Apache Kafka

for Java Developers

Reference Architecture and Examples

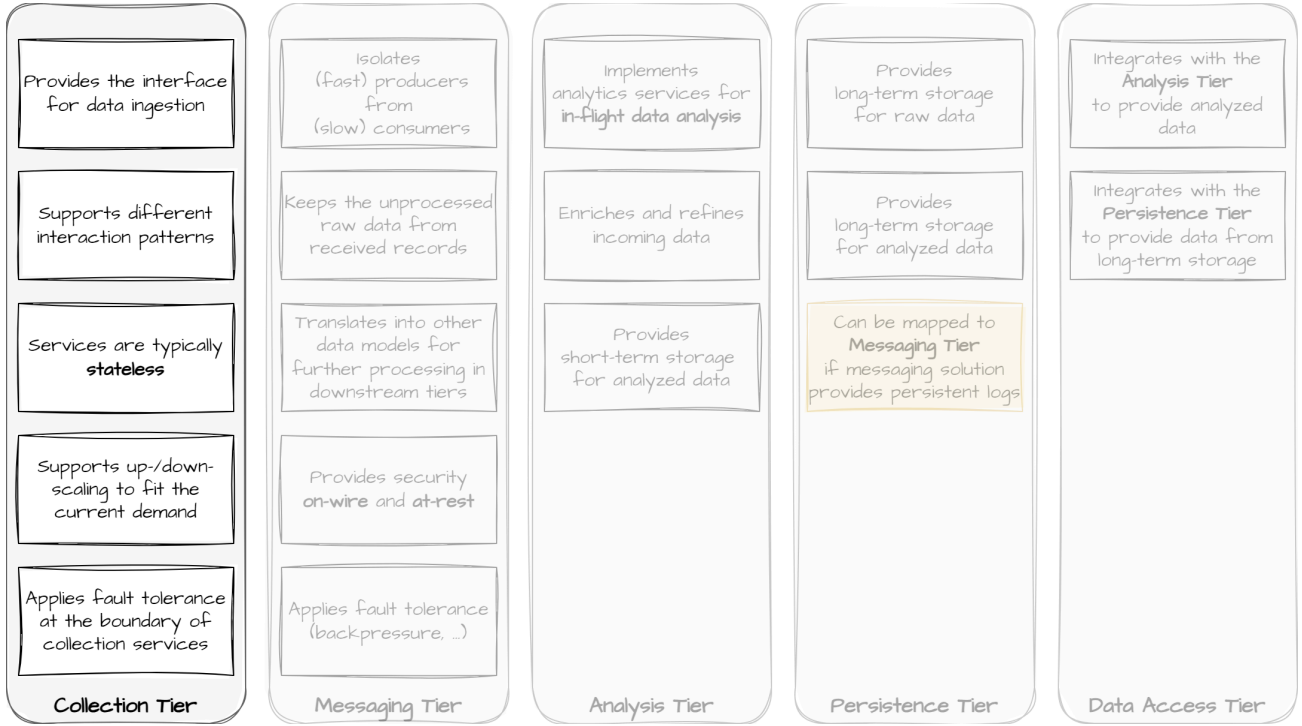
Boris Fresow, Markus Günther

JavaLand 2024, Nürburgring

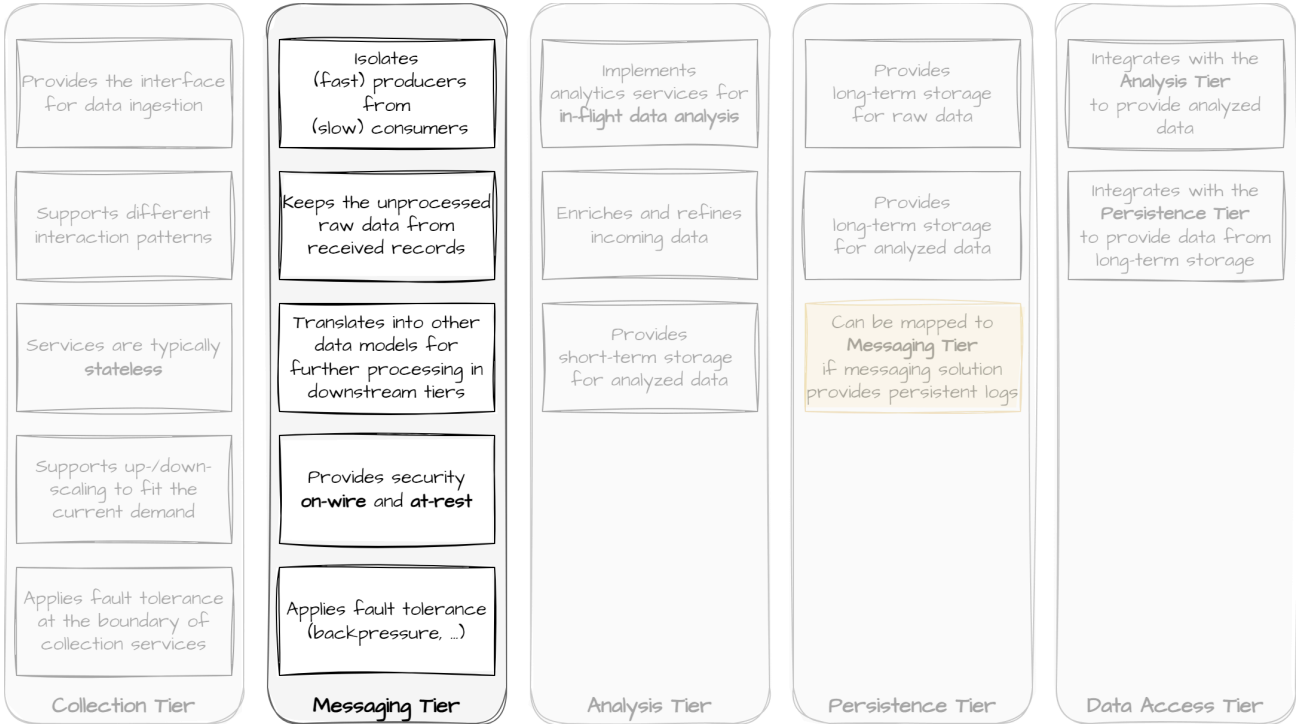
