WiXEngineering

Battle-tested event-driven patterns for your microservices architecture

Natan Silnitsky
Backend Infra Developer, Wix.com

During the past couple of years, I've implemented or have witnessed implementations of several key patterns of event-driven messaging designs on top of Kafka that have facilitated creating a robust distributed microservices system at Wix that can easily handle increasing traffic and storage needs with many different use-cases. In this talk I will share these patterns with you, including: * Consume and Project (data decoupling) * End-to-end Events (Kafka+Websockets) * In memory KV stores (consume and query with 0-latency) * Events transactions (Exactly Once Delivery)

At Wix

>200M registered users (website builders) from 190 countries

5% of all Internet websites run on Wix

2,000 Microservices

1.5B Kafka messages produced / Day

Agenda

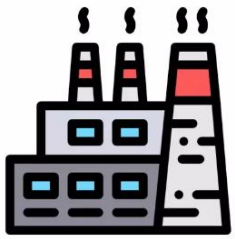
Event-driven microservices

Event-driven patterns:

- Consume and Project
- Event-driven from end to end
- 0-latency KV Store
- Events in Transactions

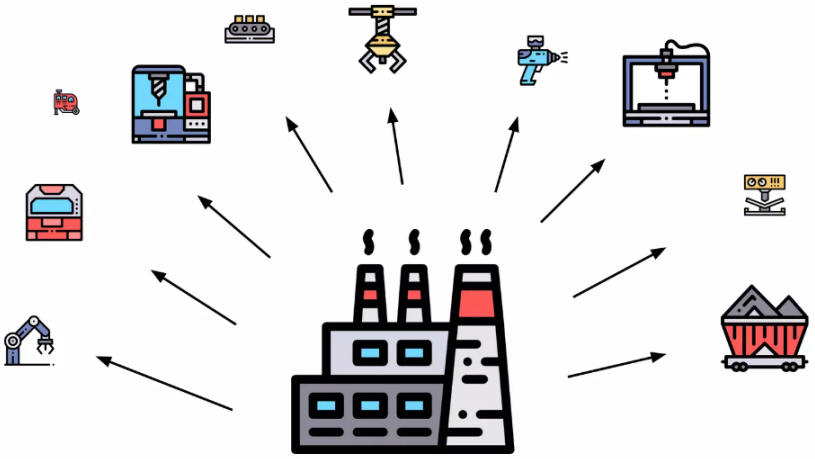
Microservices

The Monolith



Microservices

Monolith to Distributed **Microservices** System



@NSilnitsky

* Separately

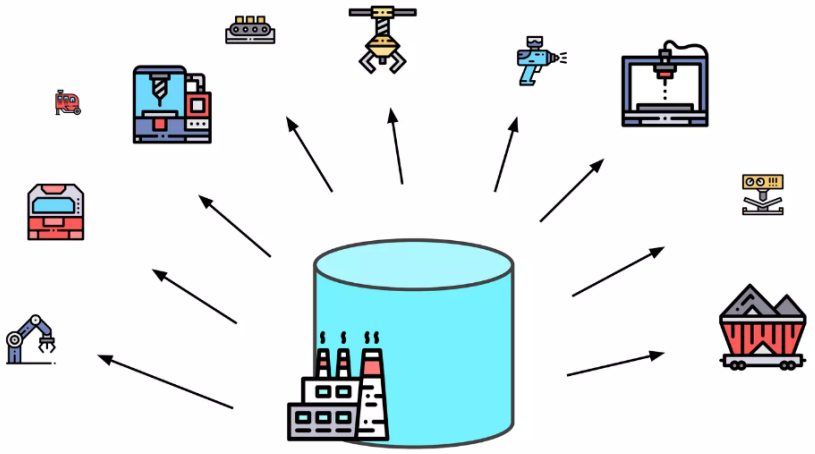
Microservices

Decoupled Microservices

Mutable

Contention

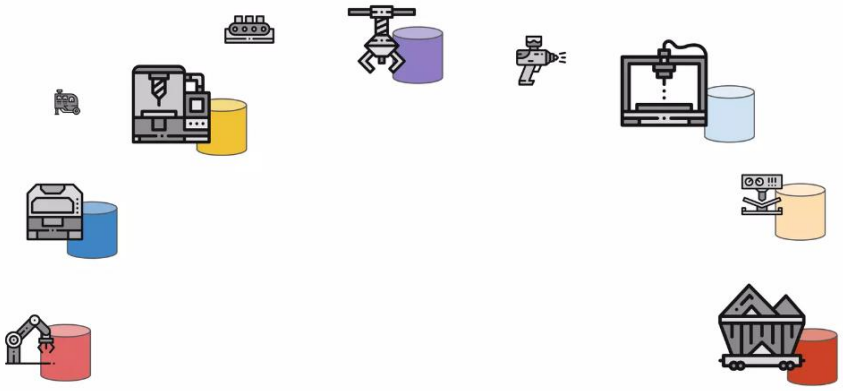
Choice



@NSilnitsky

Microservices

Data Encapsulation



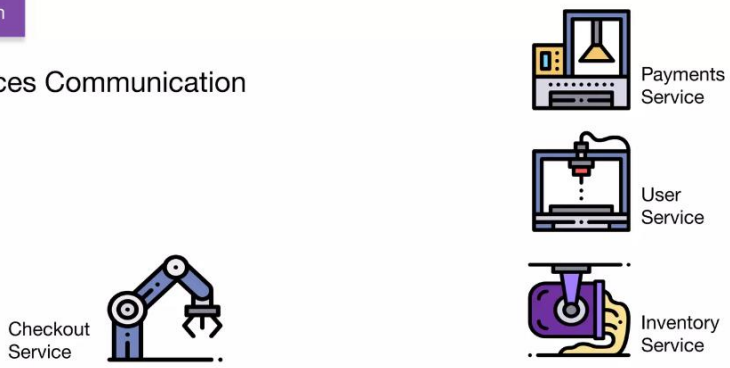
@NSilnitsky

* flexibility, schema

What about microservices communication?

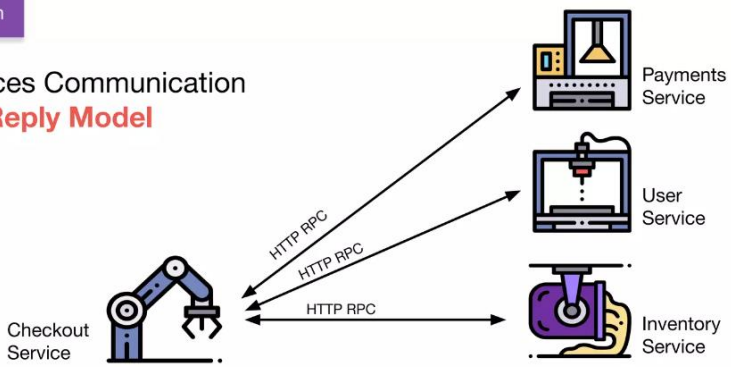
Request-reply communication

Microservices Communication



Request-reply communication

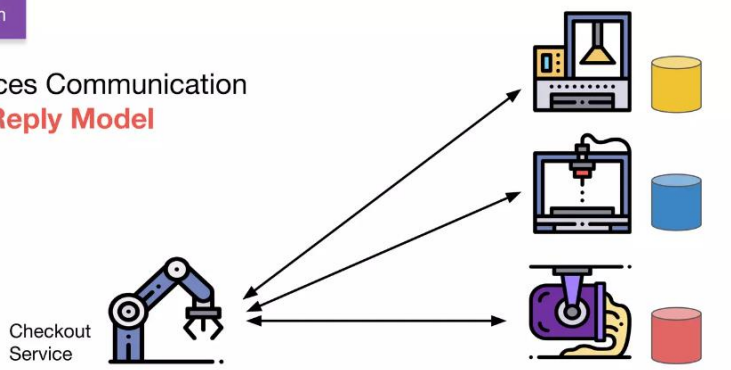
Microservices Communication
Request-Reply Model



Request-reply communication

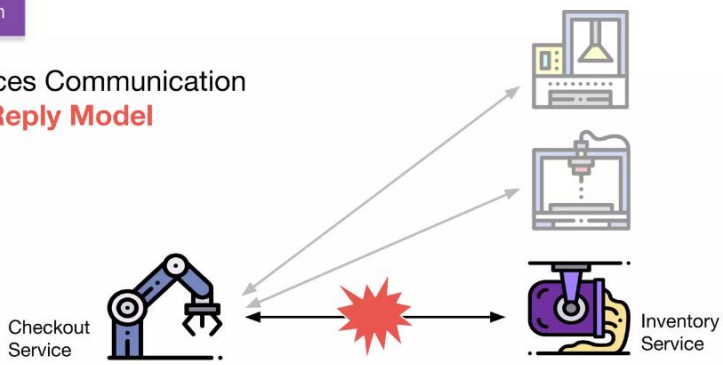
Microservices Communication
Request-Reply Model

