ETL IS DEAD; LONG-LIVE STREAMS

Neha Narkhede,
Co-founder & CTO, Confluent
Data and data systems have really changed in the past decade
Old world: Two popular locations for data

Operational databases

Relational data warehouse
Several recent data trends are driving a dramatic change in the ETL architecture
"#1: Single-server databases are replaced by a myriad of **distributed** data platforms that operate at **company-wide** scale"
#2: Many more types of data sources beyond transactional data - logs, sensors, metrics...
#3: Stream data is increasingly ubiquitous; need for faster processing than daily
The end result? This is what data integration ends up looking like in practice.
A giant mess!
we will see how transitioning to streams cleans up this mess and works towards...
Streaming platform

- App
- App
- App
- App

- changelogs
- search
- NoSQL
- DWH
- Hadoop

- request-response
- monitoring
- security
- messaging or stream processing
- streaming data pipelines
A SHORT HISTORY OF DATA INTEGRATION
Surfaced in the 1990s in retail organizations for analyzing buyer trends
Extract data from databases
Transform into destination warehouse schema
Load into a central data warehouse
BUT ... ETL tools have been around for a long time, data coverage in data warehouses is still low! WHY?
ETL has drawbacks
#1: The need for a global schema
#2: Data cleansing and curation is manual and fundamentally error-prone
#3: Operational cost of ETL is high; it is slow; time and resource intensive
#4: ETL tools were built to narrowly focus on connecting databases and the data warehouse in a batch fashion
Early take on real-time ETL = Enterprise Application Integration (EAI)
EAI: A different class of data integration technology for connecting applications in real-time
EAI employed Enterprise Service Buses and MQs; weren't scalable
ETL AND EAI ARE OUTDATED!
Old world: scale or timely data, pick one

- **EAI**: Real-time but not scalable
- **ETL**: Scalable but batch
Data integration and ETL in the modern world need a complete revamp
NEW WORLD: STREAMING, REAL-TIME AND SCALABLE

- **EAI**: Real-time but not scalable
- **ETL**: Scalable but batch
- **Streaming Platform**: Real-time and scalable

Diagram:
- **Real-time** vs. **Scale**
  - EAI: Real-time but not scalable
  - ETL: Scalable but batch
  - Streaming Platform: Real-time and scalable
Modern streaming world has new set of requirements for data integration
#1: Ability to process high-volume and high-diversity data
#2 Real-time from the grounds up; a fundamental transition to event-centric thinking
Event-Centric Thinking

web app → Streaming Platform → Hadoop

"A product was viewed"
Event-Centric Thinking

- mobile app
- web app
- APIs

"A product was viewed"

Streaming Platform

Hadoop
Event-Centric Thinking

Streaming Platform

- mobile app
- web app
- APIs

"A product was viewed"

- Security
- Hadoop
- Monitoring
- Rec engine
Event-centric thinking, when applied at a company-wide scale, leads to this simplification ...
Streaming platform

- DWH
- Hadoop
- App
  - request-response
  - changelogs
  - messaging or stream processing
  - streaming data pipelines
#3: Enable forward-compatible data architecture; the ability to add more applications that need to process the same data ... differently
To enable forward compatibility, redefine the T in ETL:

Clean data in; clean data out
#1: Extract as unstructured text

#2: Transform 1 = data cleansing = "what is a product view"

#3: Load into DWH

#4: Transform 2 = drop PII fields
#1: Extract as unstructured text

#2: Transform 1 = data cleansing = "what is a product view"

#3: Load cleansed data

#4: Transform 2 = drop PII fields

DWH

Cassandra
#1: Extract as structured product view events

#2: Transforms = drop PII fields

#4.1 Load product view stream

#4.2 Load filtered product view stream

#4: Load filtered product view stream

Streaming Platform

DWH

Cassandra
To enable forward compatibility, redefine the $T$ in ETL: Data transformations, not data cleansing!
#1: Extract once as structured product view events

#2: Transform once = drop PII fields" and enrich with product metadata

#4.1: Load product views stream

#4.2: Load filtered and enriched product views stream

#4: Load filtered and enriched product views stream
Forward compatibility = Extract clean-data once; Transform many different ways before loading into respective destinations ... as and when required
"In summary, needs of modern data integration solution? Scale, diversity, latency and forward compatibility"
Requirements for a modern streaming data integration solution

- Fault tolerance
- Parallelism
- Latency
- Delivery semantics
- Operations and monitoring
- Schema management
DATA INTEGRATION: PLATFORM VS TOOL