Reference Architecture

for Data Streaming Applications

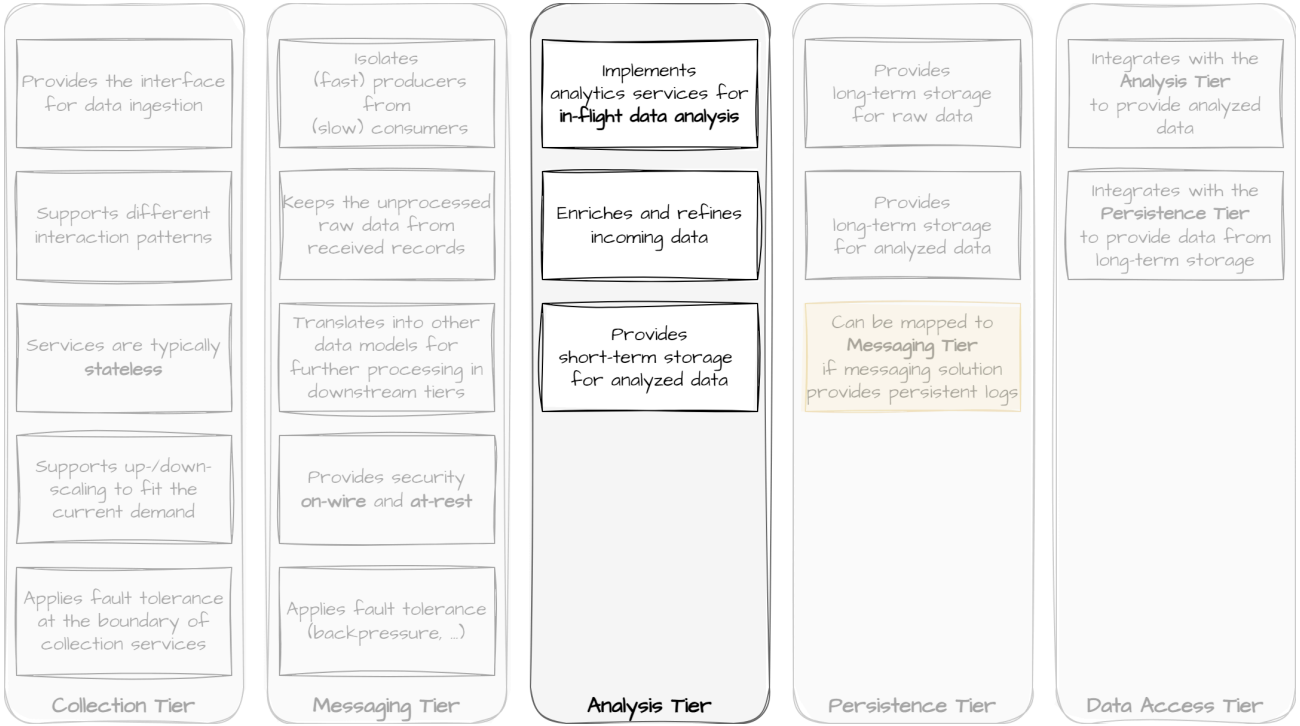
The *Collection Tier* is the entry point for bringing data into our streaming system.



