Datenmanagement
Datenverarbeitungsarchitekturen / Data Streaming

Prof. Dr. Markus Grüne, FB03
Wirtschaftsinformatik

Lernziele

Sie wissen, wie Streaming-Plattformen architektonisch aufgesetzt sind.

Sie kennen architektonische Muster wie die Lambda und Kappa-Architektur und können beurteilen, wann welche eingesetzt werden kann.

Sie wissen, wie die Architekturen mit dem Begriff "Complex Event Processing" zusammenhängen.

Von ETL zu modernen Architekturen

Alle Software-Architekturen haben Trade-Offs.

Die Entscheidung über die Wahl einer Architektur hängt von diesen Trade-Offs ab.

ETL-Pipelines führen zu Latenzen, nächtlichen Verarbeitungs-Jobs. Business Intelligence-Werkzeuge sind für die Geschwindigkeit der heutigen Datenverarbeitung oft nicht angemessen.

Eine Idee wäre es, analytische und transaktionsverarbeitende Datenbanken zusammen zu fassen, um Geschwindigkeit zu gewinnen.

Warum benötigen wir neue Architekturen?

Say we wanted to count tweet impressions. Furthermore, not only do we want real-time updates as users are tapping, swiping, and clicking right now, but we want historic counts dating back to the moment a tweet was posted (for example, consider a tweet by Donald Trump last year that's receiving a new burst of engagement).

In Twitter, impression data are captured by frontend logs. After a multistage aggregation pipeline, the logs are deposited into a Hadoop data warehouse (…)

In the lambda architecture at Twitter, circa 2012, the batch processing layer was MapReduce, where such aggregations were already routine.

However, the logging pipeline introduced a delay: even in the best possible case, logs were a few hours old (…). This meant that a dashboard of tweet impressions powered by MapReduce alone would always be a few hours out of date. Not good enough!

Quelle: Lin, Jimmy (2017): The Lambda and the Kappa. In: *IEEE Internet Comput.* 21 (5), S. 60–66. DOI: 0.1109/MIC.2017.3481351

Warum benötigen wir neue Architekturen?

Ah, but that's when the real-time layer comes in — which in this case was Storm, a framework for real-time processing that provides a dataflow graph abstraction (…)

Storm performed real-time aggregations, and a merging layer encapsulated the logic to integrate batch and real-time results.

Once the (delayed) results from the batch layer arrived, results from the real-time layer could be discarded.