* 1 by 1- slow



Request-reply communication

Microservices Communication
Request-Reply Model



Event-driven communications

Introduce a **message broker** in between services

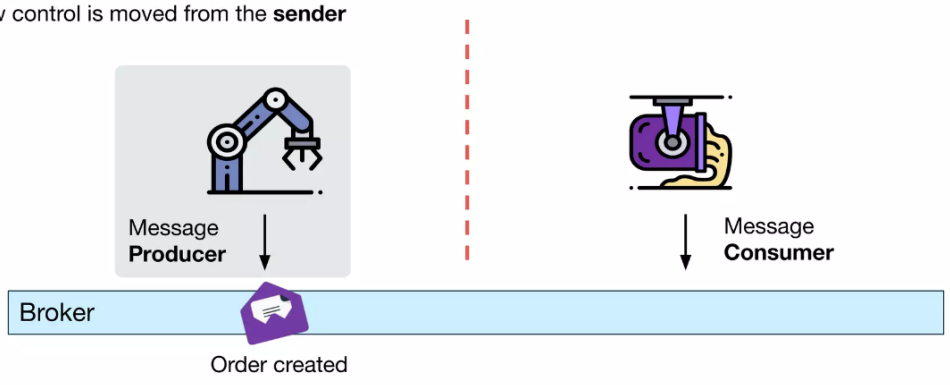
@NSilnitsky

* Reliability

Event-driven communications

Introduce a message broker in between services
Event-driven model

Flow control is moved from the **sender**



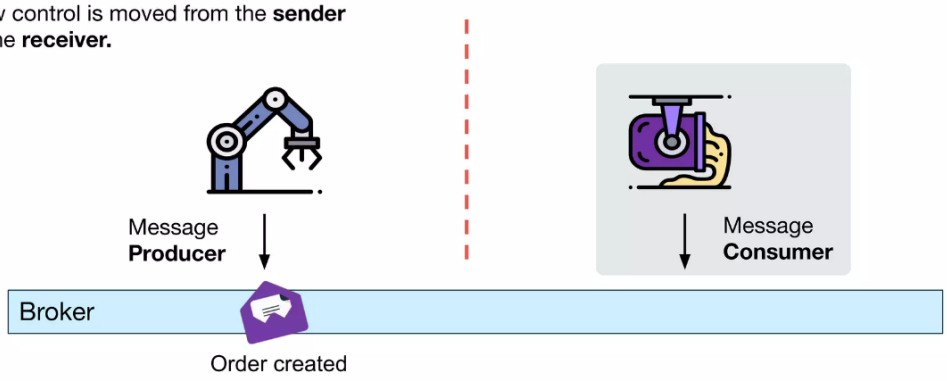
@NSilnitsky

* Not ineteract directly

Event-driven communications

Introduce a message broker in between services
Event-driven model

Flow control is moved from the **sender** to the **receiver.**

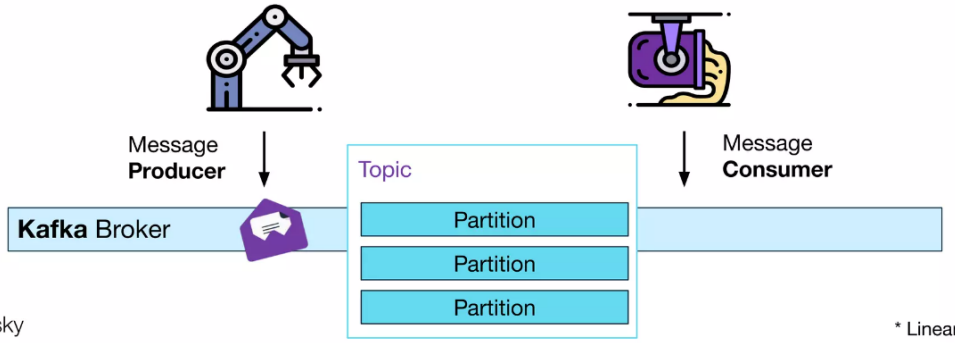


@NSilnitsky

* more robust

Event-driven communications

Introduce a message broker in between services
Distributed log broker

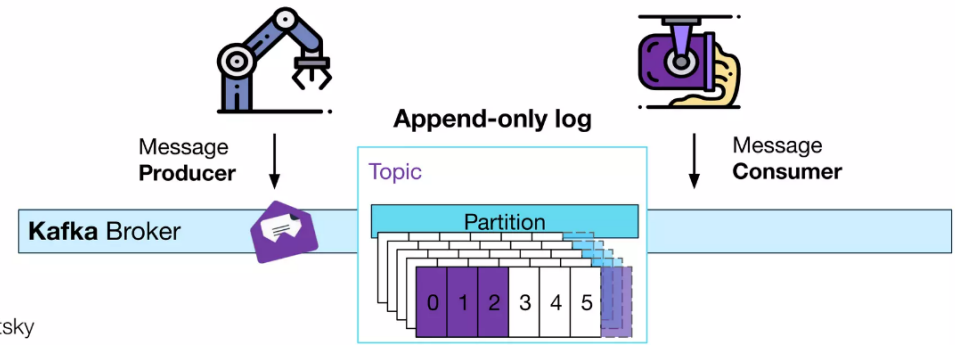


@NSilnitsky

* Linear scaling

Event-driven communications