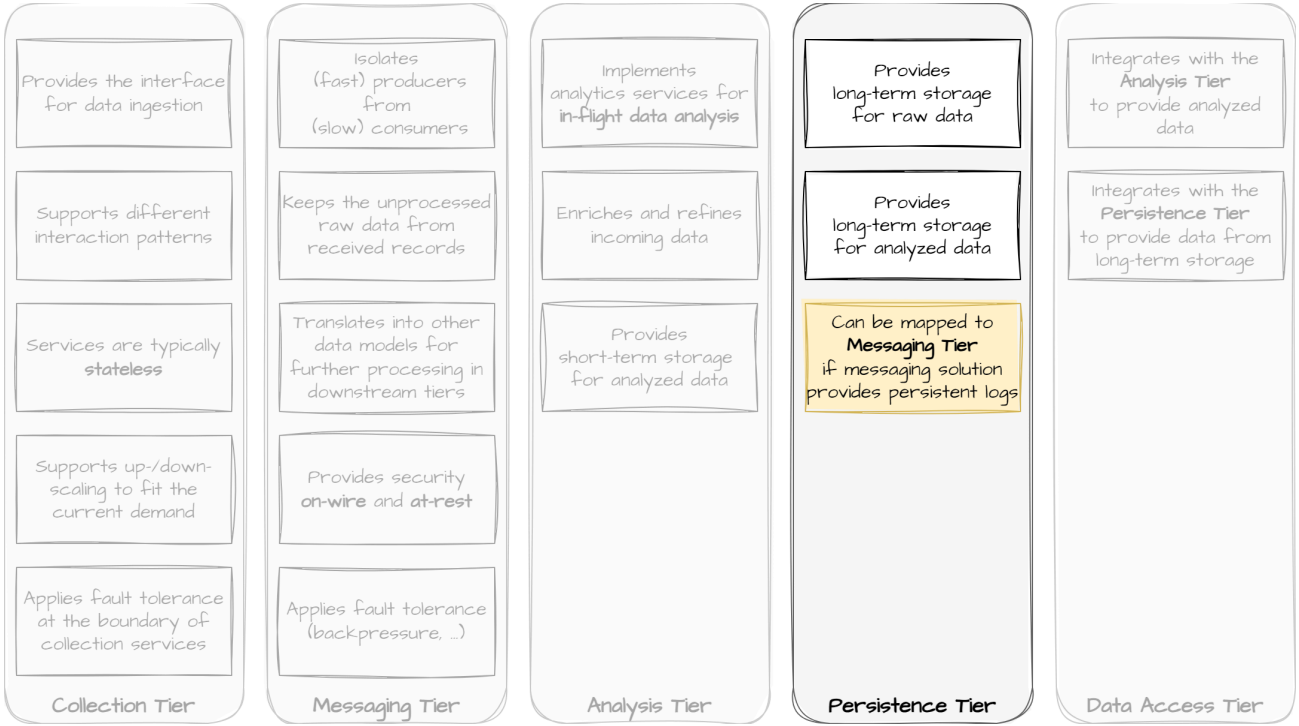
The *Messaging Tier* transports data from the *Collection Tier* to the rest of the pipeline.



