dependency Mgmt issues	• Cloud-native, easy deploy, CI/CD
• Complex to deploy and Mng	• Declarative (rather than imperative)

Imperative vs Declarative

Imperative -
the "how"
Spell out the exact steps

Declarative -
the "what"
Specify the desired outcome

reactJS / Kubernetes

Ingestion -

examples = Fivetran / Airbyte / Meltano / DLT hub

Transform -

dbt core -
- Supercharged SQL (is not a compute engine)
- Code-like powers
  - templates
  - tests
  - Version Control
  - Documentation / more

AWS(S3) + Snowflake
- write SQL inside Snowflake
- fully managed

Visualization -

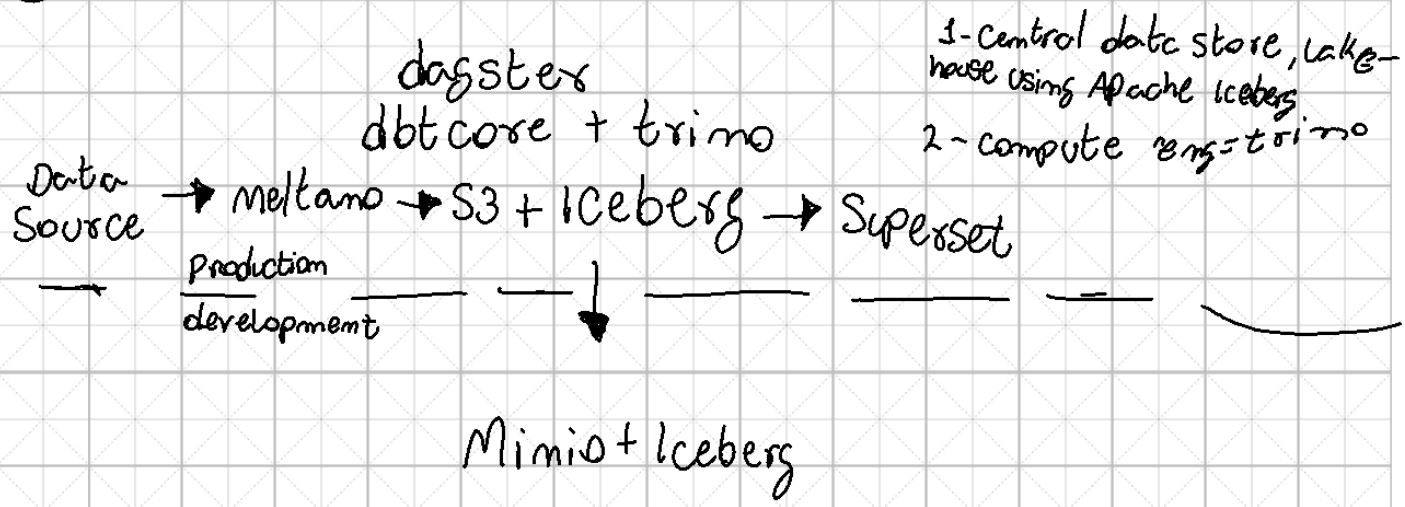
tools -

- Superset
- tableau
- PowerBI
- Looker

what tool you use depends heavily on what your data analysts use

BI tooling is not easy to change once in place

BI Stack v2 - Data Lakehouse



Streaming Stack

unbounded data

Latency where it matters

typical order data

Data Source → (latency: hours) → Data consumer

Amazon

Data Source → (latency: seconds) → Data Consumer

more examples:

click data for real time marketing

IOT device data

Streaming Camera data

Stack -

Stream data SRC → kafka → Flink → S3 + Iceberg → trino → data consumers

Production / Development

minio + Iceberg

Kafka = event broker = is used to ingest events from source systems ("event log")

+

Flink = Streaming Processing engine = it can transform continuous, unbounded streaming events in real time

=

Ingest and transform massive volumes of data at a very low latency

The Basics - ML vs AI vs Deep Learning?