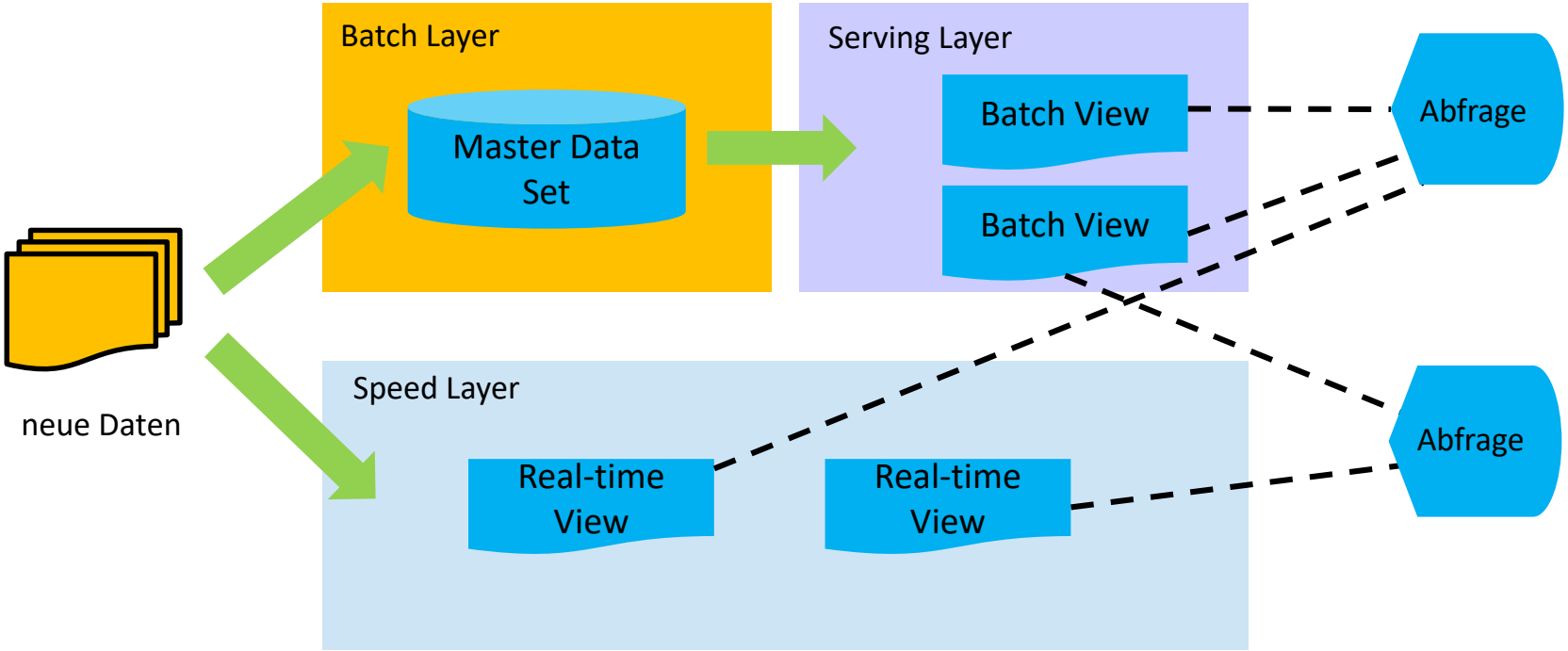
In other words, the batch computations provided truth, while the real-time results were transient.

Quelle: Lin, Jimmy (2017): The Lambda and the Kappa. In:
IEEE Internet Comput. 21 (5), S. 60–66. DOI: 0.1109/MIC.2017.3481351

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

LAMBDA-ARCHITEKTUR

Lambda-Architektur (bis etwa 2011)



In Anlehnung an: https://www.databricks.com/glossary/lambda-architecture

Lambda-Architektur

Neue Daten kommen kontinuierlich an.

Z.B. als Batch, Micro-Batch oder real-time. Diese werden simultan in die Batch Layer und Speed Layer geladen.

Die Batch Layer übernimmt folgende Aufgaben:

- Management der Masterdaten und Korrektur der Speed Layer
- Vorberechnung der Batch Views

Die Speed Layer (Stream Layer) übernimmt Aufgaben der Real-time-Verarbeitung.

- Anzeige der Daten, die noch nicht in der Batch-Layer verarbeitet wurden.
- Die Speed Layer arbeitet nur auf kürzlich eingetroffenen Daten.

Die Serving Layer stellt die Ergebnisse der Verarbeitungsstufen bereit und sorgt für eine Low-Latency-Darstellung der Batch Views.

Bewertung der Lambda-Architektur

Vorteile

Historische und aktuelle Kennzahlen können berechnet werden.

Business Agility – Reaktion auf Marktereignisse wird schnell möglich.

Die Konsistenz wird letztlich durch die Batch Layer gesichert.

Skalierbarkeit, Ausfallsicherheit.

Nachteile

Komplexität: Hohe Kosten durch die doppelte Entwicklung von Funktionen für die Speed und Batch Layer. Teils sehr unterschiedlich.

Zwei Implementierungen müssen dauerhaft unterhalten werden. Oft von getrennten Teams.

Aggregatwerte können sich bei Downtimes ändern.