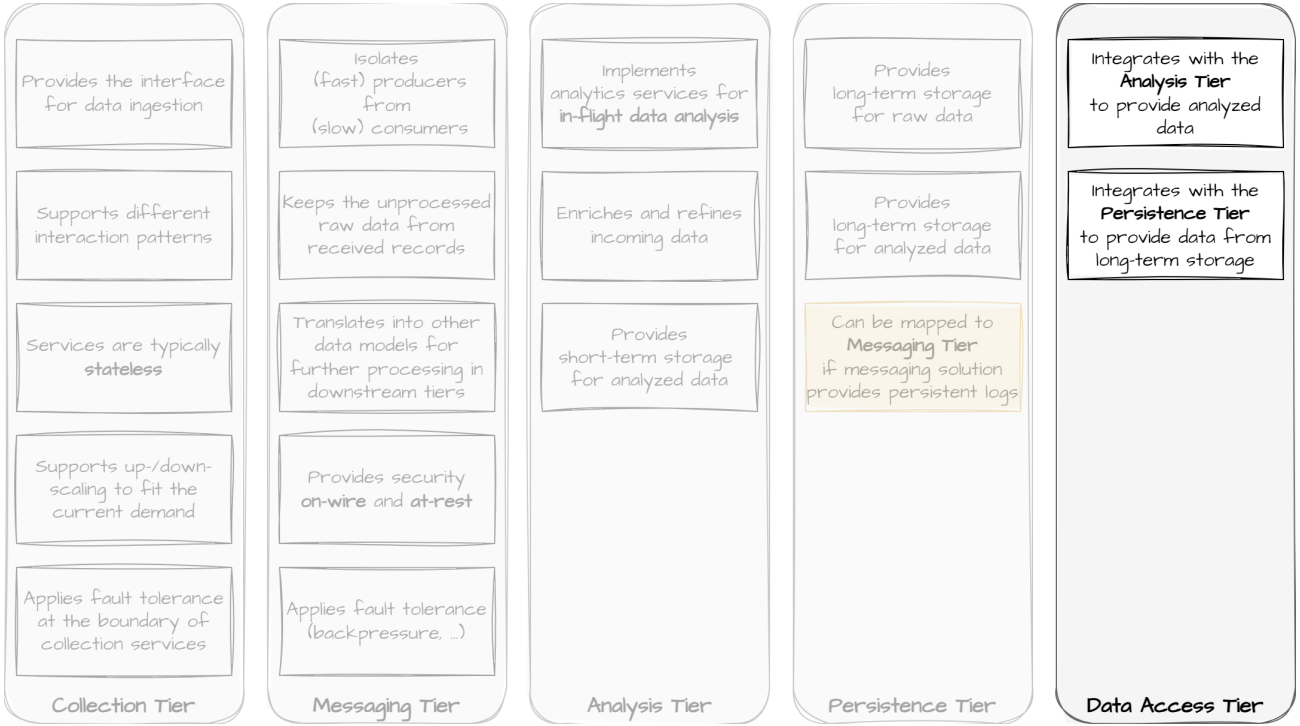
The *Analysis Tier* provides the capability of in-flight data analysis.