Introduce a message broker in between services
Distributed log broker

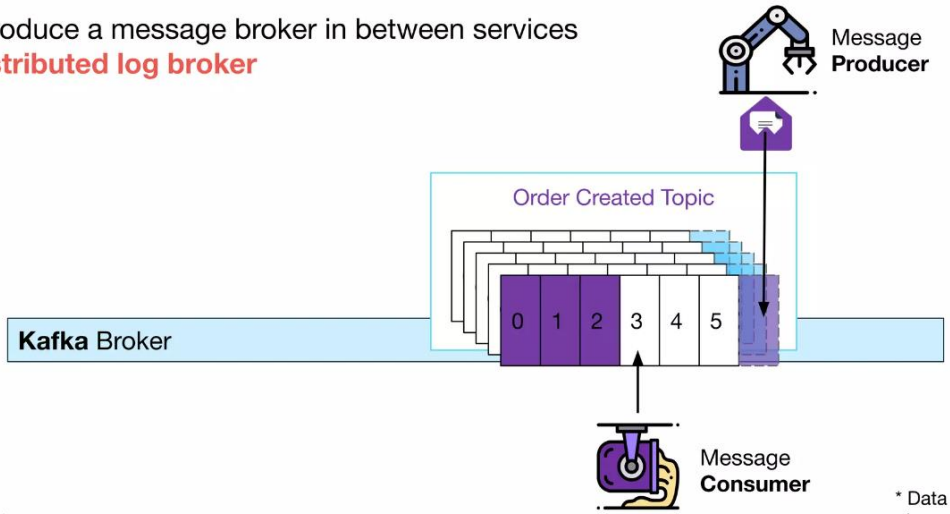


@NSilnitsky

Event-driven communications

Introduce a message broker in between services

Distributed log broker

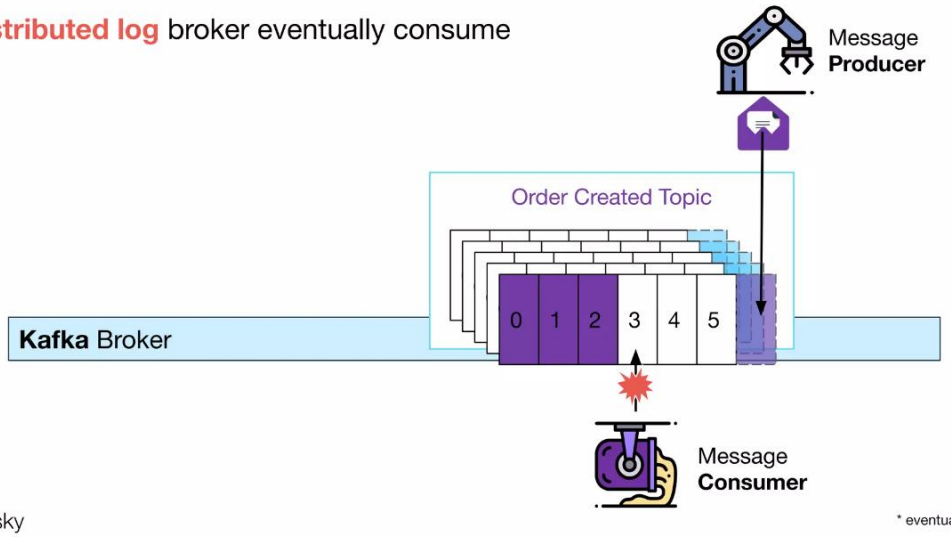


* Data retained, Immutable

@NSilnitsky

Event-driven communications

Distributed log broker eventually consume



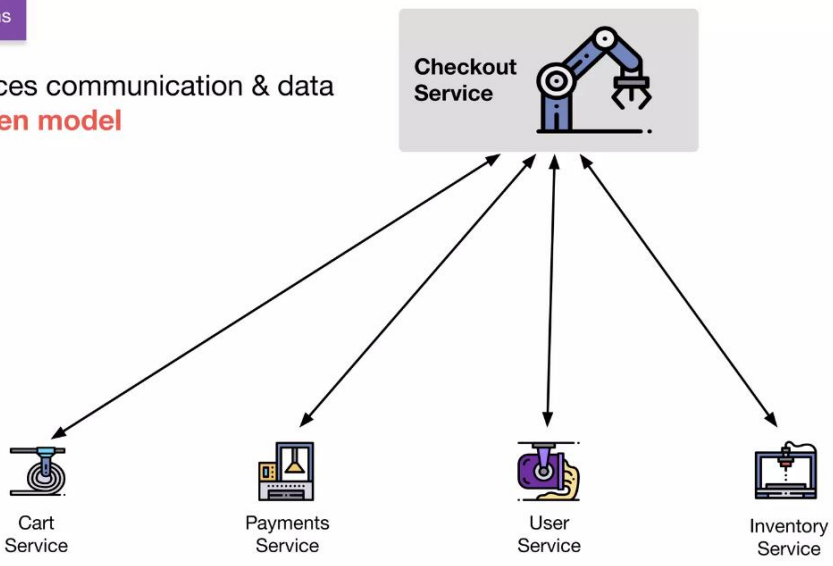
@NSilnitsky

* eventually consumed

Event-driven communications

Microservices communication & data

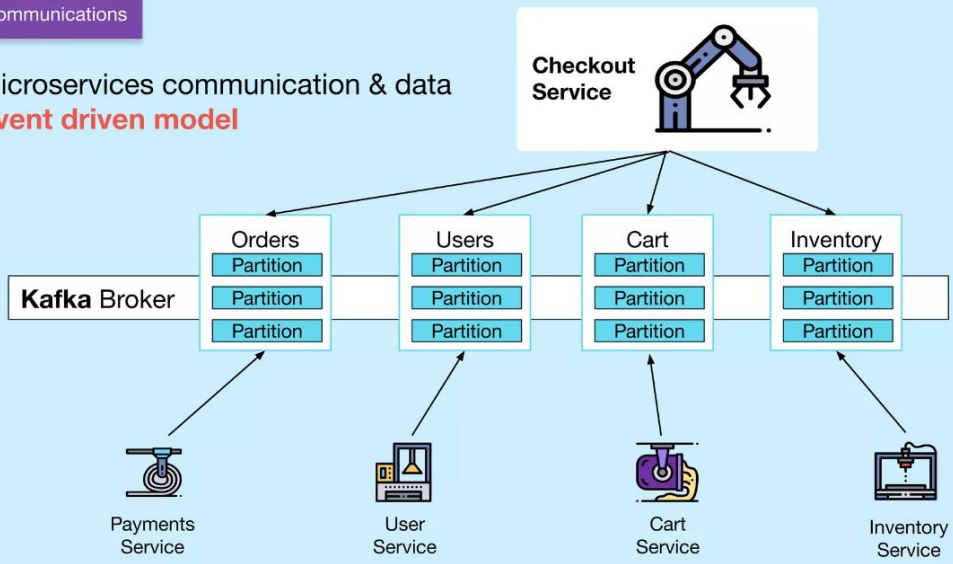
Event driven model



@NSilnitsky

Event-driven communications

Microservices communication & data
Event driven model



Agenda

Event-driven microservices

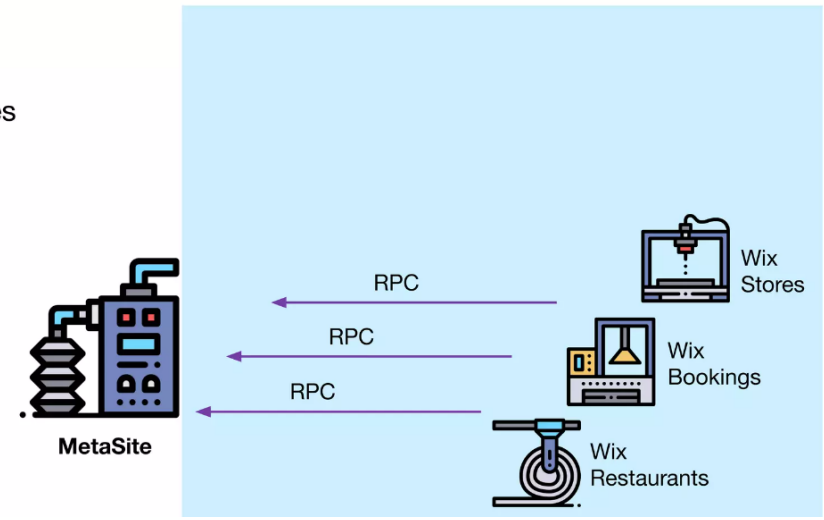
Event-driven patterns:

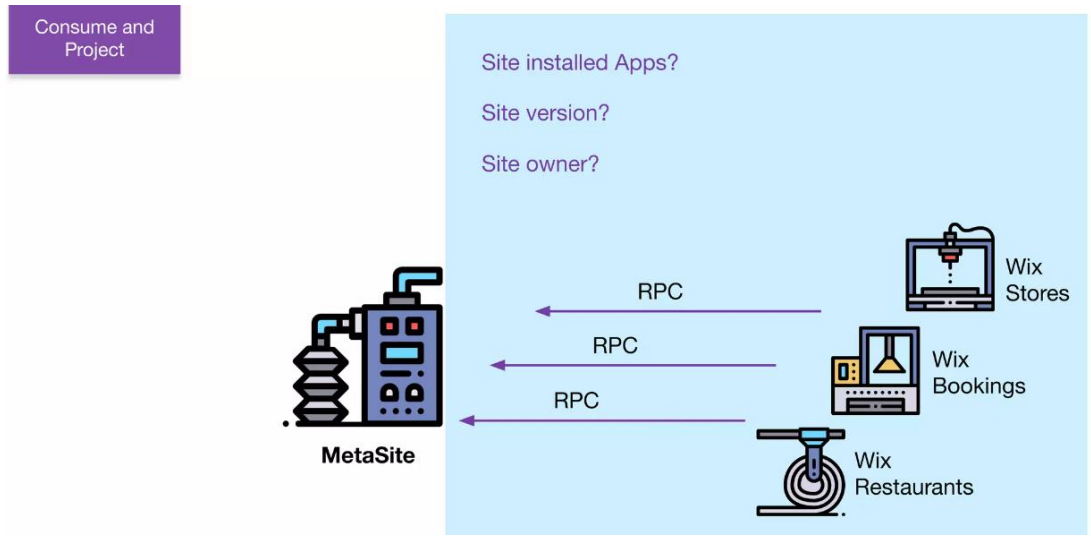
- Consume and Project
- Event-driven from end to end
- 0-latency KV Store
- Events in Transactions