Einsatzmöglichkeiten Lambda-Architektur

- Geeignet für ein breites Spektrum von Datenverarbeitungsaufgaben.
  - Verarbeitung großer Datenmengen
  - Bereitstellung von Abfrageergebnissen mit geringer Latenz
  - Echtzeit-Analyseanwendungen wie Dashboards und Berichte
  - Batch-Verarbeitungsaufgaben wie Datenbereinigung, -umwandlung und -aggregation
- Anwendungsfelder
  - Stream-Verarbeitungsaufgaben wie Ereignisverarbeitung
  - maschinelle Lernmodelle
  - Anomalieerkennung
  - Betrugserkennung
  - Gut für die Verarbeitung der von IoT-Geräten erzeugten großen Datenströme eignen.

Häufig für den Aufbau / das Befüllen von Data Lakes verwendet.

KAPPA-ARCHITEKTUR

Kappa-Architektur

Die Architektur wurde 2014 von Jay Kreps, einem Autor von Kafka vorgestellt.

Alles ist ein Stream → nur eine Engine wird für das Stream Processing benötigt. (Streaming first!)

Statt Batch Processing der Lambda-Architektur hier "Streaming" historischer Daten.

Ein Stream ist eine Collection, an die nur Elemente angehängt werden können (log).

Der Log drückt alle Updates auf die Tabelle aus.

Die Tabelle ist eine Abstraktion auf die Logs (mit weiteren technischen Details).

Kappa-Architektur - Layers

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Speed Layer (Stream Layer)

- Erfassung, Verarbeitung und Speicherung von Live-Streaming-Daten in großer Menge
- wird in der Regel mit einer Stream-Processing-Engine wie Apache Flink, Apache Storm, Apache Kinesis, Apache Kafka implementiert
- zuverlässiger Zugriff auf Abfrageergebnisse.

Die Speed Layer in der Kappa-Architektur ist in zwei Hauptkomponenten unterteilt: die Ingestion Component und die Processing Component.

Ingestion Component

- Erfassung eingehender Daten und Speicherung von Rohdaten wie Protokolldateien, Sensoren und APIs
- Daten werden in einem verteilten Datenspeicher gespeichert, z. B. in einer Nachrichtenwarteschlange oder einer NoSQL-Datenbank.

In Anlehnung an: https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/

Kappa-Architektur - Layers

Processing Component

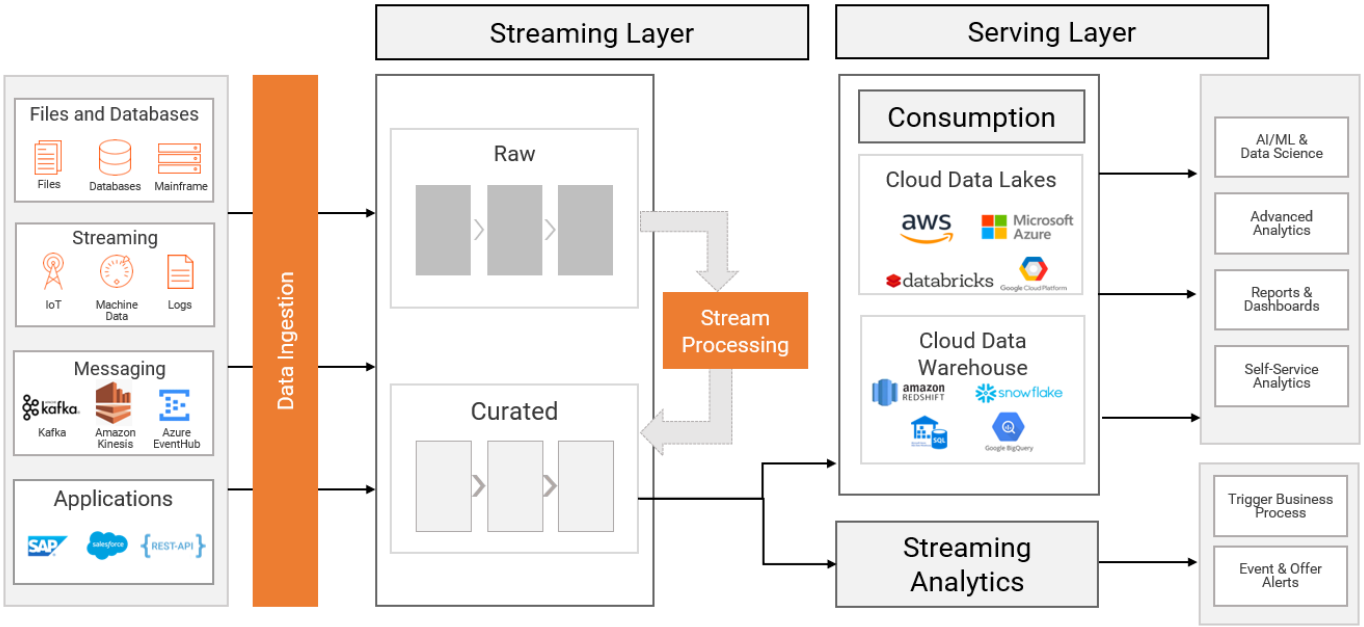
- Verarbeitung der ankommenden Daten und die Speicherung der Ergebnisse in einem verteilten Datenspeicher
- In der Regel mit einer Stream-Processing-Engine wie Apache Flink oder Apache Storm implementiert