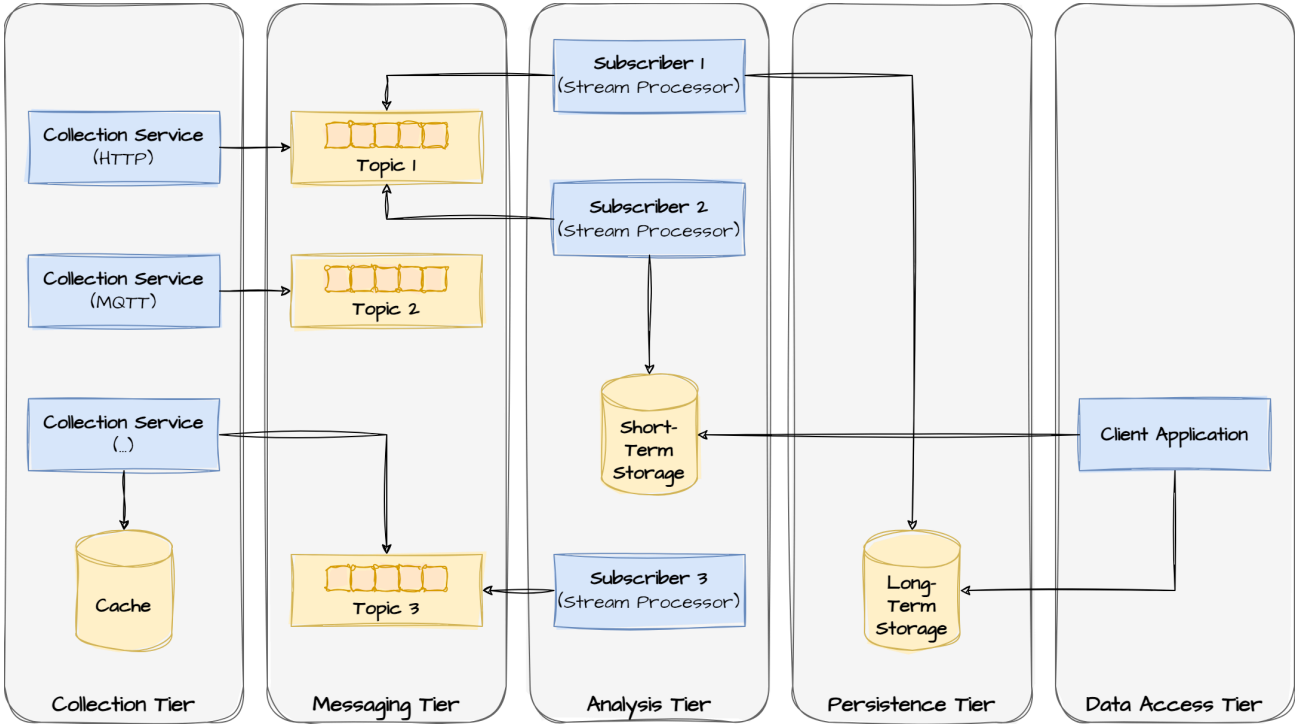
The *Persistence Tier* allows us to store raw, refined, and enriched data for later usage.



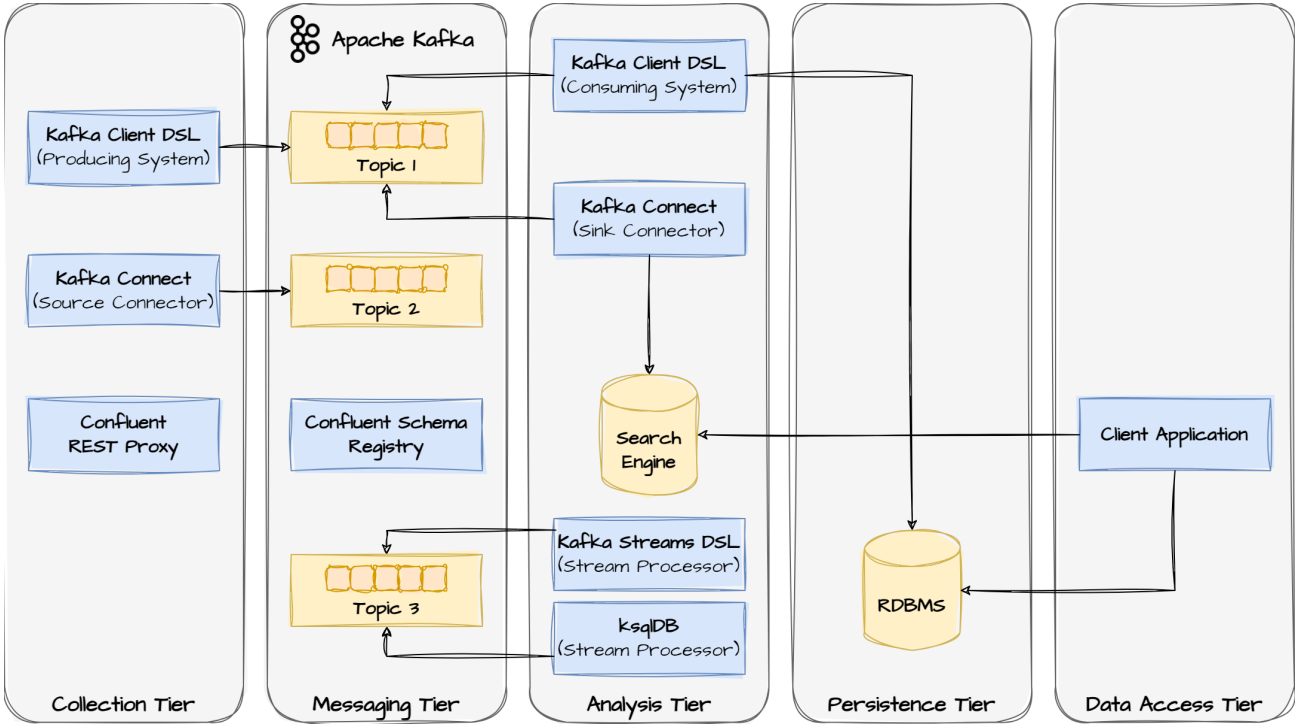
The *Data Access Tier* integrates with *Analysis* and *Persistence Tier* to make data available.