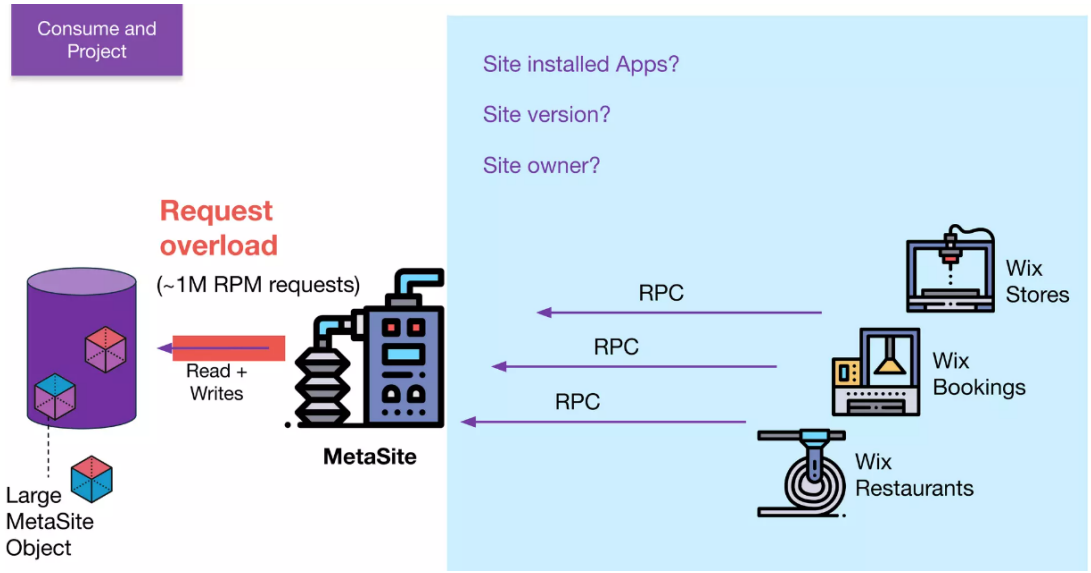
01 Consume and Project

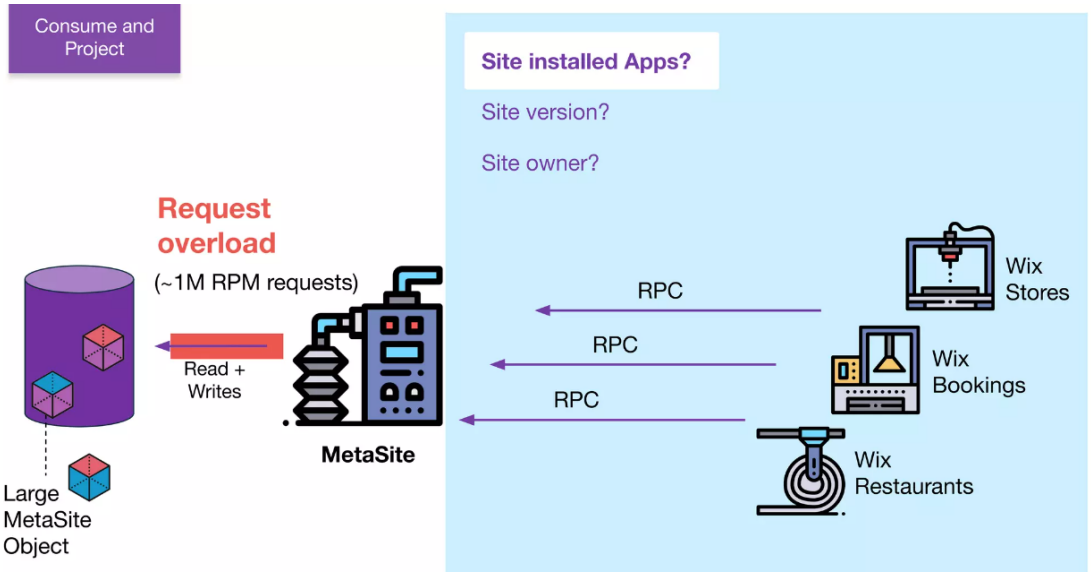
Consume and Project

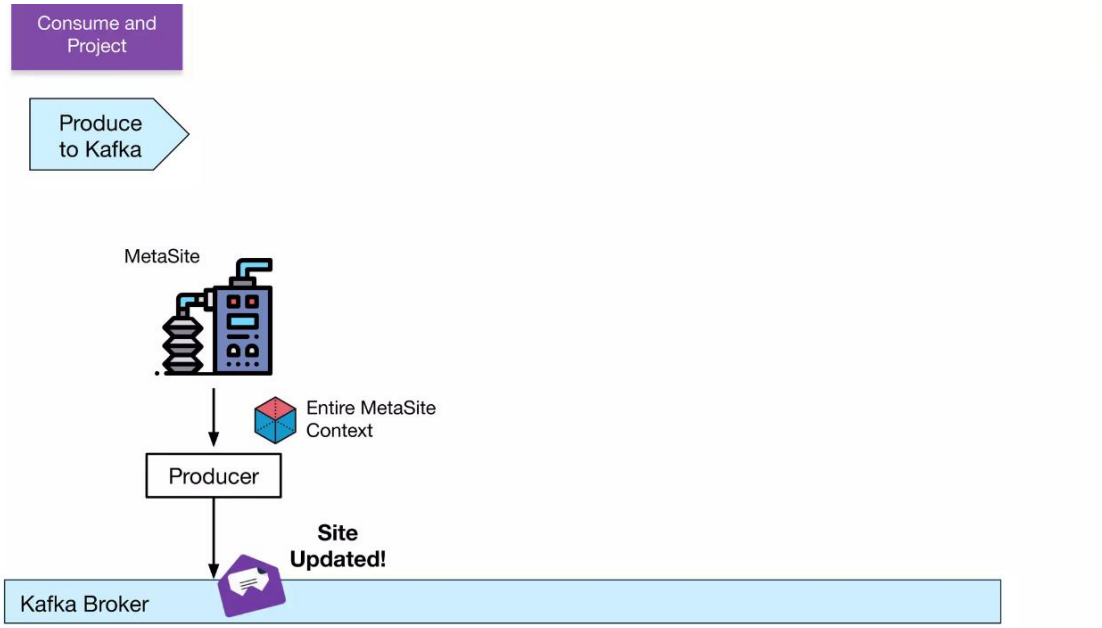
For Popular Services

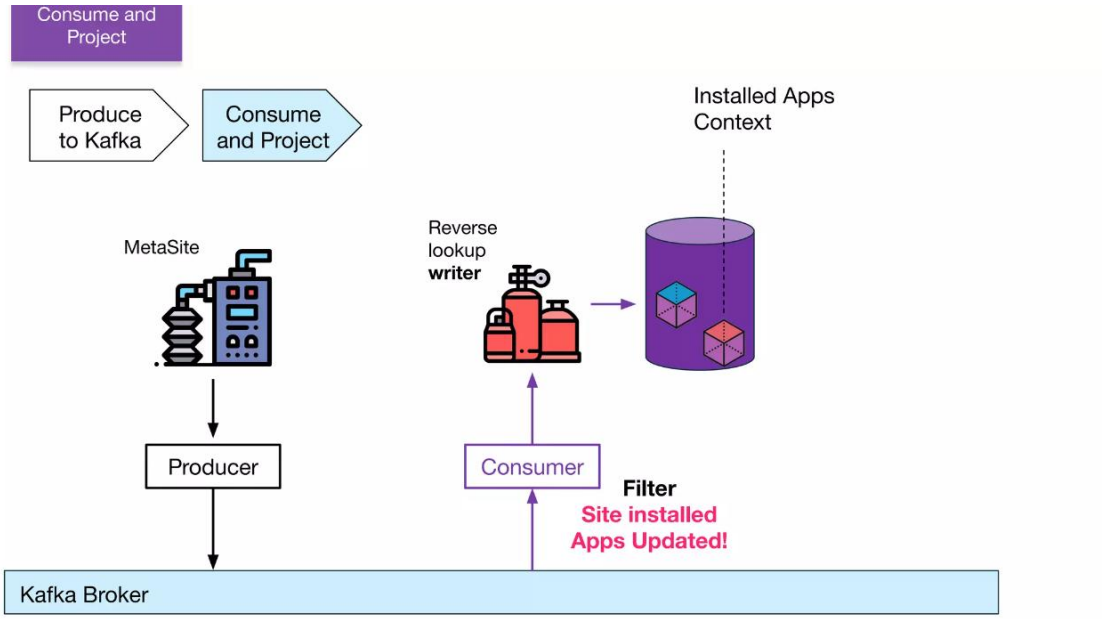


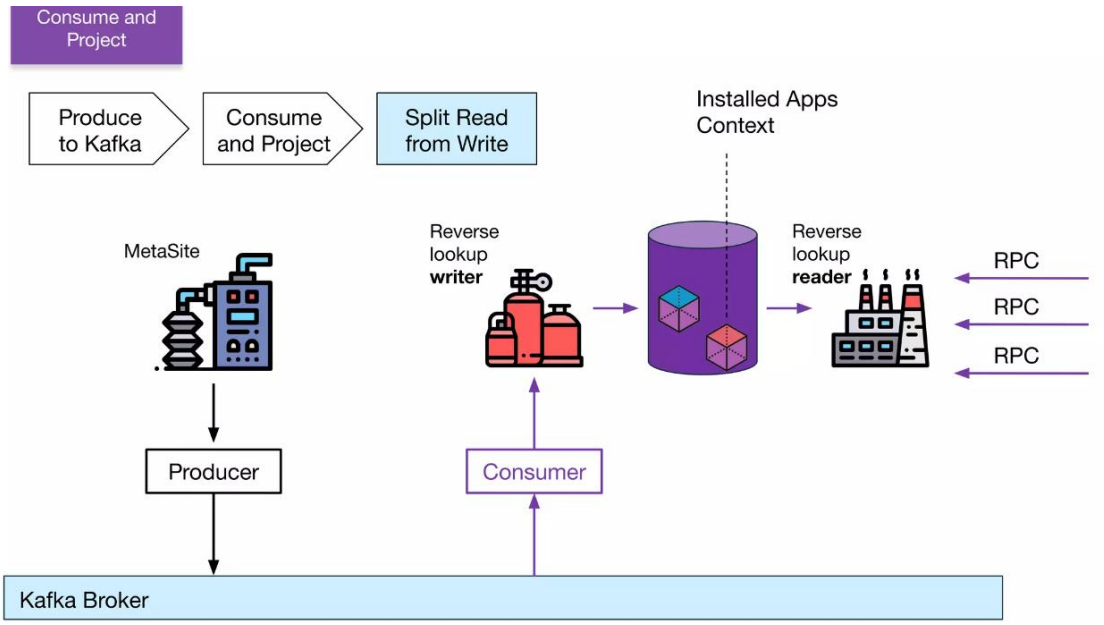

Consume and Project
Site installed Apps?
Site version?
Site owner?
MetaSite
RPC
RPC
RPC
Wix Stores
Wix Bookings
Wix Restaurants


Consume and Project
Request overload
(~1M RPM requests)
Read + Writes
Large MetaSite Object
MetaSite
Site installed Apps?
Site version?
Site owner?
RPC
RPC
RPC
Wix Stores
Wix Bookings
Wix Restaurants


Consume and Project
Request overload
(~1M RPM requests)
Read + Writes
Large MetaSite Object
MetaSite
Site installed Apps?
Site version?
Site owner?
RPC
RPC
RPC
Wix Stores
Wix Bookings
Wix Restaurants


Consume and Project
Produce to Kafka
MetaSite
Entire MetaSite Context
Producer
Site Updated!
Kafka Broker


Consume and Project
Produce to Kafka
Consume and Project
Installed Apps Context
MetaSite
Reverse lookup writer
Producer
Consumer
Filter
Site installed Apps Updated!
Kafka Broker


Consume and Project
Produce to Kafka
Consume and Project
Split Read from Write
Installed Apps Context
MetaSite
Reverse lookup writer
Reverse lookup reader
RPC
RPC
RPC
Producer
Consumer
Kafka Broker