Ziel: große Datenströme verarbeiten und schneller Zugriff auf Abfrageergebnisse

Die Serving Layer stellt weiterhin die Datenzugriffe bereit

In Anlehnung an: https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/

Beispiel einer Kappa-Architektur



Stream Processing-Engine (Spark / Flink usw.) liest Daten aus Messaging-System, wandelt sie um und published die angereicherten Daten zurück im Messaging-System für Echtzeitanalysen. Daten werden auch an Serving Layer verteilt (Cloud Data Lake, ein Cloud Data Warehouse…). Hierauf setzen die Analysesysteme auf

Quelle: https://www.informatica.com/blogs/adopt-a-kappa-architecture-for-streaming-and-ingesting-data.html

Bewertung der Kappa-Architektur

Vorteile

Recherchieren Sie!

Nachteile

Recherchieren Sie!

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

COMPLEX EVENT PROCESSING

Complex Event Processing

Die Lambda- und Kappa-Architekturen verarbeiten Datenströme.

Diese können abstrakt als Events (Ereignisse) aufgefasst werden.

Das Complex Event Processing verarbeitet Ströme von Ereignissen.

Ziel der Verarbeitung ist es, rechtzeitig auf die Ereignisse reagieren zu können.

Da Ereignisse per se nicht "komplex" sein können, wäre der Begriff "Composite Event Processing" eigentlich besser.

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Complex Event Processing

System: Menge von Objekten, die Attribute (Eigenschaften) haben können und die untereinander in Beziehung stehen können.

Ereignis: verändert die Menge der Objekte im System oder mindestens einen Attributwert eines Objekts oder eine Beziehung zwischen Objekten. Geschehnis, das keine Realzeit in Anspruch nimmt, sondern immer in einem Zeitpunkt eintritt.

Der Zustand eines dynamischen Systems setzt sich im Allgemeinen aus mehreren Zustandsgrößen zusammen:

- diskrete Zustandsgrößen, bei denen sich die Zustandswerte in diskreten Zeitpunkten ändern
- kontinuierliche Zustandsgrößen, bei denen sich die Zustandswerte kontinuierlich ändern

Beispiel: Zimmer mit den diskreten Zustandsgrößen *Fensterzustand* ("auf", "zu") und *Anzahl der Personen im Zimmer* sowie mit der kontinuierlichen Zustandsgröße *Temperatur.*

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Beispiel für ein elementares Ereignis

Beispiel	Interpretation
Ereignistyp: Kursänderung	Klasse von gleichartigen Ereignissen.
Zeitstempel: 2023-04-23 10:10:09	Eintrittsdatum / Signal-Ankunft
Ereignis-ID: 106342	Individuelle ID

Name: xyzCompany
Einkaufskurs: 32,5
Letzter Kurs: 40,8
Aktueller Kurs: 41,1

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Komplexes Ereignis, Ereignisstrom

Ein komplexes Ereignis ist

- eine endliche Menge von atomaren Ereignissen,
- die passend zu einem vorgegebenen Ereignismuster zueinander in Beziehung stehen.

