Building Event Driven Services with Apache Kafka and Kafka Streams

Ben Stopford
@benstopford
There is a book!

Event Driven Architectures
Event Sourcing (DDD)

Stream Processing
Today’s ecosystems get pretty big

- 2.2 trillion messages per day (6 Petabytes)
- Up to 400 Microservices per cluster.
- 20-200 Brokers per cluster
Today’s ecosystems get pretty big

- 1 billion messages per day
- 20,000 messages per second
- 100 teams
Event Driven Architectures
Event Sourcing (DDD)

Events

Stream Processing
Notification

Data replication
Streaming Platforms

Kafka Streams / KSQL

Serving Layer (Cassandra etc.)

High Throughput Messaging

Data is embedded in each engine

Clustered Java App
Streaming Pipeline

(a) Apps Opened, Per App, Per day

opened_per_day

(b) Crashes, Per App, Per day

app_crashes

crashed_per_day

(c) Unstable Applications

unstable_apps

apps_opened
Streaming Platforms

KAFKA

Kafka Streams / KSQL

Data is embedded in each engine

High Throughput Messaging

Serving Layer (Cassandra etc.)

Clustered Java App
Streaming Platforms

KAFKA

Kafka Streams / KSQL

High Throughput Messaging

Data is embedded in each engine

Clustered Java App

Serving Layer (Cassandra etc.)
An event log is a simple idea

Messages are added at the end of the log
Readers have a position all of their own

Old

Fred is here  Scan

New

George is here  Scan

Sally is here  Scan
You can rewind and replay, just like Tivo!
The hard part: Tying it all together!
Many "logs" over many machines
Resistant to Failure

Producing Services

Kafka

Consuming Services
Streaming Platforms

KAFKA

High Throughput Messaging

Data is embedded in each engine

Kafka Streams / KSQL

Clustered Java App

Serving Layer (Cassandra etc.)
Streaming Example

apps_opened → changelog → opened_per_day
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
```
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
CREATE TABLE opened_per_day AS
SELECT app_id, count(*)
FROM apps_opened
WINDOW TUMBLING (SIZE 1 DAY)
GROUP BY app_id;
Streaming is manipulating events in flight, at scale.
Event Driven Architectures

Stream Processing
Increasingly we build ecosystems
The Problem is DATA
Most services share the same core facts.

Most services live in here
Events have two hats

Notification

Data replication
Buying an iPad (with REST/RPC)
Events for Notification Only

Notification

Webserver

Submit Order

Orders Service

Order Created

Shipping Service

Customer Service

getCustomer()

KAFKA

REST

confluent
Pluggability

Webserver

Submit Order

Orders Service

Shipping Service

Customer Service

getCustomer()

REST

Kafka

Repricing

Orders

payments

Notification

Order Created
Events for Data Locality

Data is replicated

Webserver

Orders Service

Shipping Service

Customer Service

Submit Order

Order Created

Order Updated

KAFKA
Events have two hats

- Notification
- Data replication
Stateless / Stateful Stream Processing
Relates to these hats
Stateless Stream Processing

Notification

Submit Order

Orders Service

Kafka Streams

Shipping Service

Webserver

Customer Service

Rest/RPC

getCustomer()

Order Created

KAFKA
Stateful Stream Processing

Data replication
KSQL ~ KStreams
Orders

Any code you desire

Customers

Kafka Streams API

Table
It's just java

builder.stream("Orders")
  .join("Customers",...)
  .transform((key, value) ->
    {
      //Any code your heart desires!
    }
  )
  .to("Shipments");
Streaming can be stateful or stateless.

1. Joining & Operating on Streams

2. Joining & Operating on Materialized Tables
Streaming Platform

Producer

Consumer

Connectors

The Log

Connectors

Streaming Engine

KAFKA

confluent
Event Driven Example
1. Use events to decouple and to collaborate
Event Collaboration

Browser

Webserver

Orders Service

Order Received
Order Validated
Order Completed

KAFKA

Notification
2. Use the Single Writer Principal
State changes to a topic owned by one service

Browser

Webserver

Orders Service

Order Received

Order Validated

Order Completed

KAFKA
Local consistency points in the absence of Global Consistency
3. Convert legacy databases to Events (with CDC)
Make Legacy Datasets Available via the Log

Browser

Webserver

Orders Service

Order Received
Order Validated
Order Completed

Connect

Products

KAFKA
4. Use Kafka as an Event Store
Product Catalogue stored in 3 places

- Browser
- Webserver
- Orders Service
- Product Catalogue stored in 3 places

Reporting view may be “thinner”
5. Derive “Materialized Views” instead of caching
Materialize Stock ‘View’ Inside Service

Browser

Webserver

Orders Service

Stock

Order Received

Order Validated

Order Completed

Products

KAFKA

Data Replication

Connect

Browser

Webserver

Orders Service

Stock

Order Received

Order Validated

Order Completed

Products

KAFKA

Data Replication

Connect
Is stateful a good idea?

- Standby Replicas
- Disk Checkpoints
- Compacted topics
- Or just use a database!
Database Inside Out Pattern

KAFKA: Retained Event Streams

Payments
Orders
Products
Customers

Queries run locally (in database or streaming engine)
6. Write to State Stores, just like a local ‘database’, backed up in Kafka
State stores behave like local databases

Browser

Webserver

Orders Service

Reserved Stocks

Reserved Stocks

State Store

Browser

Webserver

Connect

Orders

Reserved Stocks

Order Received

Order Validated

Order Completed

Products

Stock

Kafka
7. Use Transactions to tie All Interactions Together
8. Evolve and Grow through Streaming Functions
Tiered Contexts

Finance

Operations

Front Office
Span regions or clouds
Handle Disconnectedness
So...
Optimize for complexity vs optimize for scale

Event Driven Architectures

Stream Processing
Events provide the key to evolutionary architectures

Notification

Data replication
Spectrum of use cases

- Finer Grained, Collaborative, Connected
- Coarser Grained, Non-collaborative, Disconnected

- Notification
- Data Replication
Events to transcend individual services
Start Simple and Evolve

1. Broadcast events
2. Retain them in the log
3. Evolve the event-stream with streaming functions
4. Recasting the event stream into views when you need to query.
Find out more


**Software:** [https://confluent.io/download/](https://confluent.io/download/)

**Cloud:** [https://www.confluent.io/confluent-cloud/](https://www.confluent.io/confluent-cloud/)

**Kubernetes Operator:** [https://www.confluent.io/confluent-operator/](https://www.confluent.io/confluent-operator/)


**Twitter:** @benstopford