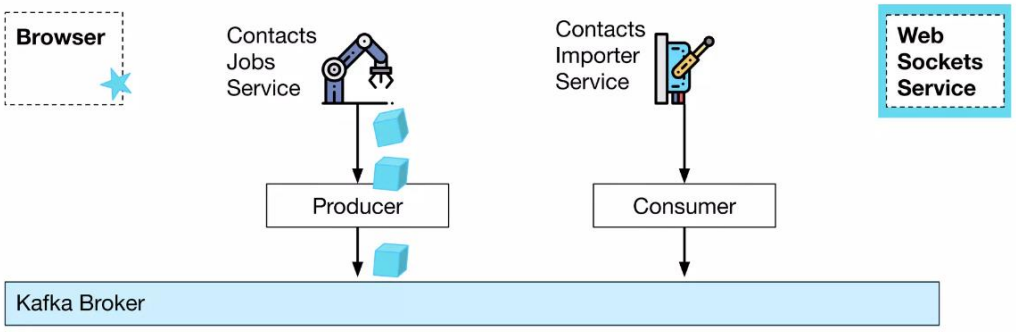
Consume and Project

- **Produce instead of serve** - massive load reduction
- **Project** - client-query-optimized “Materialized View”
- **Read/write split** - only scale read-only DB. more resilient

02 Event-driven from end to end

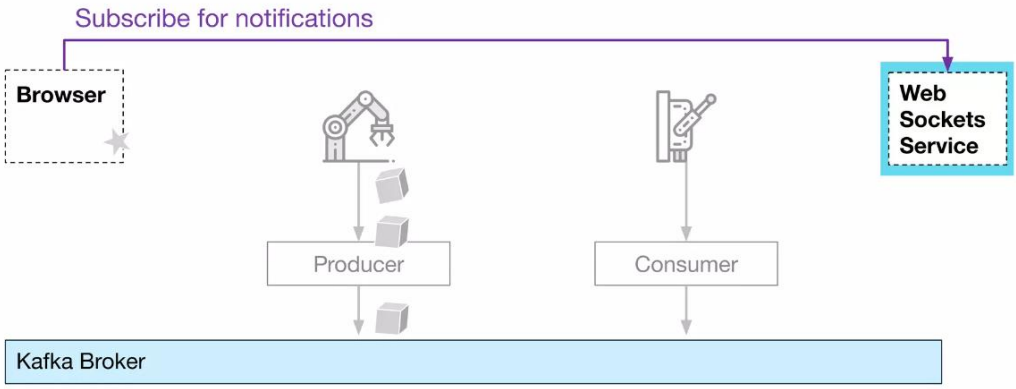
Event-driven from end to end

Use case:
Long-running async business process



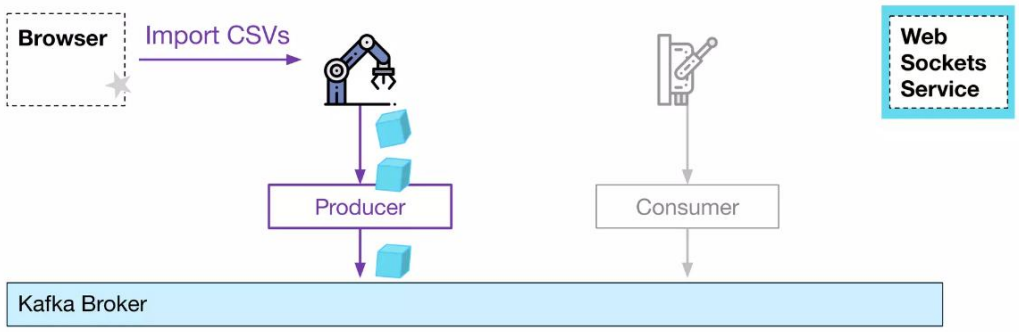
Event-driven from end to end

Use case:
Long-running async business process



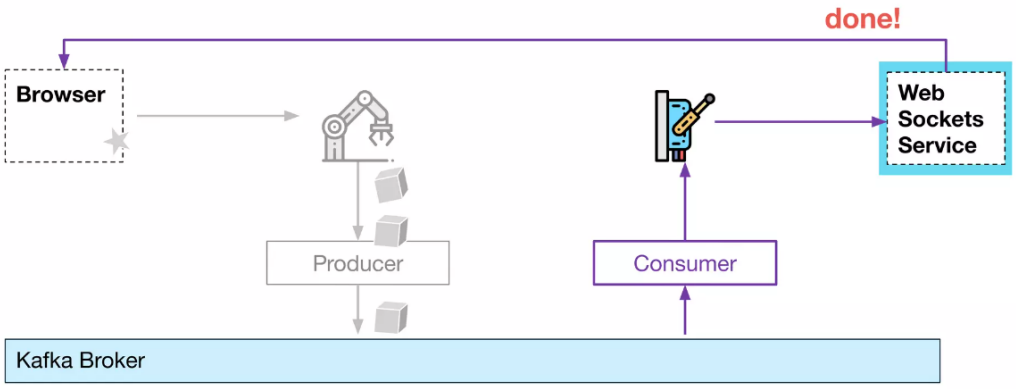
Event-driven
from end to end

Use case:
Long-running async business process



Event-driven
from end to end

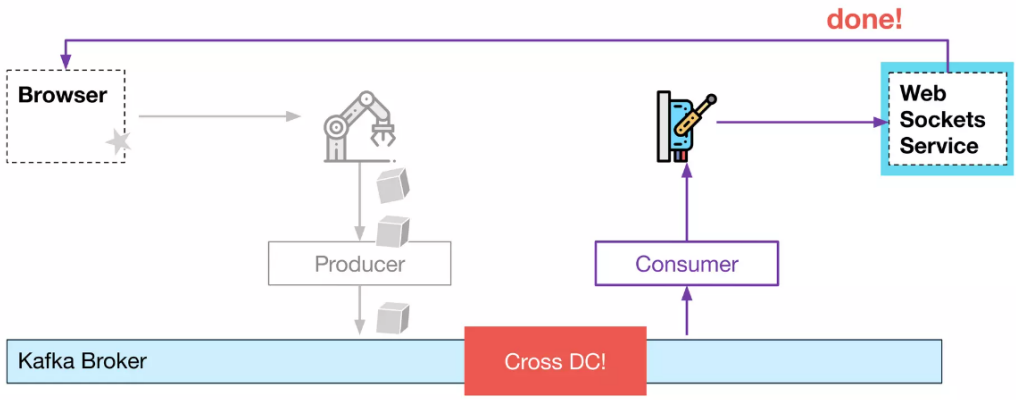
Use case:
Long-running async business process



* Distributed

Event-driven
from end to end

Use case:
Long-running async business process



* Distributed

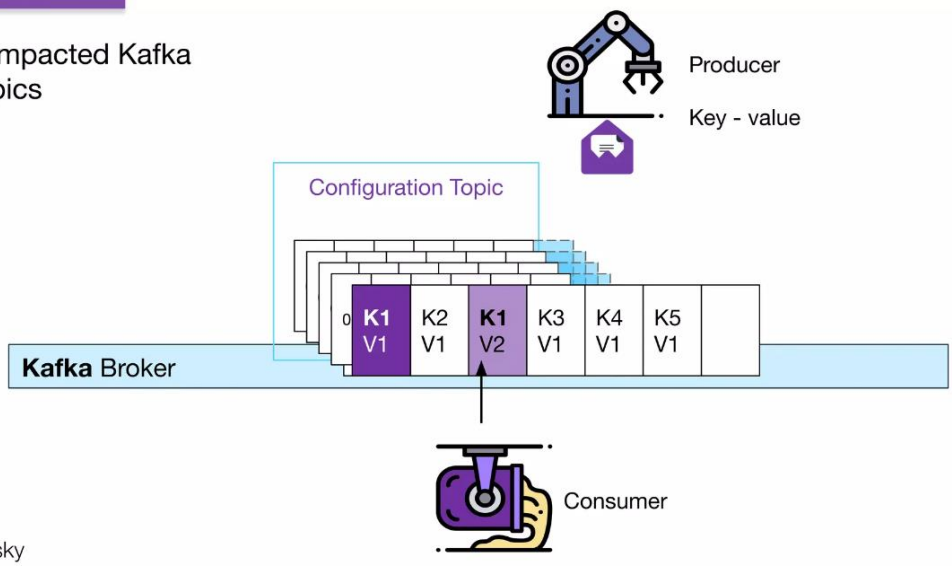
Event-Driven from End to End

- **No polling** - notify the user on any status change
- **Scalable** - spin up many import job workers
- **Replication** - cross-DC requests are easy to set up