A reference architecture helps to reason about tiers and (non-)functional requirements.



Kafka features a rich ecosystem of services that fit into the tiers of a streaming architecture.



Real-World Examples

Example #1: Efficient persisting and querying of audit trail data and sensor events.

Status

- Data origins at
  - Backend service (HTTP)
  - Sensors in the field (MQTT)
- RDBMS for write- and read-side
  - Optimized for writes, not reads
  - Moderate write requests: ~235 req. / s

Example #1: Efficient persisting and querying of audit trail data and sensor events.

Status

- Data origins at
  - Backend service (HTTP)
  - Sensors in the field (MQTT)
- RDBMS for write- and read-side
  - Optimized for writes, not reads
  - Moderate write requests: ~235 req. / s

Analysis

- Writes introduce heavy load on the DB
- (Non-optimized) Reads perform abysmal

Example #1: Efficient persisting and querying of audit trail data and sensor events.

Status

- Data origins at
  - Backend service (HTTP)
  - Sensors in the field (MQTT)
- RDBMS for write- and read-side
  - Optimized for writes, not reads
  - Moderate write requests: ~235 req. / s

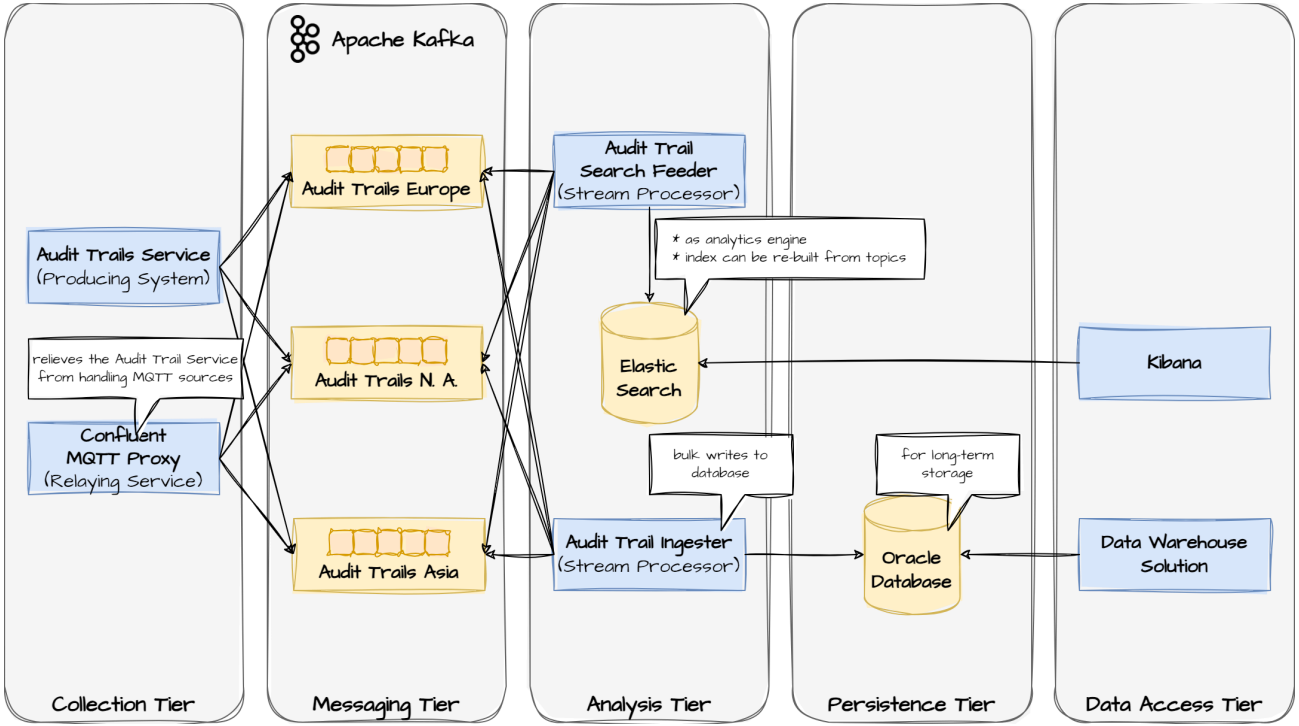
Analysis

- Writes introduce heavy load on the DB
- (Non-optimized) Reads perform abysmal

Proposal and implementation

- Decouple write- from read-side
- Use optimized store for reads

Example #1: Efficient persisting and querying of audit trail data and sensor events.



Example #2: Collecting and analyzing sensor data for alerting in near real-time

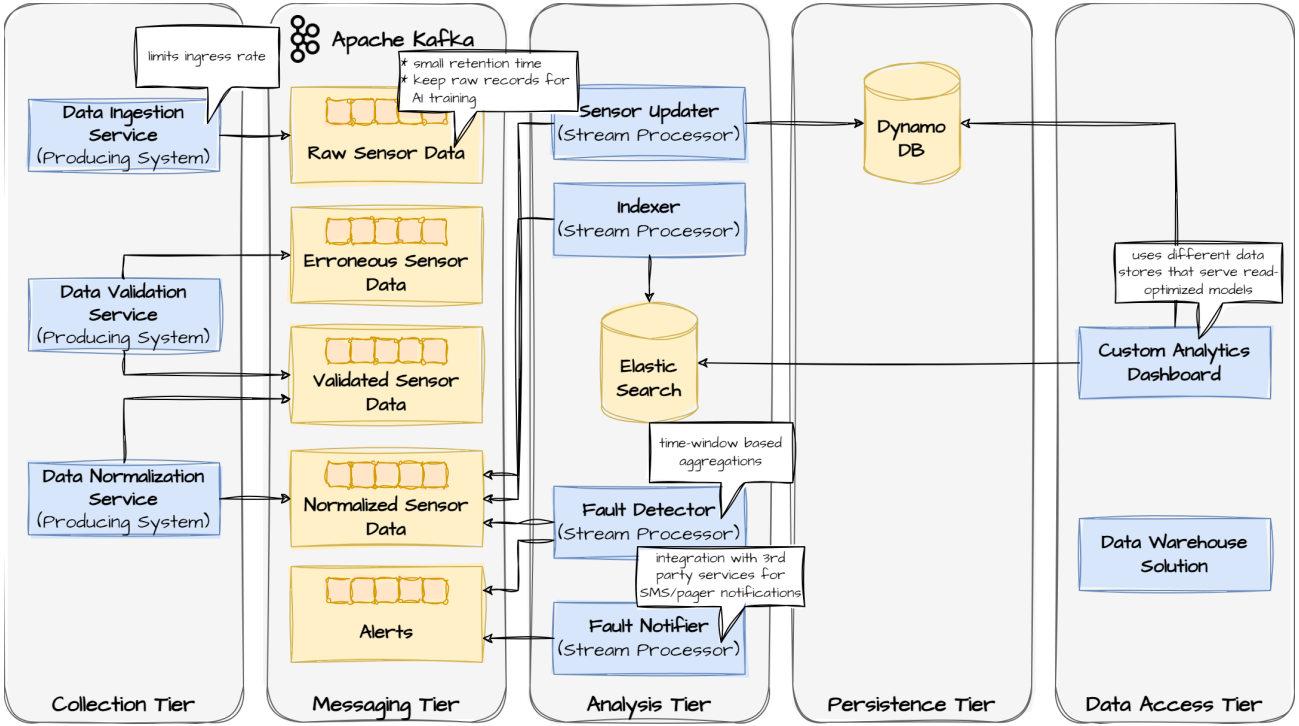
Project

- Sensors periodically push health data
- Single machine houses 10-50+ sensors
  - i machines per site
  - j sites per tenant
  - k tenants that need supervision
- Near real-time processing to detect errors
- Manufacturers use different data models