In einem Ereignisstrom kann es

- mehrere komplexe Ereignisse geben,
- die zu einem Ereignismuster passen.

Komplexe Ereignisse werden auch als Ereignisinstanz bezeichnet, manchmal auch als Ereignismusterinstanz oder kurz Musterinstanz (engl. im Zusammenhang mit patterns auch als match).

Im Spezialfall kann ein komplexes Ereignis eine einelementige Menge sein, dann besteht es aus einem atomaren Ereignis.

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Ereignisverarbeitungssprachen

Für die Beschreibung von Ereignismustern verwendet man Beschreibungssprachen, die als Event Pattern Language oder Event Processing Language bezeichnet werden.

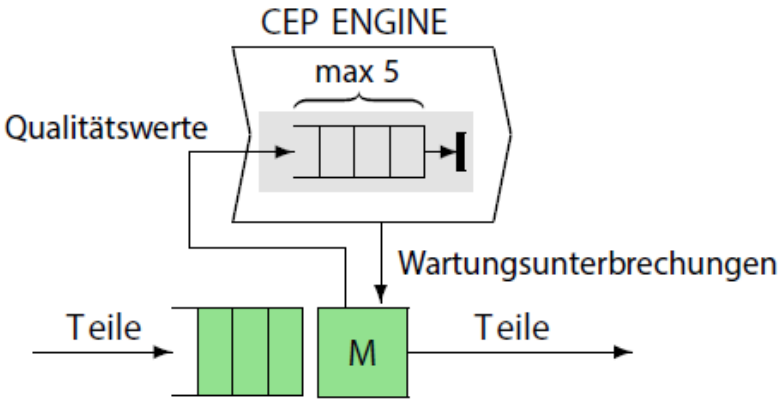
- Nahezu jede Event Processing Language verfügt über die Kern-Operatoren Sequenz (zeitbasiert), Konjunktion, Disjunktion und Negation.
- Zusätzlich gibt es meist Operatoren für Wiederholungen sowie spezifische Funktionen bezogen auf die Attributwerte.

Semantik von Ereignismuster-Operatoren:

- Bei der Zeitintervall-Semantik wird einem CE das Zeitintervall zugeordnet, das durch das früheste und das späteste atomare Ereignis der Ereignismenge festgelegt ist.
- Die Zeitpunkt-Semantik ordnet einem CE den Zeitpunkt des spätesten zugehörigen atomaren Ereignisses zu. Auf diese Weise wird auch ein komplexes Ereignis mit einem Eintrittszeitpunkt in Verbindung gebracht.

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

CEP am Beispiel Maschinenwartung
Predictive Maintenance



Maschine M bearbeitet Teile und leitet sie anschließend weiter.

Wenn 2 der letzten 5 bearbeiteten Teile schlechter Qualität waren, wird vor der Bearbeitung des nächsten Teils eine Wartung der Maschine durchgeführt.

Zur Speicherung der Information über die Qualität der Teile wird eine Queue mit Kapazität 5 verwendet.

Für jedes bearbeitete Teil wird ein Qualitätswert in diese Queue eingefügt, bei voller Queue wird vorher der vorderste Eintrag der Queue entfernt

Die Entscheidung, ob gewartet werden muss, erfolgt immer erst, wenn mindestens 5 Teile gefertigt worden sind.

Qualitätswerte werden jeweils im Zeitpunkt eines Bedienungsende-Ereignisses als atomares Ereignis zur CEP Engine geschickt.