03 0-latency KV Store

0-latency KV Store

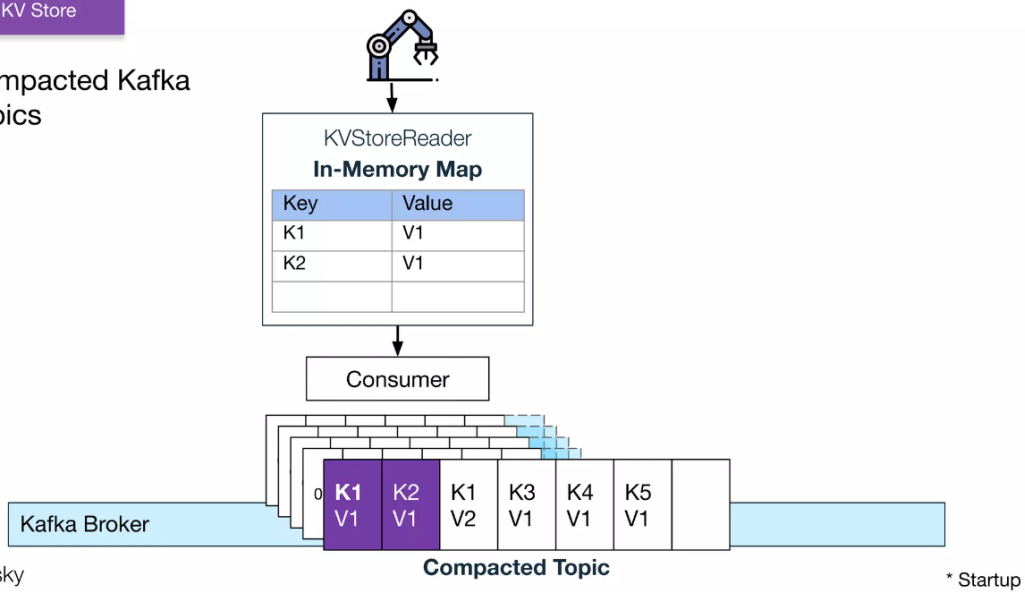
Compacted Kafka Topics



@NSilnitsky

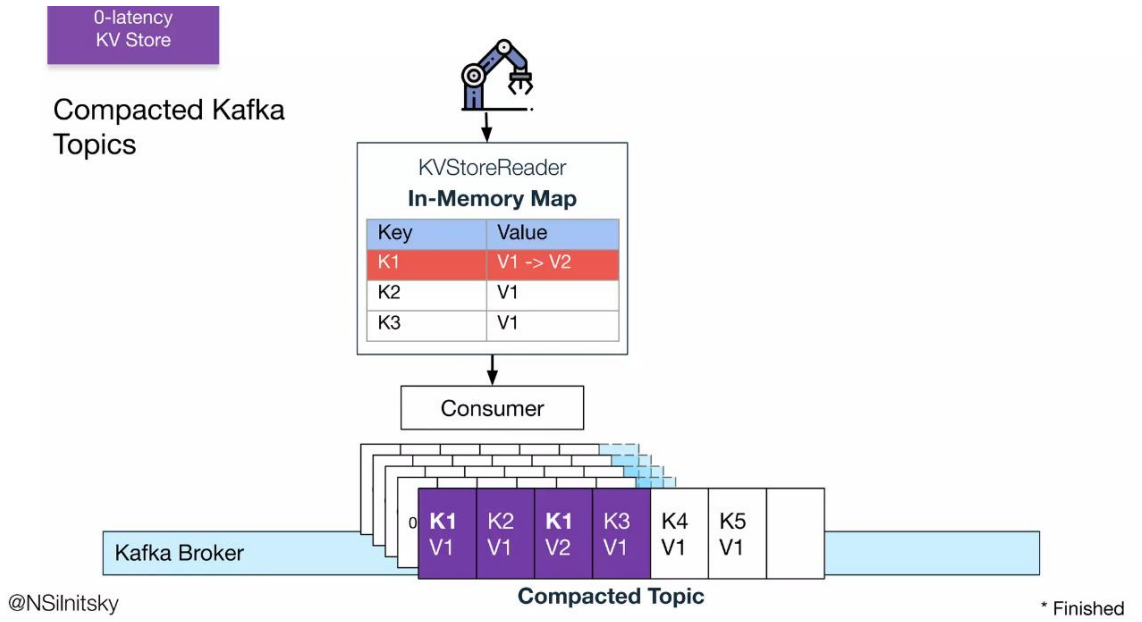
0-latency KV Store

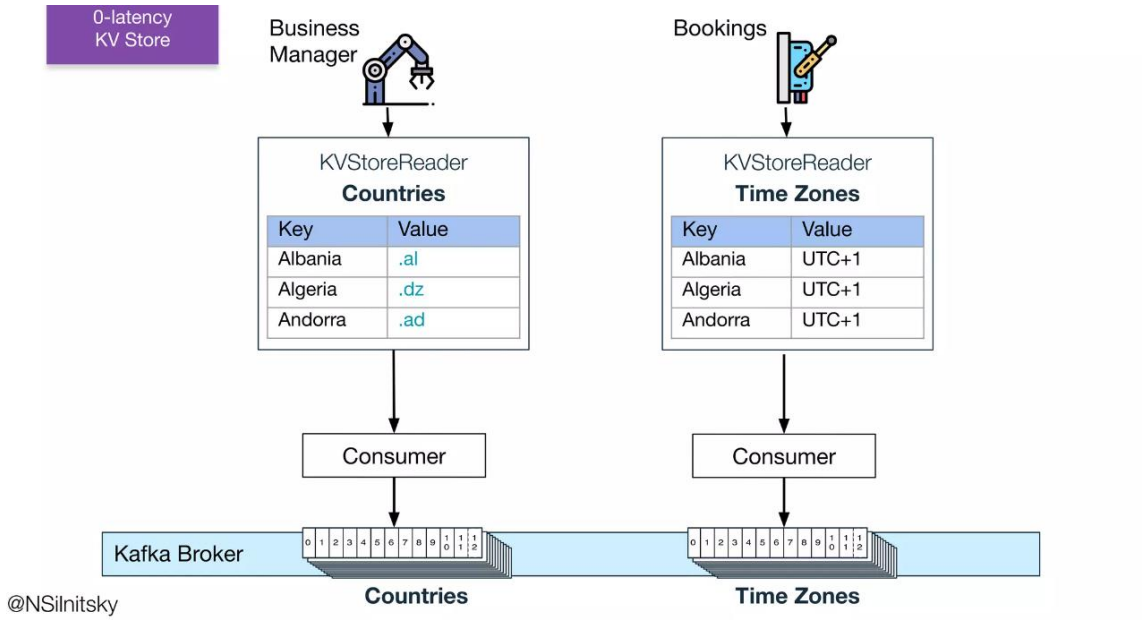
Compacted Kafka Topics

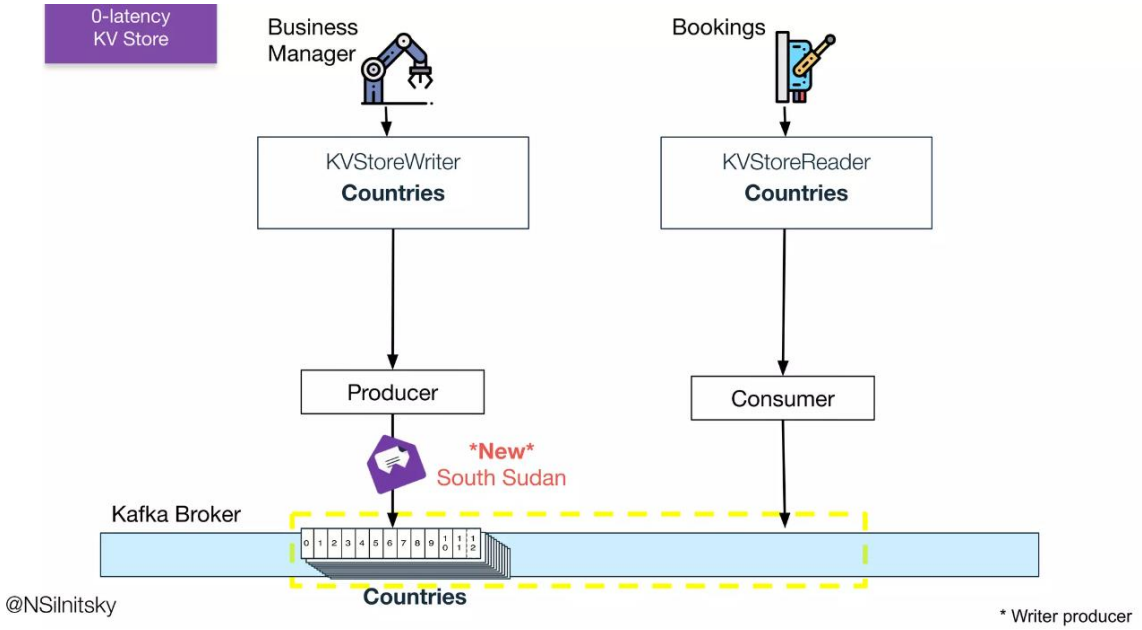


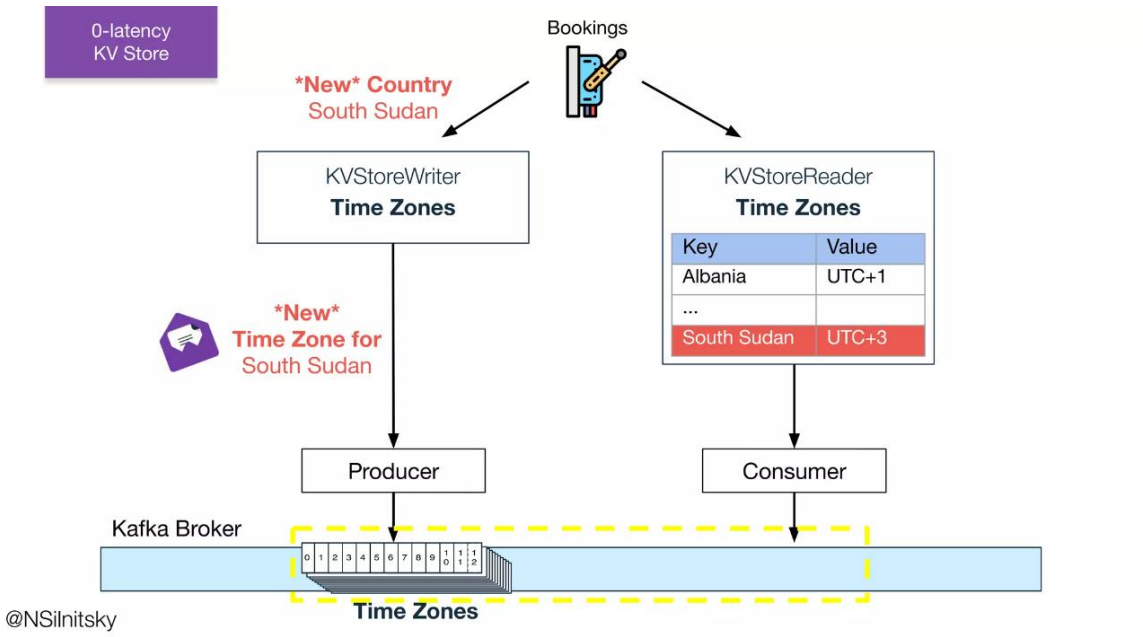
@NSilnitsky

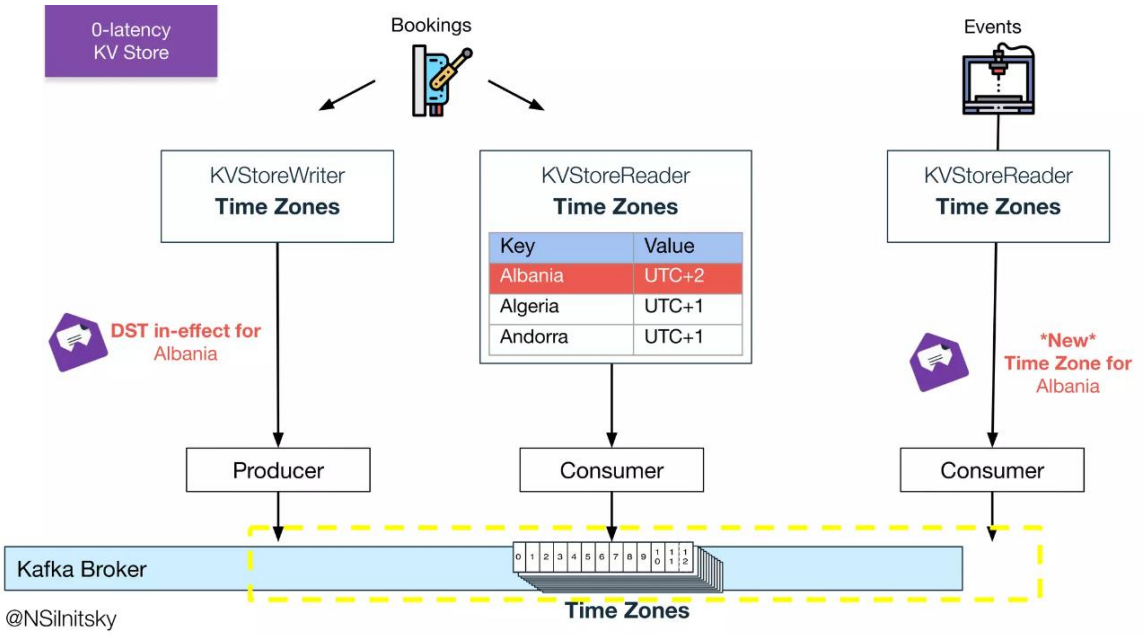
* Startup


0-latency
KV Store
Compacted Kafka
Topics
KVStoreReader
In-Memory Map
Key
Value
K1
V1 -> V2
K2
V1
K3
V1
Consumer
0
K1
V1
K2
V1
K1
V2
K3
V1
K4
V1
K5
V1
Kafka Broker
Compacted Topic
@NSilnitsky
* Finished


0-latency
KV Store
Business
Manager
Bookings
KVStoreReader
Countries