AI = catch all term for a field of study in computer science that seeks to develop machines with "human-like" intelligence

ML = machine learning is a specific field within AI that develops algorithims that can learn from data rather than hand writing

Deep Learning = is a specific type of ML algorithim called neural networks, because it was loosely based on the human brain, it's the most complete kind of ml algorithm, as it can compute anything, it's also called a Universal function

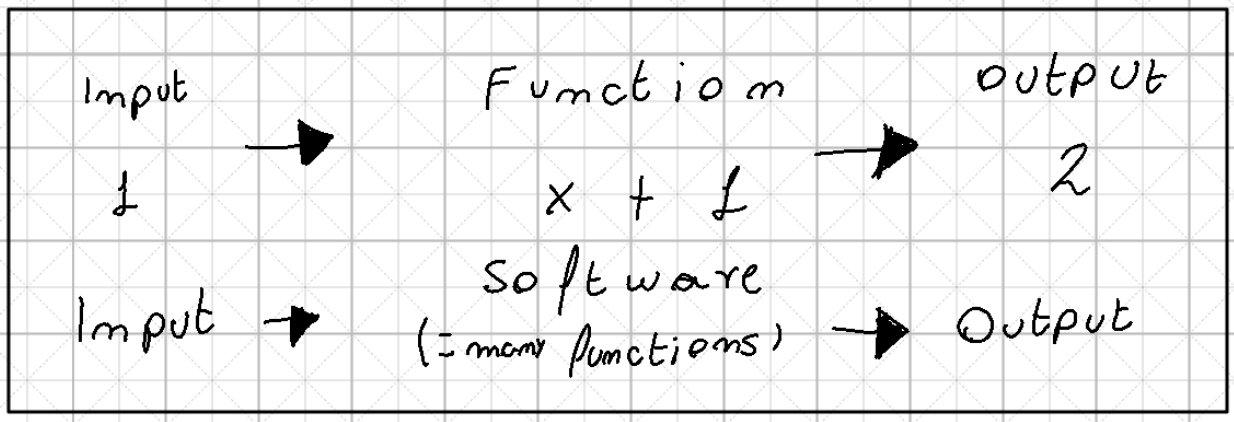
ML vs Deep Learning

ML -
- Classical ml algorithims
- Structured Data
- Small Datasets
- Less compute required (CPU)

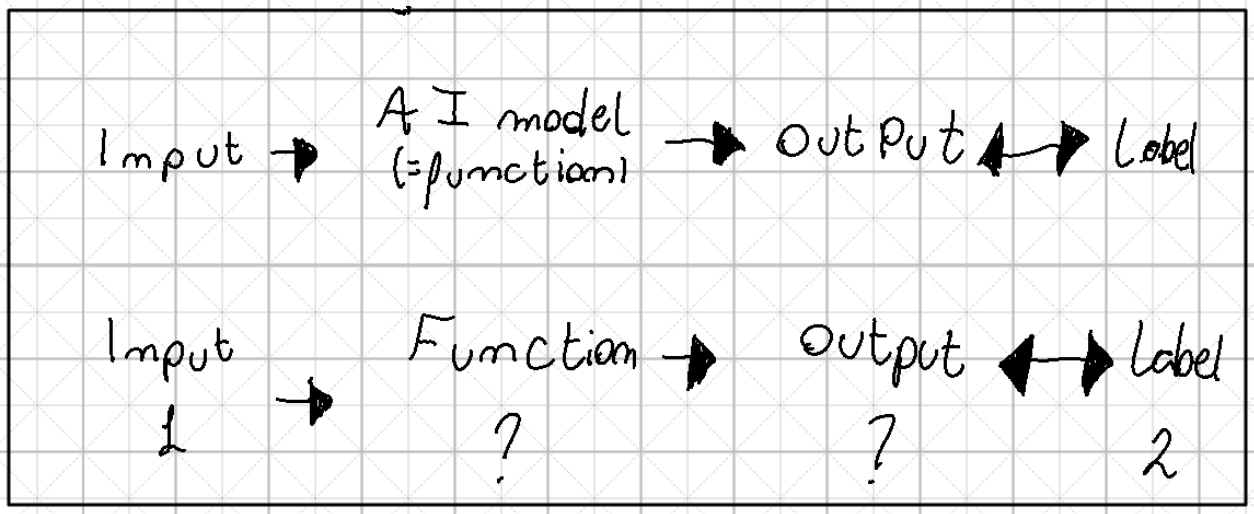
Deep Learning -
- neural networks
- Unstructured data
- Large datasets
- Large compute required (gpu)