Central, reusable infrastructure for many use cases

one-off, non-reusable solution for a particular use case
New shiny future of ETL: a streaming platform
Streaming platform serves as the central nervous system for a company's data in the following ways ...
#1: Serves as the real-time, scalable messaging bus for applications; no EAI
#2: Serves as the source-of-truth pipeline for feeding all data processing destinations; Hadoop, DWH, NoSQL systems and more
#3: Serves as the **building block** for stateful **stream processing** microservices
Streaming

Batch data integration
"Streaming
Batch ETL"
a short history of data integration

drawbacks of ETL

needs and requirements for a streaming platform

new, shiny future of ETL: a streaming platform

what does a streaming platform look like and how it enables Streaming ETL?
Apache Kafka: a distributed streaming platform
Apache Kafka 6 years ago
> 1,400,000,000,000,000 messages processed / day
Now Adopted at 1000s of Companies Worldwide
what role does Kafka play in the new shiny future for data integration?
#1: Kafka is the de-facto storage of choice for stream data
The log

next write

reader 1

reader 2
The log + pub-sub

publisher

subscriber 1

subscriber 2
#2: Kafka offers a scalable messaging backbone for application integration
Kafka messaging APIs: scalable EAI

produce(message) -> Messaging APIs
consume(message)
#3: Kafka enables building streaming data pipelines (E & L in ETL)
KAFKA’S CONNECT API: STREAMING DATA INGESTION

Diagram showing the process of data ingestion with Kafka’s Connect API, involving an app, messaging APIs, source, extract, load, and sink.
#4: Kafka is the basis for stream processing and transformations
Kafka’s Streams API: Stream Processing (transforms)
Kafka’s Connect API = E and L in Streaming ETL
Connectors!

Apps

NoSQL

RDBMS

Search

Apps

Monitoring

Hadoop

DWH

RT analytics
How to keep data centers in-sync?
Sources and Sinks

Extract

Connect API

Load

source

sink
CHANGELOGS
Transforming changelogs
Kafka’s Connect API = Connectors Made Easy!

- **Scalability**: Leverages Kafka for scalability
- **Fault tolerance**: Builds on Kafka’s fault tolerance model
- **Management and monitoring**: One way of monitoring all connectors
- **Schemas**: Offers an option for preserving schemas from source to sink
KAFKA ALL THE THINGS!
Kafka's Streams API = The T in Streaming ETL
Stream processing = transformations on stream data
2 visions for stream processing

Real-time Mapreduce VS Event-driven microservices
2 visions for stream processing

Real-time Mapreduce VS Event-driven microservices

- Central cluster
- Custom packaging, deployment & monitoring
- Suitable for analytics-type use cases

- Embedded library in any Java app
- Just Kafka and your app
- Makes stream processing accessible to any use case
Vision 1: real-time mapreduce
Vision 2: event-driven microservices => Kafka’s streams API
Kafka's Streams API = Easiest way to do stream processing using Kafka
#1: Powerful and lightweight Java library; need just Kafka and your app
#2: Convenient DSL with all sorts of operators: join(), map(), filter(), windowed aggregates etc
Word count program using Kafka’s streams API

```java
KStreamBuilder builder = new KStreamBuilder();
KStream<String, String> textLines = builder.stream(StringDeserializer, StringDeserializer, "TextLinesTopic");

KStream<String, Long> wordCounts = textLines
    .flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\W+")))
    .map((key, value) -> new KeyValue<>(value, value))
    .countByKey(StringSerializer, LongSerializer, StringDeserializer, LongDeserializer, "Counts")
    .toStream();

wordCounts.to("WordsWithCountsTopic", StringSerializer, LongSerializer);

KafkaStreams streams = new KafkaStreams(builder, config);
streams.start();
```
#3: True event-at-a-time stream processing; no microbatching
#4: Dataflow-style windowing based on event-time; handles late-arriving data
#5: out-of-the-box support for local state; supports fast stateful processing
EXTERNAL STATE
LOCAL STATE
Fault-tolerant local state
#6: Kafka's Streams API allows reprocessing; useful to upgrade apps or do A/B testing
REPROCESSING
Real-time dashboard for security monitoring
Kafka’s Streams API: Simple is Beautiful

Vision 1

Vision 2
Logs unify batch and stream processing
Streams API

app

New shiny future of ETL: Kafka
A GIANT MESS!
All your data ... everywhere ... now

Streaming platform

request-response

messaging

stream processing

streaming data pipelines

changelogs
VISION: ALL YOUR DATA ... EVERYWHERE ... NOW

Streaming platform

App

App

App

App

request-response

messaging

OR

stream

processing

change

Apps

DWH

Hadoop

streaming data pipelines

confluent
Thank you!

@nehanarkhede