Key
Value
Albania
.al
Algeria
.dz
Andorra
.ad
KVStoreReader
Time Zones
Key
Value
Albania
UTC+1
Algeria
UTC+1
Andorra
UTC+1
Consumer
Consumer
Kafka Broker
Countries
Time Zones
@NSilnitsky


0-latency
KV Store
Business
Manager
Bookings
KVStoreWriter
Countries
KVStoreReader
Countries
Producer
Consumer
New
South Sudan
Kafka Broker
Countries
@NSilnitsky
* Writer producer





0-latency KV Store

- Loading the compacted topic to memory –

  a 0-latency kv store for dynamic configuration

- Compacted topic is still a topic –

  other services can consume its update events

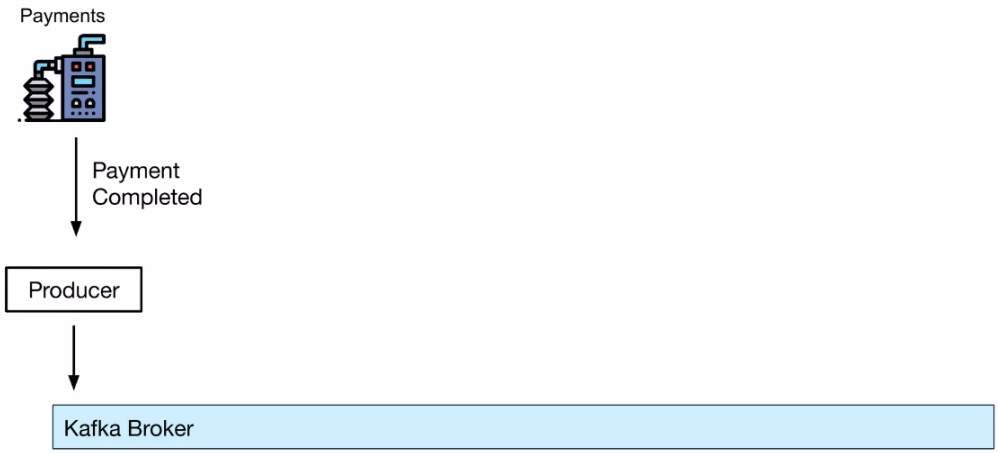
- Small Datasets - for large datasets, the topic should be

