Harmonized solution design of data stations

We take Klepmann (2017) as our starting point, who states that “Many applications today are *data-intensive*, as opposed to *compute-intensive*. Raw CPU power is rarely a limiting factor for these applications—bigger problems are usually the amount of data, the complexity of data, and the speed at which it is changing.”

Generically, we want:

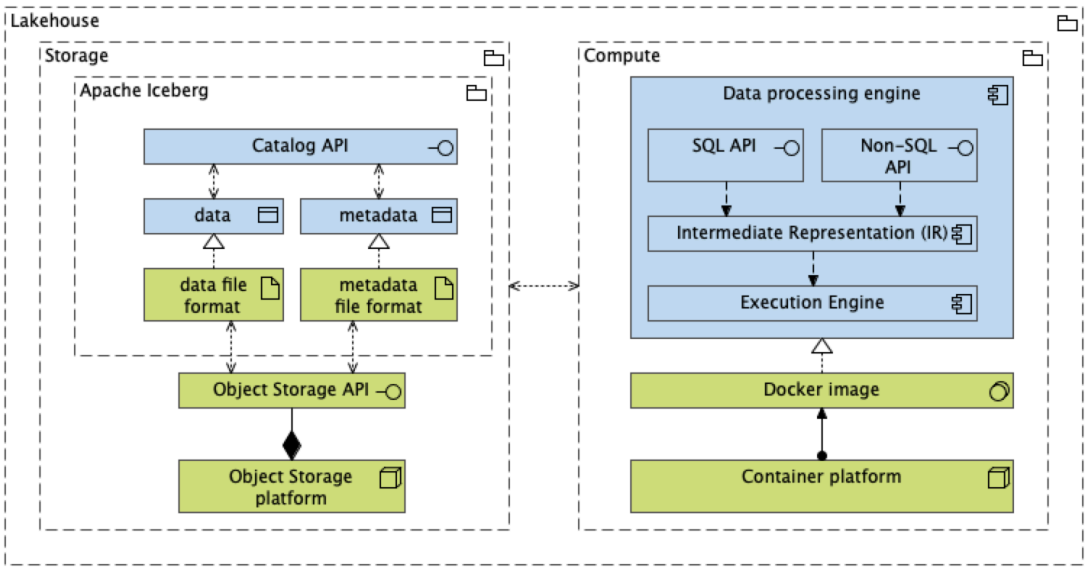
Reliability	Scalability	Maintainability
tolerating hardware & software vaults	Measuring load & performance	Operability, simplicity & evolvability
human error	Latency percentiles, throughput	

We focus on analytical data systems, with different patterns from transactional data systems.

0.1 Detailing the layers of a data station

TO DO: provide detailed layers, and explain how interoperability works across the layers:

- Storage layer (technology): all reference architectures stipulate use of S3-compliant blob storage
- Data and metadata (application): resides in the data station
- We propose to move towards open table formats, that is, Apache Iceberg, whereby storage and compute can be separated



Figuur 1: Solution of a minimal lakehouse that sits at the core of a data station

0.2 Detailing the data conformity zone

TO DO: explain that

- data conformity zone is essentialy a lakehouse pattern
- the architecture of a lakehouse has stabilized and converged towards:
  - **Colum-oriented storage and memory layout:** Apache Arrow ecosystem, including Apache Flight
  - **Late-binding with logical data models most suited for analytics:** ELT pattern with zonal architecture
    - *staging zone:* hard business rules (does incoming data comply to syntactic standard), change data capture
    - *linkage & conformity zone:* concept-oriented tables, typically following a data vault modeling principle, ascertain referential integrity across resources, with tables per concept and linking tables. Mapping to coding systems. Entity resolution for record linkage at the subject level
    - *consumption zone:* convenient standardized views like an event table (patient journey, layout for process mining) with uniformity of dimensions using a star schema

0.3 Detailing the trains

TO DO: explain

- difference between centralized and distributed federated learning (causes lots of confusion)
- basically Train is a generalization of all types of computes
- difference between
  - Train for secondary use, which usually with batch-wise, less strict latency requirements
  - Train for primary use, like API call and messaging, with stricter latency requirements. This also includes deployment of AI for inference

Bibliografie