How AI models work

In programming



In AI models



Training vs Inference

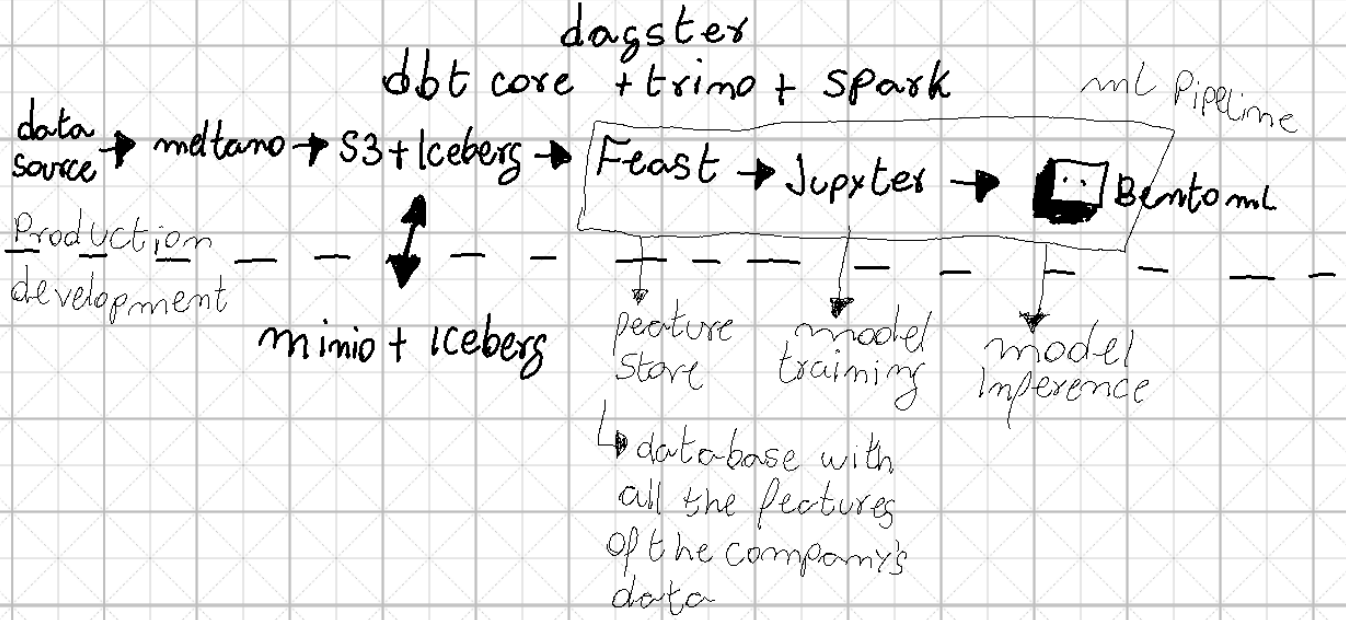
Training (building) * requires lots of data and compute

Input → AI model (?) → Output ↔ Label

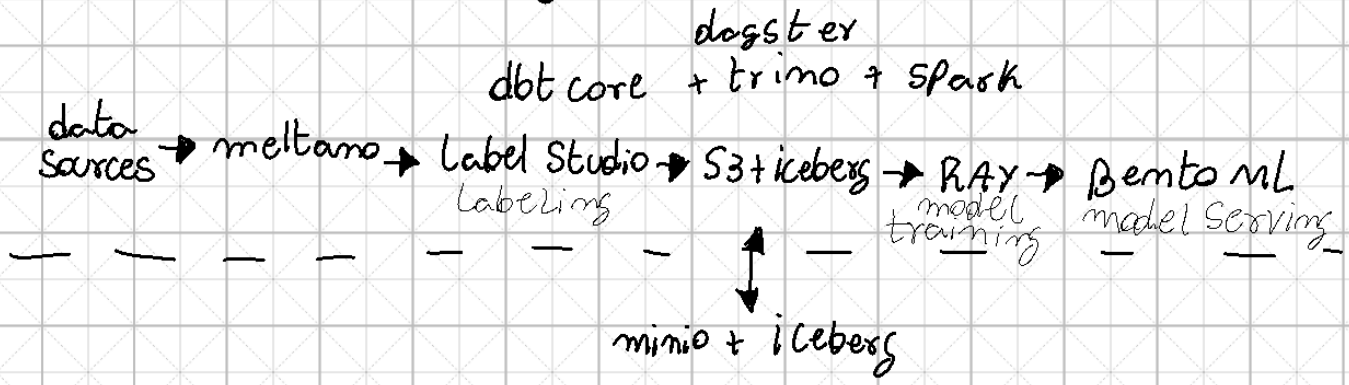
Inference (usage)

Input → AI model (Already trained) → Output

ML Stack



Deep Learning stack



Other Considerations

Data validations

- Type validation
- Constraint validation
- Code validation

*dagster/dbtcore/great expectations

testing

- Unit testing
- Statistical testing

*dagster

observability

- monitoring, alert, notifs
- Upstream schema changes
- Detailed logs and stack traces
- ALWAYS expect failures
- Observe infra, not just pipeline

CI/CD

- Automate CI/CD process
- Automation = more frequent releases

*terraform/Pulumi/Github Actions