  streamed to a disk-based DB + In-memory Cache

04 Events in Transactions

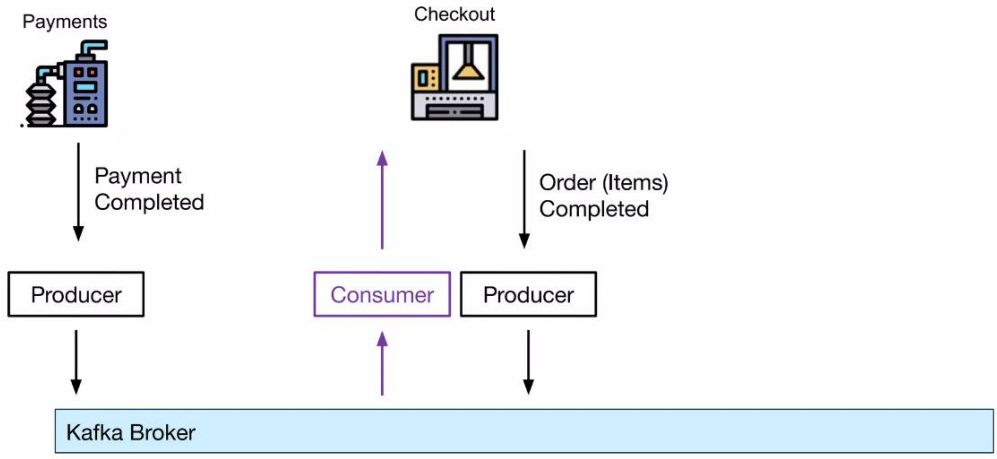
Events in Transactions

Classic Ecommerce Flow



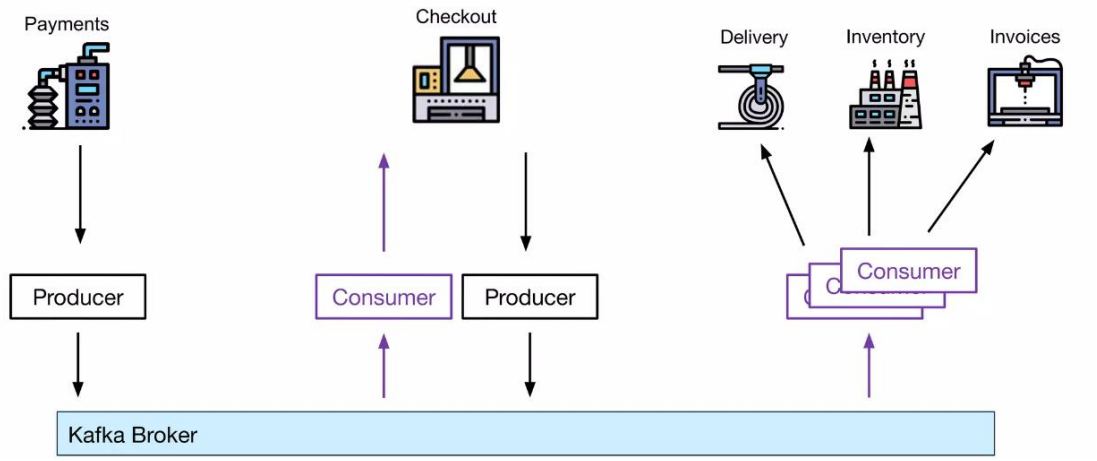
Events in Transactions

Classic Ecommerce Flow



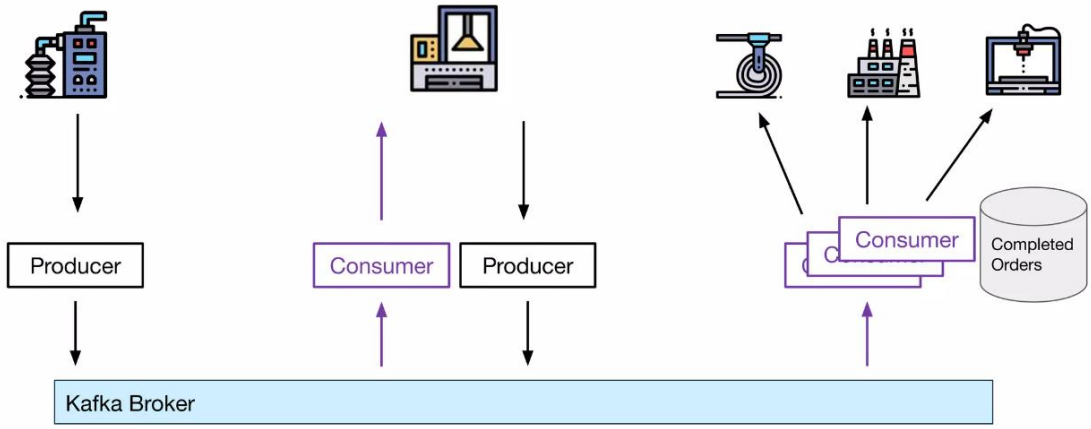
Events in Transactions

Classic Ecommerce Flow



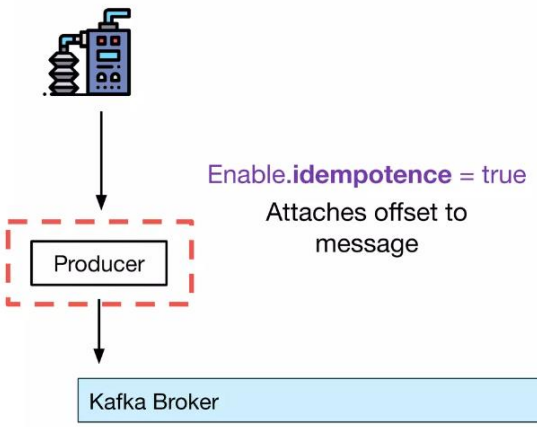
Events in Transactions

Classic Ecommerce Flow



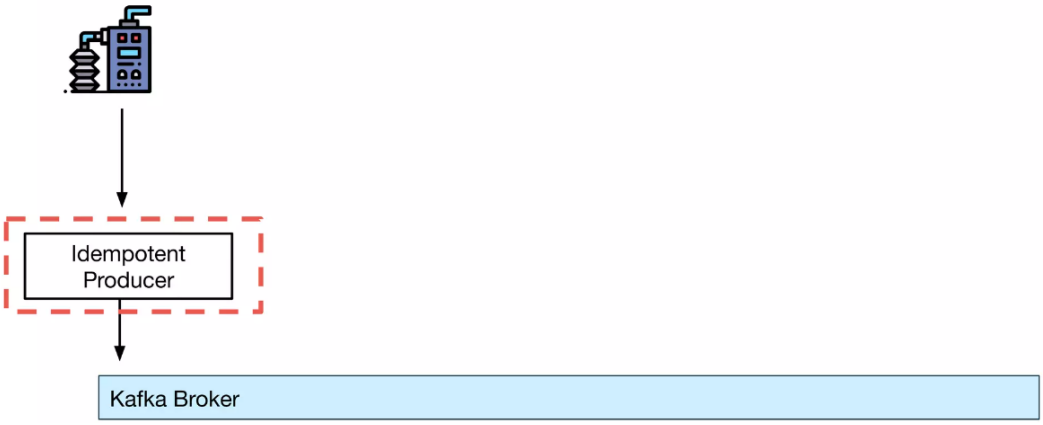
Events in Transactions

Classic Ecommerce Flow



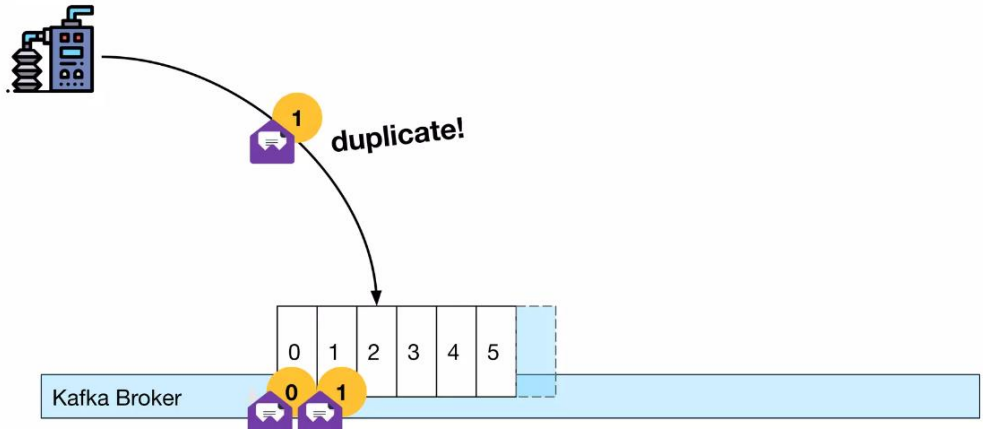
Events in Transactions

Classic Ecommerce Flow

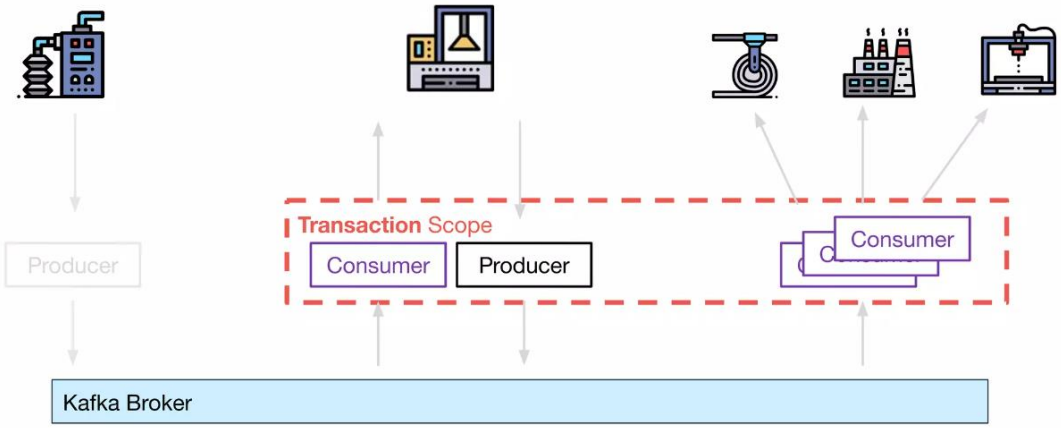


Events in Transactions

Payments
Idempotent Producer



Events in Transactions



Events in Transactions

- Events transactions are quite complex –

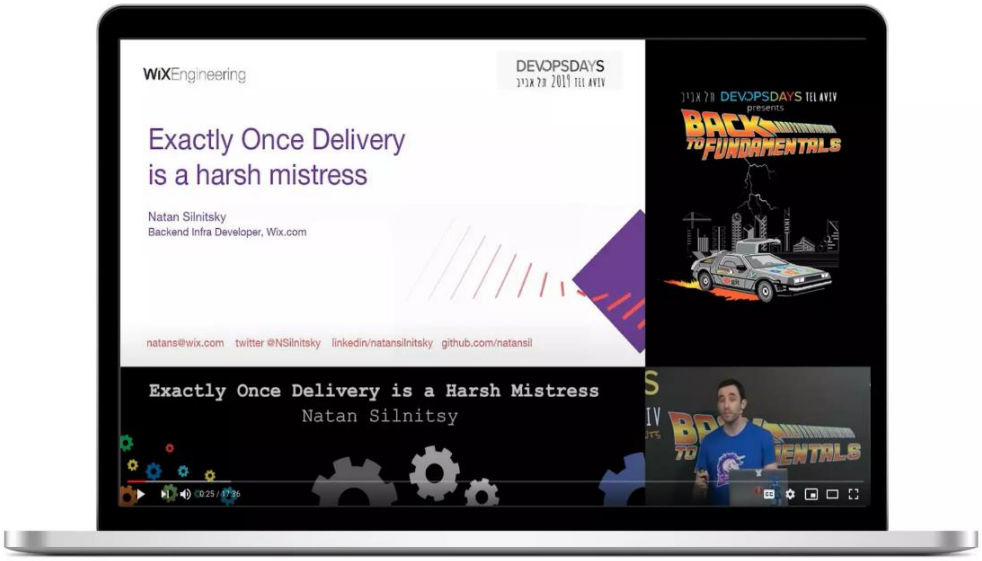
  More boilerplate, performance tuning via transaction size.

- DB updates/3rd party calls are not covered –

  You can dedup by using Kafka record offset as version

- Don't turn on transactions by default -

  Use in critical flows like payments processing



Wix developers have employed these event-driven patterns to make their microservices more **decoupled**, **resilient** and **scalable**.

Resources

6 Event Driven Architecture Patterns - Part 1 & Part 2 - by Natan Silnitsky

The Data Dichotomy: Rethinking the Way We Treat Data and Services - by Ben Stopford

Shared Database - by Chris Richardson

Exactly once delivery - DevOpsDay TLV 2019 - By Natan Silnitsky