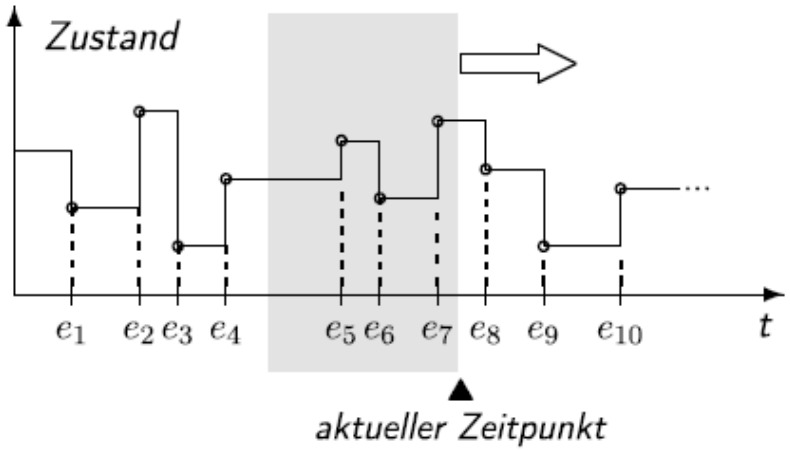
Falls Wartung notwendig ist, wird die Bearbeitung gestoppt und eine Wartung durchgeführt. Andernfalls wird aus dem Teilepuffer das nächste Teil entfernt und bearbeitet.

Die Arbeit der CEP Engine besteht darin, in dem eingehenden Strom von Qualitätswert-Ereignissen ein ganz bestimmtes Muster zu erkennen.

Sind die Qualitätswerte „gut“ mit 0 und „schlecht“ mit 1 codiert, so muss einfach überprüft werden, ob zwei Qualitätswerte mit dem Wert 1 vorliegen.

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Auswertung von Ereignisströmen mittels "Fenstern"



gleitendes Zeitfenster

Ereignisströme sind kontinuierlich

Daher bei Suche nach einem Ereignismuster den Suchraum auf eine endliche Teilmenge beschränken; meist mittels gleitenden Längen- oder Zeitfenstern (engl. Sliding Window).

Gleitendes Längenfenster: feste Anzahl von Ereignissen überprüfen. Vor Einfügen eines neuen Ereignisses wird ältestes Ereignis entfernt.

Gleitendes Zeitfenster: feste Zeitdauer; wird auf Zeitachse entlang geschoben. Ereignisse werden berücksichtigt, die im aktuellen Zeitfenster „sichtbar“ sind.

Ein Nachfolgeereignis wird durch die CEP Engine oftmals dann initiiert, wenn innerhalb eines gleitenden Längen- oder Zeitfensters ein Schwellenwert (engl. threshold) unter- oder überschritten wird.

In Anlehnung an: Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

Schlussbetrachtung CEP

Das CEP kann als Erklärungskonzept zur Analyse vieler Architekturmuster verwendet werden.

Lambda und Kappa-Implementierung setzen das Event Processing in unterschiedlicher Form um.

Lernziel / Fragen

Sie sind mit den wesentlichen Konzepten der Lambda- und Kappa-Architekturen vertraut, die in heutigen Anwendungslandschaften eingesetzt werden.

✓ Lambda-Architekturen wurden bis in die 2010er durchgängig eingesetzt. Sie bieten Vorteile bei Reporting-Anwendungen, die klassische BI-Strukturen nachempfinden.

✓ Kappa-Architekturen sind leichtgewichtiger und wartungsärmer. Sie benötigen jedoch leistungsfähige Datenseen, um die generierten Daten zu speichern. Oft verarbeiten Sie die ankommenden Daten zweimal, um klassische Reporting-Lösungen nachzuempfinden.

✓ Die Konzepte der Architekturen können mit Hilfe der Elemente des CEP beschrieben werden.

Literatur

Lin, Jimmy (2017): The Lambda and the Kappa. In: IEEE Internet Comput. 21 (5), S. 60–66. DOI: 0.1109/MIC.2017.3481351

Hedtstück, Ulrich (Hg.) (2020): Complex Event Processing. Berlin, Heidelberg: Springer Berlin Heidelberg.

https://www.informatica.com/blogs/adopt-a-kappa-architecture-for-streaming-and-ingesting-data.html

https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/