Status

- Existing system does not scale
  - due to used broker
  - due to software design
- Services / tiers intermixed
  - hard to optimize for non-functional requirements
- **Proposal: Redesign core messaging**

Example #2: Collecting and analyzing sensor data for alerting

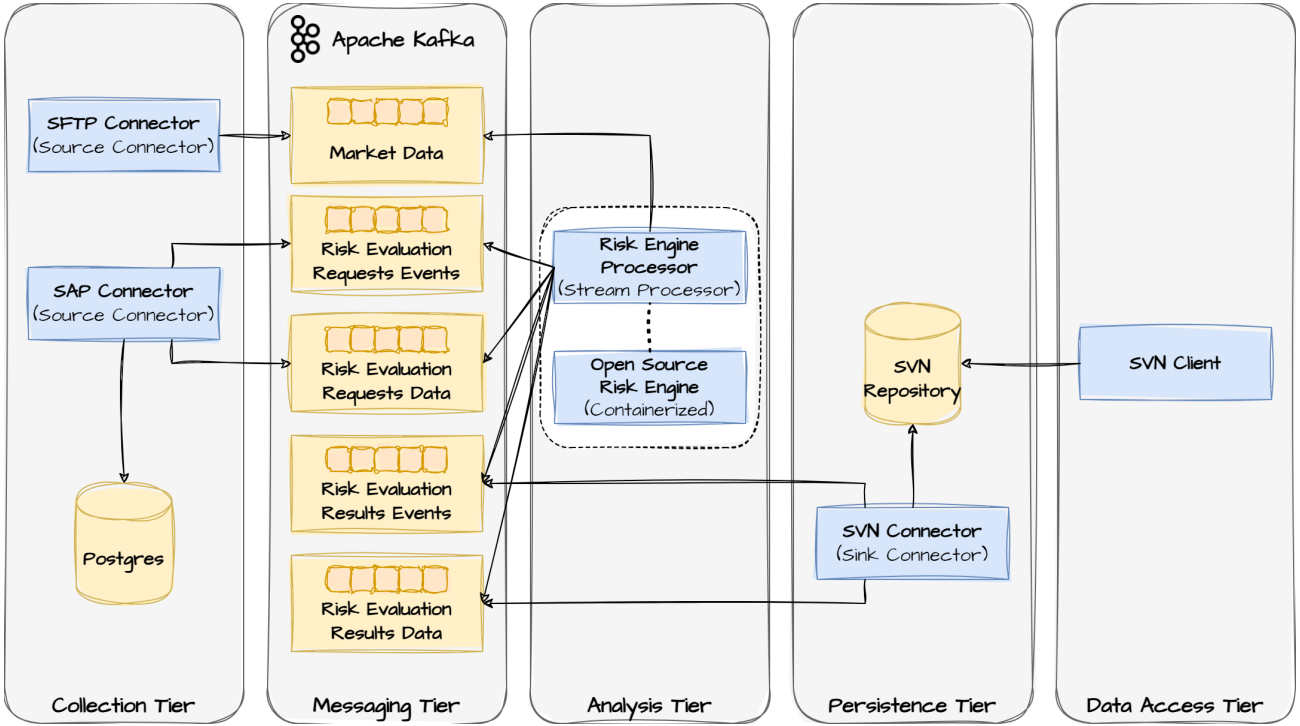


Example #3: On-the-fly risk estimation on mortgages (greenfield)

Project

- Financial risk assesors evaluate the risk on mortgages using software tools
  - given current market data
  - given current the property in question
  - given the investor's / house builder's financial situation
- Necessary data channels do not have streaming capabilities (SAP, ...)
- Risk evaluation software is a C++ application with manual feed-in of data

Example #3: On-the-fly risk estimation on mortgages (greenfield)



Questions?